SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Leonardo Jelenković

VIŠEDRETVENI UGRAĐENI SUSTAVI ZASNOVANI NA MONITORIMA

DOKTORSKA DISERTACIJA

Zagreb, 2005.

Doktorska disertacija je izrađena na:

Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave,
Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu.

Mentor: akademik Leo Budin

Doktorska disertacija ima 131 stranicu

Disertacija br.: _______

Povjerenstvo za ocjenu doktorske disertacije:

1. Dr.sc. Mario Žagar, red. prof. FER-a Zagreb
2. Dr.sc. Leo Budin, akademik red. prof. FER-a Zagreb
3. Dr.sc. Željko Hocenski, izv. prof. Elektortehnički fakultet Osijek

Povjerenstvo za obranu doktorske disertacije:

1. Dr.sc. Mario Žagar, red. prof. FER-a Zagreb
2. Dr.sc. Leo Budin, akademik red. prof. FER-a Zagreb
3. Dr.sc. Željko Hocenski, izv. prof. Elektortehnički fakultet Osijek
4. Dr.sc. Siniša Srbljić, red. prof. FER-a Zagreb
5. Dr.sc. Marin Golub, doc. FER-a Zagreb

Datum obrane disertacije: 30. studenog 2005.

Zahvale

Iskreno zahvaljujem mentoru akademiku Leu Budinu na vodstvu i pomoći koju mi pruža godinama, od izrade diplomskog rada, zaposlenju na FER-u, izrade magistarskog rada te sada i doktorske disertacije.

Zahvaljujem se i članovima ZEMRIS-a za pomoć i razumijevanje koje su mi pružali tijekom studija.

Posebnu zahvalu upućujem roditeljima koji su mi dali veliku podršku i pomoć tijekom školovanja.

SADRŽAJ

1. UVOD **1**

2. UGRAĐENI RAČUNALNI SUSTAVI **5**

2.1. Operacijski sustavi 5
2.2. Ugrađeni sustavi 6
2.3. Dretve u ugrađenim sustavima 9
2.4. Pregled izvedbe i mogućnosti nekih jezgri za ugrađene sustave 12
2.4.1. Operacijski sustav Windows CE 12
2.4.2. Operacijski sustav QNX Neutrino 13
2.4.3. Operacijski sustav VxWorks 5.4 14

3. MONITORI **16**

3.1. Semafori 16
3.2. Klasifikacija monitora 18
3.3. Uporaba monitora u ugrađenim sustavima 21
3.3.1. Problem inverzije prioriteta 21
3.3.2. Odabir monitora za ugrađene sustave 26

4. OSTVARENJE MONITORA **30**

4.1. Podatkovna struktura jezgre 30
4.2. Funkcije za ostvarenje monitora 31

5. SINKRONIZACIJA POMOĆU MONITORA **37**

5.1. Kritični odsječak 37
5.2. Sinkronizacija sustava zadataka 39
5.3. Problem proizvođača i potrošača 42
5.4. Problem čitača i pisača 44
5.5. Problem pušača cigareta 49
5.6. Problem pet filozofa 50

6. FUNKCIJE OPERACIJSKOG SUSTAVA OSTVARENE MONITORIMA **54**

6.1. Pristup ulazno-izlaznim napravama 55
6.2. Ostvarivanje kašnjenja 55
6.3. Raspoređivači dretvi 57
6.3.1. Raspoređivanje prema trenutku nužnog završetka 58
6.3.2. Raspoređivač kružnog posluživanja 60
6.4. Ostvarenje semafora 61
6.5. Redovi poruka 64

7. MODEL IZGRADNJE MONITORA ZA ARM7 ARHITEKTURU **66**

7.1. Programski model ARM7 procesora 66
7.2. Jezgrine funkcije ostvarene u strojnom jeziku 68
7.2.1. Aktiviranje dretve 70
7.2.2. Prihvat programskog prekida 70
7.2.3. Jezgrina funkcija `Ući_u_monitor()` *72*
7.2.4. Jezgrina funkcija `Izaći_iz_monitora()` *73*
7.2.5. Jezgrina funkcija `Uvrstiti_u_red_uvjeta()` *74*

7.2.6. Jezgrina funkcija `Osloboditi_iz_reda_uvjeta()` ... *75*
7.2.7. Prihvat prekida sata ... *76*
7.3. VELIČINA JEZGRINIH FUNKCIJA ... 79
7.4. TRAJANJE IZVOĐENJA JEZGRINIH FUNKCIJA ... 81
7.5. SKLOPOVSKA PODRŠKA OSTVARENJU MONITORA ... 83
8. UPORABA MONITORA ... 85
8.1. RUČNI UREĐAJ ... 85
8.1.1. Opis sustava ... 85
8.1.2. Ostvarenje sustava ... 85
8.2. UPRAVLJANJE RASPODIJELJENIM OBJEKTIMA ... 90
8.2.1. Opis sustava ... 90
8.2.2. Ostvarenje upravljanja ... 92
8.2.3. Mogućnosti proširenja ... 104
9. ZAKLJUČAK ... 105
DODATAK A – OSTVARENJE MONITORA U STROJNOM JEZIKU ARM7TDMI PROCESORA ... 107
POPIS LITERATURE ... 125
POPIS KORIŠTENIH OZNAKA I KRATICA

1. UVOD

Ugrađeni računalni sustavi postaju sve značajnije područje u znanosti i industriji. Zahvaljujući napretku poluvodičke tehnologije procesori, kao osnovne komponente upravljanja ugrađenih sustava, postaju sve brži uz istodobno manju potrošnju energije i manje dimenzije što izravno pogoduje njihovoj uporabi. Njihova uporaba se iz tih razloga sve više proširuje na nova područja. Ugrađeni sustavi čine ogromno područje od, primjerice, najjednostavnijih igračaka pa do složenih sustava upravljanja letjelicama. Prema podacima iz 2002. godine [2] proizvelo se 6,2 milijardi procesora od kojih je više od 98% namijenjeno ugrađenim sustavima.

Ovaj se rad ograničava samo na dio tog područja. Razmatraju se sustavi s minimalnim sklopovljem, tj. sustavi koji koriste jednostavnije izvedbe procesora te imaju ograničen spremnički prostor i koji za upravljanje koriste više dretvi kojima je potreban određeni mehanizam raspoređivanja, sinkronizacije i međusobne komunikacije. Korištenje gotovih rješenja u obliku operacijskih sustava u takvim slučajevima najčešće nije isplativo zbog njihova zahtjeva za velikim spremničkim prostorom.

U ovom se radu razmatra te potom i definira minimalni skup funkcija pomoću kojih je moguće zasnovati višedretvene ugrađene sustave. Jednostavnost funkcija trebala bi omogućiti lakše i brže shvaćanje njihova rada te, po potrebi, njihovu modifikaciju i proširivanje dodatnim mogućnostima. Minimalnim brojem funkcija trebalo bi se ograničiti zauzeće spremničkog prostora, a mogućnostima funkcija dozvoliti izgradnju složenih mehanizama, kao što su razne metode raspoređivanja, sinkronizacije i komunikacije, tamo gdje su takvi mehanizmi potrebni.

Ugrađeni su sustavi posebna skupina računalnih sustava kod kojih se, zbog njihovih svojstava kao što su ograničene dimenzije i masovna proizvodnja, pokušava smanjiti proizvodna cijena uređaja kako bi on bio konkurentan. Smanjenje cijene može se ostvariti i jednostavnošću sklopovlja te smanjivanjem potrebnog spremničkog prostora. Zbog toga se programi koje uređaj obavlja optimiraju obzirom na veličinu. Ukoliko je potrebno istovremeno upravljati s više zadataka, za koje se koriste različite dretve, potreban je i mehanizam koji će omogućiti potrebnu sinkronizaciju i raspoređivanje. Operacijski sustavi nude mnoštvo sinkronizacijskih i komunikacijskih funkcija. Uz to operacijski sustavi obično i upravljaju okolinom, prekidnim sustavom, spremničkim prostorom, odnosno u sebi imaju ugrađeno sučelje za upravljanje. Veličina spremničkog prostora koju zahtjeva operacijski sustav za svoje strukture podataka i programe može znatno premašiti veličinu koju zauzimaju primjenske dretve koje obavljaju zadaću uređaja. Korištenje operacijskih sustava u nekim bi slučajevima znatno povećalo potreban spremnički prostor, a samim time i cijenu uređaja. U ovom su radu razmatrani pokušaji koji vode do odstranjivanja operacijskog sustava s takvih uređaja, odnosno određivanja skupa funkcija dovoljnih za rad višedretvenog ugrađenog sustava. Promatrani su i postupci reduciranja jezgre operacijskog sustava kojima nije prvenstven cilj sama veličina jezgre već ubrzanje, sigurnost, odvajanje bitnih osnovnih funkcija ili nešto drugo.

Dosadašnji radovi na temu ostvarenja funkcija za raspoređivanje i međusobnu

sinkronizaciju i komunikaciju višedretvenog sustava uz minimalno zauzeće spremničkog prostora nastoje ostvariti određenu mini, mikro ili čak nanojezgru. Takva su rješenja, iako smanjenih mogućnosti, zadovoljavajuća u određenim primjenama. Jasno je da će sustav s minimalnom jezgrom, samo s osnovnim skupom funkcija, u slučajevima korištenja složenijih metoda međusobne sinkronizacije i komunikacije dretvi biti sporiji od klasičnih jezgri. Složene će se metode u tom slučaju morati ostvariti skupom odabranih funkcija dok su u standardnim (monolitnim) jezgrama te funkcije izravno ugrađene i optimirane. U nekim je slučajevima ipak mnogo bitnije da jezgra bude manja te se određeni pad učinkovitosti može prihvatiti. Nedostatak monolitne jezgre koji se želi izbjeći u ugrađenim sustavima jest njena složenost koja znatno otežava razumijevanje, razvoj i nadogradnju novim funkcijama.

Koncept za smanjenje složenosti jezgre može biti izgradnja tzv. mikrojezgre. Mikrojezgra treba sadržavati samo osnovne mehanizme jezgre, a sve ostalo ostvaruje se korištenjem biblioteka izvan jezgre [Buh95b, Eng95b]. Na taj se način izvođenje u nadglednom načinu, unutar jezgrinih funkcija koje su neprekidive, znatno smanjuje. Upravljački se programi izvode izvan jezgrinih funkcija u korisničkom načinu te se u slučaju grešaka ne mora odmah narušiti stabilnost cijelog sustava već samo dotične dretve. Različiti koncepti, stabilnost i brzina mikrojezgri razmatrani su sredinom 90-tih u okviru raznih projekata.

J. Liedtke tako promatra organizaciju jezgre s bržom međuprocesnom komunikacijom [Lie93] te osmišljava i izgrađuje *L3* i potom *L4* sučelje mikrojezgre [Lie95]. U svojim daljnjim radovima [Lie96a, Lie96b, Lie97, Lie01] Liedtke nastoji ukazati na mogućnosti poboljšavanja učinkovitosti takvih jezgri budući su prva ostvarenja mikrojezgri razočarala svojom sporošću, najčešće zbog učestale promjene konteksta pri obavljanju nekih funkcija međuprocesne komunikacije. Na osnovu *L4* opisa sučelja mikrojezgre razvijeno je nekoliko raznih ostvarenja [3].

Mach operacijski sustav [4] koji je prethodio te se potom i paralelno razvijao s idejom mikrojezgri, pored toga što je od početka namijenjen za uporabu na višeprocesorskim i raspodijeljenim sustavima, među prvima je donio neke novosti u organizaciji operacijskog sustava. U samu jezgru ugrađeni su mehanizmi međuprocesne komunikacije, dok su istovremeno iz nje izbačeni neki drugi zadaci, kao što je gospodarenje spremničkim prostorom [Loe92]. Obećavajuće nove tehnike brzo su *Mach*-u pribavile pozornost istraživača. Međutim, kako se s vremenom pokazalo da je takva arhitektura značajno sporija od klasičnih jezgri, mnogi su se okrenuli od tog projekta. *Mach* je unatoč tome ostao dobar model za eksperimentiranje što se može vidjeti po broju objavljenih radova na konferencijama kao što su *USENIX Conference* te *Symposium on Operating Systems Principles* [5]. Nastavak razvoja na *Mach* jezgri i njegovim idejama nastavljen je u okviru raznih sveučilišnih projekata te *GNU Hurd* projekta [6].

Idejama smanjenja jezgre na minimalnu razinu bavio se i Engler [Eng95a, Eng95b, Eng98], ali s gledišta povećavanja brzine rada za sustave posebne namjene. Projekt na kome je radio, *Exokernel*, bavi se upravo smanjivanjem jezgre na način da se gotovo sve funkcije operacijskog sustava prebace u korisnički način rada. Eksojezgri osnovna je funkcija da korisničkim programima (ili operacijskom sustavu zasnovanom na bibliotekama) omogućava gotovo izravan i zaštićen pristup sklopovlju. Sklopovlje se ne skriva od korisnika preko raznih sučelja već mu se nudi direktno kako bi ga iskoristio što učinkovitije. Prikazana ostvarenja takvih jezgri na sustavima posebne namjene (web poslužitelj) znatno su brža od uobičajenih rješenja s operacijskim sustavima.

KeyKOS [Bom92] nanojezgra izgrađena je tako da osim što nudi sigurnost, pouzdanost i raspoloživost, omogućuje i istovremeno izvođenje nekoliko operacijskih sustava na istom računalu. Korištenjem *KeyKOS*-a (tj. nad njegovim primitivima) načinjena su ostvarenja operacijskih sustava *EDX*, *RPS*, *VM*, *MBS* i *UNIX*. Iako se smatra nanojezgrom, *KeyKOS* sadrži poveći skup funkcija operacijskog sustava te za svoj rad treba oko 100 kB spremničkog prostora što je čak i više od nekih kasnijih mikrojezgri (*Neutrino*, *uCOS-II*).

Nanojezgra operacijskog sustava *μChoices* [Tan95], [10] još je jedan od sustava koji se smještaju između sklopovlja i operacijskog sustava. Ideja na kojoj je jezgra nastala jest povećanje prenosivosti operacijskog sustava skrivanjem specifičnosti sklopovlja unutar nanojezgre. Različita se sklopovlja na taj način mogu koristiti kroz isto sučelje. Nanojezgra je tada jedini dio sustava koji se mora prilagoditi ciljanom sklopovlju dok ostatak sustava ostaje isti (npr. u nekom višem programskom jeziku).

Još rudimentarniju funkcionalnost posrednika između sklopovlja i operacijskog sustava pružao je *RC 4000 Nucleus* [Nut00]. *Nucleus* je uključivao jednostavno prioritetno raspoređivanje, upravljanje s procesima i međuprocesnu komunikaciju porukama te osnovno upravljanje okolinom. Operacijski sustavi koji se izvode na *Nucleus*-u sa gledišta upravljanja sredstvima sustava izgledaju kao obični procesi. Iako nije naknadno razvijan, *Nucleus* je svojim idejama doprinio razvoju operacijskih sustava.

μC/OS-II [Lab02] je operacijski sustav za rad u stvarnom vremenu, pisan u programskom jeziku C koji se ističe jednostavnošću izvedbe, malom veličinom i ostvarenjem za mnoštvo različitih procesora te dostupnošću izvornog kôda. Sve navedeno ga čini gotovo školskim primjerom izgradnje jezgre za primjenu u ugrađenim sustavima. Sam naziv skraćenica je od *Micro-Controller Operating System* (operacijski sustav za mikroupravljače) i jasno upućuje na ciljano područje uporabe.

Od ostalih radova, zanimljivih u području razvoja mikrojezgri, mogu se još izdvojiti i *Cache kernel* [Che94], *Echidna* [Ste94, Ste97], *EMERALDS* [Zub01] i *SPACE* [Pro91].

Svi opisani radovi u konačnici imaju određenu jezgru sa specifičnom zadaćom. Jedan od ciljeva ovog rada je definirati skup funkcija, koje se mogu i nazvati jezgrinim funkcijama, čija je prvenstvena namjena omogućiti sinkronizaciju dretvi u sustavu. Takve se funkcije tada mogu iskoristiti kao temelj izgradnje nekog ugrađenog sustava, ili ih se može iskoristiti za ostvarenje nekog standardiziranog sučelja kao što je *POSIX* [8] ili *OSEK/VDX* [7]. Gotovo se sve ostale funkcije trebaju moći ostvariti korištenjem tih osnovnih funkcija. Ostvarivanje najčešće korištenih funkcija operacijskog sustava za ugrađena računala također je jedan od ciljeva rada.

Za postizanje zadanog cilja predložen je koncept monitora. Iako tek nešto složeniji od semafora, monitori su mnogo snažniji sinkronizacijski mehanizam te će se razmotriti mogućnost njihova korištenja u ugrađenim sustavima kao jedinog sinkronizacijskog mehanizma.

Razmatranje mogućnosti izgradnje operacijskog sustava konceptom monitora počinje gotovo istodobno s pojavom samog koncepta monitora [Hoa74]. Ta su razmatranja ipak bila usmjerena na mehanizam sinkronizacije koji bi se mogao ugraditi u programski jezik, a koji bi bio znatno pogodniji za složenije slučajeve sinkronizacije potrebne u operacijskom sustavu. Sami monitori ostvareni su korištenjem jednostavnijih mehanizama, kao što su semafori. O nedostacima izgradnje operacijskog sustava za srednje i velike računalne sustave korištenjem monitora raspravljao je J.L. Keedy [Kee78]. Prema njemu monitori bi zbog veličine nekih jezgrinih funkcija mogli znatno smanjiti mogućnosti

paralelnog rada dretvi. Drugi nedostaci pojavljuju se kada treba ostvariti složenije mehanizme raspoređivanja za što monitori nisu dovoljno fleksibilni. Ostali navedeni problemi uglavnom su vezani uz ostvarenja u programskom jeziku *Simula* te ugniježđenim pozivima monitorskih funkcija.

U radu su analizirani razni modeli monitora te je među njima odabran onaj s najboljim svojstvima s obzirom na uporabu u ugrađenim sustavima. Za odabrani model monitora prikazan je način njegova ostvarenja korištenjem opisnog jezika. Problem inverzije prioriteta značajan je problem u višedretvenim ugrađenim sustavima. Metode za njegovo rješavanje su razmatrane te su izdvojene one koje su pogodne za ugradnju u funkcije monitora. Sinkronizacija pomoću odabranog modela monitora prikazana je na nekoliko standardnih problema sinkronizacije dretvi. Razmatrane su mogućnosti izgradnje raznih funkcija operacijskog sustava korištenjem odabranog modela monitora. Radi procjene zauzeća spremničkog prostora i trajanja određenih operacija funkcije za ostvarenje monitora načinjene su u strojnom jeziku procesora ARM7, kao jednog od predstavnika procesora koji se koriste u ugrađenim sustavima. Zasnivanje ugrađenih sustava korištenjem monitora prikazano je s dva primjera.

Rad se sastoji od devet poglavlja. U drugom poglavlju navedene su osnovne značajke, uloga i podjela operacijskih sustava. Prikazani su koncepti ostvarivanja upravljanja ugrađenim sustavima, opisani su oblici poslova ugrađenih sustava te nekoliko jednostavnih raspoređivača. Na kraju je prikazana struktura i svojstva nekih postojećih rješenja za ugrađene sustave.

Treće poglavlje opisuje monitore i razne koncepte koji se mogu koristiti pri njihovoj izgradnji. Prikazana je podjela monitora prema prioritetu redova u kojima se dretve mogu naći pozivom njegovih funkcija. S obzirom na uporabu u ugrađenim sustavima odabran je model monitora koji po svojim svojstvima najviše odgovara toj primjeni.

Ostvarenje odabranog modela monitora prikazano je u četvrtom poglavlju. Uz same funkcije za ostvarenje monitora prikazane su i funkcije protokola nasljeđivanja prioriteta i protokola vršnog prioriteta.

Korištenje monitora za rješavanje raznih problema sinkronizacije dretvi prikazano je u petom poglavlju. Uz rješenja s monitorima prikazana su i rješenja uz uporabu semafora.

Mogućnosti korištenja monitora za ostvarenje funkcija operacijskog sustava, od upravljanja pristupom ulazno-izlaznim napravama, ostvarivanju kašnjenja, dodatnih raspoređivača do komunikacije porukama prikazane su u šestom poglavlju.

U sedmom poglavlju ukratko je opisan ARM7 procesor te su za njega ostvarene funkcije za rad s monitorom. Na osnovu tog ostvarenja analizirano je zauzeće spremničkog prostora, trajanja pojedinih funkcija, učinkovitost ARM7 arhitekture za ostvarenje monitora te mogućnosti njena poboljšanja.

Dva primjera izgradnje ugrađenog sustava korištenjem monitora prikazana su u devetom poglavlju. Prvi primjer opisuje ručni uređaj dok drugi opisuje sustav nadzora i upravljanja nad skupom pokretnih raspodijeljenih objekata.

2. UGRAĐENI RAČUNALNI SUSTAVI

osophobia n. – A common fear among embedded system programmers [Bar99]

2.1. Operacijski sustavi

Operacijski sustav je skup osnovnih programa koji se nalaze između korisnika računala i sklopovlja [Sil94]. Njegova je zadaća da korisniku (primjenskim programima) pruži prikladnu okolinu za učinkovito korištenje računalnog sklopovlja.

Slika 2.1 Uloga operacijskog sustava

Slika 2.1 prikazuje mjesto operacijskog sustava u računalnom sustavu. Primjenski programi jednom dijelu sklopovlja pristupaju preko operacijskog sustava, a drugome izravno.

Operacijski sustav je najkritičniji dio programskog okruženja nekog računalnog sustava te ga kao takvog smišljaju, izgrađuju i mijenjaju iskusni stručnjaci s ovog područja [Nut00]. Njegova korisnost mjerljiva je njegovim mogućnostima i brzini rada.

Različiti računalni sustavi postavljaju vlastite zahtjeve na programsko okruženje. Operacijski sustav osobnog računala, namijenjen mnoštvu aktivnosti kao što su uredski programi, pregledavanje i stvaranje multimedijalnih sadržaja te računalne igre, bitno se razlikuje od onog koji se koristi u računalu koje upravlja nekim proizvodnim procesom.

Prvi računalni sustavi, iako ogromnih dimenzija, bili su ograničeni mogućnostima procesora i pohrane podataka. Program, koji se na razne načine učitavao s vanjskih jedinica, imao je ugrađen dio za upravljanje okolinom. U prvotnim sustavima nije bilo mogućnosti za paralelni rad više zadataka već se na njima obavljao samo jedan program. Kad je program završio s radom i rezultati bili pohranjeni na neku vanjsku jedinicu, moglo se početi s učitavanjem i izvođenjem sljedećeg programa. Svi programi izvođeni na istom računalu imali su jedan dio programa sličan – dio za upravljanje okolinom. Struktura tih programa može se opisati s dva dijela: dio koji obavlja potrebne proračune te dio koji čita podatke iz ulaznih jedinica i pohranjuje rezultate na vanjske jedinice. Drugi dio se sastoji od potrebnog broja funkcija za upravljanje okolinom te on može biti jednak za sve programe, tj. nije ga potrebno ponovo pisati za svaki od programa. Skup takvih funkcija može se smatrati jednim oblikom operacijskog sustava, mada se za taj dio sustava danas

koristi pojam upravljačkih programa.

Zahtjevi za korištenjem prvih računala ubrzano su rasli što je dodatno ubrzalo razvoj računalne industrije. Računala su tada bila iznimno skupa te se u namjeri da se istim računalom koristi više osoba pristupilo osmišljavanju novog programskog okruženja. Takvo programsko okruženje treba omogućiti prividno istovremeni rad više zadataka na istome računalu. Dijeljenje računala, kako procesorskog vremena, spremničkog prostora i ulazno-izlaznih naprava, zahtjeva skup sinkronizacijskih mehanizama koji do tada nisu bili potrebni. Pored sinkronizacije javlja se potreba za mehanizmom razmjene podataka među korisnicima i među procesima. Zaštita podataka od neovlaštenog korištenja postaje jedna od potrebnih funkcija programskog okruženja. Navedeni zahtjevi samo su neki od mnogih koji su se javljali tijekom razvoja operacijskih sustava i koji su utjecali na njihovu arhitekturu. Razvoj operacijskih sustava traje i dalje jer se i računala usavršavaju te koncepti koji su bili optimalni za prethodnu generaciju ne moraju biti i za sljedeću. Tako se događa da se ponovo prihvaćaju stari koncepti (ili njihove modifikacije) koji postaju primjenjivi na novim arhitekturama.

Većina funkcija koje operacijski sustavi danas imaju potiče od prvih operacijskih sustava razvijenih na poslužiteljskim računalima. Ti su prvi sustavi postavili temelje gotovo svim sustavima koje danas koristimo, bilo na osobnom računalu u uredu ili kući, ili poslužiteljima. Ipak, u području ugrađenih računala uporaba operacijskih sustava znatno je manja. Neki ugrađeni sustavi toliko su jednostavni da nemaju potrebe korištenja operacijskog sustava već je dovoljan upravljački program. Kada se operacijski sustavi i koriste u ugrađenim računalima tada su ti operacijski sustavi znatno drukčijih svojstava.

2.2. Ugrađeni sustavi

Ugrađene sustave može se kratko definirati kao kombinacija sklopovlja i programske podrške, uz eventualno dodatne mehaničke ili druge dijelove, načinjena da obavlja specifičnu funkciju [Bar99].

Ugrađeni se sustavi u nekoliko svojstava bitno razlikuju od ostalih računalnih sustava (npr. uredskog računala, radne stanice ili poslužitelja). Vremenska određenost je vrlo bitna u velikom dijelu ugrađenih sustava jer zakašnjela reakcija može imati katastrofalne posljedice. Stabilnost i pouzdanost ugrađenim je sustavima znatno bitnija nego u ostalim računalnim sustavima. Sustavi za rad u stvarnom vremenu, odnosno sustavi u kojima je vremenska određenost osobito važna podskup su ugrađenih sustava. Ovaj se rad u velikom dijelu bavi tim područjem ugrađenih sustava te se dosta metoda koristi iz područja sustava za rad u stvarnom vremenu koje je, na neki način, detaljnije znanstveno istraženo.

Uporaba procesora u građenim sustavima vrlo je rasprostranjena. Neki od primjera su: govorni automati, mobilni telefoni i uredske telefonske centrale, mrežna oprema (usmjernici, poslužitelji vremena, sigurnosne stijene), računalni pisači, tvrdi diskovi, upravljači motora, ABS upravljači za automobile, samostalne kućne naprave (regulatori temperature, klima uređaji, protupožarna zaštita, protuprovalni nadgledni sustavi), ručni kalkulatori, razni kućni uređaji (mikrovalna pećnica, perilica rublja/posuđa, televizije, DVD uređaji), sustavi za navođenje, upravljanje letom i ostali integrirani sustavi u avionima i raketama, medicinska oprema, mjerni instrumenti, višenamjenski satovi, ručna računala, programirljivi logički upravljači za automatizaciju i nadzor u industriji [Jak02, Jel02, Jak03, Jel04], igraće naprave.

Osobno računalo, bilo u uredu ili u kući ili radna stanica, nije osmišljeno za obavljanje samo jedne specifične funkcije već mnoštvo njih. Zbog svoje opće upotrebljivosti osobno računalo ima mnoštvo primjena: uredsko računalo, radna stanica, poslužitelj, igrača naprava, sredstvo za izvođenje/stvaranje multimedijalnih sadržaja i drugo. Međutim, zbog svoje opće namjene takvo računalo nema odgovarajuće mehanizme potrebne za uporabu u većini ugrađenih sustava. Kao prvo, dimenzijama i cijenom bitno odudara od zahtjeva ugrađenih sustava. Nadalje, što je mnogo važnije, nema ugrađene mehanizme vremenske određenosti, kako ni u sklopovlju tako ni u programskoj okolini. Npr. kod osobnog računala gotovo je sasvim nebitno je li za prikaz nekog prozora na zaslonu potrebna sekunda, dvije, tri ili se neki proračun obavlja desetak ili više sekundi ili minuta, ali je ta vrsta neodređenosti nedozvoljena u ugrađenim sustavima. Uredski operacijski sustavi i sustavi na radnim stanicama imaju vrlo lošu podršku vremenski uređenim zadacima i odzivu na vanjske događaje, pa su u većini slučajeva neupotrebljivi za ugrađene sustave.

Zbog širokog područja i upravljanje ugrađenim sustavima se međusobno znatno razlikuje. Negdje je dovoljan kratki upravljački program, dok je drugdje potreban složeni višezadaćni sustav posebnih svojstava. Ugrađene sustave s obzirom na ostvarenje načina upravljanja možemo podijeliti u nekoliko skupina [1]:

1. upravljački program,
2. neprekidiva višezadaćnost,
3. višezadaćnost,
4. uredski operacijski sustavi te
5. operacijski sustavi posebne namjene.

Upravljački program sastoji se od jedne beskonačne petlje u kojoj se na osnovu ulaza ili varijabli stanja sustava poduzimaju određene akcije. Rad više aktivnosti ostvaruje se slijednim provjeravanjem i upravljanjem svakom od njih. Prekidi izazvani vanjskim događajima najčešće su vrlo kratki i samo mijenjaju stanje pojedinih varijabli. Drugoj inačici ove skupine pripadaju sustavi u kojima je gotovo sva aktivnost raspoređena u prekidnim procedurama, tj. obrada i upravljanje se odvijaju unutar obrade prekida. Upravljački program je vrlo kratak i jednostavan te se u velikom dijelu ugrađenih sustava koristi upravo ovaj model upravljanja. Glavni nedostaci upravljačkog programa su nemogućnost pridjeljivanja prioriteta pojedinim komponentama pri upravljanju i otežano ostvarenje upravljanja nad skupom periodičnih poslova.

Kod neprekidive višezadaćnosti u sustavu se nalazi nekoliko dretvi, gdje svaka upravlja s određenim dijelom sustava. Osim vanjskog prekida dretvu koja se trenutno izvodi ne može prekinuti niti jedna druga dretva, već ona pri završetku trenutnog posla izravno poziva raspoređivač. Raspoređivanje i međusobna komunikacija obavljaju se izravnim pozivima dretvi te mehanizmi sinkronizacije nisu potrebni. Vanjski prekidi mogu biti ili vrlo kratki, mijenjajući samo stanja određenih varijabli, ili zasebne procedure koje upravljaju pojedinim dijelovima sustava. Prednosti ovakvih sustava su u jednostavnosti izvedbe i mogućnosti pridjeljivanja prioriteta pojedinim dretvama. Osnovni nedostatak jest neprekidivost izvođenja dretve, tj. raspoređivanje se poziva isključivo na kraju rada trenutno aktivne dretve, bez obzira što se i prije toga mogu stvoriti uvjeti za nastavak rada prioritetnije dretve.

U skupinu načina upravljanja navedenim pod višezadaćnost podrazumijevaju se sustavi u kojima je upravljanje raspodijeljeno po dretvama različitih prioriteta. Aktivna dretva je ona s najvećim prioritetom iz skupa pripravnih dretvi. Ukoliko se nakon prekida ili drugog događaja (npr. funkcija sinkronizacije ili međusobne komunikacije) stvore uvjeti za nastavak rada dretve višeg prioriteta, prekida se izvođenje trenutne dretve te se

prioritetnijoj dretvi omogućuje izvođenje. U višezadaćne sustave ubrajaju se svi operacijski sustavi za rad u stvarnom vremenu. U odnosu na prijašnje skupine višezadaćnost je mnogo većih mogućnosti, ali samim time i veće složenosti. Jedan od ciljeva ovoga rada jest zadržavanje osnovnih mogućnosti ove skupine uz istovremeno značajno smanjenje složenosti sustava.

U sustavima s blagim vremenskim ograničenjima moguće je koristiti i obične, uredske operacijske sustave. Zbog znatne programske podrške takvi su sustavi mnogo jeftiniji za izgradnju i održavanje, ali im je upotrebljivost ograničena. Iz sličnih su razloga nastale i izvedenice takvih operacijskih sustava s puno boljom podrškom za primjenu u ugrađenim sustavima. Zbog sličnosti sustava, dodatno obrazovanje za programere takvih sustava se minimizira te se ubrzava prenošenje aplikacija na novi sustav. Takvi sustavi, međutim, zbog nastojanja zadržavanja osnovnih koncepata početnog sustava, najčešće imaju značajno dulji odziv na vanjske događaje od operacijskih sustava posebno oblikovanih za ugrađene sustave. Primjena tih sustava je zbog toga ograničena te se isti ne primjenjuju u kritičnim sustavima kontrole i upravljanja.

Operacijski sustavi posebne namjene grade se po narudžbi te se od operacijskih sustava, bilo za ugrađene ili ostale sustave, bitno razlikuju u svojim svojstvima i ponašanju.

Jezgra operacijskog sustava može se ostvariti na nekoliko konceptualno različitih načina [Eng98, Cre04, Nut00]:

- monolitna jezgra,
- mikrojezgra,
- proširiva jezgra i
- razne hibridne kombinacije navedenih.

Kod monolitnih jezgri sve se funkcije operacijskog sustava izvode u sustavskom načinu rada, tj. s najvećim prioritetom te su kao takve neprekidive u svom izvođenju. Funkcije jezgre mogu se stoga međusobno usko povezati pridonoseći kompaktnosti i brzini. Monolitne jezgre najčešće su vrlo brze, pogotovo u operacijskim sustavima za rad u stvarnom vremenu. Primjer monolitne jezgre jest *VxWorks* operacijski sustav, klasični primjer sustava za rad u stvarnom vremenu. Stabilnost, pouzdanost te mnoštvo funkcija sinkronizacije i međusobne komunikacije uz ostale atribute (snažno razvojno okruženje, multiplatformska podrška) čine ga vrlo popularnim izborom za uporabu u ugrađenim sustavima. Nedostaci monolitne jezgre očituju se u teškoćama nadogradnje novim funkcijama ili promjenama nekih svojstava. Također, ukoliko uslijed nepredviđenog tijeka izvođenja neka od jezgrinih funkcija izazove neoporavljivu pogrešku, cijeli je sustav kompromitiran, a ne samo dotična dretva ili dio sustava. Jedan od načina poboljšanja proširivosti takvih sustava jest korištenje modularnosti, tj. jezgre koja se sastoji od osnovnog dijela i dodatnih modula. Moduli se po potrebi mogu dinamički uključivati i isključivati iz jezgre. Primjer operacijskog sustava s modularnom jezgrom jest Linux [18]. Zanimljiva rasprava o organizaciji jezgre vođena je između A. Tanenbauma i L.B. Torvaldsa [17].

Operacijski sustavi zasnovani na mikrojezgri nastoje riješiti neke probleme koji se pojavljuju kod monolitnih jezgri. Mikrojezgra se sastoji samo od skupa osnovnih funkcija, dok se ostale funkcionalnosti operacijskog sustava ostvaruju izvan jezgre, u korisničkom načinu rada. Kada se u nekoj funkciji pojavi kritična greška moguće je zaustavljanje rada samo dotičnog procesa ili dretve, dok bi ostatak sustava mogao nastaviti normalno raditi. Dio kôda koji pripada operacijskom sustavu, a izvan je jezgre, unaprijed je definiran i

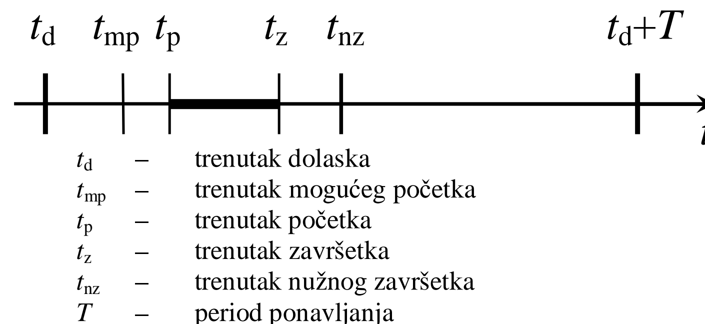
nepromjenjiv, kao npr. upravljački programi. Korisniku ili njegovu procesu nije dozvoljeno korištenje proizvoljnih programa za pristup sklopovlju već se to obavlja preko unaprijed definiranih funkcija. Tako se osigurava izvođenje samo provjerenog kôda, ali i unose ograničenja. Zbog minimalnog skupa funkcija i ostvarenja glavnine funkcija operacijskog sustava van jezgre, učestalo se mijenja način rada te su brzine rada ovih sustava lošije od sustava s monolitnom jezgrom. S druge strane prednosti su u proširivosti, održavanju i većoj stabilnosti pri ispadu pojedinih dijelova sustava. S obzirom na uporabu u ugrađenim sustavima značajna je prednost u veličini same jezgre koja može biti i za red veličine manja od uobičajenih monolitnih jezgri. Često su funkcije koje su dio mikrojezgre dovoljne za ugrađeni sustav. Primjer mikrojezgre jest *Neutrino* u sustavu *QNX* [9].

Proširiva jezgra [Eng98] je jedan od oblika mikrojezgre koji prvenstveno služi za ostvarenje mehanizama zaštite u sustavu. Takva jezgra dozvoljava korisničkim programima izravan i zaštićen pristup sklopovlju. Sloj koji ona pruža vrlo je tanak u usporedbi s klasičnim operacijskim sustavima gdje se često za pristup određenom sklopovlju mora koristiti nekoliko podsustava. *Exokernel* je primjer navedene zamisli koja je ostvarena kroz nekoliko sustava visoke učinkovitosti. Takvi sustavi, međutim, nisu sustavi opće nego specijalne namjene u kojoj se znatno ističu brzinom rada.

2.3. Dretve u ugrađenim sustavima

Pri razmatranju ugrađenih sustava potrebno je poznavati oblik poslova koji se obavljaju, odnosno oblik dretvi koje ih izvode.

Dretve najčešće obavljaju periodičke poslove s unaprijed zadanim vremenima ponavljanja. Osim periodičnih dretvi u sustavu mogu postojati i dretve za izvođenje poslova koji se javljaju sporadično, kao npr. reakcija na pojavu prekida. I jedne i druge dretve imaju neka zajednička svojstva za čije se promatranje mogu definirati određeni vremenski trenuci bitni za odvijanje dretvi [Bud99]. Slika 2.2 prikazuje svojstvene trenutke u radu dretve.



Slika 2.2 Svojstveni trenuci u životnom ciklusu jedne dretve ugrađenog sustava

Dretva se sa svojim poslom pojavljuje u sustavu u trenutku t_d. Izvođenje može započeti u trenutku t_{mp} koje može biti i jednako trenutku dolaska. Trenutak kada dretva počinje svoje izvođenje označeno je sa t_p. Moguće je da se dretva za vrijeme svog izvođenja prekida drugom dretvom većeg prioriteta ili obradom prekida. U trenutku t_z dretva završava s pridijeljenim joj poslom te završava s radom i nestaje iz sustava ili se privremeno zaustavlja do sljedećeg pojavljivanja posla. Do trenutka t_{nz} dretva mora obaviti posao ili će sustav snositi određene posljedice. Ukoliko je posao dretve periodički njeno

sljedeće aktiviranje očekuje se u trenutku t_d+T. Vremenska uređenost događaja sa slike mora se očuvati ukoliko se želi stabilan rad sustava, tj. za navedene vremenske trenutke mora vrijediti uređenje: $t_d \leq t_{mp} \leq t_p < t_z \leq t_{nz}$.

Zadaća je operacijskog sustava, ili nekog drugog rješenja koje se koristi umjesto njega, omogućiti održavanje navedenog vremenskog uređenja za sve zadatke u sustavu. Drugim riječima, zadaci moraju biti tako raspoređeni da svoje izvođenje počinju najranije u trenutku t_{mp} te da posao obave najkasnije do t_{nz}. Postoje razni algoritmi raspoređivanja koji se koriste u tu svrhu, od kojih su neki vrlo jednostavni dok su drugi vrlo složeni. Odabir algoritma ovisi o uporabi. Ponegdje će i oni najjednostavniji biti sasvim dovoljni dok će drugdje biti potrebni ostali, različite složenosti.

Problem raspoređivanja sredstava javlja se u gotovo svakom sustavu. Kako su gotovo svi sustavi upravljani računalima to postaje problem i u području računarstva. Raspoređivanje sredstava tako da se zadovolje sva ograničenja uz istovremeno postizanje očekivane učinkovitosti ili kvalitete, može biti vrlo složen problem. U nastavku se razmatra samo problem raspoređivanja dretvi u sustavu, odnosno, raspoređivanje procesorskog vremena po dretvama sustava.

Raznolikost računalnih sustava zahtjeva razne metode raspoređivanja te se iz istog razloga nove metode neprestano istražuju i usavršavaju [But99, Kuo02, Ni02, Han03, Lim03, Abd04]. Raspoređivanje u ugrađenim sustavima treba obaviti tako da sve dretve obave svoje (periodičke) poslove prije trenutka nužnog završetka. Za različite probleme najčešće se koriste i različiti algoritmi raspoređivanja. Raspoređivač koji savršeno odgovara jednom tipu problema kod drugog može dati vrlo loše rezultate. Povećanje procesorske snage najčešće daje prave rezultate. Međutim, takvo rješenje poskupljuje gotovi proizvod te ga treba uzeti kao zadnje, ako se problem ne može riješiti drugim metodama.

Najčešće korišteni algoritmi raspoređivanja dretvi u ugrađenim sustavima mogu se podijeliti u dvije skupine: statički i dinamički [Til91a]. Kod statičkih algoritama sustav se analizira prije samog rada te se unaprijed definira redoslijed izvođenja dretvi ili se dretvama statički pridijeli prioritet te ih onda sustav raspoređuje koristeći njihove prioritete. Dinamički algoritmi prate rad sustava i na osnovi njegova stanja dinamički određuju dretvu koja će se iduća izvoditi.

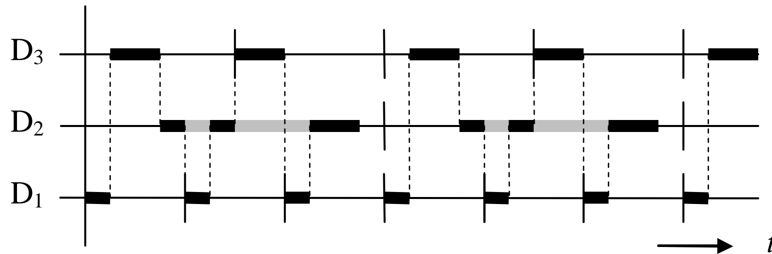
Budući se teži jednostavnosti sustava u nastavku su razmatrani algoritmi koji se često koriste i prilično su jednostavni za ostvarenje. Osnovni prioritetni raspoređivač, koji za aktivnu dretvu uzima dretvu najvećeg prioriteta iz reda pripravnih, može se jednostavno ugraditi u sinkronizacijske funkcije te će se on sigurno koristiti. Algoritam raspoređivanja radi tako da se nakon svakog događaja koji uključuje funkcije sinkronizacije (jezgrine funkcije) prije samog izlaska iz funkcije od svih pripravnih dretvi za sljedeću aktivnu odabire ona s najvećim prioritetom.

Od ostalih jednostavnijih raspoređivača koristit će se algoritam mjere ponavljanja (engl. *rate monotonic scheduling* - RMS) i raspoređivanje prema trenutku nužnog završetka (engl. *earliest deadline first* - EDF).

Algoritam mjere ponavljanja [Til91a, Liu73, Bar03, Bin03, Lau03] spada u skupinu statičkih algoritama kod kojih se dretvama pridijele prioriteti prije pokretanja sustava. Prema algoritmu dretvama s većom frekvencijom pojavljivanja, odnosno kraćim vremenom ponavljanja, pridjeljuje se veći prioritet.

Na primjer, ako u sustavu postoje dretve D_1, D_2 i D_3 s periodama ponavljanja redom

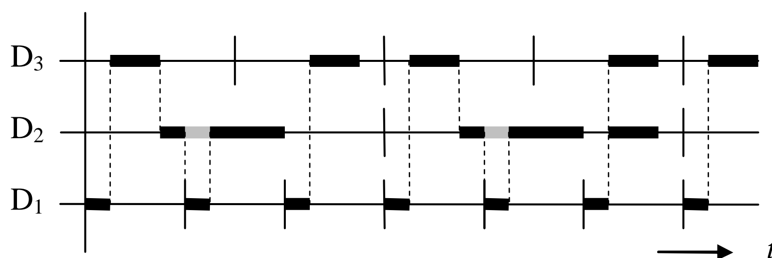
10 ms, 30 ms, 15 ms, tada najveći prioritet po zadanom algoritmu ima dretva D_1, slijedi D_3 dok dretva D_2 ima najmanji prioritet budući ima najveću periodu ponavljanja. Slika 2.3 prikazuje jedan takav sustav, odnosno raspoređivanje dretvi uz zadana trajanja izvođenja poslova dretvi od 2,5 ms, 5 ms i 10 ms. Za svaku je dretvu označen period u kome se mora obaviti, a obavljanje dretve prikazano je debljom crtom.



Slika 2.3 Primjer raspoređivanja algoritmom mjere ponavljanja

Iz slike je vidljivo da se izvođenje dretve D_2 prekida u nekoliko navrata zbog izvođenja druge dretve većeg prioriteta. U primjeru je pretpostavljeno da su dretve odmah spremne za izvođenje ($t_a = t_{mp}$) te da je trenutak nužnog završetka jednak trenutku sljedeće periode ($t_{nz} = t_a + T$).

Algoritam raspoređivanja prema trenutku nužnog završetka pridjeljuje procesoru onu pripravnu dretvu čiji je trenutak nužnog završetka najskoriji. Ako se promatra u terminima prioriteta dretva s najbližim trenutkom nužnog završetka ima najveći prioritet. Algoritam je dinamički jer se mora izvršiti nakon svake promjene stanja u sustavu, tj. nakon što jedna dretva završi svoj periodički rad ili druga postaje spremna za izvođenje. U svakom takvom trenutku ponovo treba izračunati čiji je trenutak nužnog završetka najbliži i toj dretvi pridijeliti najveći prioritet, odnosno procesor. Za sustav sa slike 2.3 svakih 30 ms poklapa se trenutak nužnog završetka svih triju dretvi te je odabir jedne od njih u tom intervalu ili proizvoljan ili se unose dodatni parametri u sustav. Zbog toga i raspored sa slike 2.3 prikazuje jedno od mogućih rješenja raspoređivanja. Ukoliko bi se kao dodatni kriterij uzeo redni broj dretve tada raspoređivanje po ovom algoritmu odgovara slici 2.4. Dretva D_2 u ovom je primjeru prekinuta samo jednom i to dretvom D_1.



Slika 2.4 Primjer raspoređivanja prema trenutku nužnog završetka

Trenuci u kojima se poziva raspoređivač su trenuci završetka rada periodičkog posla dretvi i trenuci pojave novih pripravnih dretvi u sustavu, što odgovara početku nove periode.

Osim raspoređivanja korištenjem zadanih svojstvenih trenutaka (t_d, t_{mp}, t_p, t_z, t_{nz}, T), često se u proces moraju uključiti i neki drugi parametri sustava [Chu90, Mar94, Shi95, Nic96]. Obzirom da su navedena dva algoritma jednostavna, njihova učinkovitost zaostaje za naprednijim rješenjima [Liu73, Hon89, Bin03, Lau03, Abd04], ali su zato jednostavniji za ugradnju.

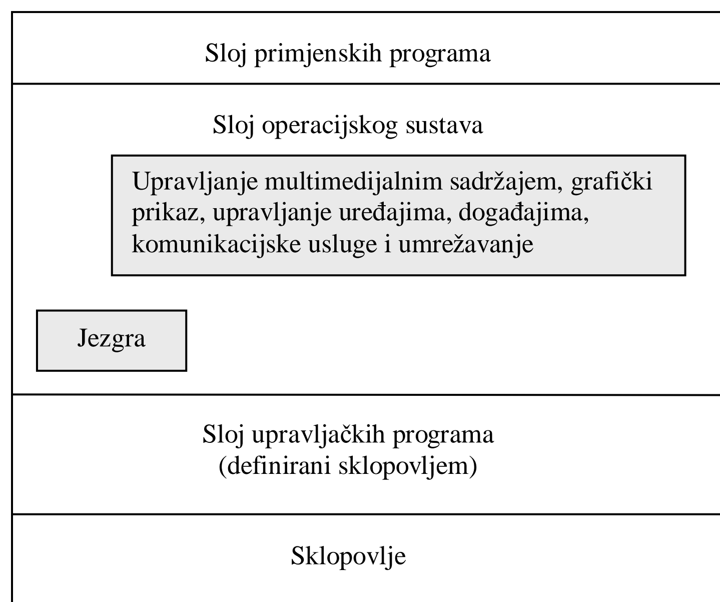
U ostalim računalnim sustavima algoritmi raspoređivanja većinom su usmjereni na pravičnost dodjele procesorskog vremena po svim dretvama sustava. Veći prioritet u takvim sustavima najčešće znači samo više procesorskog vremena, a ne kao u ugrađenim sustavima gdje prioritet određuje aktivnu dretvu. Najčešći raspoređivač u tim sustavima radi na načelu podjele procesorskog vremena u male vremenske odsječke – kvante, koji se dodjeljuju dretvama sustava.

2.4. Pregled izvedbe i mogućnosti nekih jezgri za ugrađene sustave

Budući je cilj rada definiranje osnovnih funkcija za zasnivanje ugrađenog sustava, bitno je proučiti osnovne funkcionalnosti operacijskih sustava namijenjenih ugrađenoj uporabi. U ovom potpoglavlju prikazano je nekoliko operacijskih sustava za rad u stvarnom vremenu.

2.4.1. Operacijski sustav *Windows CE*

Prema navodima proizvođača [11] porodica operacijskih sustava *Windows CE* izgrađena je tako da zadovoljava široko područje inteligentnih uređaja. Zbog sličnosti arhitekture jezgre i njenog sučelja s ostalim *Win32* arhitekturama, sustav ima potencijalno vrlo veliki broj programera koji uz vrlo malo dodatnog napora mogu izgrađivati rješenja zasnivana na ovom operacijskom sustavu. Pored poznate arhitekture i razvojni su alati jedna od prednosti ovog sustava. Slika 2.5 prikazuje arhitekturu *Windows CE* sustava.



Slika 2.5 Arhitektura Windows CE sustava

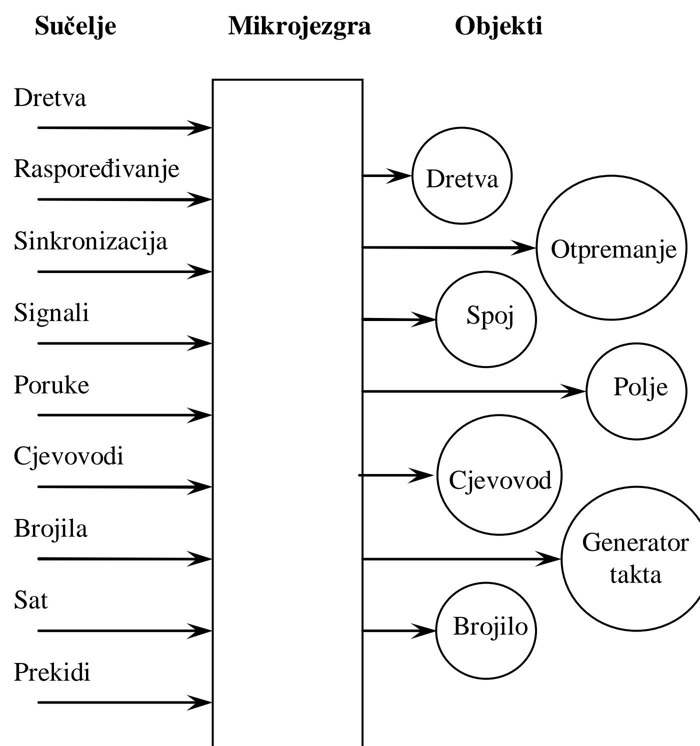
Jezgra sustava pruža osnovnu funkcionalnost operacijskog sustava u upravljanju procesima, dretvama i spremničkim prostorom za koje se koristi straničenje. Odnos procesa i dretvi jednak je kao i na ostalim *Win32* sustavima, tj. jedan proces sadržava jednu ili više dretvi koje dijele isti adresni prostor te mogu učinkovitije raditi na istom zadatku. Prioriteti dretvi koji se koriste u procesu raspoređivanja su u području od 0 do 255, gdje manji broj predstavlja veći prioritet. Prioriteti su podijeljeni u nekoliko skupina te je za svaku predviđena posebna namjena. Koristi se kružno raspoređivanje temeljeno na

prioritetima. Pripravne dretve najvećeg prioriteta dijele procesorsko vrijeme tako da svaka dobije po kvant vremena. Nakon što raspoređene dretve završe s radom ili više nisu u redu pripravnih, s izvođenjem nastavljaju dretve sljedećeg nižeg prioriteta. Problem inverzije prioriteta rješava se protokolom nasljeđivanja prioriteta.

Prihvat i obrada prekida podijeljena je na dva dijela: jezgrina prekidna rutina (engl. *kernel-mode interrupt service routine* - ISR) i dretva za obradu prekida (engl. *user-mode interrupt service thread* - IST). ISR se obavlja unutar nadglednog načina rada i namijenjen je za obavljanje kraćih poslova pri pojavi prekida. Sučelje preko kojega ISR treba koristiti jezgru omogućuje da se nakon završetka ISR-a, ukoliko je potrebno, pokreće IST koji obavlja ostatak posla, a koji može potrajati te se zato izvodi izvan jezgre u korisničkom načinu rada. Za različite prekide mogu postojati različiti ISR-ovi i IST-ovi.

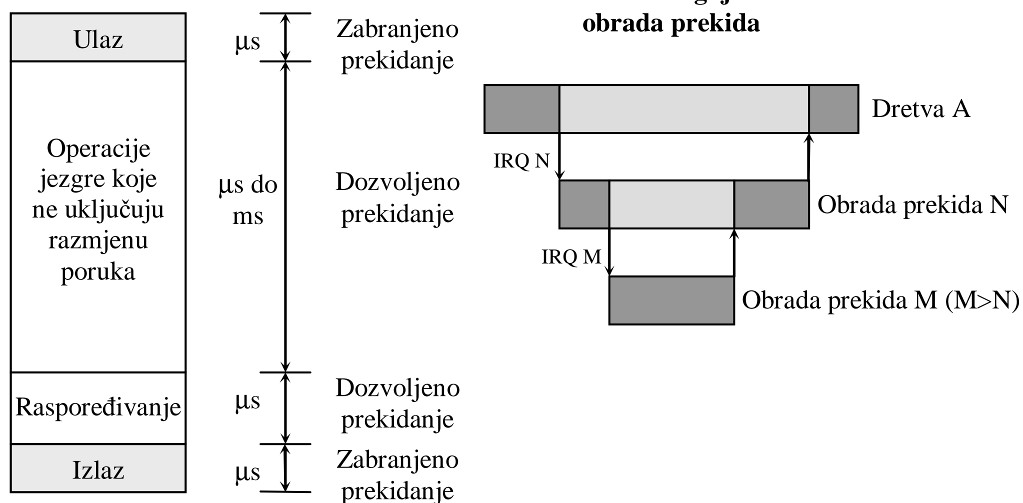
2.4.2. Operacijski sustav *QNX Neutrino*

Neutrino [9] je mikrojezgra koja uključuje osnovna sučelja definirana *POSIX* standardom za ugrađene sustave integrirane s osnovnim *QNX* sustavom prosljeđivanja poruka. U jezgru su uključeni dijelovi za upravljanje dretvama, sustavom komunikacije porukama, signalima, vremenom i brojilima, prekidima, semaforima, varijablama međusobnog isključivanja i uvjetnim varijablama, prema slici 2.6.



Slika 2.6 Sučelje prema *Neutrino* jezgri i objekti s kojima upravlja

Posebna pažnja pri izgradnji jezgre posvećena je povećanju brzine rada jezgrinih funkcija te smanjenju neprekidivog dijela jezgre. Jezgrine su funkcije zbog toga u većini svog trajanja prekidive. Isto tako prekidima su pridijeljeni prioriteti te se obrada prekida nižeg prioriteta može prekinuti radi obrade prekida većeg prioriteta. Slika 2.7 prikazuje neka svojstva jezgre u slučaju poziva funkcije jezgre, odnosno, ukoliko se prekidi ugnježđuju.



Slika 2.7 Prekidivost jezgrinih funkcija i obrada prekida *Neutrino* jezgre

Procesi i dretve organizirani su slično kao i kod *Windows CE*, tj. dretve su raspoređene po procesima. Prioriteti dretvi kreću se od 1 do 63, s time da veći broj označava veći prioritet. U sustavu se razlikuje stvarni prioritet (engl. *real priority*) i trenutni prioritet (engl. *effective priority*) koji dretve mogu imati. Trenutni prioritet uglavnom je jednak stvarnom, ali zbog nasljeđivanja prioriteta ili zbog algoritma raspoređivanja on može biti i različit. Pripravne dretve raspoređene su u jedan od 64 reda, prema prioritetu. Za izvođenje se uvijek odabire prva dretva iz reda s najvećim prioritetom. U sustavu postoji četiri algoritma raspoređivanja koji se mogu koristiti i koji se definiraju na razini svake pojedine dretve, tj. različite dretve mogu koristiti različite algoritme. To su: raspoređivanje po redu prispijeća, algoritam kružnog posluživanja, adaptivno te sporadično raspoređivanje. Raspoređivači mijenjaju stanja i prioritete dretvi kojima rukuju te ih jezgra dalje raspoređuje isključivo prema prioritetu.

Razmjena poruka osnovni je mehanizam međuprocesne komunikacije koji je naslijeđen iz *QNX*-a te je vrlo optimiran s obzirom na učinkovitost. Osim njega na raspolaganju su signali, *POSIX* mehanizmi razmjene poruka, zajednički spremnički prostor i cjevovodi.

2.4.3. Operacijski sustav *VxWorks 5.4*

Među najpoznatije operacijske sustave za rad u stvarnom vremenu spada i *VxWorks* [16], [Jon97]. Monolitna jezgra zavidne brzine, mnoštvo podržanih standarda te razvijeno razvojno okruženje neki su od uobičajenih atributa koji se vežu uz navedeni sustav. Pored vlastitog sučelja za korištenje u sustavima za rad u stvarnom vremenu, u jezgru je ugrađena podrška i za *POSIX* standarde (1003.b).

Pored pretpostavljenog prioritetnog raspoređivanja, podržano je i kružno raspoređivanje. Za razlika od prethodna dva sustava za opis osnovne jedinice rada koristi se pojam zadatka (engl. *task*) koji zapravo predstavlja dretvu, dok se koncept procesa kao okvira u kojem se dretve izvode ne koristi.

Radi bržeg prihvata, prekidi se obrađuju izvan konteksta bilo koje dretve. Iz funkcija za obradu prekida mogu se stoga pozivati gotovo sve funkcije jezgre osim onih koje bi mogle uzrokovati zaustavljanje daljnjeg izvođenja, što je nedozvoljeno.

Za komunikaciju među dretvama postoji nekoliko mehanizama. Najznačajniji načini komunikacije su: zajednički spremnički prostor, semafori, redovi poruka, cjevovodi, signali, pozivi udaljenih funkcija i priključne točke prema mrežnom sloju.

Radi rješavanja problema inverzije prioriteta koristi se protokol nasljeđivanja prioriteta. Uz njega se veže zanimljivost iz svemirske misije na Mars robotske letjelice *Mars Pathfinder* [Jon97] koja je bila upravljana upravo *VxWorks* operacijskim sustavom. Nakon uspješnog slijetanja i prvih nekoliko dana rada, na sustavu upravljanja počeli su se pojavljivati neobjašnjivi i učestali prekidi. Kako je sustav napravljen da prilikom detekcije greške prekida rad i ponovo pokreće cijeli sustav sve je do određene mjere i dalje radilo. Greška koja je za nekoliko dana pronađena te potom i uklonjena jest u protokolu nasljeđivanja prioriteta, koji je bio isključen. Problem je nastao kada je prioritetnoj dretvi na dulje vrijeme bio uskraćen pristup sabirnici što je sklop za nadzor interpretirao kao kritičnu grešku te je zaustavio i ponovo pokrenuo sustav.

3. MONITORI

Monitori se kao koncept u paralelnom programiranju pojavljuju krajem 60-tih i početkom 70-tih godina 20. stoljeća. Nastali su zbog potrebe za jačim mehanizmima sinkronizacije od semafora, do tada osnovnog mehanizma sinkronizacije. Koncept je razvijen u krugu znanstvenika C.A.R. Hoare [Hoa74], E.W. Dijkstra [Dij65, Dij68], P. Brinch Hansen [Bri72, Bri73]. Da bi se prikazao nedostatak semafora u složenijim sustavima sinkronizacije, najprije će biti prikazani semafori i njihov mehanizam rada.

3.1. Semafori

Povijesno gledano koncept semafora predstavio je Dijkstra [Dij65] sredinom 60-tih godina prošlog stoljeća. Semafor je predstavljen jednom cjelobrojnom vrijednošću i jezgrinim funkcijama `Čekati(sem)` i `Postaviti(sem)`. Originalno se poziv `Čekati(sem)` označavao sa `P(sem)` a `Postaviti(sem)` sa `V(sem)` prema izvornoj notaciji te se u tom obliku i danas često nalazi u literaturi [Sil94].

Dretva koja poziva funkciju `Čekati(sem)` će ili proći i nastaviti raditi ili će biti zaustavljena i smještena u red. Postoji nekoliko inačica semafora koje se najčešće razlikuju samo u načinu smanjivanja/povećavanja vrijednosti semafora. Npr. vrijednost se može smanjiti/povećati odmah pri ulazu u funkciju ili tek kasnije kada se ustanovi da je to moguće/potrebno. Funkcionalnost svih njih gotovo je identična osim u nekim specijalnim slučajevima. Ovdje je prikazana inačica koja se najčešće koristi. Funkcija `Čekati(sem)` pokušava smanjiti vrijednost semafora, a da vrijednost ne postane negativna. Ako to nije moguće, tj. vrijednost semafora već je nula, dretva će biti smještena u red dotičnog semafora. Ako se vrijednost može smanjiti ona se smanjuje i dretva može nastaviti s radom.

Pri pozivu funkcije `Postaviti(sem)` prva dretva u redu semafora premješta se iz reda semafora u red pripravnih dretvi te se onda na osnovi prioriteta dretvi u tom redu određuje koja će nastaviti rad. Ukoliko je red semafora prazan, vrijednost semafora se poveća za jedan.

Jezgrine funkcije za rad sa semaforima mogu se prikazati sljedećim tekstom [Bud03]:

```
j-funkcija Čekati(J) {
   pohraniti kontekst u opisnik aktivne dretve
   ako je (Sem[J].v ≥ 1) {
      Sem[J].v = Sem[J].v - 1
   } inače {
      premjestiti opisnik iz reda aktivne dretve u red Sem[J]
      premjestiti opisnik prve pripravne u opisnik aktivne dretve
   }
   obnoviti kontekst iz opisnika aktivne dretve
}
```

```
j-funkcija Postaviti(J) {
   pohraniti kontekst u opisnik aktivne dretve
   premjestiti opisnik iz reda aktivne dretve u red pripravnih dretvi
   ako je ((Sem[J].v = 0) ∧ (red Sem[J] nije prazan)) {
      premjestiti prvu iz reda Sem[J] u red pripravnih dretvi
   } inače {
      Sem[J].v = Sem[J].v + 1
   }
   premjestiti opisnik prve pripravne u opisnik aktivne dretve
   obnoviti kontekst iz opisnika aktivne dretve
}
```

Prikazani semafor najčešće se naziva općim semaforom jer mu pozitivne vrijednosti nisu ograničene. Drugi, također prikladni nazivi jesu brojeći semafor ili brojilo događaja jer se može iskoristiti za navedene primjene. Drugačiji algoritam rada semafora prikazan u [Sil94] ima malo jednostavniju strukturu, tj. uvijek se obavlja smanjivanje vrijednosti semafora pri funkciji `Čekati()` te povećavanje u funkciji `Postaviti()`. Provjera prije ulaza u red ili vađenja dretve iz reda je nešto drukčija, ali je konačna funkcionalnost ista.

Semafori se koriste za ograničavanje istovremenog pristupa sredstvu sustava od strane više dretvi. Na primjer, ukoliko se želi ograničiti istovremeni pristup određenom sredstvu na najviše pet dretvi može se koristiti semafor s početnom vrijednošću pet. Dretve koje žele pristupiti sredstvu morale bi sadržavati sljedeće retke:

```
   [...]
   Čekati(J)
   [...] //korištenje sredstva
   Postaviti(J)
   [...]
```

Ostvarenje semafora u suvremenim operacijskim sustavim slična je gore navedenom algoritmu uz eventualno dodatne mogućnosti i provjere. Kod operacijskih sustava za rad u stvarnom vremenu te funkcije najčešće imaju dodatnu funkcionalnost rješavanja nekih problema koji su u tim sustavima kritični, npr. problem inverzije prioriteta (objašnjen u odjeljku 3.3.1).

Ograničenja semafora dolaze do izražaja u situacijama kada se koristi više od jednog semafora. U takvim slučajevima može se dogoditi da u nekom proizvoljnom slijedu izvođenja nastane potpuni zastoj [Bud03, Sil94, Til91a, Kos73, Nut00]. Potpuni zastoj je stanje sustava u kojem sve dretve ne mogu nastaviti svoj rad jer za nastavak trebaju određeno sredstvo (semafor) koje nije dostupno, odnosno već ga je zauzela neka druga dretva koja je također blokirana na nekom trećem sredstvu.

Najjednostavniji primjer je s dvije dretve koje u svom izvođenju zauzimaju sredstva `I` i `J` proizvoljnim redoslijedom te su im ponekad potrebna oba sredstva istovremeno. Ako se slučajno takve situacije poklope, tj. obje dretve trebaju oba sredstva, onda svaka od njih treba pozvati funkcije `Čekati(I)` i `Čekati(J)`. Ukoliko ti pozivi nisu uzastopni i izvedeni istim redoslijedom, može se dogoditi da svaka od dretvi zauzme po jedan semafor i ostane blokirana na drugom.

`[...] //dretva N`
`Čekati(I)`
`[...]`
`Čekati(J) //zastoj`
`[...]`
	`[...] //dretva M`
	`Čekati(J)`
	`[...]`
	`Čekati(I) //zastoj`
	`[...]`

Jedno od rješenja problema potpunog zastoja može se zasnivati na određenim pravilima kojima se programer mora pridržavati, kao npr. uvijek isti redoslijed poziva semafora, ali takva rješenja nisu primjenjiva na sve probleme. Drugi način, koji je i primijenjen na nekim sustavima, jest da se prošire pozivi za rad sa semaforima tako da se jednom pozivu jezgrine funkcije obavi više operacija nad jednim ili više semafora. Operacije nad skupom semafora će se tada ili sve obaviti ili se neće obaviti niti jedna i dretva će ostati blokirana. Takvim pozivom jedna dretva istovremeno može zauzeti više sredstava odjednom. Ako makar i jedno nije dostupno neće zauzeti niti jedno. Dosta se problema može riješiti na taj način, ali ne i svi, kao što je to prikazano u [Kos73]. Ostali algoritmi za rješenje problema potpunog zastoja su složeni [Raj91] i zahtijevaju poznavanje budućih zahtjeva svake dretve koja se natječe za zadana sredstva.

Iz navedenih razloga prišlo se razmatranju drugih mehanizama sinkronizacije kojima se problem potpunog zastoja može lakše izbjeći. Jedan od takvih mehanizama su monitori.

3.2. Klasifikacija monitora

Sam pojam *monitor* prvi puta uvodi Hoare [Hoa74]. Monitor je tada zamišljen kao klasa koja se sastoji od posebnih funkcija, nazvanih *monitorskim funkcijama*. Svojstvo tih funkcija jest da je najviše jedna dretva aktivna unutar njih. Odnosno, ukoliko je jedna dretva aktivna u nekoj od monitorskih funkcija sve ostale dretve ili čekaju na ulaz u monitorsku funkciju ili su u redu čekanja u nekom od redova uvjeta monitora ili su izvan monitora i ne traže ulaz. Pod frazom *unutar monitora* podrazumijeva se stanje dretve kada se ona nalazi u nekoj monitorskoj funkciji. Isto tako *ući u monitor* označava postupak ulaska dretve u monitorsku funkciju. Iako se u literaturi najčešće monitorska funkcija opisuje kao funkcija, u programu to može biti i samo dio kôda koji je omeđen odgovarajućim pozivima pomoćnih funkcija koje ostvaruju mehanizme *ući u monitor* i *izaći iz monitora*.

Za dretvu koja je unutar monitora kaže se da je vlasnik monitora ili da drži monitor. U nekom trenutku samo jedna dretva može biti unutar monitora. Dretva unutar monitorske funkcije može obavljati neke kritične operacije, obično nad dijeljenim podacima, ispitivati neke uvijete koje ostale dretve neće moći promijeniti dok ne uđu u monitor, itd. Dretva unutar monitora može ustanoviti da joj uvjeti za nastavak rada nisu ispunjeni te privremeno napustiti monitor i smjestiti se u odgovarajući red uvjeta i istovremeno osloboditi monitor. Neka od idućih dretvi će eventualno ispuniti uvjet prvoj dretvi, i osloboditi ju iz reda.

Koja će dretva nastaviti rad nakon oslobađanja dretve iz reda uvjeta ovisi o modelu monitora. Neka se dretva koja je u monitoru i poziva funkciju za oslobođenje druge dretve iz reda uvjeta nazove *signalna dretva*, a dretva koja je u redu uvjeta i biva oslobođena nazove *čekajuća dretva*. Monitor prikazan u [Hoa74] ili Hoareov monitor radi tako da signalna dretva u pozivu funkcije za oslobađanje čekajuće dretve bude stavljena u red signalnih dretvi, dok se čekajućoj dretvi istovremeno dozvoljava nastavak rada unutar monitora. Ideja ovog pristupa jest da se čekajućoj dretvi omogući nastavak točno u

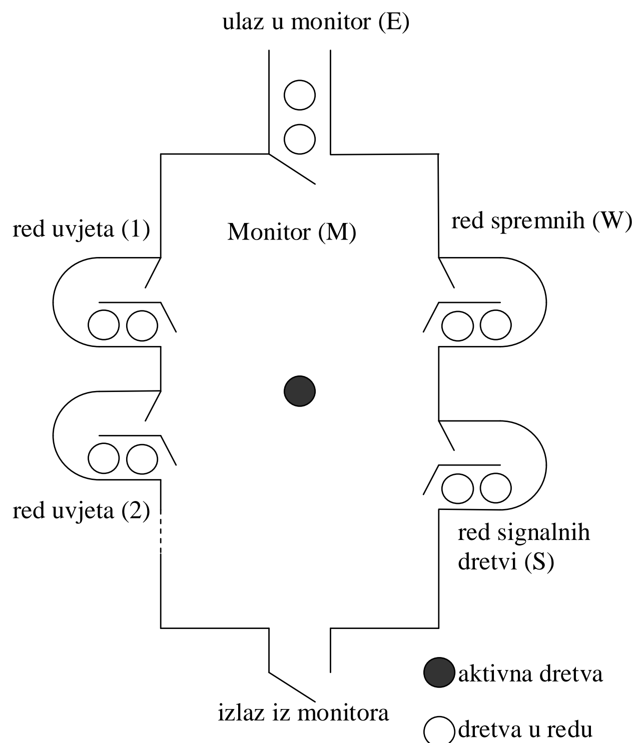
trenutku kada su joj uvjeti za nastavak rada ispunjeni i nepromijenjeni od strane neke druge dretve. Kako je monitor predložen kao koncept koji bi se ugradio u programski jezik (*PASCAL*), a ne u sam operacijski sustav, ostvarenje takvog monitora je problematično, pogotovo stoga što su se tada monitori ostvarivali koristeći semafore. Problem je kako osigurati da baš određena dretva bude sljedeća, a ne neka druga. To se ne može jednostavno napraviti bez mijenjanja raspoređivača u operacijskom sustavu. Osim navedenog, razlozi za traženje drugih modela monitora su u povećanju učinkovitosti.

Drugi algoritam rada funkcije za oslobađanje dretve iz reda uvjeta prikazao je Brinch-Hansen [Bri72], mada to on nije nazvao monitorima nego dijeljeno sredstvo (engl. *shared*). Kod ovih monitora signalna dretva ostaje u monitoru i nakon poziva funkcije za oslobađanje čekajuće dretve. Tek kada signalna dretva napusti monitor čekajuća će dretva dobiti priliku za ulaz u monitor, ako ima veći prioritet od ostalih dretvi koje također žele u monitor. Ovaj je algoritam mnogo bliži radu standardnih raspoređivača u jezgrama operacijskih sustava te je iz istog razloga taj koncept najčešće i korišten.

Osim navedena dva postoje još neki bitno drukčiji koncepti. Buhr [Buh95a] je podijelio monitore u tri skupine:

- eksplicitno signalizirajući monitori (engl. *explicit-signal monitors*),
- izravno izlazeći monitori (engl. *extended immediate return monitors*) i
- automatsko signalizirajući monitori (engl. *automatic signal monitors*).

Kod eksplicitno signalizirajućih monitora postoji funkcija *Signaliziraj* koja oslobađa jednu dretvu iz zadanog reda uvjeta monitora. Oslobođena dretva onda ili odmah ulazi u monitor ili odlazi u neki od red uvjeta prije nego što se propušta u monitor. Takvi monitori mogu se prikazati slikom 3.1 prema [Buh95a].



Slika 3.1 Opći model monitora

Monitor se sastoji od nekoliko redova. Prvi je red za ulaz u monitor (E). Kada dretva želi ući u monitor u kome se već nalazi druga dretva ona ulazi u ovaj red i čeka

oslobođenje monitora.

Pri izlazu dretve iz monitora mora se nekoj drugoj dretvi omogućiti ulazak u monitor, ako takva dretva postoji.

Dretva unutar monitora može ustanoviti da joj uvjeti za nastavak rada nisu ispunjeni te odlazi u red čekanja, odnosno red uvjeta. Pritom oslobađa monitor i propušta sljedeću dretvu u monitor.

Prema općenitom modelu postoje još dva reda, red spremnih (W) te red signalnih (S) dretvi. Dretva koja je unutar monitora i poziva jezgrinu funkciju za oslobođenje dretve iz reda uvjeta, u toj operaciji biva svrstana u red signalnih dretvi (S), dok dretva koja je oslobođena svrstava se u red spremnih dretvi. U ovisnosti o modelu monitora, tj. odnosu prioriteta redova S, W i E ovisiti će koja će sljedeća dretva ući u monitor.

U mnogim slučajevima je zapaženo da poziv jezgrine funkcije `Signalizirati()` slijedi poziv funkcije za izlaz iz monitora [How76]. Brinch-Hansen i Hoare su raspravljali o pojednostavljenoj inačici monitora koja bi pogodovala takvoj uporabi te su nastali monitori naziva izravno izlazeći monitori. Kod tih monitora dretva koja poziva funkciju `Signalizirati()`, nakon što oslobodi dretvu iz reda uvjeta sama izlazi iz monitora, oslobađajući time monitor za sljedeću dretvu, najčešće upravo oslobođenu iz reda uvjeta. Cijela je ta funkcionalnost ugrađena u funkciju `Signalizirati()`. U još jednostavnijoj inačici [Bud03] funkcija `Signalizirati()` obje dretve, ona koja signalizira i ona kojoj se signalizira ispunjenje uvjeta, izlaze iz monitora i mogu nastaviti s izvođenjem. U ovoj inačici u monitor ulazi sljedeća dretva koja čeka u ulaznom redu monitora. Oba navedena modela monitora mnogo su manje ekspresivna od eksplicitno signalizirajućih monitora i neki se programi koji se mogu napisati s eksplicitnim monitorima s izravno izlazećim monitorima ne mogu se jednostavno načiniti [How76, Buh95a]. Tako je predložena nova inačica monitora [How76] kod koje funkciji `Signaliziraj()` neposredno slijedi funkcija za ulaz u red uvjeta `Čekati()`, odnosno funkcionalnost obje funkcije ugrađena je u novu funkciju `Signalizirati()`. Ti su monitori nazvani prošireni izravno izlazeći monitori.

Automatsko signalizirajući monitori nemaju funkciju `Signalizirati()` te je funkcija `Čekati()` promijenjena u `Čekati(uvjetni_izraz)` [Buh95a]. Kod ovih monitora jezgrina funkcija za izlaz iz monitora mora izračunati uvijete svih dretvi iz svih redova uvjeta. Ako je uvjet nekoj od njih zadovoljen dretva se propušta u ulazni red monitora. Funkcija izlaza iz monitora na ovaj način može postati, ne samo složena za izvedbu, već i vrlo zahtjevna, zbog mogućeg velikog broja izračunavanja i provjera uvjeta za nastavak rada dretvi.

Tablica 3.1 prikazuje korisne modele monitora u ovisnosti o prioritetu redova S, W i E te ostale značajnije modele monitora [Buh95a].

Monitori su podijeljeni u dvije skupine: prioritetni i neprioritetni. Kod prioritetnih monitora dretve koje su unutar monitora ili u nekom od reda uvjeta imaju veći prioritet od dretvi koje čekaju na ulaz u monitor, tj. ulazni red ima najmanji prioritet. Iduća podjela na blokirajuće i neblokirajuće odnosi se na dretvu koja signalizira ispunjenje uvjeta, odnosno je li ta funkcija blokira ili ne signalizirajuću dretvu. Kod polublokirajućih dretvi nije unaprijed određeno koja će dretva ostati u monitoru, signalizirajuća ili ona kojoj se signalizira, već to ovisi o nekim drugim parametrima, npr. o prioritetima samih dretvi. Posljednja dva retka u tablici zauzimaju proširene izravno-izlazeći i automatsko-signalizirajući monitori.

Tablica 3.1 Klasifikacija monitora

Signalna svojstva	Prioritetni	Neprioritetni
Blokirajući	$E_P < S_P < W_P$	$E_P = S_P < W_P$
Neblokirajući	$E_P < W_P < S_P$	$E_P = W_P < S_P$
Polublokirajući	$E_P < W_P = S_P$	$E_P = W_P = S_P$
Prošireni izravno-izlazeći	$E_P < W_P$	$E_P = W_P$
Automatsko-signalizirajući	$E_P < W_P$	$E_P = W_P$

Formalna semantika te pravila potvrđivanja ispravnosti svih modela iz tablice 3.1 dana su u [Buh95a]. U istome je radu i prikazana ocjena učinkovitosti pojedinih modela.

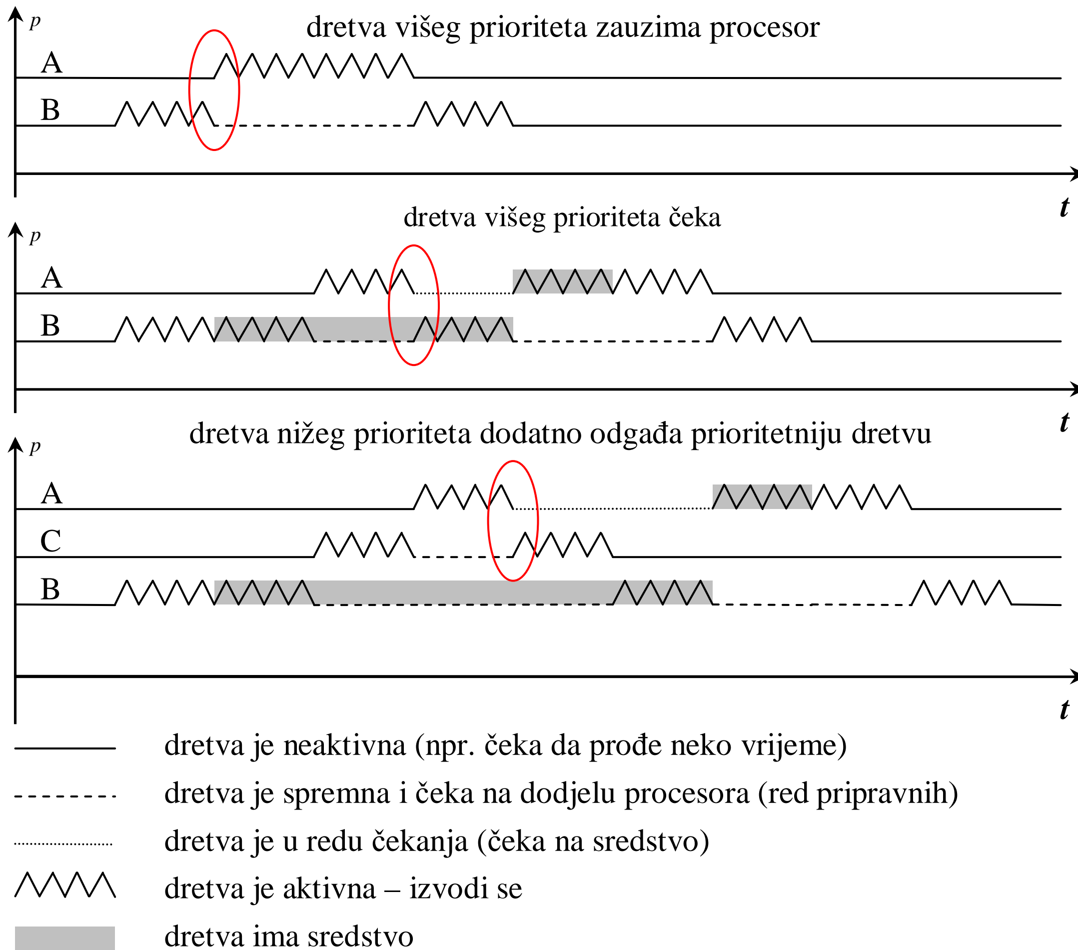
3.3. Uporaba monitora u ugrađenim sustavima

Različiti modeli monitora prikazani u prethodnom potpoglavlju iako su svi semantički ispravni nisu jednako dobar izbor za uporabu u ugrađenim sustavima. U ugrađenim sustavima određenost, pouzdanost i jednostavnost mnogo su značajniji od prosječne učinkovitosti (prikazane u [Buh95a]). Dretve u ugrađenim sustavima uglavnom se međusobno razlikuju po važnosti, odnosno prioritetu. Pri dodjeli procesora dretva većeg prioriteta ima prednost ispred one manjeg prioriteta. Iako je takva ili slična situacija i u ostalim sustavima, odnosno, ostalim operacijskim sustavima koji nisu namijenjenih ugrađenoj uporabi, kod njih je prioritet nešto što određuje koliko će pojedina dretva dobiti procesorskog vremena, a ne kada će ta dretva postati aktivna. U operacijskim sustavima za rad u stvarnom vremenu jasno je definirano da kada dretva višeg prioriteta postaje spremna za izvođenje ona istiskuje dretvu nižeg prioriteta koja se trenutno izvodi. Međutim, i u takvim se sustavima događaju slučajevi kada dretva nižeg prioriteta blokira izvođenje dretve višeg prioriteta, odnosno dolazi do problem inverzije prioriteta.

3.3.1. Problem inverzije prioriteta

Problem inverzije prioriteta nastaje kada dretva višeg prioriteta za nastavak rada treba sredstvo koje je zauzela druga dretva nižeg prioriteta. Na slici 3.2 prikazana su tri slučaja koja se mogu pojaviti u višedretvenom sustavu.

Problem se u raznim slučajevima rješava na razne načine. Kada je sustav dovoljno određen i poznate su potrebe za sredstvima svih dretvi u sustavu, problem se može izbjeći tako da se dretvi nižeg prioriteta privremeno onemogući zauzimanje sredstava koja će uskoro biti potrebna dretvi višeg prioriteta [Raj91]. Općenito su sustavi s unaprijed poznatim vremenski određenim potrebama dretvi za sredstvima rijetki te gornja metoda u općim slučajevima nije odgovarajuća. U ugrađenim se sustavima ipak se češće susreću takvi slučajevi te je ova metoda kod njih prihvatljiva. Budući je ideja ovog rada definirati sučelje za širi krug ugrađenih sustava ova se metoda neće razmatrati.



Slika 3.2 Problem inverzije prioriteta

Metode koje se najčešće koriste u slučajevima problema inverzije prioriteta ne rješavaju sam problem nego ublažavaju njegove posljedice. To rade tako da se dretvi koja je zauzela sredstva potrebna prioritetnijoj dretvi (korištenjem sinkronizacijskog mehanizma) omogući što brži rad do oslobađanja dotičnih sredstava. Dva najpoznatija takva protokola su protokol nasljeđivanja prioriteta (engl. *priority inheritance protocol*) i protokol vršnog prioriteta (engl. *priority ceiling protocol*). Važna pretpostavka za oba protokola je da dretve nakon što sredstvo zauzmu isto će i osloboditi nakon nekog vremena.

3.3.1.1. Protokol nasljeđivanja prioriteta

Kod protokola nasljeđivanja prioriteta [Loc88, Sha90, Raj91] dretvi nižeg prioriteta, koja je zauzela određeno sredstvo podiže se prioritet na razinu blokirane dretve. To se zbiva u samoj funkciji sinkronizacije koju poziva prioritetnija dretva. Povećanje prioriteta treba zapamtiti u odgovarajućoj podatkovnoj strukturi da bi se kasnije omogućilo vraćanje prioriteta na prijašnju razinu. Podatkovna struktura mora biti dinamičkog tipa, radi pohranjivanja podataka o povećanju prioriteta, jer ono može biti i višestruko. Smanjivanje prioriteta, odnosno, vraćanje na razinu koju je dretva imala prije nasljeđivanja, treba obaviti u funkcijama sinkronizacije i to u funkcijama za oslobađanje sredstva. U poboljšanoj inačici protokola, nasljeđivanje prioriteta se može primijeniti i na više od jedne dretve, odnosno i tranzitivno. Na primjer, ukoliko dretva kojoj se prioritet povećava i sama

čeka na drugo sredstvo, tada treba povećati prioritet i dretvi koja drži to drugo sredstvo, i tako dalje. U jednom se pozivu funkcije sinkronizacije mogu tako povećati prioriteti više dretvi.

Funkcija za jednostruko nasljeđivanje prioriteta koja se poziva iz funkcije za zauzimanje sredstva, npr. funkcije `Čekati(J)` kod semafora, može izgledati kao u nastavku:

```
funkcija nasljedni_prioritet_povećati(S, prioritet) {
   dretva = vlasnik_sredstva(S)
   ako je (dretva.prioritet < prioritet) {
      zadnji = dretva.dp_zadnji+1
      dretva.din_pri[zadnji].prioritet = dretva.prioritet
      dretva.din_pri[zadnji].uzrok = S
      dretva.dp_zadnji = zadnji
      dretva.prioritet = prioritet
   }
}
```

Kao što je vidljivo, funkciji su potrebni neki podaci sustava o dretvama i njihovim prioritetima, sredstvima i trenutnim vlasnicima tih sredstava. Slovo `S` identificira sredstvo koje je uzrok povećanja prioriteta, a može biti semafor ili monitor ili neki drugi sinkronizacijski mehanizam.

Funkciju nasljeđivanja treba pozvati isključivo ako se dotično sredstvo ne može zauzeti, tj. kada dretva ulazi u red. Funkciju sinkronizacije mora slijediti raspoređivač koji će uzeti u obzir promijenjene prioritete dretve. Može se primijetiti da je struktura podataka za spremanje povijesti prioriteta dinamička, tj. nije unaprijed definirana veličina polja. U primjeni se ipak može pretpostaviti ograničena dubina povijesti tako da se dinamičkim zauzimanjem spremnika ne pridonese složenosti sustava. Pored prioriteta dretve, potrebno je zapisati i sredstvo koje je uzrokovalo povećanje prioriteta.

Pri povratku sredstva dretvi treba vratiti njen prijašnji prioritet, tj. treba pozvati funkciju `nasljedni_prioritet_smanjiti()`, koja slijedi u nastavku, iz funkcije za vraćanje sredstva, npr. `Postaviti(J)` kod semafora.

```
funkcija nasljedni_prioritet_smanjiti(S, dretva) {
   za i=1 do dretva.dp_zadnji {
      ako je (dretva.din_pri[i].uzrok = S) {
         dretva.din_pri[zadnji].uzrok = 0
   }  }
   n = dretva.dp_zadnji
   dok je ((n > 0) ∧ (dretva.din_pri[n].uzrok = 0)){
      dretva.prioritet = dretva.din_pri[n].prioritet
      n = n-1
   }
   dretva.dp_zadnji = n
}
```

Kao i prethodnu funkciju i ovu mora slijediti poziv raspoređivača koji će uzeti u obzir nove prioritete.

U slučaju višestrukog nasljeđivanja prioriteta funkcija za nasljeđivanje postaje nešto složenija. U toj funkciji se osim dretvi koja drži potrebno sredstvo, prioritet može podići i

dretvi koja blokira tu dretvu, te tako dalje, dretvi koja blokira prethodnu dretvu itd. Kôd funkcije slijedi u nastavku:

```
funkcija Nasljedni_prioritet_povećati(S, prioritet) {
   dretva = vlasnik_sredstva(S)
   dok je (dretva.prioritet < prioritet) {
      zadnji = dretva.dp_zadnji+1
      dretva.din_pri[zadnji].prioritet = dretva.prioritet
      dretva.din_pri[zadnji].uzrok = S
      dretva.dp_zadnji = zadnji
      dretva.prioritet = prioritet
      R = dretva.red //sredstvo na koje čeka 'dretva'
      ako je (R ≠ red pripravnih dretvi) {
         dretva_vlasnik = vlasnik_sredstva(R)
         ako je (dretva.prioritet > dretva_vlasnik.prioritet) {
            dretva = dretva_vlasnik
            S = R
   }  }  }
}
```

Ova metoda za razliku od jednostavnije inačice unosi nešto nedeterminizma u trajanju izvođenja funkcije zbog tranzitivnog povećavanja prioriteta.

Protokol nasljeđivanja prioriteta ne rješava problem potpunog zastoja već se za sprječavanje i rješavanje istog moraju upotrijebiti dodatni algoritmi ili postupci.

3.3.1.2. Protokol vršnog prioriteta

Za protokol vršnog prioriteta postoje dvije inačice: originalni protokol vršnog prioriteta [Sha90, Raj91] i pojednostavljeni protokol vršnog prioriteta [Bur01]. Pojednostavljeni protokol ima još nekoliko naziva: izravni protokol vršnog prioriteta (engl. *Immediate ceiling priority protocol*), protokol zaštite prioritetom (engl. *Priority protect protocol*) kod POSIX standarda te oponašanje protokola vršnog prioriteta (engl. *Priority ceiling emulation*) kod programskog jezika Java.

Kod obje inačice protokola svakom se sredstvu pridjeljuje vršni prioritet, tj. prioritet najznačajnijeg zadatka koji može zauzeti dotično sredstvo.

Originalni, složeniji protokol ima za cilj i izbjegavanje nastajanja potpunog zastoja te višestrukog blokiranja zadataka. Osnovna ideja protokola je da kada jedan zadatak pri ulasku u kritični odsječak zaustavi izvođenje kritičnog odsječka nekog drugog zadatka, tada to obavlja s prioritetom većim od vršnog prioriteta svih zaustavljenih kritičnih odsječaka. Ukoliko kritični odsječak u koji zadatak želi ući nema dovoljno velik prioritet, ulazak mu se privremeno zabranjuje dok se taj uvjet ne ispuni, tj. dok se prije ne izvrše svi kritični odsječci s većim vršnim prioritetom.

Definicija protokola koja slijedi preuzeta je iz [Raj91].

Zadatku J koji ima najveći prioritet među zadacima pripravnim za izvođenje dodjeljuje se procesor. Neka S^* označava binarni semafor najvećeg prioriteta koji je zaključan od strane nekog drugog zadatka.

1. Prije nego li zadatak J uđe u kritični odsječak mora zaključati semafor S koji zaštićuje pristup zajedničkim podacima. Zadatak J biti će blokiran na zaključavanju semafora S, ako prioritet zadatka J nije veći od vršnog prioriteta semafora S^*. U

ovom slučaju kaže se da je *J* blokiran na semaforu S^* i da je blokiran zadatkom koji je zaključao taj semafor. U protivnom, ako je prioritet od *J* veći, tada mu se dozvoljava zaključavanje semafora *S* i ulazak u kritični odsječak. Kada *J* završi sa svojim kritičnim odsječkom i oslobodi semafor *S*, zadatak najvećeg prioriteta koji je čekao na taj semafor, ako postoji, biti će aktiviran.

Treba primijetiti da ukoliko je semafor *S* već zaključan tada je to s zadatkom koji ima osnovni ili naslijeđeni prioritet jednak ili veći od prioriteta zadatka *J* koji će i u tom slučaju biti blokiran na semaforu *S*.

2. Zadatak *J* koristi pridijeljeni mu prioritet osim ako se nalazi u kritičnom odsječku kojim blokira prioritetnije zadatke. Ako blokira prioritetnije zadatke tada on poprima prioritet $p(J_H)$ koji je jednak prioritetu blokiranog zadatka s najvećim prioritetom. Kada *J* izlazi iz kritičnog odsječka vraća mu se prioritet koji je imao prije ulaska u kritični odsječak. Nasljeđivanje prioriteta je tranzitivno. Operacije nasljeđivanja prioriteta i povratka prioriteta su nedjeljive operacije.
3. Zadatak *J* izvan kritičnog odsječka može blokirati zadatak J_L ukoliko je prioritet zadatka *J* veći od prioriteta, naslijeđenog ili zadanog, s kojim se izvodi zadatak J_L.

Analizom ovog protokola može se ustanoviti sličnost s protokolom nasljeđivanja prioriteta. I kod ovog se protokola prioritet blokiranog zadatka nasljeđuje u trenutku blokiranja. Osnovna je razlika što se u nekim slučajevima zadatku neće dozvoliti zaključavanje semafora iako je on slobodan, a zbog mogućeg blokiranja zadataka većeg prioriteta nad drugim semaforima. Na prvi pogled to blokiranje izgleda nepotrebno, ali ono zapravo pridonosi rješavanju problema potpunog zastoja, barem osnovnih oblika potpunog zastoja [Bur01]. U slučajevima gdje je problem potpunog zastoja značajan i često se pojavljuje ovaj je protokol dobar odabir.

Primjena navedenog protokola u monitorima zbog složenosti zahtjeva posebnu pažnju. Za razliku od protokola nasljeđivanja prioriteta i ako je monitor slobodan treba provjeriti smije li zadatak ući u monitor. Ako se u monitoru već nalazi neka dretva treba joj eventualno podići prioritet na razinu pozivajuće dretve.

Funkcija `provjera_ulaza()` poziva se prije zauzeća sredstva (semafor, monitor) kada je on slobodan ili kada se oslobodi. Slovo `S` označava sredstvo, `pri` prioritet pozivajuće dretve, `vlasnik[i]` označava dretvu koja je zaključala sredstvo `i`, `red[i]` označava red dretvi koji čekaju na sredstvo `i` te `VP(i)` vršni prioritet sredstva `i`.

```
funkcija provjera_ulaza(S, pri) {
   ULAZ = DOZVOLJEN
   za i=1 do broj_sredstava {
      ako je ((i ≠ S) ∧ ((vlasnik[i] ≠ NITKO) ∨ (red[i] ≠ PRAZAN))
             ∧ (VP(i)≥ pri)) {
         ULAZ = ZABRANJEN
   } }
   vrati ULAZ
}
```

Ukoliko dretva većeg prioriteta ne može zauzeti sredstvo jer ga ima dretva nižeg prioriteta, tada se dretvi nižeg prioriteta povećava prioritet i to se radi tranzitivno, ako je potrebno. Funkcije za povećavanje i smanjivanje prioriteta identične su funkcijama prikazanim za protokol nasljeđivanja prioriteta

Kod pojednostavljenog protokola vršnog prioriteta dretvi se odmah pri zauzimanju sredstva podigne prioritet na unaprijed određenu vršnu vrijednost. Na taj se način za vrijeme korištenja nekog sredstva dretvama povećava prioritet da bi one što prije završile s njegovim korištenjem i oslobodile ga za dretve višeg prioriteta. Protokol je jednostavniji za ostvarenje od prethodnog, ali je po pitanju učinkovitosti lošiji. Dretva nižeg prioriteta zauzećem određenog sredstva dobiva veći prioritet i istiskuje prioritetnije dretve čak i onda kada od dretvi višeg prioriteta ne postoji trenutna potreba za sredstvom. Funkcija `podignuti_prioritet()` može se pozivati u funkciji sinkronizacije pri zauzimanju sredstva. Najjednostavniji oblik tih funkcija dan je u nastavku.

```
funkcija podignuti_prioritet(S, dretva) {
   pohranjeni_prioriteti[S] = dretva.prioritet
   dretva.prioritet = Prioriteti[S]
}
funkcija spustiti_prioritet(S, dretva) {
   dretva.prioritet = pohranjeni_prioriteti[S]
}
```

U elementu polja `Prioriteti[S]` pohranjeni su prioriteti koje dretve poprimaju kada zauzmu sredstvo `S`.

Navedene funkcije neće ispravno raditi ukoliko dretve zauzimaju više sredstava te se ta sredstva ne oslobađaju obrnutim redoslijedom od redoslijeda zauzeća. Npr. ako dretva zauzima redom sredstva S1, S2, S3 i S4 tada, da bi gornji algoritam ispravno radio, potrebno je sredstva oslobađati redoslijedom: S4, S3, S2 i S1.

Općenitiji te nešto složeniji algoritam koji radi u općem slučaju zahtjeva dodatnu podatkovnu strukturu istu kao i kod protokola nasljeđivanja prioriteta:

```
funkcija podignuti_prioritet(S, prioritet) {
   zadnji = dretva.dp_zadnji+1
   dretva.din_pri[zadnji].prioritet = dretva.prioritet
   dretva.din_pri[zadnji].uzrok = S
   dretva.dp_zadnji = zadnji
   dretva.prioritet = Prioriteti[S]
}
```

Smanjivanje prioriteta obavlja se na isti način kao i kod protokola nasljeđivanja prioriteta te se koristi i ista funkcija `nasljedni_prioritet_smanjiti()`.

Kao i kod prethodnog protokola obje funkcije sinkronizacije (za zauzimanje i za oslobađanje sredstva) treba slijediti raspoređivač. Raspoređivač može biti i dio samih funkcija sinkronizacije.

3.3.2. Odabir monitora za ugrađene sustave

Od monitora navedenih u tablici 3.1 potrebno je odabrati onaj model koji će poštovati prioritete svih dretvi bez obzira na red u kojem se nalaze te tako izbjeći nepotrebne slučajeve inverzije prioriteta.

Prioritetni monitori i oni kod kojih pojedini redovi imaju različite prioritete nisu prihvatljivi zbog mogućnosti pojave inverzije prioriteta. Da bi se to prikazalo neka se u sustavu nalaze tri dretve:

- D_Q – dretva koja čeka u redu uvjeta,

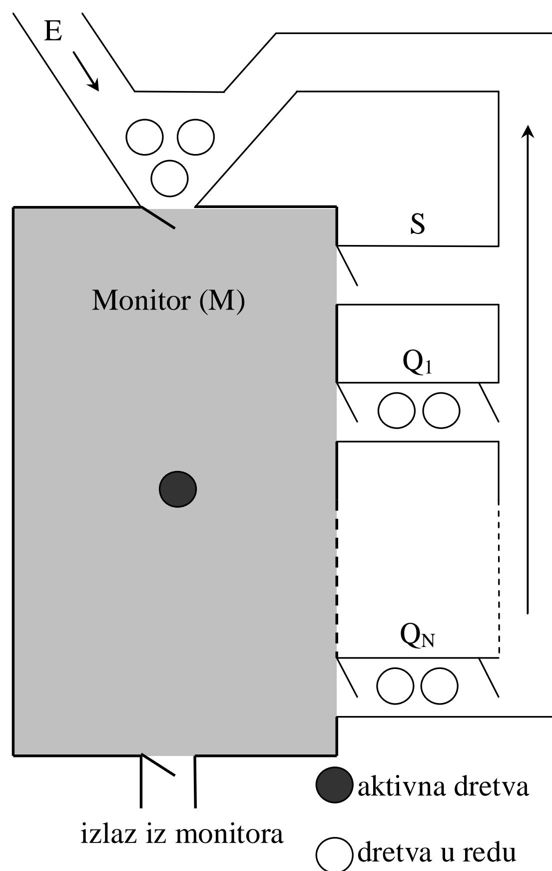
- D_E – dretva koja čeka na ulaz u monitor te
- D_A – aktivna dretva unutar monitora.

Ukoliko je prioritet redova uvjeta veći od ulaznog reda tada dretva u ulaznom redu monitora D_E koja može imati veći prioritet od ostalih dretvi u sustavu, pored toga što mora čekati da monitor napusti dretva koja je trenutno aktivna unutar monitora D_A, morati će čekati i na dretve D_Q koje ta dretva oslobađa iz reda uvjeta istog monitora. U obrnutom slučaju, kada ulazni red ima veći prioritet, može se dogoditi da je dretva najvećeg prioriteta baš oslobođena iz reda uvjeta, ali ipak ne nastavlja rad jer ulazni red ima veći prioritet.

Monitor kod kojeg su prioriteti svih redova isti $E_P = W_P = S_P$, tj. ulaznog reda za monitor, reda spremnih dretvi i reda signalnih dretvi, omogućuje da nakon bilo kojeg poziva funkcija za rad s monitorom prioritet samih dretvi utječe na to koja će se nastaviti izvoditi unutar ili izvan monitora, a ne to iz kojeg su reda izašle.

Zbog navedenog je razloga upravo taj model monitora odabran. U tablici 3.1 odabrani se monitor može nazvati *neprioritetni polublokirajući*. U nastavku rada kada se spominje monitor misli se na dotični model monitora.

Umjesto tri reda (E,W,S) moguće je koristiti samo jedan red za ulaz u monitor E prema slici 3.3, uz odgovarajuće funkcije za rad s monitorom.



Slika 3.3 Odabrani model monitora

Osim ulaznog reda E te redova uvjeta Q_1, Q_2, ... Q_N u monitoru nema drugih redova. Redovi S i W iz općenitog modela u ovom se slučaju spajaju s redom E, tj. odgovarajuće ih funkcije prebacuju u taj red.

Za ostvarenje monitora potrebne su četiri funkcije:

(1) funkcija za ulaz u monitor: `Ući_u_monitor(M)`,

(2) funkcija za izlaz iz monitora: `Izaći_iz_monitora(M)`,

(3) funkcija za ulaz u red uvjeta: `Uvrstiti_u_red_uvjeta(M, K)` te

(4) funkcija za oslobađanje iz reda uvjeta: `Osloboditi_iz_reda_uvjeta(M, K)`.

Navedene funkcije su jezgrine funkcije i to osnovne jezgrine funkcije. Raspoređivač koji radi na načelu prioriteta dretvi može se ugraditi u ove funkcije tako da bi minimalna jezgra za ugrađeni sustav mogla sadržavati samo dotične funkcije, uz prateću podatkovnu strukturu.

Funkcijom `Ući_u_monitor(M)` dretva će ili ući u monitor i nastaviti raditi unutar monitora, ili će biti stavljena u red E dotičnog monitora u kom slučaju aktivna dretva postaje prva iz reda pripravnih.

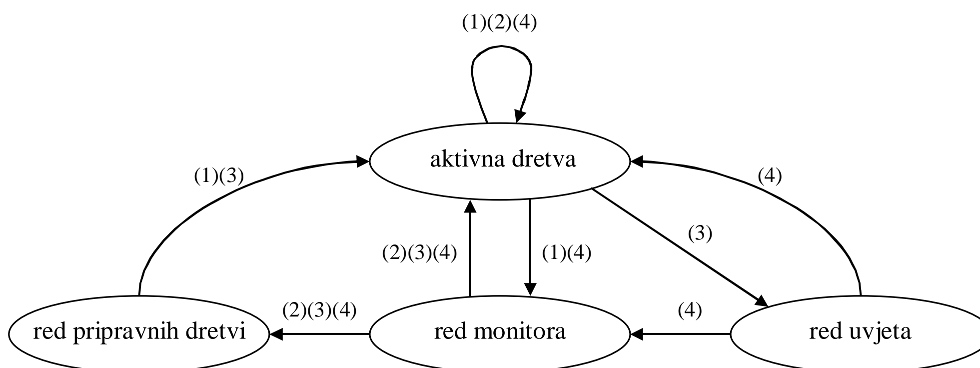
Pozivom funkcije `Izaći_iz_monitora(M)` dretva oslobađa monitor. Ukoliko ima dretvi u ulaznom redu tada se prvoj iz tog reda dodjeljuje monitor i premješta ju se u red pripravnih.

Funkcija `Uvrstiti_u_red_uvjeta(M, K)` smije se pozvati jedino ako je dretva unutar monitora. Pozivom se dretva stavlja u red uvjeta te se oslobađa monitor. Ukoliko je neka dretva čekala na ulaz u monitor tada se toj dretvi dodjeljuje monitor i premješta ju se u red pripravnih dretvi.

Funkcija `Osloboditi_iz_reda_uvjeta(M, K)` najčešće sama određuje ponašanje monitora, tj. model monitora. U odabranom modelu ta će funkcija prvo dretvu koja ju poziva premjestiti u ulazni red monitora. Potom će u isti red premjestiti prvu dretvu iz zadanog reda uvjeta. Iz tog će reda tada odabrati dretvu najvećeg prioriteta kojoj će dodijeliti monitor te će ju premjestiti u red pripravnih dretvi.

Svaka od navedenih funkcija završava aktiviranjem dretve najvećeg prioriteta iz reda pripravnih dretvi.

Iz navedenog opisa vidljivo je da svaki poziv funkcije može rezultirati promjenom aktivne dretve. Pozivom `Uvrstiti_u_red_uvjeta(M, K)` to će se sigurno i dogoditi, dok za ostale pozive to ovisi o prioritetu i stanju ostalih dretvi u sustavu. Dretva se može nalaziti u četiri stanja kako je to prikazano na slici 3.4.



Slika 3.4 Stanje dretve u sustavu

Iz aktivnog stanja dretva s nekim od poziva za rad s monitorom može otići u jedan od

drugih redova te također može promijeniti stanje druge dretve u sustavu. U jednom pozivu stanje dretve može se više puta promijeniti, npr. može ju se najprije pomaknuti iz reda uvjeta u red monitora, potom u red pripravnih te konačno u aktivnu, ukoliko ta dretva ima veći prioritet od ostalih.

U nastavku rada razmatrati će se jednostavni oblici monitora zasnovani na predloženom modelu te sinkronizacija pomoću njih. Složeni problemi kao što su rekurzivno pozivanje funkcija za ulaz i izlaz iz monitora neće se razmatrati. Rješenje takvih problema zahtijevalo bi dodatnu podatkovnu strukturu i povećalo kôd i složenost funkcija, a to se upravo želi izbjeći.

4. OSTVARENJE MONITORA

Monitori mijenjaju stanja dretvi u sustavu te kao takvi moraju imati pristup njihovim podacima, tj. podacima jezgre. Budući se razmatraju ugrađeni sustavi s minimalnim sklopovljem poželjno je da podatkovna struktura jezgre bude što manja, sa što manje ili čak bez podataka za koje je potrebno dinamičko zauzimanje spremničkog prostora.

4.1. Podatkovna struktura jezgre

Odabrana podatkovna struktura usko je povezana s ostvarenjem monitora te s odabranim modelom ugrađenih sustava. Sve su dretve statičke i inicijalizirane prije pokretanja sustava. Tijekom rada sustava nema stvaranja novih dretvi niti postojeće nestaju iz sustava. Proširenje na općenitiji model s dinamičkim stvaranjem i uništavanjem dretvi zahtijevalo bi dodatnu podatkovnu strukturu i dodatne funkcije za dinamičko upravljanje spremničkim prostorom jezgre. Te metode nisu složene, ali budući se želi ostvariti minimalna jezgra one nisu razmatrane u ovom radu.

Struktura podataka jezgre definirana je sljedećim elementima:

- `Dretva[]` – struktura s podacima o dretvama,
- `Monitor[]` – struktura podataka o monitorima,
- `Aktivna` – kazaljka na trenutno aktivnu dretvu te
- `Pripravne` – kazaljka na red pripravnih dretvi.

Aktivna dretva je ona dretva koja se trenutno izvodi, odnosno, u jezgrinim funkcijama to je dretva koja je pozvala jezgrinu funkciju. Pripravne dretve su one dretve koje su spremne za izvođenje te čekaju da im raspoređivač dodjeli procesor. U redu pripravnih dretvi s najmanjim prioritetom nalazi se i tzv. zamjenska (engl. *idle*) dretva, tj. dretva koja će se izvoditi kada niti jedna druga trenutno nije pripravna.

U sustavu postoje tri različita tipa redova: red pripravnih dretvi, redovi za ulaz u monitor i redovi na uvjetima unutar monitora. Odabran je pristup kod kojeg redovi nisu složeni po prioritetima već se pri stavljanju dretve u red ona stavlja na fizički prvo mjesto. Prilikom oslobađanja dretve iz reda, red se pretražuje i oslobađa se dretva s najvećim prioritetom najbliže kraju reda. Oslobađanje je time nešto dulje, ali je izbjegnuto preslagivanje liste prilikom ulaska dretve u red tako da je ukupna složenost otprilike ista. Prednost takve liste je što se promjenom prioriteta neke dretve njen položaj u listi ne mora mijenjati.

Osnovni podaci o dretvama organizirani su u strukturi `Dretva`:

- `id` – identifikacijski broj dretve,
- `pri` – prioritet dretve,
- `red` – stanje ili red u kojem se dretva nalazi: `AKTIVNA`, `PRIPRAVNA`, `RED_UVJETA` ili indeks monitora na čiji ulaz čeka,
- `iduća` – kazaljka na iduću dretvu u redu, ako se nalazi u nekom redu i
- `kasniti` – vrijeme kada dretvu treba ponovo aktivirati, tj. premjestiti iz reda

čekanja (red uvjeta) u red pripravnih.

Osim navedenih podataka ukoliko se koristi protokol nasljeđivanja prioriteta ili pojednostavljeni protokol vršnog prioriteta za svaku je dretvu potrebno dodatno rezervirati podatkovnu strukturu za spremanje povijesti prioriteta:

- `din_pri` – polje strukture s podacima:
  - `uzrok` – monitor zbog kojeg je prioritet uvećavan,
  - `pri` – prijašnji prioritet dretve,
- `dp_zadnji` – trenutni broj zapisa u strukturi `din_pri`.

Originalni protokol vršnog prioriteta nije napravljen jer on bitno utječe na funkcije za ostvarenje monitora. Dok se protokol nasljeđivanja prioriteta i pojednostavljeni protokol vršnog prioriteta lako ugrade u pojedini dio kôda, kod originalnog protokola vršnog prioriteta to nije moguće. Korišteni koncept monitora jest da je neki monitor slobodan ili zaključan. Ako je slobodan tada je red dretvi koje čekaju na ulaz prazan. To međutim ne bi bio slučaj kada bi se koristilo originalni protokol vršnog prioriteta. Za taj bi protokol trebalo promijeniti funkcije za ostvarenje monitora.

Ostali podaci, kao što su adrese kôda, podataka dretve i stoga potrebni su pri ostvarenju u strojnom jeziku te su oni definirani u poglavlju ostvarenja monitora na specifičnoj arhitekturi (poglavlje 7).

Struktura podataka o monitoru sadrži sljedeće elemente:

- `vlasnik` – kazaljka na dretvu koja je unutar monitora ili ima vrijednost `NITKO` ukoliko niti jedna dretva nije u monitoru,
- `red` – kazaljka na prvu dretvu u redu monitora, `KRAJ_LISTE` ako je red prazan,
- `red_uvjeta[]` – polje kazaljki kod kojeg se svaki element koristi za jedan red uvjeta, npr. kazaljka na prvu dretvu u trećem redu uvjeta je `red_uvjeta[3]`,
- `prioritet` – ako se koristi protokol vršnog prioriteta tada se uz svaki monitor veže i prioritet s kojim će se dretve izvoditi unutar njega.

Za ostvarenje redova nije potrebna nikakva dinamička struktura podataka. Budući se svaka dretva može naći samo u jednom redu korišten je element dretvene strukture `iduća`.

4.2. Funkcije za ostvarenje monitora

Funkcije za ostvarenje monitora su jezgrine funkcije te su kao takve i označene. Svaka jezgrina funkcija počinje s instrukcijama za zabranu prekida i završava s dozvolom prekidanja. U naprednijim arhitekturama se jezgrine funkcije mogu ostvariti ulaskom u nadgledni način rada korištenjem programskog prekida. Kod arhitektura koje nemaju takav način rada dovoljno je zabraniti prekide nekom od instrukcija ili postavljanjem odgovarajućih zastavica u za to određeni registar procesora (statusni registar). Daljnja pretpostavka je da se odmah pri ulasku u jezgrinu funkciju kontekst aktivne dretve pohranjuje te da se pri izlasku obnavlja kontekst aktivne dretve. Aktivna dretva pri izlasku iz jezgrine funkcije ne mora biti ista dretva koja je pozvala jezgrinu funkciju. Kontekst dretve čine sadržaji registara procesora.

Pretpostavljeno ponašanje jezgrinih funkcija je prema tome:

```
j-funkcija funkcija(parametri) {
   zabraniti prekidanje
   pohraniti kontekst u opisnik aktivne dretve
   KÔD JEZGRINE FUNKCIJE
   obnoviti kontekst iz opisnika aktivne dretve
   dozvoliti prekidanje
}
```

Zbog sažetijeg zapisa u jezgrinim funkcijama navedenim u nastavku to se pretpostavljeno ponašanje neće ispisivati, tj. ispisati će se jedino „KÔD JEZGRINE FUNKCIJE“, ali će se podrazumijevati da postoje ostali dijelovi kôda prije i poslije.

Dijelovi kôda koji se ponavljaju u više funkcija izdvojeni su u zasebne funkcije. Te se funkcije smiju pozivati isključivo iz jezgrinih funkcija.

Slijede funkcije za ostvarenje monitora.

```
j-funkcija Ući_u_monitor(M) {
   ako je (Monitor[M].vlasnik = NITKO) {
      Monitor[M].vlasnik = Aktivna.id
      #povećati_prioritet(Aktivna, M)
   }
   inače {
      *nasljedni_prioriteti_povećati(Aktivna.pri, M)
      Aktivna.red = M
      Aktivna.iduća = Monitor[M].red
      Monitor[M].red = Aktivna
      Prebaciti_prvu_iz_reda_Pripravnih_u_red_Aktivna()
   }
}
```

Jezgrina funkcija za ulaz u monitor najprije provjerava je li monitor slobodan. Ukoliko jest on se dodjeljuje pozivajućoj dretvi koja i nastavlja s radom nakon završetka te funkcije. Ukoliko se unutar monitora nalazi neka druga dretva pozivajuća se dretva stavlja u red čekanja na ulaz u monitor. Ukoliko se koristi jedan od protokola za rješavanje inverzije prioriteta onda se poziva odgovarajuća funkcija. Znakovi `*` i `#` ispred imena funkcija označavaju da su ti pozivi uvjetni. Znak `*` označava korištenje protokola nasljeđivanja prioriteta dok znak `#` označava protokol vršnih prioriteta. Podizanje prioriteta treba obaviti kada dretva ulazi u monitor, tj. odmah nakon što se dretvi pridijeli monitor. Kod protokola nasljeđivanja prioriteta, ako pozivajuća dretva bude stavljena u red monitora treba provjeriti prioritet dretve koja je unutar monitora. Ako joj je prioritet manji tada joj ga treba povećati i time uvećati joj mogućnost da prije nastavi i završi s radom unutar monitora. Smanjivanje prioriteta za oba protokola obavlja se u funkciji za izlaz iz monitora.

```
j-funkcija Izaći_iz_monitora(M) {
   *smanjiti_prioritet(Aktivna, M)
   #smanjiti_prioritet(Aktivna, M)
   Aktivna.red = PRIPRAVNE
   Aktivna.iduća = Pripravne
   Pripravne = Aktivna
   osloboditi_prvu_iz_reda_monitora(M)
   Prebaciti_prvu_iz_reda_Pripravnih_u_red_Aktivna()
}
```

Nakon smanjenja prioriteta (ako je potrebno), dretva se premješta u red pripravnih te se monitor dodjeljuje prvoj dretvi iz reda monitora. Konačno se dretvu najvećeg prioriteta iz reda pripravnih aktivira. Prilikom dodjeljivanja monitora potrebno je odabranoj dretvi povećati prioritet, ako se koristi protokol vršnog prioriteta. Funkcije za smanjivanje prioriteta rade identično za oba protokola te se koristi ista funkcija.

```
j-funkcija Uvrstiti_u_red_uvjeta(M, K) {
   *smanjiti_prioritet(Aktivna, M)
   Aktivna.red = RED_UVJETA
   Aktivna.iduća = Monitor[M].red_uvjeta[K]
   Monitor[M].red_uvjeta[K] = Aktivna
   osloboditi_prvu_iz_reda_monitora (M)
   Prebaciti_prvu_iz_reda_Pripravnih_u_red_Aktivna()
}
```

Funkcija za stavljanje dretve u red uvjeta najprije premješta aktivnu dretvu u red uvjeta. Potom monitor dodjeljuje dretvi najvećeg prioriteta iz ulaznog reda monitora, koju pomiče u red pripravnih. Budući dretva oslobađa monitor, prilikom korištenja protokola nasljeđivanja prioriteta treba još i smanjiti prioritet dretve na vrijednost koju je imala prije ulaska u monitor.

```
j-funkcija Osloboditi_iz_reda_uvjeta(M, K) {
   *smanjiti_prioritet(Aktivna, M)
   Aktivna.iduća = Monitor[M].red
   Monitor[M].red = Aktivna
   Aktivna.red = M
   ako je (Monitor[M].red_uvjeta[K] ≠ KRAJ_LISTE) {
      prva = izvaditi_prvu_iz_reda(Monitor[M].red_uvjeta[K])
      prva.iduća = Monitor[M].red
      Monitor[M].red = prva
      prva.red = M
   }
   osloboditi_prvu_iz_reda_monitora(M)
   Prebaciti_prvu_iz_reda_Pripravnih_u_red_Aktivna()
}
```

Funkcija za oslobađanje dretve iz reda uvjeta definira model monitora. Za odabrani model funkcija najprije premješta aktivnu dretvu u red monitora. Potom, prvu iz zadanog reda uvjeta premješta u isti red monitora te konačno iz toga reda odabire dretvu najvećeg prioriteta kojoj dodjeljuje monitor.

Pomoćnim funkcijama koje slijede nastoji se smanjiti veličinu kôda izdvajanjem dijelova koji se ponavljaju u posebne funkcije.

```
funkcija prebaciti_prvu_iz_reda_Pripravnih_u_red_Aktivna() {
   Aktivna = izvaditi_prvu_iz_reda(Pripravne)
   Aktivna.red = AKTIVNA
}
```

U pomoćnoj funkciji je pretpostavljeno da red pripravnih dretvi nikada neće biti prazan pri njenom pozivu, tj. da će uvijek bar jedna dretva biti pripravna. Ako takva dretva ne postoji onda ju treba dodati u sustav iako neće obavljati nikakav koristan posao.

```
funkcija izvaditi_prvu_iz_reda(Red) {
   prva = Red
   pom = prva
   prije = prva
   dok je (pom.iduća ≠ KRAJ_LISTE) {
      ako je (pom.iduća.pri ≤ prva.pri) {//manji broj = veći prioritet
         prije = pom
         prva = pom.iduća
      }
      pom = pom.iduća
   }
   ako je (prije = prva) { //vadi se prva iz reda
      Red = Red.iduća
   } inače {
      prije.iduća = prva.iduća
   }
   vratiti(prva)
}
```

Gornja će pomoćna funkcija dretvu najvećeg prioriteta izvaditi iz zadanog reda i vratiti kazaljku na nju. Kao i prethodna i ova funkcija pretpostavlja da u zadanom redu postoji barem jedna dretva.

```
funkcija osloboditi_prvu_iz_reda_monitora(M) {
   ako je (Monitor[M].red ≠ KRAJ_LISTE) {
      prva = izvaditi_prvu_iz_reda(Monitor[M].red)
      Monitor[M].vlasnik = prva.id
      prva.red = PRIPRAVNE
      prva.iduća = Pripravne
      Pripravne = prva
      #povećati_prioritet(prva, M)
   }
   inače {
      Monitor[M].vlasnik = NITKO
   }
}
```

Kada ulazni red monitora nije prazan, među dretvama tog reda odabire se dretva s najvećim prioritetom i pridjeljuje joj se monitor. Ukoliko se koristi protokol vršnog prioriteta toj se dretvi tada poveća prioritet na potrebnu vrijednost.

Funkcije koje slijede su ostvarenja protokola za rješavanje problema inverzije prioriteta.

Prve dvije funkcije se koriste u slučaju uporabe protokola nasljeđivanja prioriteta. Funkcija za povećavanje prioriteta dretvi `nasljedni_prioriteti_povećati()` poziva se kada dretva, koja želi ući u monitor, ostaje zaustavljena u ulaznom redu monitora. U toj se funkciji po potrebi povećava prioritet dretvi koja je unutar monitora, a sama je u nekom drugom redu.

```
funkcija nasljedni_prioriteti_povećati(prioritet, M) {
   povećana = Dretva[Monitor[M].vlasnik]
   red = M
   dok je (povećana.pri > prioritet) {
      kraj = povećana.dp_zadnji+1
      povećana.din_pri[kraj].stari_pri = povećana.pri
      povećana.din_pri[kraj].uzrok = red
      povećana.dp_zadnji = kraj
      povećana.pri = prioritet
      *red = povećana.red
      *ako je (red ≥ 0) { //red≥0 – u redu za ulaz u monitor
      *   povećana = Dretva[Monitor[red].vlasnik]
      *}
   }
}
```

U prikazanom rješenju manji broj označava veći prioritet. Pri povećavanju i smanjivanju prioriteta koristi se struktura podataka koja je vrlo slična stogu. Prije povećanja prioriteta prijašnja se vrijednost prioriteta pohranjuje zajedno s indeksom monitora zbog koje je prioritet dretvi i povećan.

Redci koji započinju zvjezdicom (*) koriste se ukoliko se nasljeđivanje prioriteta obavlja tranzitivno. Odnosno ukoliko je dretva A kojoj se prioritet podiže sama već u nekom redu drugog monitora, dretvi B koja je unutar tog monitora podiže se prioritet na istu vrijednost s ciljem da bi ona što prije završila rad u tom monitoru i oslobodila ga. Ukoliko se radi jednostavnosti to ne želi koristiti, onda se ti redci na koriste.

Funkcija `smanjiti_prioritet()` smanjuje prioritet dretve koja oslobađa monitor na vrijednost koju je imala prije ulaska u monitor.

```
funkcija smanjiti_prioritet(dretva, M) {
   za i=1 do dretva.dp_zadnji {
      ako je (dretva.din_pri[i].uzrok = M) {
         dretva.din_pri[i].uzrok = -1
   }  }
   i = dretva.dp_zadnji
   dok je ((i>0) ∧ (dretva.din_pri[i].uzrok = -1)) {
      dretva.pri = dretva.din_pri[i].stari_pri
      i = i-1
   }
   dretva.dp_zadnji = i
}
```

Pri vraćanju prioriteta dretvi u funkciji oslobađanja monitora `M`, sva povećanja uzrokovana tim monitorom se poništavaju, tj. umjesto oznake monitora stavlja se vrijednost -1. Nakon toga kreće se od kraja podatkovne strukture te se dok god je oznaka

monitora poništena dretvi vraća prioritet na prijašnju razinu.

Kod protokola vršnih prioriteta funkcija za podizanje prioriteta treba pamtiti povijest mijenjanja prioriteta. Za to će poslužiti ista podatkovna struktura. Funkcija za podizanje prioriteta slijedi u nastavku dok je funkcija za vraćanje prioriteta `smanjiti_prioritet()` identična funkciji kao kod protokola nasljeđivanja prioriteta.

```
funkcija podignuti_prioritet(dretva, M) {
   ako je (dretva.pri > Prioriteti[M]) {
      zadnji = dretva.dp_zadnji+1
      dretva.din_pri[zadnji].prioritet = dretva.pri
      dretva.din_pri[zadnji].uzrok = M
      dretva.dp_zadnji = zadnji
      dretva.pri = Prioriteti[M]
   }
}
```

Kao što je vidljivo iz gornjeg kôda, funkcija za podizanje prioriteta kod protokola vršnih prioriteta vrlo je jednostavna.

5. SINKRONIZACIJA POMOĆU MONITORA

Sinkronizacija pomoću monitora konceptualno je različita od sinkronizacije pomoću semafora. Svakom je semaforu pridijeljena vrijednost dok monitorima nije. Monitor može biti ili prolazan ili neprolazan. Iz navedenog moglo bi se zaključiti da je mehanizam monitora sličan binarnom semaforu, tj. semaforu koji ima samo dva stanja: prolazno i neprolazno. Međutim kod monitora jedino dretva koja je ušla u monitor može izaći, tj. pozvati funkciju `Izaći_iz_monitora(M)`. Kod semafora pozivi `Čekati(S)` i `Postaviti(S)` ne moraju slijediti jedan drugoga. Dapače, jedna dretva može postavljati dugoj dretvi semafor N, a sama čekati na semafor M. Ukoliko se međutim binarnim semaforom zaštićuje kritični odsječak tada su pozivi `Ući_u_monitor()` i `Čekati()` te `Izaći_iz_monitora()` i `Postaviti()` identični po funkcionalnosti.

Monitori su mehanizam koji je predviđen za složenije probleme sinkronizacije te se za neke slučajeve u kojima su potrebni brojači događaja moraju koristiti dodatne varijable. U slučajevima gdje je bio dovoljan jedan semafor ovdje će pored monitora biti potrebna i varijabla. Međutim, u slučajevima u kojima se koristi više semafora ovdje će i dalje biti dovoljan jedan monitor uz eventualno dodatne redove uvjeta [Jel97, Jel98].

Postupci sinkronizacije monitorima opisani u ovom poglavlju počinju s jednostavnim problemima kao što je ostvarivanje kritičnih odsječaka i sinkronizacije uređenog skupa zadataka. Slijede poznati problemi sinkronizacije najčešće spominjani u literaturi te na kraju složeni problemi sinkronizacije kod kojih monitori znatno olakšavaju sinkronizaciju. Za većinu problema prikazano je i rješenje sa semaforima radi usporedbe broja poziva jezgrinih funkcija i jednostavnosti algoritama.

5.1. Kritični odsječak

Kritični je odsječak najčešći problem sinkronizacije višedretvenog sustava. Potreba za ostvarenjem zaštite u kritičnom odsječku pojavljuje se kada određeno sredstvo želi koristiti više dretvi. Istovremeni pristup se najčešće ne smije dopustiti jer bi to moglo narušiti ispravnost rješenja na kome dretve rade ili ukoliko se radi u pristupu uređaju, grešku u radu uređaja. Funkcije za ostvarenje kritičnog odsječka moraju osigurati da se on izvodi slijedno, tako da se u njemu nalazi najviše jedna dretva.

Dretva koja obavlja neki ciklički posao (tipična dretva u ugrađenim sustavima) i koja tijekom svog rada ulazi u kritični odsječak može se prikazati sljedećim redcima:

```
Dretva posao_dretve() {
   ponavljati {
      NEKRITIČNI DIO
      ulaz u kritični odsječak
      KRITIČNI ODSJEČAK
      izlaz iz kritičnog odsječka
   }
}
```

Algoritmi za ostvarenje kritičnog odsječka, ponekad nazivani i algoritmi međusobnog isključivanja moraju udovoljiti sljedeća četiri uvjeta [Bud03]:

1. u kritičnom se odsječku smije nalaziti samo jedna dretva,
2. odabir dretve koja će ući u kritični odsječak treba obaviti u konačnom vremenu,
3. dretva zaustavljena izvan kritičnog odsječka ne smije spriječiti ulazak druge dretve u kritični odsječak i
4. mehanizam međusobnog isključivanja mora raditi i kada su brzine izvođenja dretvi proizvoljne.

Uobičajeno je da jezgra operacijskog sustava pruža funkcije za ostvarenje međusobnog isključivanja. Najčešće su to semafori, ali se osim njih mogu koristiti i drugi mehanizmi.

Ukoliko jezgra sustava nema funkcije za međusobno isključivanje ono se može ostvariti korištenjem drugih algoritama kao što su Dekkerov, Petersonov, Lamportov ili neki drugi. Nedostatak tih algoritama je radno čekanje. Svaki od tih algoritama zahtjeva korištenje pomoćnih varijabli koje dretve u petlji čitaju dok se ne dogodi promjena koja im omogućuje ulazak u kritični odsječak. Tako će dretva iako ne može ući u kritični odsječak i dalje trošiti procesorsko vrijeme neprestano čitajući vrijednosti varijabli.

Ostvarenje kritičnog odsječka funkcijama jezgre, dretve koje pokušaju ući u kritični odsječak u kome se već nalazi dretva smještaju se u odgovarajući red čekanja. Izuzetak su neki višeprocesorski i raspodijeljeni sustavi kod kojih je radno čekanje neizbježno.

Uobičajeno ostvarenje međusobnog isključivanja nad kritičnim odsječkom jest pomoću binarnih semafora.

```
Dretva posao_dretve() {
   ponavljati {
      nekritični dio
      ČekatiBSem(S)
      KRITIČNI_ODSJEČAK
      PostavitiBSem(S)
      nekritični odsječak
   } do zauvijek
}
```

Budući se koriste binarni semafori korištene su oznake `ČekatiBSem(S)` i `PostavitiBSem(S)` tako da se te funkcije razlikuju od općih semafora.

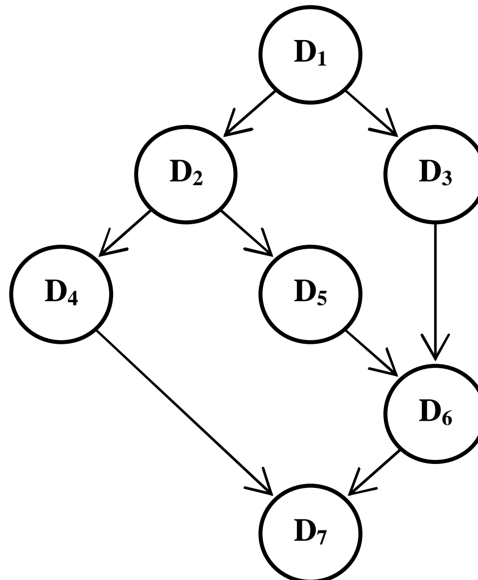
Ostvarenje međusobnog isključivanja nad kritičnim odsječkom s monitorima je identično kao i s binarnim semaforima.

```
Dretva posao_dretve() {
   ponavljati {
      nekritični dio
      Ući_u_monitor(M)
      KRITIČNI_ODSJEČAK
      Izaći_iz_monitora(M)
      nekritični odsječak
   } do zauvijek
}
```

Složenost ostvarenja kritičnog sektora je dakle ista koriste li se binarni semafor ili monitor.

5.2. Sinkronizacija sustava dretvi

Vrlo se često neki problemi rješavaju pomoću sustava dretvi čije je izvođenje vremenski međusobno uređeno. U takvom će sustavu dretva D_i treba početi obavljati svoj posao tek nakon što dretve D_m, $m \in P_i$ završe s poslom, gdje je P_i skup indeksa dretvi koje prethode dretvi i. Može se uvesti relacija parcijalnog uređenja $D_m<D_i$ koja označava da se dretva D_m treba dogoditi prije dretve D_i. Dretvi može, ali i ne mora, prethoditi jedna ili više dretvi. Međusobno uređenje dretvi najčešće se može izraziti i grafički kao na primjeru sa slike 5.1.



Slika 5.1 Sustav vremenski uređenih dretvi

Za zadani primjer vrijede relacije: $D_1<D_2$, $D_1<D_3$, $D_2<D_4$, $D_2<D_5$, $D_3<D_6$, $D_4<D_7$, $D_5<D_6$, i $D_6<D_7$. Relacije su tranzitivne, tj. iz $D_1<D_2$ i $D_2<D_5$ slijedi $D_1<D_5$. Tranzitivne relacije nisu potrebne pri sinkronizaciji, dapače, one su suvišne i samo zahtijevaju dodatna nepotrebna sredstva sinkronizacije.

Sinkronizacija uređenog sustava dretvi pomoću binarnih semafora ostvaruje se tako da se za svaku relaciju $D_m<D_n$ koristi po jedan semafor $S_{m,n}$, tj. za svaku strelicu na grafu potreban je po jedan binarni semafor. U navedenom primjeru potrebno je osam binarnih semafora. Prije početka obavljanja svog zadatka dretva mora čekati na semafore koje postavljaju dretve koje joj prethode. Kada dretva završi sa svojim zadatkom ona postavlja semafor svim dretvama koje joj neposredno slijede. Početno su svi semafori neprolazni.

Tako će tekst dretve D_6 biti:

```
Dretva D6() {
   ČekatiBSem(S3,6)
   ČekatiBSem(S5,6)

   Posao_zadatka_D6

   PostavitiBSem(S6,7)
}
```

Općeniti kôd za sinkronizaciju sustava vremenski uređenih dretvi s binarnim semaforima može se napisati kao u nastavku. Sa N_i označen je skup indeksa dretvi koje neposredno slijede nakon dretve D_i.

```
Dretva Di() {
   za svaki j ∈ Pi {
      ČekatiBSem(Sj,i)
   }

   Posao_zadatka_Di

   za svaki k ∈ Ni {
      PostavitiBSem(Si,k)
   }
}
```

Nešto manje semafora biti će potrebno ukoliko se koriste opći semafori koji mogu imati proizvoljnu pozitivnu vrijednost i kod kojih postoje funkcije kojima se s jednim pozivom vrijednost semafora povećava ili smanjuje i za vrijednosti veće od 1. U tom će slučaju biti potrebno onoliko semafora koliko ima dretvi s prethodnicima. U gornjem je primjeru to šest semafora (za dretve D_2-D_7). Dretva D_6 bi s općim semaforima izgledala:

```
Dretva D6() {
   ČekatiOSem(S6,2)

   Posao_zadatka_D6

   PostavitiOSem(S7,1)
}
```

Poziv `ČekatiOSem(S,m)` označava da opći semafor `S` treba smanjiti za vrijednost `m`. Ukoliko je trenutna vrijednost semafora `m` ili veća, vrijednost semafora se smanjuje i dretvi se dopušta nastavak rada. Ukoliko je trenutna vrijednost manja od `m` tada se dretva svrstava u red tog semafora sve dok neka druga dretva ne poveća vrijednost tog semafora na barem traženu vrijednost. Vrijednost `m` mora biti jednaka broju dretvi koje prethode pozivajućoj dretvi. U navedenom primjeru za dretvu D_6 to je vrijednost 2 jer dretva mora čekati na dretve D_3 i D_5.

Općeniti kôd sinkronizacije s općim semaforima nešto je jednostavniji od onog s binarnim semaforima.

```
Dretva Di() {
   ČekatiOSem(Si,m) //m-broj dretvi koje neposredno prethode

   Posao_zadatka_Di

   za svaki k ∈ Ni {
      PostavitiOSem(Sk,1)
   }
}
```

Za sinkronizaciju s monitorima potreban je jedan monitor sa `n` redova uvjeta, gdje je `n` broj dretvi s prethodnicima te još po jedna varijabla za svaku dretvu. Ta će varijabla

iskazivati stanje dotične dretve, tj. je li dotična dretva obavila svoj zadatak ili nije. Za dretvu D_6 sinkronizacija s monitorima mogla bi izgledati kao u nastavku:

```
Dretva D₆() {
   Ući_u_monitor(M)
   dok je ((Zad₃ ≠ GOTOV) ∨ (Zad₅ ≠ GOTOV)) {
      Uvrstiti_u_red_uvjeta(M,6)
   }
   Izaći_iz_monitora(M)

   Posao_zadatka_D₆

   Ući_u_monitor(M)
   Zad₆ = GOTOV
   Osloboditi_iz_reda_uvjeta(M,7)
   Izaći_iz_monitora(M)
}
```

U početku su sve varijable Zad_i postavljene na vrijednost različitu od GOTOV. Navedeni kôd može se poopćiti za bilo koju dretvu.

```
Dretva Dᵢ() {
   Ući_u_monitor(M)
   dok je (∃j ∈ Pᵢ, Zadⱼ ≠ GOTOV) {
      Uvrstiti_u_red_uvjeta(M,i)
   }
   Izaći_iz_monitora(M)

   Posao_zadatka_Dᵢ

   Ući_u_monitor(M)
   Zadᵢ = GOTOV
   za svaki k ∈ Nᵢ {
      Osloboditi_iz_reda_uvjeta(M,k)
   }
   Izaći_iz_monitora(M)
}
```

Usporedbom rješenja sa semaforima i monitorima mogu se uočiti neke sličnosti i razlike. Rješenje s monitorima ima četiri poziva funkcija sinkronizacije više od rješenja s binarnim semaforima (ulazak/izlazak iz monitora). Osim toga za to rješenje potrebne su dodatne varijable Zad_i, jer se uz monitor ne pamte nikakve vrijednosti. Ostali dio kôda je vrlo sličan. Može se zaključiti da je za sinkronizaciju sustava dretvi lakše koristiti semafore jer je s njima kôd i manji i brži, iako je potrebno puno više semafora. Međutim, rješenje s monitorima nije značajno lošije, pogotovo ako sam posao koji dretve obavljaju traje znatno duže od poziva nekoliko jezgrinih funkcija za rad s monitorima.

Ukoliko je sustav dretvi ciklički, tj. ukoliko nakon zadnje dretve treba ponovo početi prva dretva, tada treba dodati dio kôda ili u prvu dretvu ili u zadnju ili na neki drugi način promijeniti sinkronizaciju. Ono na što je potrebno pripaziti su početne vrijednosti semafora za sljedeću iteraciju, odnosno, vrijednosti varijabli koje označavaju stanje izvođenja

pojedinih dretvi.

Jedno od mogućih rješenja jest da svaka dretva nakon što završi svoj ciklički posao čeka na završetak zadnje dretve sustava koja će po završetku ponovo postaviti varijable sinkronizacije te svima signalizirati kraj jednog ciklusa, odnosno, početak novog.

Drugo rješenje za ciklički sustav jest da svaka dretva broji koliko je puta izvršila zadatak. Tada dretva neće početi izvoditi svoj zadatak dok god neka o njenih neposrednih prethodnika nema taj broj veći od nje. Npr. neka postoji polje `brojač[n]` koji sadrži broj izvođenja svakog zadatka.

```
Dretva Di(){
   Ući_u_monitor(M)
   ponavljati {
      dok je (∃j ∈ Pi, brojač[j] ≤ brojač[i]) {
         Uvrstiti_u_red_uvjeta(M,i)
      }
      Izaći_iz_monitora(M)

      Posao_zadatka_Di

      Ući_u_monitor(M)
      brojač[i] = brojač[i] + 1
      za svaki (k ∈ Ni) {
         Osloboditi_iz_reda_uvjeta(M,k)
      }
   } do zauvijek
   Izaći_iz_monitora(M)
}
```

Za početnu dretvu uvjet ispitivanja treba promijeniti u: `brojač[j] < brojač[i]` budući ona prva kreće u novu iteraciju sustava.

5.3. Problem proizvođača i potrošača

Problem sinkronizacije sustava dretvi kod kojeg neke dretve proizvode poruke koje koriste druge dretve vrlo je poznati problem jer je njegova primjena vrlo česta [Bud03, Sil94, Rob00, Kli05]. Svaka dretva proizvođač radi ciklički posao proizvodnje poruka, ili ih izračunava ili dohvaća s određene lokacije unutar ili izvan računalnog sustava. Potom ih pohranjuje u određeno mjesto dijeljenog međuspremnika i signalizira potrošačima da ih mogu koristiti. Mehanizmi sinkronizacije trebaju omogućiti ispravnu komunikaciju među entitetima. Potrošača treba zaustaviti ako nema poruka u međuspremniku, kao i proizvođača ukoliko je međuspremnik pun. Isto tako, ukoliko se radi o više proizvođača stavljanje poruke u međuspremnik treba serijalizirati, odnosno, u istom trenutku samo jedan proizvođač smije pristupiti međuspremniku. U protivnom, moglo bi se zbog paralelnog rada (stvarnog ili virtualnog) dogoditi da se na isto mjesto u međuspremnik postavi najprije jedna, a potom preko nje i druga poruka. Slični mehanizam treba dodati ako u sustavu ima više potrošača.

U nastavku potpoglavlja razmatrani su slučajevi kada u sustavu može biti i više

proizvođača i više potrošača te kada se komunikacija obavlja preko međuspremnika ograničene duljine. Neovisno o mehanizmu sinkronizacije, za komunikaciju je potrebno nekoliko zajedničkih varijabli. Prve su međuspremnik `M` i broj poruka `N` koje se u njega mogu smjestiti. Kazaljke `ULAZ` i `IZLAZ` pokazuju na prvo prazno mjesto, odnosno na prvo puno mjesto u međuspremniku. Inicijalno obje kazaljke pokazuju na prvo mjesto u međuspremniku. Proizvođači proizvode poruku funkcijom `Proizvesti_poruku()`, a potrošači troše poruku funkcijom `Potrošiti_poruku(poruka)`.

Rješenje sa semaforima ovisi je li se koriste samo binarni ili i opći semafori. U ovom su slučaju opći korisni budući mogu poslužiti kao brojila za brojanje punih i praznih mjesta u međuspremniku te tada nisu potrebne dodatne varijable za istu namjenu. Osim dva opća semafora koriste se još i dva binarna semafora kojima se proces stavljanja i uzimanja poruka iz i u međuspremnik zaštićuje.

```
Dretva Proizvođač() {
   ponavljati {
      poruka = Proizvesti_poruku()

      ČekatiOSem(Prazna_mjesta)
      ČekatiBSem(Proizvođači)
      M[ULAZ] = poruka
      ULAZ = (ULAZ+1) mod N
      PostavitiBSem(Proizvođači)
      PostavitiOSem(Puna_mjesta)
   }
}

Dretva Potrošač() {
   ponavljati {
      ČekatiOSem(Puna_mjesta)
      ČekatiBSem(Potrošači)
      poruka = M[IZLAZ]
      IZLAZ = (IZLAZ+1) mod N
      PostavitiBSem(Potrošači)
      PostavitiOSem(Prazna_mjesta)

      Potrošiti_poruku(poruka)
   }
}
```

Početno su binarni semafori prolazni, opći semafor `Prazna_mjesta` ima vrijednost veličine međuspremnika `M`, tj. vrijednost `N`, dok semafor `Puna_mjesta` ima vrijednost nula. Ukupno, za rješenje sa semaforima potrebna su 4 semafora.

Za rješenje korištenjem monitora osim jednog monitora s dva reda uvjeta potrebna je i dodatna varijabla `Broj_punih` koja broji puna mjesta u međuspremniku. Početna vrijednost te varijable treba biti 0.

```
Dretva Proizvođač() {
   ponavljati {
      poruka = Proizvesti_poruku()

      Ući_u_monitor(P)
      dok je (Broj_punih = N) {
         Uvrstiti_u_red_uvjeta(P, red_proizvođača)
      }
      M[ULAZ] = poruka
      ULAZ = (ULAZ+1) mod N
      Broj_punih++
      Osloboditi_iz_reda_uvjeta(P, red_potrošača)
      Izaći_iz_monitora(P)
   }
}

Dretva Potrošač {
   ponavljati {
      Ući_u_monitor(P)
      dok je (Broj_punih = 0) {
         Uvrstiti_u_red_uvjeta(P, red_potrošača)
      }
      poruka = M[IZLAZ]
      IZLAZ = (IZLAZ+1) mod N
      Broj_punih--
      Osloboditi_iz_reda_uvjeta(P, red_proizvođača)
      Izaći_iz_monitora(P)

      Potrošiti_poruku(poruka)
   }
}
```

Rješenje sa semaforima i monitorima ne razlikuju se po broju pozvanih jezgrinih funkcija. Za rješenje s monitorima potrebna je dodatna varijabla, ali je potrebno manje sredstava sustava, samo jedan monitor s dva reda uvjeta, nasuprot četiri semafora. Nedostatak koje ima prikazano rješenje s monitorima u odnosu na rješenje sa semaforima jest u smanjenoj mogućnosti paralelnog rada. Kod rješenja sa semaforima jedna dretva proizvođača i jedna dretva potrošača mogu istovremeno stavljati i uzimati poruke dok kod rješenja s monitorima ne. Rješenje koje bi uklonilo taj problem zahtjeva dva dodatna monitora koji bi se koristili samo za zaštitu pristupa međuspremniku od strane proizvođača i potrošača. Ipak, dio kôda koji proizvođači i potrošači obavljaju unutar monitora vrlo je kratak i razumna je pretpostavka da dobavljanje podataka i njihovo trošenje traju znatno dulje. U tom slučaju nedostatak zbog smanjenih mogućnosti paralelnog rada nestaje.

5.4. Problem čitača i pisača

Kod problema čitača i pisača u sustavu se nalaze dvije skupine dretvi: dretve čitači, koje samo čitaju vrijednost iz zajedničkog spremnika te dretve pisači koje pišu u zajednički

spremnik [Sil94]. Ukoliko jedna dretva čitača čita iz zajedničkog spremnika to ne treba spriječiti druge dretve čitača da također čitaju. Međutim kada jedna dretva pisača želi pisati u spremnik tada treba osigurati da niti jedna druga dretva, bilo to dretva čitača ili pisača ne pristupa tom spremniku.

Postoje različita rješenja problema [Cou71, Sil94] koja imaju različite prednosti i nedostatke. Jedno od najlakše ostvarivih rješenja jest takvo da dretva pisača zauzme spremnik tek kada niti jedna druga dretva čitača i pisača ne pristupa spremniku, dok će se pisačima uvijek dozvoliti pristup osim kada dretva pisača u njega piše. Navedeno rješenje ima problem izgladnjivanja dretvi pisača jer se te dretve mogu proizvoljno odgađati zbog stalnog dolazaka novih čitača. Drugo rješenje daje prednost pisačima na način da se čitačima ne dopusti pristup spremniku ako neka dretva pisača čeka na spremnik, iako trenutno spremniku pristupaju čitači. Nedostatak ovog rješenja može biti upravo povlašteni status dretvi pisača. Rješenje koje bi bilo najpravednije zahtijevalo bi pamćenje događaja, odnosno, redoslijed zahtjeva za pristup zajedničkom spremniku. Ako su dva ili više uzastopnih zahtjeva zahtjevi čitača onda se svima njima može dopustiti pristup spremniku.

U nastavku su prikazana rješenja sa semaforima i monitorima.

Najjednostavnije rješenje [Cou71] ostvareno binarnim semaforima zahtjeva dodatnu varijablu koja broji koliko trenutno čitača čita iz zajedničkog spremnika. Pristup spremniku pisačima je ograničeno korištenjem posebnog binarnog semafora kojeg mora proći i prva dretva čitača. Taj semafor ponovo postavlja zadnja dretva pisača pri završetku čitanja kao i svaki pisač po završetku svog rada.

```
Dretva Čitač() {
   rad koji ne uključuje zajednički spremnik
   ČekatiBSem(S1)
   br_čitača++
   ako je (br_čitača = 1) {
      ČekatiBSem(P)
   }
   PostavitiBSem(S1)

   čitanje iz zajedničkog spremnika

   ČekatiBSem(S1)
   br_čitača--
   ako je (br_čitača = 0) {
      PostavitiBSem(P)
   }
   PostavitiBSem(S1)
}

Dretva Pisač() {
   rad koji ne uključuje zajednički spremnik
   ČekatiBSem(P)

   pisanje u zajednički spremnik

   PostavitiBSem(P)
}
```

Rješenje s monitorima zahtjeva jedan monitor s dva reda uvjeta. Dretve pisači prvo ulaze u monitor te ukoliko trenutno niti jedna dretva ne čita mogu pristupiti i pisati u zajednički spremnik. Pri tome ne napuštaju monitor i na taj način sprječavaju čitačima da pristupe spremniku. Dretve čitači zahtijevaju ulaz u monitor prije i poslije čitanja. Unutar monitora, prije čitanja još se povećava brojač dretvi čitača, dok se nakon čitanja brojač smanjuje. Ukoliko se radi o zadnjoj dretvi čitača onda ona poziva funkciju za oslobađanje dretve pisača iz reda uvjeta. Ukoliko takva dretva postoji ona će pokušati ući u monitor i pisati.

```
Dretva Čitač() {
   rad koji ne uključuje zajednički spremnik
   Ući_u_monitor(M)
   br_čitača++
   Izaći_iz_monitora(M)

   čitanje iz zajedničkog spremnika

   Ući_u_monitor(M)
   br_čitača--
   ako je (br_čitača = 0) {
      Osloboditi_iz_reda_uvjeta(M,P)
   }
   Izaći_iz_monitora(M)
}

Dretva Pisač() {
   rad koji ne uključuje zajednički spremnik
   Ući_u_monitor(M)
   dok je (br_čitača > 0) {
      Uvrstiti_u_red_uvjeta(M,P)
   }

   pisanje u zajednički spremnik

   Osloboditi_iz_reda_uvjeta(M,P)
   Izaći_iz_monitora(M)
}
```

Rješenje [Cou71] koje favorizira pisače, prikazano u nastavku, radi tako da se čitačima ne dozvoli pristup zajedničkom spremniku ako neki od pisača čeka na sredstvo. Za ostvarenje takvog rješenja sa semaforima potreban je veći broj semafora.

```
Dretva Čitač() {
   rad koji ne uključuje zajednički spremnik
   ČekatiBSem(S3)
   ČekatiBSem(Č)
   ČekatiBSem(S1)
   br_čitača++
   ako je (br_čitača = 1) {
```

```
      ČekatiBSem(P)
   }
   PostavitiBSem(S1)
   PostavitiBSem(Č)
   PostavitiBSem(S3)

   čitanje iz zajedničkog spremnika

   ČekatiBSem(S1)
   br_čitača--
   ako je (br_čitača = 0) {
      PostavitiBSem(P)
   }
   PostavitiBSem(S1)
}

Dretva Pisač() {
   rad koji ne uključuje zajednički spremnik

   ČekatiBSem(S2)
   br_pisača++
   ako je (br_pisača = 1) {
      ČekatiBSem(Č)
   }
   PostavitiBSem(S2)
   ČekatiBSem(P)

   pisanje u zajednički spremnik

   PostavitiBSem(P)
   ČekatiBSem(S2)
   br_pisača--
   ako je (br_pisača = 0) {
      PostavitiBSem(Č)
   }
   PostavitiBSem(S2)
}
```

Semafor `S3` na prvi pogled može izgledati suvišan. Međutim, ukoliko se on izostavi moguće je, da se u nekom slijedu izvođenja, u trenutku kada jedna dretva čitača postavlja semafor `Č` na njega čekaju i čitač i pisač. Budući se u ovom rješenju daje prednost pisačima semafor `S3` osigurava da se takav slučaj ne može dogoditi.

Rješenje s monitorima prikazano u nastavku mnogo je jednostavnije. Nakon ulaska dretve u monitor provjeravaju se uvjeti napredovanja te ako oni nisu zadovoljeni dretva se svrstava u red uvjeta.

```
Dretva Čitač() {
   rad koji ne uključuje zajednički spremnik

   Ući_u_monitor(M)
   dok je (br_pisača > 0){
      Uvrstiti_u_red_uvjeta(M,Č)
      Osloboditi_iz_reda_uvjeta(M,Č)
   }
   br_čitača++
   Izaći_iz_monitora(M)

   čitanje iz zajedničkog spremnika

   Ući_u_monitor(M)
   br_čitača--
   ako je (br_čitača = 0) ∧ (br_pisača > 0) {
      Osloboditi_iz_reda_uvjeta(M,P)
   }
   Izaći_iz_monitora(M)
}

Dretva Pisač() {
   rad koji ne uključuje zajednički spremnik

   Ući_u_monitor(M)
   br_pisača++
   dok je (br_čitača > 0) {
      Uvrstiti_u_red_uvjeta(M,P)
   }

   pisanje u zajednički spremnik

   br_pisača--
   ako je (br_pisača > 0) {
      Osloboditi_iz_reda_uvjeta(M,P)
   } inače {
      Osloboditi_iz_reda_uvjeta(M,Č)
   }
   Izaći_iz_monitora(M)
}
```

Složenije rješenje koje bi uzelo u obzir redoslijed prispijeća čitača i pisača moralo bi koristiti dodatno polje za zapis tih zahtjeva koje bi se pregledavalo i po njemu propuštalo dretve u kritični odsječak.

5.5. Problem pušača cigareta

Problem pušača cigareta koji je predstavio S.S. Patil [Kos73, Par75] odnosi se na problem sinkronizacije tri dretve pušača cigareta i jedne dretve trgovca. Svaki od pušača ima u neograničenoj količini jednu od komponenti potrebnu za pušenje: papir, šibice ili duhan. Trgovac ima na raspolaganju sve tri komponente u neograničenim količinama. Trgovac nasumice odabire dvije različite komponente i postavlja ih na stol. Pušač koji ima treću komponentu uzima sve sa stola, signalizira trgovcu da je stol prazan, savija cigaretu i puši. Trgovac tada ponovo odabire dvije komponente i stavlja ih na stol te se postupak ponavlja.

Sinkronizacija pomoću semafora je problematična [Kos73] zbog više sredstava koja se koriste, ako se za svaku komponentu koristi po jedan semafor. U ovom bi slučaju najjednostavnije bilo definirati da jedan semafor označava dvije komponente, npr. semafor „1“ predstavlja papir i šibice, „2“ papir i duhan te „3“ šibice i duhan. Međutim, na taj je način problem pretvoren u puno jednostavniji problem sinkronizacije, dok je početna ideja problema pokazati mogućnost rješenja složenih sinkronizacija gdje se koristi više semafora, odnosno, sredstava koje dretve trebaju zauzeti.

Jedno od predloženih rješenja sa semaforima [Par75] uz nešto veći skup semafora koristi i jednostavne matematičke operacije. Osnovna ideja jest da se od cijelog skupa semafora samo neki koriste za sinkronizaciju između trgovca i pušača dok su ostali tu jedino radi jednostavnosti algoritma. Pušači i u ovom rješenju čekaju samo na jedan semafor te ni ovo rješenje iako rješava dotični problem nije opće rješenje početne klase problema već zapravo vrlo slično prethodnom rješenju.

Pravo rješenje problema koristeći semafore uključuje jezgrine funkcije koje nedjeljivo obavljaju skup operacija nad semaforima kod kojih ako se pojedina operacija ne može izvesti ne izvodi se niti jedna. Kod takvih funkcija moguće je u istom pozivu čekati na dva semafora, tj. zaustaviti pozivajuću dretvu dok god oba semafora ne postanu prolazna. Rješenje s takvim funkcijama tada postaje jednostavno. Međutim, ni s takvim jezgrinim funkcijama nije uvijek moguće riješiti sve probleme sinkronizacije [Kos73].

Rješenje s monitorima je u ovom slučaju jednostavno kao što se može vidjeti iz nastavka.

```
Dretva Trgovac() {
   Ući_u_monitor(M)
   ponavljati {
      ako je (stol ≠ PRAZAN){
         Uvrstiti_u_red_uvjeta(M,T)
      }

      sastojak[1] = slučajno odabrati prvi sastojak
      sastojak[2] = slučajno odabrati drugi sastojak //različit od prvog

      stol = PUN
      Osloboditi_iz_reda_uvjeta(M,P)
   }
   Izaći_iz_monitora(M)
}
```

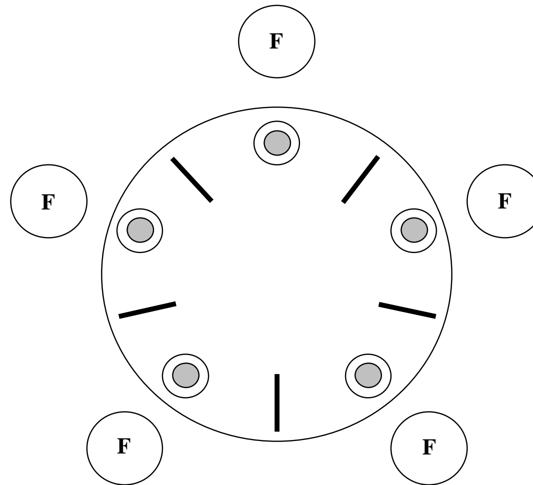
```
Dretva Pušač {
   ima = komponenta koju pušač ima u neograničenim količinama
   Ući_u_monitor(M)
   ponavljati {
      dok je (stol = PRAZAN) {
         Uvrstiti_u_red_uvjeta(M,P)
      }
      ako je ((sastojak[1] = ima) ∨ (sastojak[2] = ima)) {
         Osloboditi_iz_reda_uvjeta(M,P)
         Uvrstiti_u_red_uvjeta(M,P2)
      } inače {
         stol = PRAZAN
         Osloboditi_iz_reda_uvjeta(M,T)
         Osloboditi_iz_reda_uvjeta(M,P2)                              ##
         Osloboditi_iz_reda_uvjeta(M,P2)                              ##
         Izaći_iz_monitora(M)

         zamotati cigaretu i pušiti

         Ući_u_monitor(M)
   }  }
}
```

Red uvjeta `P2` koristi se da bi se izbjegao zastoj u slučaju da dretve pušači imaju različite prioritete. Tada bi se, u nekom mogućem slijedu izvođenja, dvije dretve pušača većeg prioriteta neprestano vrtjele u petlji provjeravajući što ima na stolu dok treća dretva zbog manjeg prioriteta ne bi došla na red za provjeru i samim time sustav bi stao. Kada bi sve dretve pušača imale isti prioritet red `P` bi bio dovoljan te bi se `P2` zamijenio sa `P` i redci koje završavaju sa `##` mogli bi se izbaciti.

5.6. Problem pet filozofa

Klasični primjer gdje se klasičnom uporabom semafora pojavljuje potpuni zastoj jest problem pet filozofa [Bud03, Sil94, Rob01]. Problem se sastoji u sinkronizaciji pet procesa filozofa koji ciklički obavljaju iste radnje. Filozofi sjede za okruglim stolom, ispred svakog se nalazi tanjur s hranom, a između svaka dva tanjura postavljen je jedan štapić. Životni ciklus filozofa sastoji se u misaonom procesu nakon kojeg slijedi hranjenje te ponovo razmišljanje, pa hranjenje itd. Problem je u tome što filozofi jedu tek kada dohvate oba štapića (lijevi i desni), a na stolu nema dovoljno štapića da svi filozofi jedu kad im se prohtje.

Slika 5.2 prikazuje filozofe označene slovom F, njihov raspored oko stola te tanjure i štapiće raspoređene po stolu.



Slika 5.2 Problem pet filozofa

Životni ciklus filozofa k može se opisati petljom:

```
Dretva Filozof(k){
   ponavljati {
      razmišljati
      uzeti_štapiće(k)
      jesti
      vratiti_štapiće_na_stol(k)
   } do zauvijek
}
```

Kritični redci kôda su uzimanje i vraćanje štapića. Filozof neće moći uzeti oba štapića ako neki od njegovih susjeda s lijeve ili desne strane u tom trenutku jede.

Rješenje sinkronizacije sa semaforima kod kojeg je svaki štapić zaštićen s jednim semaforom (binarnim), prikazano u nastavku, može dovesti do potpunog zastoja.

```
funkcija uzeti_štapiće(k) {
   ČekatiBSem(k)
   ČekatiBSem((k+1) mod 5)
}

funkcija vratiti_štapiće_na_stol(k) {
   PostavitiBSem(k)
   PostavitiBSem((k+1) mod 5)
}
```

Do potpunog zastoja može doći kada svi filozofi istovremeno pristupe stolu. Tada se može dogoditi da svaki od njih prođe jedno čekanje semafora (uzima lijevi štapić). Međutim, budući je i sljedeći filozof prošao prvi semafor, drugi je semafor neprolazan. Svi se filozofi zaustave na čekanju drugog semafora (desni štapić) te nastaje potpuni zastoj.

Ukoliko bi se koristile funkcije jezgre za nedjeljivo obavljanje više operacija nad više semafora, tj. ako bi se u gornjem kôdu čekanje na oba semafora moglo obaviti u jednom pozivu tada bi se problem riješio. U takvom bi pozivu filozof bio blokiran i ako je samo jedan od semafora neprolazan, odnosno, ne bi zauzeo niti onaj drugi eventualno prolazan semafor.

Druga rješenja sa semaforima zahtijevaju ili dodatni semafor koji će ograničiti istovremeni pristup stolu na npr. četiri filozofa tako da uvijek bar jedan može jesti ili korištenjem dodatnih varijabli i semafora.

Rješenje korištenjem monitora mnogo je jednostavnije. Pored jednog monitora s pet reda uvjeta (po jedan za svakog filozofa) potrebne su dodatne varijable za označavanje stanja štapića i filozofa. Neka su to polja `štapić[5]` i `filozof[5]`. Sinkronizacija se tada može obaviti prema:

```
funkcija uzeti_štapiće(k) {
   Ući_u_monitor(M)
   filozof[k] = ŽELI_JESTI                                          **
   dok je ((štapić[k] = ZAUZET) ∨ (štapić[(k+1) mod 5] = ZAUZET)) {
      Uvrstiti_u_red_uvjeta(M, k)
   }
   štapić[k] = ZAUZET
   štapić[(k+1) mod 5] = ZAUZET
   filozof[k] = JEDE                                                **
   Izaći_iz_monitora(M)
}

funkcija vratiti_štapiće_na_stol(k) {
   Ući_u_monitor(M)
   štapić[k] = SLOBODAN
   štapić[(k+1) mod 5] = SLOBODAN
   filozof[k] = MISLI
   ako je ((filozof[(k-1) mod 5] = ŽELI_JESTI) ∧                    **
                              štapić[(k-1) mod 5] = SLOBODAN)) {    **
      Osloboditi_iz_reda_uvjeta(M, (k-1) mod 5)
   }                                                                **
   ako je ((filozof[(k+1) mod 5] = ŽELI_JESTI) ∧                    **
                              štapić[(k+2) mod 5] = SLOBODAN)) {    **
      Osloboditi_iz_reda_uvjeta(M, (k+1) mod 5)
   }                                                                **
   Izaći_iz_monitora(M)
}
```

Logički ispravno je i rješenje koje ne koristi retke koje završavaju s zvjezdicama (`**`), ali je to rješenje neučinkovito. Funkcija za oslobađanje iz reda uvjeta poziva se i ako dotični filozof nema uvjete za uzimanje oba štapića, odnosno, čak i ako ne traži štapiće.

U funkciji dobavljanja štapića petlja `dok` može se promijeniti u jednostavno ispitivanje uvjeta `ako`, ali se tada u funkciji vraćanja štapića unutar odgovarajućeg `ako` bloka trebaju označiti odgovarajući štapići kao zauzeti prije oslobađanja dotične dretve iz reda uvjeta.

Općenito se preporuča uporaba petlje `dok` jer je lakše ponovo provjeriti uvjete nastavka nego pratiti moguće scenarije odvijanja dretvi u sustavu. Mnogo je teže osigurati da dretva koja je oslobođena iz reda uvjeta zaista zateče stanje u kojem može nastaviti rad. Uporabom petlje `dok` omogućava se dretvama da se nadmeću za sredstva i to korištenjem svog prioriteta i/ili redoslijeda zahtjeva.

U složenim slučajevima sinkronizacije monitori su mnogo lakši za uporabu od semafora. Jednostavnije je dodatnom logikom (izvan jezgrinih funkcija) provjeriti može li dretva nastaviti rad ili mora čekati, nego koristiti semafore, koji u takvim slučajevima vrlo često mogu dovesti do potpunog zastoja. Kôd pisan korištenjem monitora u takvim je slučajevima mnogo jednostavniji i za razumijevanje jer je sve eksplicitno napisano u njemu. Kod semafora je potrebno paziti tko postavlja pojedine semafore, koje su moguće vrijednosti određenih semafora, redoslijed poziva semafora i slično.

Kod monitora struktura kôda sinkronizacije u gotovo svim slučajevima izgleda isto:

```
...
Ući_u_monitor(M)
dok je (uvjeti za nastavak rada dretve nisu zadovoljeni) {
   Uvrstiti_u_red_uvjeta(M,K)
}
Izaći_iz_monitora(M)
...
obaviti posao
...
Ući_u_monitor(M)
ako je (uvjet za nastavak rada dretve R zadovoljen) {
   Osloboditi_iz_reda_uvjeta(M,R)
}
Izaći_iz_monitora(M)
...
```

Prije obavljanja zadanog posla dretva ispituje uvjete i to unutar monitora. Nakon obavljanja posla provjerava jesu li uvjeti ispunjeni za neku drugu dretvu i ukoliko jesu dozvoljava joj nastavak rada. Unutar monitora, i prije i poslije obavljanja posla dretve, često dolazi i dodatni dio kôda u kojem se postavljaju varijable koje definiraju stanje sustava te im se ne smije dozvoliti istovremeno mijenjanje.

6. FUNKCIJE OPERACIJSKOG SUSTAVA OSTVARENE MONITORIMA

Monitori se kao jedno od sredstava sinkronizacije dretvi mogu upotrijebiti za izgradnju i drugih sinkronizacijskih mehanizama. Ostvarenje semafora pomoću monitora je jednostavno (potpoglavlje 6.4). Mogu li se s monitorima ostvariti i ostale funkcije koje su dio jezgre operacijskog sustava? U ovom je poglavlju razmatran i prikazan način ostvarenja određenih funkcija jezgre operacijskog sustava korištenjem monitora i dodatnih potrebnih mehanizama. U te funkcije spadaju pristup ulazno-izlaznim napravama, ostvarivanje kašnjenja, raspoređivanje dretvi, ostvarenje semafora i redova poruka. Međusobno isključivanje i uvjetne varijable direktno su ostvarene monitorima dok su za korištenje zajedničkog spremničkog prostora monitori potrebni jedino kao sredstvo međusobnog isključivanja.

Prikazani načini ostvarenja nekih funkcija, koje se uobičajeno nalaze u operacijskim sustavima, napravljeni su radi prikaza mogućnosti monitora. Funkcije koje se inače ugrađuju u jezgru sustava u usporedbi s tim istim funkcijama ostvarenih monitorima biti će i brže i manje, ali pridonose složenosti jezgre. Prednost ostvarenja s monitorima, osim u smanjenoj složenosti jezgre, mogla bi biti u onim funkcijama koje su po trajanju izvođenja dulje. Te se funkcije ukoliko ostvarene unutar jezgre ne mogu prekidati za vrijeme svog izvođenja dok se u ostvarenju s monitorima mogu. Tako bi se vrijeme u kojem je zabranjeno prekidanje smanjilo na vrijeme izvođenja funkcija za rad s monitorima.

Prihvat i obrada vanjskih prekida ne utječe na funkcije monitora budući se funkcije za ostvarenje monitora ne mogu prekidati. Prilikom ostvarenja prihvata prekida i njegove obrade ipak treba pripaziti na nekoliko stvari. Najjednostavnije ostvarenje, koje će najčešće biti i dovoljno, jest da se tijekom obrade prekida zabrani daljnje prekidanje. Dovoljno je prije početka obrade pohraniti kontekst procesora te ga po obradi obnoviti. Ostala rješenja zahtijevaju uplitanje u strukturu jezgre. Ukoliko se želi dozvoliti prekidanje obrade prekida drugim prekidima (npr. ugnježđivanje prekida) tada bi bilo potrebno osigurati spremnički prostor za kontekst tih prekida i mogućnost povratka na prethodnu obradu. Ukoliko procesor ne razlikuje sklopovski prekid od programskog tada bi trebalo dodati odgovarajuću logiku na početku obrade prekida koja će ih razlikovati i preusmjeriti obradu. Pozivanje funkcija monitora iz prekidne rutine nije dozvoljeno jer bi to moglo narušiti konzistentnost podataka jezgre. Monitorske funkcije smiju se pozivati jedino iz konteksta dretve, a budući se prekidi prihvaćaju van konteksta bilo koje dretve te se funkcije ne smiju izravno pozivati iz obrade prekida. Ukoliko se ipak pojavi takva potreba, tj. potrebno je pozvati funkciju `Osloboditi_iz_reda_uvjeta()`, tada treba koristiti posebne dretve za obradu prekida koje će tu funkciju pozvati. Dretva za obradu prekida se tada premješta iz prekidne rutine u odgovarajući red (u aktivnu ili red pripravnih ili red monitora). U ovom se dijelu koda zadire u strukturu jezgre, ali ako se koriste posebni monitori onda se neće narušiti stabilnost sustava. Na ovaj je način ostvarena i dretva za kašnjenje prikazana u potpoglavlju 6.2. Za razliku od monitora, semaforska funkcija `Postavi()` ne zahtjeva kontekst dretve te su semafori u prednosti pri ostvarenju sinkronizacije iz prekidne rutine.

6.1. Pristup ulazno-izlaznim napravama

Ostvarenje upravljanja ulazno-izlaznim (UI) napravama ovisi o njihovom ponašanju. Ukoliko se njima pristupa isključivo korištenjem spremničkih lokacija, tj. funkcijama *čitaj*() i *piši*(), onda je dovoljno te odsječke kôda proglasiti kritičnim odsječcima. Prije korištenja određene ulazno-izlazne naprave „`S`“ potrebno je pozvati funkciju `Ući_u_monitor(UI[S])` te po završetku korištenja iste naprave funkciju `Izaći_iz_monitora(UI[S])`. Sljedeći tekst prikazuje jednu tipičnu uporabu opisanog mehanizma:

```
//korištenje UI naprave rednog broja S – samo čitanje/pisanje
Ući_u_monitor(UI[S])
korištenje UI naprave: čitanje/pisanje
Izaći_iz_monitora(UI[S])
```

Ukoliko UI naprava generira prekide, odnosno, dretva mora čekati na neki događaj te naprave potrebno je koristiti i redove uvjeta. Dretva će kao u prvom slučaju najprije ući u monitor, eventualno obaviti neke operacije sa UI napravom te nakon toga čekati u redu uvjeta da se naprava javi. Naprava će generirati prekid u čijoj će se obradi osloboditi odgovarajuća dretva. U kôdu programa dretve trebali bi se naći sljedeći redci:

```
//korištenje UI naprave rednog broja S – čitanje/pisanje + čekanje
Ući_u_monitor(UI[S])
korištenje UI naprave: čitanje/pisanje
Uvrstiti_u_red_uvjeta(UI[S], 1)
korištenje UI naprave: čitanje/pisanje
Izaći_iz_monitora(UI[S])
```

U dretvu za obradu prekida koji uzrokuje tražena UI naprava trebalo bi dodati retke:

```
//u proceduri za obradu prekida
Ući_u_monitor(UI[S])
Osloboditi_iz_reda_uvjeta(UI[S], 1)
Izaći_iz_monitora(UI[S])
```

Iz navedenog se primjećuje da je ostvarenje pristupa UI napravama, bilo samo čitanjem ili pisanjem ili korištenjem prekida, uporabom monitora jednostavno. Ukoliko, međutim, na prekid jedne naprave može čekati više dretvi, tada treba koristiti dodatne metode za određivanje kojoj će se dretvi omogućiti korištenje naprave.

6.2. Ostvarivanje kašnjenja

Dretve u ugrađenim sustavima najčešće obavljaju neki ciklički posao. Vremenska skica takvih dretvi dana je u potpoglavlju 2.3. Dretva po obavljanju svog posla, treba pričekati do sljedećeg trenutka mogućeg početka obavljanja posla, koje je određeno periodom `T` i prethodnim trenutkom završetka.

Očito je da se javlja potreba za funkcijom `Zakasniti(t)` koja će zaustaviti rad pozivajuće dretve za određeno vrijeme `t`. Za brojanje vremena potrebna je sklopovska podrška. U daljem je radu pretpostavljeno da se logički sat može dohvatiti funkcijom `Dohvatiti_sat()`. Isto tako pretpostavljeno je da postoji nekakvo brojilo koje broji

unatrag od zadane vrijednosti do nule kada šalje prekid. Funkcija za upravljanje tim brojilom je `Postaviti_brojilo(vrijeme)` koja inicijalizira početnu vrijednost brojila na vrijednost `vrijeme` te ono tada počinje odbrojavati. Funkcija koja vraća trenutno stanje brojila je `Dohvatiti_brojilo()`. Ukoliko sklopovski nije podržana funkcija `Dohvatiti_sat()` ona se može programski ostvariti uz male izmjene u funkciji `Postaviti_brojilo()`.

Ostvarenje kašnjenja pomoću monitora zahtjeva jedan monitor s po jednim redom uvjeta za svaku dretvu u sustavu (svaki dodatni red uvjeta unutar istog monitora zahtjeva samo jednu dodatnu kazaljku). Osim monitora potrebno je pri obradi prekida brojila dodati poziv funkcije `Istek_vremena()`, prikazane u nastavku, u dretvu za obradu tog prekida. Za svaku je dretvu potrebno zapisati i vrijeme kašnjenja pa je potrebna ili dodatna varijabli u strukturi dretve ili kao zasebno polje. Prikazano rješenje pretpostavlja korištenje varijable `kasniti` koja je dio strukture `Dretva`. Funkcije za ostvarivanje kašnjenja `Zakasniti()` i `Istek_vremena()` izgrađene korištenjem monitora prikazane u nastavku su monitorske funkcije, u kojima je istovremeno aktivna samo jedna dretva. Iz tog su razloga označene sa `m-funkcija`. Prikazano rješenje radi tako da svakoj dretvi prije uvrštenja u red uvjeta zapiše vrijeme kada se treba ponovo aktivirati. Pri prekidu brojila pregledavaju se redom sve dretve koje čekaju i premještaju se u red pripravnih one dretve kojima je vrijeme čekanja isteklo.

```
m-funkcija Zakasniti(vrijeme) {
   Ući_u_monitor(M_Zakasniti)
   *zabraniti_prekidanje
   Aktivna.kasniti = Dohvatiti_sat() + vrijeme
   do_prekida = Dohvatiti_brojilo()
   ako je ((do_prekida = 0) || (do_prekida > vrijeme)) {
      Postaviti_brojilo(vrijeme)
   }
   *dozvoliti_prekidanje
   Uvrstiti_u_red_uvjeta(M_Zakasniti, Aktivna.id)
   Izaći_iz_monitora(M_Zakasniti)
}

m-funkcija Istek_vremena() { // poziva ju posebna dretva
   Ući_u_monitor(M_Zakasniti)
   V_prvi = VRIJEME_KRAJ
   *zabraniti prekidanje
   sada = Dohvatiti_sat()
   za i=1 do BROJ_DRETVI {
      ako je ((Dretva[i].kasniti ≠ 0) ∧ (Dretva[i].kasniti ≤ sada)) {
         Dretva[i].kasniti = VRIJEME_KRAJ
      } inače {
         ako je (Dretva[i].kasniti - sada < V_prvi) {
            V_prvi = Dretva[i].kasniti - sada
   }  }  }
   ako je (V_prvi < VRIJEME_KRAJ) {
      Postaviti_brojilo(V_prvi)
   }
```

```
   *dozvoliti prekidanje
   za i=1 do BROJ_DRETVI {
      ako je (Dretva[i].kasniti = VRIJEME_KRAJ){
         Dretva[i].kasniti = 0
         Osloboditi_iz_reda_uvjeta(M_Zakasniti, i)
   }  }
   Izaći_iz_monitora(M_Zakasniti)
}
```

U slučaju da je precizna vrijednost brojila bitna, treba uključiti i retke označene sa *, te se onda i funkcije za zabranu i dozvolu prekidanja moraju ugraditi. Jedno jednostavno rješenje kojim bi se moglo spriječiti prekidanje ovih funkcija jest podizanjem prioriteta dretvi za vrijeme tih poziva (npr. protokolom vršnih prioriteta).

6.3. Raspoređivači dretvi

Ukoliko se sve u sustavu ostvaruje pomoću monitora, ugrađeni raspoređivač koji postoji u tim funkcijama za mnoge će sustave biti i dovoljan. Raspoređivač dretvi ugrađen u funkcije monitora radi tako da uvijek pripravna dretva najvećeg prioriteta bude i aktivna dretva. Ako su monitori temelj svih funkcija jezgre i dretve obavljaju ili periodički posao ili čekaju i obrađuju određene događaje u sustavu ugrađeni raspoređivač će biti dostatan.

Ukoliko statički prioriteti nisu dovoljni za željeno raspoređivanje, treba ostvariti neki od dinamičkih algoritama.

Ugrađeni raspoređivač će biti dostatan i ako u sustavu postoji samo jedna radna dretva (koja ima uvijek nešto za raditi) koja pri tome ima i najmanji prioritet od svih dretvi u sustavu. Međutim, ukoliko u sustavu postoji više od jedne radne dretve, ugrađeni raspoređivač neće biti dovoljan. U takvom je slučaju potrebno ugraditi dodatni raspoređivač.

Pri ostvarenju raznih postupaka raspoređivanja postoje razni pristupi. Najlakši jest da se funkcije raspoređivanja ostvare kao jezgrine funkcije ili dio jezgrinih funkcija s pristupom podacima jezgre i mogućnošću rukovanja dretvama. U razmatranom sustavu podatkovna struktura jezgre je jednostavna i mala te nije teško napraviti dodatne funkcije. Međutim, dodavanjem tih funkcija jezgri ona se povećava i samim tim postaje složenija. Ako bi se svi mehanizmi ugradili unutar jezgre izgubila bi se početna zamisao da jezgra bude što manja i jednostavnija. Raspoređivanje unutar jezgre se zbog toga neće razmatrati zbog porasta složenosti kao i zbog toga što to uključuje poznato rješenje koje se najčešće i koristi. Jednostavno raspoređivanje koje koristi prioritete ionako je ugrađeno u funkcije za rad s monitorima. Drugi način ostvarenja raspoređivača je preko dodatnih funkcija koje nisu jezgrine funkcije, a koje koriste samo jezgrine funkcije. Treće rješenje uključuje dretvu zaduženu za raspoređivanje, koja kombinacijom poziva jezgre mijenja stanje ostalih dretvi u sustavu i na taj način vrši raspoređivanje. Prednost zadnja dva pristupa jest što se problem rješava izvan jezgre te se ona sama ne mijenja. Nedostaci su brzina i nešto veća složenost samih mehanizama naspram onih kada se raspoređivači ugrađuju unutar jezgre.

U radu je prikazan način ostvarenja nekoliko algoritama raspoređivanja: mjera ponavljanja, raspoređivanje prema trenutku nužnog završetka te kružno posluživanje.

Ugrađeni raspoređivač u monitorima dovoljan je za ispravno raspoređivanje dretvi za

algoritam mjere ponavljanja budući je to statički algoritam. Prije pokretanja sustava dovoljno je na osnovi učestalosti ponavljanja izračunati i pridijeliti prioritete dretvama te ih raspoređivač ugrađen u monitorima ispravno raspoređuje. Nikakav dodatni posao i kôd nije potreban.

6.3.1. Raspoređivanje prema trenutku nužnog završetka

Opis rada algoritma prikazan je u potpoglavlju 2.3. Ostvarenje tog algoritma pomoću monitora koje bi zahtijevalo dinamičku promjenu prioriteta dretvi želi se izbjeći budući je prioritet dretve dio podataka jezgre, te ukoliko bi se takva promjena obavljala izvan jezgre mogla bi se narušiti stabilnost sustava. Zbog toga je odabrano rješenje koje ne mijenja prioritete dretvi, već se sve dretve, osim one koja se treba izvoditi, stavljaju u red uvjeta. Ipak, da bi algoritam ispravno radio potrebno je dretvama pridijeliti prioritete na isti načina kao i kod algoritma mjera ponavljanja. Odnosno, dretve s kraćom periodom ponavljanja kao i sporadične dretve trebaju imati veći prioritet. To je potrebno zato da se obavljanje posla može prekinuti kada se pojavi nova spremna dretva u sustavu (kada nekoj dretvi započne novi ciklus). Nova će dretva budući da ima veći prioritet provjeriti je li ona treba započeti s obavljanjem posla ili treba pričekati neku drugu dretvu ili više njih.

Za svaku je dretvu potrebno poznavati vremenska svojstva i to barem trenutak mogućeg početka, trenutak nužnog završetka te periodu ponavljanja. Neka su navedene vrijednosti pohranjene u sljedećim poljima:

- `TNZ` – trenutak nužnog završetka,
- `TMP` – trenutak mogućeg početka i
- `Perioda` – perioda ponavljanja.

Sve navedene varijable su polja s po jednim elementom za svaku dretvu u sustavu. Ukoliko neke dretve ne treba raspoređivati s zadanim algoritmom dovoljno je u `TMP` dotične dretve postaviti maksimalnu vrijednost te se ona neće razmatrati u dotičnom procesu raspoređivanja (što ne znači da se neće nikad i izvoditi).

Pretpostavljeno je da dretve koje treba rasporediti ovim algoritmom obavljaju posao oblika:

```
Dretva posao_dretve(i) {
   ponavljati {
      započeti(i)
      periodički posao dretve „i“
      završiti(i)
   } do zauvijek
}
```

Dretva najprije poziva funkciju `započeti()`. Funkcija će najprije odgoditi izvođenje dretve do trenutka mogućeg početka izvođenja dretve. Potom se provjerava koja dretva ima najbliži trenutak nužnog završetka. Ako je to pozivajuća dretva onda ona nastavlja s radom (`periodički posao dretve „i“`), a ako nije onda se propušta dretva s takvim vremenom, a pozivajuća se stavlja u red uvjeta. Nakon obavljanja svog periodičkog posla dretva poziva funkciju `završiti()` koja će ažurirati vremena dretve te eventualno propustiti neku drugu dretvu u obavljanje njena posla. Funkcije `započeti()` i `završiti()` prikazane u nastavku su monitorske funkcije.

```
m-funkcija započeti(i) {
   Ući_u_monitor(M)
   t = Dohvatiti_sat()
   ako je (t < TMP[i]) {
      Izaći_iz_monitora(M)
      Zakasniti (TMP[i] - t)
      Ući_u_monitor(M)
   }
   ponavljati {
      prva = i
      za j=1 do broj_dretvi {
         ako je ((j ≠ i) ∧ (TMP[j] ≤ t) ∧ (TNZ[prva] > TNZ[j])) {
            prva = j
      }  }
      ako je (prva ≠ i) {
         Osloboditi_iz_reda_uvjeta(M, prva)
         ako je ((TNZ[prva] < TNZ[i]) ∧ (TMP[j] ≤ Dohvatiti_sat()) {   *
            Uvrstiti_u_red_uvjeta(M, i)
      }  }
   } dok je (prva ≠ i)
   Izaći_iz_monitora(M)
}

m-funkcija završiti(i) {
   Ući_u_monitor(M)
   // ažuriranje vremena za dretvu „i"
   TNZ[i] += Perioda[i]
   TMP[i] += Perioda[i]

   t = Dohvatiti_sat()
   j = 1
   dok je ((j ≤ broj_dretvi) ∧ (TMP[j] > t)) {
      j++
   }
   ako je (j ≤ broj_dretvi){
      za k = j+1 do broj_dretvi {
         ako je ((TMP[k] ≤ t) ∧ (TNZ[j] > TNZ[k])) {
            prva = k
      }  }
      ako je (j ≠ i) {
         Osloboditi_iz_reda_uvjeta(M, prva)
   }  }
   Izaći_iz_monitora(M)
}
```

Provjera u programu označena zvjezdicom (*) na kraju retka u funkciji `započeti()` potrebna je stoga što funkcija `Osloboditi_iz_reda_uvjeta()` ne mora odmah vratiti procesor pozivajućoj dretvi već to može biti i dretva oslobođena iz reda uvjeta te kasnije i

druge dretve. Ukoliko su se uvjeti u međuvremenu promijenili, tj. ukoliko je oslobođena dretva bila većeg prioriteta te ukoliko je ona u međuvremenu završila svoj periodički posao potrebno je ponovo izračunati koja je sljedeća dretva na redu.

Navedeno rješenje nešto je složenije od onog koje bi se ugradilo u jezgru, ali ta složenost ne utječe na složenost jezgre koja i dalje ostaje ista. Nedostatak prema rješenju ugrađenim u jezgru jest u mogućnosti prekida funkcija raspoređivanja. Vremena dohvaćena iz brojila tada bi odstupala od stvarnih vrijednosti za kašnjenje uzrokovano tim prekidom.

6.3.2. Raspoređivač kružnog posluživanja

Ponekad se i u ugrađenim sustavima pojavljuju radne dretve koje su uvijek pripravne za rad. Već je spomenuto da ukoliko postoji samo jedna takva dretva u sustavu i ugrađeni će raspoređivač poslužiti. Ako ih ima više tada je potrebno ostvariti dodatni raspoređivač za takve dretve. Raspoređivač kružnog posluživanja procesora je jedan od prikladnih algoritama u takvom slučaju. Algoritam radi tako da dijeli procesorsko vrijeme u vremenske odsječke (kvante) te ih redom dodjeljuje pripravnim dretvama. Ukoliko su dretvama pridijeljeni prioriteti tada će vrijeme dijeliti dretve većeg prioriteta i tek kada te dretve završe svoj rad na red će doći dretve sljedećeg nižeg prioriteta.

Ostvarenje s monitorima nešto je složenije. Problem jest u tome što treba mijenjati prioritete dretvama nakon isteka nekog vremenskog intervala, a to znači mijenjati podatke jezgre izvan jezgrinih funkcija. To nikako nije preporučeno te se prikazano rješenje oslanja na funkcije zabrane prekida koje su u ovom slučaju neophodne. Zabrana prekida izvan jezgrinih funkcija dodatno povećava složenost sustava te to treba imati na umu ako se koristi ovakav raspoređivač. Mijenjanje prioriteta dretvi u ovom slučaju može se obaviti na dva načina. Prvi je da se raspoređivač ugradi u prekidnu rutinu koja se poziva pri isteku vremenskog intervala, kao što je to i najčešće u operacijskim sustavima. Drugi način jest da se on ostvari u posebnoj dretvi. U oba slučaja dio kôda u kojem se prioriteti dretvi mijenjaju mora se zaštititi od prekidanja.

U nastavku je prikazana jednostavna inačica druge metode, s dodatnom dretvom koja mora imati veći prioritet od ostalih dretvi koje raspoređuje. Ona po isteku unaprijed definiranog vremenskog odsječka odabire sljedeću dretvu iz reda te joj poveća prioritet. Nakon toga dretva raspoređivača zakasni sebe za trajanje vremenskog odsječka. Potom smanjuje prioritet odabranoj dretvi i odabire sljedeću dretvu te se postupak ponavlja. Podatkovna struktura potrebna za ovakav raspoređivač mora imati jedan red s kazaljkom na trenutno aktualnu dretvu.

Neka je podatkovna struktura definirana sa:

- `KP_red[N]` – polje s indeksima dretvi koje se raspoređuju,
- `N` – broj dretvi koje se raspoređuju,
- `KP_akt` – prethodno odabrana dretva te
- `T` – trajanje vremenskog odsječka.

Uz zadanu strukturu podataka, dretva raspoređivača može se prikazati sljedećom funkcijom:

```
dretva raspoređivač(){
   i = 1
   ponavljati {
      brojač = 0
```

```
      dok je (dretva[KP_red[i]].red ≠ PRIPRAVNE) ∧ (brojač < N){
         i = (i + 1) mod N
         brojač++
      }
      ako je (brojač < N) {
         KP_akt = i
         zabraniti_prekidanje
         dretva[KP_red[i]].pri = dretva[KP_red[i]].pri + 1
         dozvoliti_prekidanje
         Zakasniti(T)
         zabraniti_prekidanje
         dretva[KP_red[i]].pri = dretva[KP_red[i]].pri - 1
         dozvoliti_prekidanje
         i = (i + 1) mod N
      } inače
         Zakasniti(T) //nema pripravnih dretvi
   } do zauvijek
}
```

Zbog jednostavnosti, algoritmom nije predviđeno mjerenje stvarnog vremena koje je dretva provela u izvođenju, a koje može biti i manje od proteklog intervala. To se ne može ostvariti bez mijenjanja funkcija jezgre, odnosno, u ovom slučaju funkcija za ostvarenje monitora. U nekom slijedu izvođenju može se dogoditi da se prioritetnija dretva aktivira i potisne dretvu odabranu ovim raspoređivačem. Jasno je da se o pravednoj raspodjeli procesorskog vremena u ovakvim slučajevima može raspravljati. Treba međutim imati na umu da je zadani algoritam jednostavan i da nudi mogućnost paralelnog rada više radnih dretvi.

Problemi koji mogu nastati korištenjem ove metode jesu istodobno korištenje drugih algoritama koji dinamički mijenjaju prioritete, kao što su metoda nasljeđivanja prioriteta ili metoda vršnih prioriteta. Ukoliko se na takve stvari pripazi dovoljnim razmicanjem radnih dretvi od ostalih prioritetnijih dretvi u sustavu korištenjem različitih monitora, ova metoda može poslužiti kao jednostavan kružni raspoređivač.

Ukoliko se u sustavu nalazi više skupina radnih dretvi različitog prioriteta bilo bi potrebno ili dodati po jednu dretvu raspoređivača za svaku razinu, ili malo promijeniti algoritam uz dodatne informacije u podatkovnoj strukturi.

6.4. Ostvarenje semafora

U poglavlju 5 prikazano je kako se problemi riješeni sa semaforima mogu riješiti s monitorima. Međutim, u sustavu se ipak može pojaviti potreba za funkcionalnošću semafora. Binarni semafori koriste se za ostvarenje međusobnog isključivanja nad kritičnim odsječkom za što su funkcije `Ući_u_monitor` i `Izaći_iz_monitora` dovoljne. Kako ostvariti opće semafore pomoću monitora prikazano je u nastavku.

Da bi se semafor ostvario s monitorom potreban je jedan monitor s jednim redom uvjeta, varijabla za zapisivanje stanja semafora te druga varijabla za brojanje dretvi koje čekaju na semafor. U nekom minimalnom sustavu mogao bi se koristiti i samo jedan monitor s po jednim redom uvjeta za svaki semafor. Zapravo ukoliko se traži egzaktno

ostvarenje semafora onda je korištenje jednog monitora nužan uvjet, jer je u operacijskim sustavima poziv za rad sa semaforima jezgrina funkcija koja se ne može pozvati paralelno iz više dretvi. Ovdje je to dopušteno ukoliko se radi o različitim semaforima jer su tada dretve koji sudjeluju u sinkronizaciji nezavisne.

Dodatna struktura podataka za skup od `S` semafora koje treba rezervirati jest polje `Sem[S]` kod kojeg se svaki element sastoji od:

- `M` – indeks monitora koji se koristi za taj semafor,
- `V` – vrijednost semafora (0 ili veća), i
- `broj_čekača` – broj dretvi koje čekaju na semafor.

Redovi uvjeta kao i red za ulaz u monitor prioritetni su redovi te će i semafori ostvareni imati prioritetni red čekanja. Dretve istog prioriteta u redu su složene po redu prispijeća.

U nastavku slijedi opći brojeći semafori, kod kojih je moguće povećavati i smanjivati vrijednost semafora za jedan, te potom semafori kod kojih se povećavanje i smanjivanje može obaviti za proizvoljnu pozitivnu vrijednost.

```
m-funkcija ČekatiOSem(semID) {
   Ući_u_monitor(Sem[semID].M)
   ako je (Sem[semID].V > 0) {
      Sem[semID].V--
   } inače {
      Sem[semID].broj_čekača++
      Uvrstiti_u_red_uvjeta(Sem[semID].M, 1)
   }
   Izaći_iz_monitora(Sem[semID].M)
}

m-funkcija PostavitiOSem(semID) {
   Ući_u_monitor(Sem[semID].M)
   ako je (Sem[semID].broj_čekača > 0){
      Sem[semID].broj_čekača--
      Oslobodi_iz_reda_uvjeta(Sem[semID].M, 1)
   }
   inače {
      Sem[semID].V++
   }
   Izaći_iz_monitora(Sem[semID].M)
}

m-funkcija ČekatiOpćiSem(semID, broj) {
   Ući_u_monitor(Sem[semID].M)
   Sem[semID].broj_čekača++
   dok je (Sem[semID].V < broj) {
      Uvrstiti_u_red_uvjeta(Sem[semID].M, 1)
   }
   Sem[semID].broj_čekača--
   Sem[semID].V = Sem[semID].V - broj
```

```
   ako je ((Sem[semID].V > 0) ∧ (Sem[semID].broj_čekača > 0)) {
      Osloboditi_iz_reda_uvjeta(Sem[semID].Monitor, 1)
   }
   Izaći_iz_monitora(Sem[semID].M)
}

m-funkcija PostavitiOpćiSem(semID, broj) {
   Ući_u_monitor(Sem[semID].M)
   Sem[semID].V = Sem[semID].V + broj
   ako je (Sem[semID].broj_čekača > 0) {
      Osloboditi_iz_reda_uvjeta(Sem[semID].Monitor, 1)
   }
   Izaći_iz_monitora(Sem[semID].M)
}
```

Prikazano rješenje općeg semafora ima prioritetni red. Ponašanje semafora se nešto razlikuje od standardnog ostvarenja općih semafora u tome što se pri povećanju vrijednosti semafora, tj. pri pozivu funkcije `PostavitiOpćiSem()` oslobađa prva dretva iz reda čekanja. Ona tada pokušava smanjiti vrijednost semafora. Ako uspije, prolazi dalje te prije izlaska iz funkcije oslobađa sljedeću iz reda i daje joj priliku za prolaz semafora. Ukoliko prva iz reda ne uspije smanjiti vrijednost semafora neće se dati prilika ostalim u redu, iako možda tamo postoji dretva koja bi mogla proći semafor. To svojstvo može biti i poželjno, ali i neželjeno, ovisno u uporabi.

Rješenje koje nema taj nedostatak, ali je nešto neučinkovitije prikazano je u nastavku.

```
m-funkcija ČekatiOpćiSem(semID, broj) {
   Ući_u_monitor(Sem[semID].M)
   Sem[semID].broj_čekača++
   dok je (Sem[semID].V < broj){
      Uvrstiti_u_red_uvjeta(Sem[semID].M, 1)
      ako je (oslobodi_sve = 1 ) {
         oslobodi_sve = 0
         za i=1 do Sem[semID].broj_čekača-1 {
            Osloboditi_iz_reda_uvjeta(Sem[semID].Monitor, 1)
   }  }  }
   Sem[semID].broj_čekača--
   Sem[semID].V = Sem[semID].V - broj
   Izaći_iz_monitora(Sem[semID].M)
}

m-funkcija PostavitiOpćiSem(semID, broj) {
   Ući_u_monitor(Sem[semID].M)
   Sem[semID].V = Sem[semID].V + broj
   ako je (Sem[semID].broj_čekača > 0) {
      oslobodi_sve = 1
      Osloboditi_iz_reda_uvjeta(Sem[semID].Monitor, 1)
   }
   Izaći_iz_monitora(Sem[semID].M)
}
```

Ukoliko je vrijednost nakon uvećavanja veća od nule svim se dretvama iz reda na taj semafor daje prilika da smanje vrijednost semafora. Najprije će to pokušati dretva s najvećim prioritetom te nakon nje ostale, po prioritetima.

Oslobađanje svih dretvi iz reda uvjeta (osim prve) obavljeno je u funkciji `ČekatiOpćiSem()` zbog toga što prva dretva iz tog reda može imati prioritet veći i od dretve koja poziva funkciju `PostavitiOpćiSem()`. Ako bi se to pokušalo napraviti iz funkcije `PostavitiOpćiSem()` moglo bi se dogoditi da prva dretva koja se oslobodi iz reda uvjeta ima veći prioritet, ali ne može smanjiti vrijednost semafora. Ona bi zbog većeg prioriteta odmah ušla u monitor te potom opet završila kao prva u redu uvjeta te bi opet bila oslobođena, itd.

Prikazano je rješenje idejno vrlo slično ostvarenjima semafora u standardnim operacijskim sustavima gdje se dretve nadmeću za semafor koristeći svoj prioritet.

6.5. Redovi poruka

Komunikacija porukama najpoznatiji je način komunikacije među dretvama. U raspodijeljenim je sustavima to i jedini način komunikacije. U procesu komunikacije razlikujemo dretve koje šalju poruke i dretve koje ih primaju. Poslane poruke moraju biti prihvaćene istim redoslijedom kojim su i poslane. Složeniji mehanizmi komunikacije porukama koji uključuju prioritet ili tipove poruka nisu razmatrani u nastavku, ali predloženo rješenje može biti temelj nekih složenijih ostvarenja.

Slanje poruke ne obavlja se direktno od pošiljatelja ka primatelju već se kao posrednik koristi operacijski sustav i njegove funkcije. Operacijski sustav se brine o rezervaciji spremničkih lokacija za privremenu pohranu poruka, redoslijedu prihvaćanja i čitanja poruka, potrebama sinkronizacije kada se pokušava čitati poruka iz praznog reda ili ako se pokušava poslati poruka a red je pun.

Kod ostvarenja reda poruka s monitorom, uz same monitore potrebno je rezervirati spremnički prostor za poruke i dodatne pomoćne varijable i to posebno za svaki red. Pretpostavljeno je da su poruke istog formata i veličine te da se za to može koristiti međuspremnik ograničene duljine. Komunikacija porukama na taj način postaje jednostavna komunikacija preko ograničenog međuspremnika, vrlo slična prikazanom rješenju u potpoglavlju 5.3.

Neka je za podatke o redovima poruka definirano polje `Poruke[]` kod kojeg se svaki element sastoji od:

- `M` – indeks monitora s dva reda uvjeta, jedan za čekanje na slanje a drugi za čitanje,
- `MS[]` – međuspremnik veličine N poruka,
- `N` – veličina međuspremnika,
- `broj_praznih` – broj praznih mjesta u međuspremniku,
- `IZ` – indeks prvog punog mjesta u međuspremniku,
- `UL` – indeks prvog praznog mjesta u međuspremniku,
- `čeka_na_slanje` – broj dretvi koje čekaju na slanje poruke, i
- `čeka_na_poruku` – broj dretvi koje čekaju na poruku.

Spremnički prostor za potreban broj redova poruka unaprijed treba rezervirati te sve varijable moraju se inicijalizirati. Za ostvarenje reda poruka potrebne su dvije funkcije, jedna za stavljanje poruke u red i druga za čitanje poruke iz reda. Jedno rješenje tih

funkcija prikazano je u nastavku.

```
m-funkcija Poslati_poruku(red, P) {
   Ući_u_monitor(Poruke[red].M)
   Poruke[red].čeka_na_slanje++
   dok je (Poruke[red].broj_praznih = 0) {
      Uvrstiti_u_red_uvjeta(Poruke[red].M, 1)
   }
   Poruke[red].čeka_na_slanje--
   Poruke[red].MS[Poruke[red].UL] = P
   Poruke[red].UL = (Poruke[red].UL + 1) mod Poruke[red].N
   Poruke[red].broj_praznih--
   ako je (Poruke[red].čeka_na_poruku > 0) {
      Osloboditi_iz_reda_uvjeta(Poruke[red].M, 2)
   }
   Izaći_iz_monitora(Poruke[red].M)
}

m-funkcija Primiti_poruku(red, R) {
   Ući_u_monitor(Poruke[red].M)
   Poruke[red].čeka_na_poruku++
   dok je (Poruke[red].broj_praznih = Poruke[red].N) {
      Uvrstiti_u_red_uvjeta(Poruke[red].M, 2)
   }
   Poruke[red].čeka_na_poruku--
   R = Poruke[red].MS[Poruke[red].IZ]
   Poruke[red].IZ =(Poruke[red].IZ + 1) mod Poruke[red].N
   Poruke[red].broj_praznih++
   ako je (Poruke[red].čeka_na_slanje > 0) {
      Osloboditi_iz_reda_uvjeta(Poruke[red].M, 1)
   }
   Izaći_iz_monitora(Poruke[red].M)
   vratiti(R)
}
```

Spremnički prostor za poruke treba osigurati u prostoru kojeg mogu dohvaćati sve dretve. Isto kao i kod semafora moguće je koristiti samo jedan monitor s po dva reda uvjeta za svaki red poruka. Ograničenja koji iz tog proizlaze su neznatna, a takvo je rješenje bliže ostvarenjima reda poruka u jezgri jer su ti pozivi tada neprekidivi. U ovom je slučaju to ograničenje suvišno, tj. dozvoljeno je prekidanje funkcije za slanje i primanje poruka drugim funkcijama. Ono što nije dozvoljeno jest istovremeni rad nad istim redom poruka i to je zaštićeno korištenjem istog monitora.

7. MODEL IZGRADNJE MONITORA ZA ARM7 ARHITEKTURU

ARM7 porodica mikroprocesora čest je izbor za ugradnju u sustavima upravljanja te je stoga izabrana za analizu uporabe monitora. Relativno jednostavna jezgra procesora dostupna u raznim opisnim jezicima lako se nadograđuje potrebnim funkcionalnostima, kao što su to komunikacijski i drugi upravljači. ARM7 odlikuje se i vrlo niskom potrošnjom energije, kao i dostupnošću mnoštva razvojnih okruženja. ARM7 je zbog toga vrlo čest izbor za *sustav na čipu* (engl. *System-on-Chip*) arhitekture.

ARM7 temelji se na arhitekturi *pročitaj-pohrani* (engl. *load-store*), 32-bitovnim instrukcijama te na tri načina adresiranja. Instrukcijski se cjevovod sastoji od tri dijela: pribavljanje, dekodiranje te izvođenje instrukcije. Taj je sustav trodijelnog cjevovoda vrlo učinkovit što se tiče ostvarenja u sklopovlju. Međutim, za veću učinkovitost potreban je dulji cjevovod, što je napravljeno u nekim inačicama ARM procesora [Fur00]. Detaljniji opis ARM arhitekture može se naći u dokumentima tvrtke ARM [12], kao i u knjigama [Fur00, Wol01]. Pored 32-bitovnih instrukcija ARM7 podržava i način rada u kojem koristi kraće, 16-bitovne instrukcije, tzv. *Thumb* instrukcije, povećavajući gustoću programa, odnosno, smanjujući njegovu ukupnu veličinu.

7.1. Programski model ARM7 procesora

Programski model ARM7 procesora sastoji se od skupa instrukcija i registara dostupnih procesoru. Na raspolaganju je 16 registara opće namjene, `r0-r15`, s time da se neki od tih registara koriste za posebne namjene. `r15` predstavlja programsko brojilo i često se označava sa `pc` (engl. *Program Counter*). `r13` se koristi za rad sa stogom te se za njega koristi i oznaka `sp` (engl. *Stack Pointer*), dok `r14` služi kao registar za privremenu pohranu sadržaja programskog brojila pri pozivu potprograma ili pojavi prekida te ima i dodatnu oznaku `lr` (engl. *Link Register*). Uz navedene registre opće uporabe postoji još i statusni registar `cpsr/spsr` (engl. *Current/Saved Program Status Register*). Neki od navedenih registara različiti su za različite načine rada procesora.

ARM7 može biti u jednom od 6 načina rada, koji su određeni vrijednostima zadnjih pet bitova statusnog registra:

- korisnički način rada (engl. *user mode*),
- brzi prekidni načina rada (engl. *fast interrupt mode* – fiq),
- prekidni načina rada (engl. *interrupt mode* – irq),
- način rada u slučaju nedefinirane instrukcije (engl. *undefined mode* – und),
- način rada u slučaju nemogućnosti dohvata operandi (engl. *abort mode* – abt),
- programski prekidi (engl. *software interrupt* – svc),
- sustavski način rada (engl. *system mode*).

Od registara opće namjene neki su registri različiti za pojedine načine rada, osim za sustavski način rada koji koristi registre korisničkog načina. Tako se npr. pri prijelazu iz korisničkog načina rada u prekidni način rada registri `r13`, `r14` i statusni registar

zamjenjuju registrima novog načina rada, tj. registrima `r13_irq`, `r14_irq` te `spsr_irq`. Vidljivi registri za pojedine načine rada prikazani su na slici 7.1.

dostupni u korisničkom načinu rada

dostupni u sustavskim načinima rada

user mode system mode	*fiq mode*	*svc mode*	*abort mode*	*irq mode*	*undefined mode*
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8_fiq				
r9	r9_fiq				
r10	r10_fiq				
r11	r11_fiq				
r12	r12_fiq				
r13 (sp)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14 (lr)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (pc)					
cpsr	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_und

Slika 7.1 Vidljivi registri ARM7 procesora

Promjena načina rada obavlja se pri pojavi prekida ili direktnim mijenjanjem određenih zastavica u statusnom registru, koje je moguće napraviti iz svih načina rada osim iz korisničkog.

Ponašanje procesora pri pojavi prekida može se opisati u četiri koraka:

1. zabranjuje se daljnje prekidanje,
2. procesor se prebacuje u novi način rada,
3. programsko brojilo se pohranjuje u registar `r14_`*`mode`* i `cpsr` u `spsr_`*`mode`*, gdje je *`mode`* oznaka prekida,
4. u programsko brojilo upisuje se vrijednost između 00_{16} i $1C_{16}$, ovisno o prekidu.

Na adresama 00_{16} do $1C_{16}$ treba postaviti instrukcije grananja koje će preusmjeriti obradu dotičnog prekida na željeni dio kôda. Tablica 7.1 sadrži adrese pojedinih prekida.

Tablica 7.1 Vektor adresa prekida

Prekid	Oznaka	Adresa
Reset	SVC	0x0000
Nedefinirana instrukcija	UND	0x0004
Programski prekid	SVC	0x0008
Greška pri dohvatu instrukcije	Abort	0x000C
Greška pri dohvaćanju podatka iz spremnika	Abort	0x0010
Prekid	IRQ	0x0018
Brzi prekid	FIQ	0x001C

Brzi prekidi imaju zadnju adresu u tablici te je moguće da kôd za obradu slijedi odmah od te adrese, bez grananja. Brzi prekidi imaju na raspolaganju više privatnih registara od ostalih načina rada ciljajući na poboljšanje učinkovitosti pri obradi takvih prekida. U jednostavnijim slučajevima ti će registri biti dovoljni za obradu prekida te se neće trošiti vrijeme na spremanje konteksta, tj. sadržaja registara.

Pri povratku iz obrade prekida potrebno je vratiti procesor u stanje koje je bilo prije

pojave prekida, tj. potrebno je vratiti stanje svih registara. Vrijednost programskog brojila te statusni registar treba vratiti u istoj operaciji te za to postoje posebne inačice instrukcija bilo da je povratna adresa spremljena na stogu ili u registru `r14`. Pri povratku iz nekih prekida potrebno je najprije korigirati vrijednost programskog brojila. Pri završetku obrade prekida i brzih prekida vrijednost programskog brojila treba umanjiti za 4 dok za prekide dohvata za 8.

Zbog mogućnosti pojave više prekida u isto vrijeme pojedinim je prekidima pridijeljen različiti prioritet. Kod ARM7 arhitekture prekidima su pridijeljeni prioriteti redom: reset (najveći prioritet), greška pri dohvata podataka iz spremnika, brzi prekid, prekid, greška pri dohvatu instrukcije (ili njenih operandi), programski prekidi i nedefinirana instrukcija.

Za pristup i upravljanje ulazno izlaznim napravama koristi se mapiranje adresa spremničkih lokacija te prekidima. Registri ulazno-izlaznih naprava dostupni su na određenim adresama adresnog prostora.

Način pohrane registara u spremnik može biti u *Big-endian*[1] ili *Little-endian* načinu, tj. podržana su oba načina i odabir se obavlja postavljanjem određenih premosnika na samom sklopu.

Instrukcijski skup ARM procesora može se podijeliti u tri skupine: instrukcije dohvata i pohranjivanja podataka, instrukcije obrade podataka te instrukcije grananja. Pregled 32-bitovnih instrukcija korišten u izgradnji monitora prikazan je na [13].

7.2. Jezgrine funkcije ostvarene u strojnom jeziku

Ostvarenje jezgrinih funkcija, tj. funkcija za rad s monitorima prikazano u nastavku, napravljeno je korištenjem *RealView Developer Suite v2.1* razvojnog alata i simulatora te je strojni jezik prilagođen tome alatu. Od okoline korišteni su prekidni međusklop te brojila čije je upravljanje opisano u [14].

Prije ispisa kôda funkcija navedene su bitne vrijednosti i strukture podataka koje se koriste te kraći opis načina njihove definicije i pohrane.

Konstantne vrijednosti definiraju se korištenjem oznake `EQU`.

```
MUTEX                          EQU   -1
UVJET                          EQU   -2
KRAJ_LISTE                     EQU   -4
Uci_u_monitor                  EQU   1
Izaci_iz_monitora              EQU   2
Uvrstiti_u_red_uvjeta          EQU   3
Osloboditi_iz_reda_uvjeta      EQU   4
```

Spremničke lokacije za pojedine varijable i polje definiraju se oznakom `DCD`. Tako se mjesto za kazaljku na aktivnu dretvu, prvu iz reda pripravnih te podatke dretvi i monitora

[1] „Little-endian“ način zapisa na rastućim spremničkim adresama sprema oktete od najmanje značajnih prema značajnijim, obrnuto od „Big-endian“ načina. Sam je naziv potekao iz knjige „Gulliverova putovanja“ Jonathana Swifta gdje su stanovnici Lilliputa izglasali zakon po kojem se jaja smiju razbijati samo na užoj strani. Taj je zakon izazvao sukobe i građanski rat s mnogo žrtava nakon kojeg su zagovornici „Big-endian“ načina razbijanja jaja bili prognani s otoka.

mogu zauzeti sljedećim dijelom programa:

```
Aktivna      DCD         Dretva_1
Pripravne    DCD         Dretva_2

Dretva       DCD   Dretva_1, Dretva_2, Dretva_3, Dretva_IRQ
;Dretva_n
;      DCD   id, prioritet, red, iduca, kasniti, dp_zadnji, din_pri
;      DCD   r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12
;      DCD   sp, lr, pc, cpsr
;
Dretva_3
       DCD   3, 50, PRIPRAVNE, Dretva_2, 0, 0, KRAJ_LISTE
       DCD   0,1,2,3,4,5,6,7,8,9,10,11,12
       DCD   Stog_Dretva3, 14, Dretva_3_start, Mode_USR
```

Ukoliko iza `DCD` slijedi labela onda ta spremnička lokacija sadrži adresu podataka ili kôda te labele. Na ovaj način `Aktivna` se može inicijalno postaviti na određenu dretvu, ali se tijekom izvođenja vrijednost može mijenjati. Najprije je prikazana struktura podataka o dretvi (sve iza znaka `;` je komentar) te primjer za jednu dretvu.

```
Monitor
      DCD   Mon1         ; adresa prvog monitora
      DCD   Mon2
;Mon1
;   DCD     vlasnik
;   DCD     prva_u_redu_za_ulaz_u_monitor
;   DCD     prva_u_redu_uvjeta_1
;   DCD     prva_u_redu_uvjeta_2

Mon1
   DCD      3
   DCD      KRAJ_LISTE
   DCD      KRAJ_LISTE
   DCD      KRAJ_LISTE
```

Funkcije za rad s monitorima pozivaju se korištenjem programskih prekida uz prethodno pohranjivanje parametara na stog. Primjer poziva prikazan je u nastavku.

```
   STMFD    sp!, {r0}   ; r0 - indeks monitora
   SWI      Uci_u_monitor
   LDMFD    sp!, {r1}   ; r1 ce sadržati povratnu vrijednost J-funkcije
```

Inicijalizacija sustava, postavljanje funkcija za obradu prekida, postavljanje prekidnog sustava i inicijalizacija brojila prikazani su u Dodatku A. U nastavku su prikazani dijelovi kôda za programske prekide preko kojih se pozivaju jezgrine funkcije, prihvat prekida sata te jednostavne primjere kôda koje obavljaju dretve.

Pozivi za ostvarenje dodatnih funkcionalnosti, tj. algoritam nasljeđivanja prioriteta i algoritam vršnih prioriteta, su komentirani budući se oni ne mogu koristiti istovremeno. Pojedini se algoritam uključuje tako da se dotični pozivi dodaju u kôd (makne se oznaka komentara).

7.2.1. Aktiviranje dretve

Pod aktiviranjem dretve podrazumijeva se učitavanje konteksta dretve koja je označena kao aktivna te njen nastavak rada od mjesta na kojem je bila prekinuta ili u stanje prije poziva jezgrine funkcije. Sljedeći kôd se poziva iz nadglednog načina rada procesora, ali se prilikom prebacivanja na sljedeću instrukciju aktivne dretve procesor vraća u korisnički način rada.

```
Aktivirati_aktivnu
      LDR         lr, =Aktivna
      LDR         lr, [lr]
      MOV         r0, #AKTIVNA
      STR         r0, [lr, #DRETVA_RED]
      ADD         lr, lr, #KONTEKST
      LDR         r1, [lr, #KONTEKST_SPSR]
      MSR         SPSR_cf, r1
      LDMFD       lr, {r0-r14}^
      ADD         lr, lr,#KONTEKST_PC
      LDMFD       lr, {pc}^
```

Navedeni kôd se poziva pri završetku jezgrinih funkcija te je sadržaj registara nebitan i nije ga potrebno pohranjivati. Kao pomoćni registar koristi se `lr` (`r14`) iz razloga što je to registar trenutnog privilegiranog načina rada i smije se promijeniti budući je povratna adresa pohranjena u kontekstu dretve.

Navedeni kôd može se opisati u nekoliko koraka. Najprije se dohvaća adresa na kojoj se nalazi kazaljka na spremničku lokaciju aktivne dretve. Potom se dohvaća adresa aktivne dretve. U opisnik te dretve upisuje se njen novi status `AKTIVNA` te se kazaljka pomiče na početak konteksta te dretve, tj. sadržaja registara. Iz konteksta se najprije učitava sadržaj statusnog registra te se ta vrijednost postavlja u `spsr` koji će se prilikom prijelaza u korisnički način rada prepisati u `cpsr`. Nakon toga se iz konteksta obnavljaju registri `r0-r14`. Zbog znaka ^ obnoviti će se korisnički registri, tj. registri `sp` (`r13`) i `lr` (`r14`) iz korisničkog načina rada, dok su i dalje aktivni registri iz nadglednog načina rada. Konačno, `lr` se podešava da pokazuje na pohranjeno programsko brojilo te se s te adrese ono učitava u programsko brojilo. Istovremeno se procesor prebacuje u način rada definiranom u `spsr`-u zbog oznaka ^ koji u ovom slučaju, budući se učitava i `pc`, ima drugo značenje.

Nakon izvršenja navedenog dijela kôda stanje registara vratiti će se u ono u kojem je procesor zatečen prije prekida.

7.2.2. Prihvat programskog prekida

Prihvat programskog prekida treba započeti sa spremanjem konteksta aktivne dretve. Potom se određuje uzrok prekida te poziva odgovarajuća funkcija. Kôd u nastavku radi upravo navedeno. Zbog preglednosti opis kôda dodan je kao komentar.

```
SWI_obrada
; Najprije treba pohraniti sadržaj lr-a, jer se on koristi kao pomoćni
; registar.Potom se mijenjanjem statusnog registra zabranjuju svi prekidi
      STMFD       sp!, {lr}
      MOV         lr, #Mode_SVC|I_Bit|F_Bit
      MSR         cpsr_cf, lr
```

```
; Spremanje konteksta aktivne dretve obavlja se slično kao i kod
; učitavanja konteksta, tj. aktiviranja dretve.
      LDR         lr, =Aktivna
      LDR         lr, [lr]
      ADD         lr, lr, #KONTEKST
      STMEA       lr, {r0-r14}^
      MRS         r0, spsr
      STR         r0, [lr, #KONTEKST_SPSR]
      LDMFD       sp!, {r3}
      STR         r3, [lr, #KONTEKST_PC]

; Adresa konteksta aktivne dretve postavlja se na stog da bi se iz
; jezgrinih funkcija mogli dohvatiti parametri.
      STMFD       sp!, {lr}

; Iz instrukcije prekida („SWI broj“) računa se „broj“ kojim se poziva
; odgovarajuća jezgrina funkcija.
      LDR         r3, [r3,#-4]
      BIC         r3, r3, #0xFF000000

; Budući sve četiri jezgrine funkcije rade s monitorima, indeks monitora
; dohvaća se ovdje.
      LDR         r1, [lr, #KONTEKST_SP]
      LDMFD       r1!, {r0}
      STR         r1, [lr, #KONTEKST_SP]

; Uz adresu traženog monitora dohvaća se i adresa aktivne dretve.
      LDR         r1, =Monitor
      LDR         r1, [r1, r0, LSL #2]
      LDR         r2, =Aktivna
      LDR         r2, [r2]

; Nakon što registar r0 sadrži indeks monitora, r1 njegovu adresu te
; r2 adresu aktivne dretve može se pozvati tražena funkcija koja je
; definirana vrijednošću u r3, tj. broju u SWI instrukciji.
      BL          pozovi_j_funkciju

; Pri povratku iz funkcije, sa stoga se skida povratna vrijednost i
; sprema ju se na korisnički stog.
      LDMFD       sp!, {r0}
      LDMFD       sp!, {r1}
      LDR         r2, [r1, #KONTEKST_SP]
      STMFD       r2!, {r0}
      STR         r2, [r1, #KONTEKST_SP]

; Na kraju se aktivira dretva koja je označena kao aktivna.
      B           Aktiviraj_aktivnu
```

```
; Pozivanje određene funkcije napravljeno je preko tablice.
pozovi_j_funkciju
      LDR         r4, =adresa_j_funkcija
      LDR         pc, [r4, r3, LSL #2]

; Tablica koja sadrži adrese traženih funkcija
adresa_j_funkcija
      DCD         0
      DCD         J_uci_u_monitor
      DCD         J_izaci_iz_monitora
      DCD         J_uvrstiti_u_red_uvjeta
      DCD         J_osloboditi_iz_reda_uvjeta
```

Iz programskog se prekida pozivaju sve četiri navedene funkcije, označene kao jezgrine funkcije. Registri `r0`, `r1` i `r2` sadrže definirane vrijednosti i na stogu se nalazi kazaljka na kontekst dretve koja poziva jezgrinu funkciju.

7.2.3. Jezgrina funkcija `Ući_u_monitor()`

Funkcijom `Ući_u_monitor()` pozivajućoj se dretvi dozvoljava ulaz u monitor ili ju se stavlja u ulazni red traženog monitora ukoliko je neka druga dretva već u monitoru.

```
J_uci_u_monitor
; r3 = Aktivna.id
      LDR         r3, [r2, #DRETVA_ID]

; r4 = Monitor[M].vlasnik
      LDR         r4, [r1, #MONITOR_VLASNIK]
      CMP         r4, #MONITOR_NITKO
; Ukoliko niti jedna dretva nije unutar monitora, tražena dretva ulazi.
; Monitor[M].vlasnik = Aktivna.id
      STREQ       r3, [r1, #MONITOR_VLASNIK]

; Povećati_prioritet(Aktivna, M)-protokol vršnog prioriteta
;     STMEQFD     sp!, {lr}
;     BLEQ        Povecati_prioritet
;     LDMEQFD     sp!, {lr}

      BEQ         vrati_ok__J_uci_u_monitor

; Ukoliko se neka druga dretva nalazi u monitoru, pozivajuća se smješta
; u ulazni red.
      LDR         r3, [r1, #MONITOR_RED]
; Aktivna.iduća = Monitor[M].red
      STR         r3, [r2, #DRETVA_IDUCA]
      STR         r2, [r1, #MONITOR_RED]
; Monitor[M].red = Aktivna
      STR         r0, [r2, #DRETVA_RED]

; Nasljedni_prioriteti_povecati(Aktivna, M)-prot. nasljeđivanja pri.
```

```
;     STMFD sp!, {lr}
;     BL          Nasljedni_prioriteti_povecati ;
;     LDMFD sp!, {lr}

; Operacije koje se pozivaju iz različitih jezgrinih funkcija odvojene su
; u posebnom modulu. Prije poziva tih funkcija potrebno je pohraniti
; povratnu adresu programskog brojila.
      STMFD       sp!, {lr}
      BL          j_prebaci_prvu_pripravnu_u_aktivne
      LDMFD       sp!, {lr}

; Prije povratka na stog se pohranjuje povratna vrijednost.
vrati_ok__J_uci_u_monitor
      MOV         r0, #0
      STMFD       sp!, {r0}
      MOV         pc, lr
```

Ova funkcija kao i sve naredne pretpostavlja da nema grešaka u argumentima, tj. da odgovarajuće dretve i monitori postoje.

7.2.4. Jezgrina funkcija `Izaći_iz_monitora()`

Dretva koja poziva ovu jezgrinu funkciju napušta monitor te ga prepušta prvoj iz reda monitora ili ukoliko tamo nema niti jedne dretve označava monitor slobodnim.

```
J_izaci_iz_monitora
; smanjiti_prioritet(Aktivna, M) - prot. nasljeđivanja/vršnih prior.
;     STMFD sp!, {lr}
;     BL          Smanjiti_prioritet           ; r0=M, r2=Aktivna
;     LDMFD sp!, {lr}

; Najprije se aktivna dretva premješta u red pripravnih.
      LDR         r3, =Pripravne
      LDR         r4, [r3]
      STR         r4, [r2, #DRETVA_IDUCA]
      STR         r2, [r3]
      MOV         r4, #PRIPRAVNE
      STR         r4, [r2, #DRETVA_RED]

; Potom treba prvu iz reda monitora prebaciti u red pripravnih.
; Budući se ovaj dio kôda poziva i iz sljedećih jezgrinih funkcija
; potrebna je i dodatna labela
J_izaci_iz_monitora_sljedeca
      STMFD       sp!, {lr}
; Adresa Monitor[M] postavlja se na stog
      STMFD       sp!, {r1}
      BL          j_osloboditi_prvu_iz_reda_monitora

; prebaci prvu iz reda pripravnih u red aktivne
      BL          j_prebaci_prvu_pripravnu_u_aktivne
```

```
; Sa stoga se dohvaća pohranjena adresa programskog brojila.
      LDMFD       sp!, {lr}
; Prije povratka na stog se pohranjuje povratna vrijednost.
      MOV         r0, #0
      STMFD       sp!, {r0}
      MOV         pc, lr
```

Pomoćna funkcija `j_osloboditi_prvu_iz_reda_monitora()` mogla se i izravno ovdje ugraditi, ali je radi preglednosti i čitljivosti kôda ipak odvojena.

7.2.5. Jezgrina funkcija `Uvrstiti_u_red_uvjeta()`

Kada dretva želi zaustaviti svoje izvođenje poziva se ova funkciju. Prva iz ulaznog reda ulazi sljedeća u monitor ili ako u tom redu nema dretvi monitor se označava kao slobodan.

```
J_uvrstiti_u_red_uvjeta
; smanjiti_prioritet(Aktivna, M) - prot. nasljeđivanja/vršnih pri.
;     STMFD sp!, {lr}
;     BL          Smanjiti_prioritet            ; r0=M, r2=Aktivna
;     LDMFD sp!, {lr}

; S korisničkog stoga potrebno je još dohvatiti indeks reda uvjeta.
      LDR         r4, [sp]
      LDR         r0, [r4, #KONTEKST_SP]
      LDMFD       r0!, {r3}
      STR         r0, [r4, #KONTEKST_SP]

; Dretva se označava da je u redu uvjeta, te ju se tamo i pomiče.
      MOV         r0, #RED_UVJETA
      STR         r0, [r2, #DRETVA_RED]
      ADD         r0, r1, #MONITOR_RED_UVJETA
      ADD         r0, r0, r3, LSL #2
      LDR         r3, [r0]
      STR         r3, [r2, #DRETVA_IDUCA]
      STR         r2, [r0]

; U slučaju da se radi o dretvi za obradu prekida sata treba označiti
; da je obrada gotova te eventualno i pozvati dodatnu funkciju.
; Dretva koja obrađuje prekide - dretva na adresi Dretva_IRQ
      LDR         r3, =Dretva_IRQ
      CMP         r2, r3
; Aktivna = Dretva_IRQ ?
      BNE         J_uvrstiti_u_red_uvjeta_kraj

; Radi se o Dretvi_IRQ
; Označiti da je prekid obrađen
      LDR         r0, =Prekid_aktivan
      MOV         r2, #0
```

```
      STR         r2, [r0]

;     BL          IRQ_kraj_obrade
; Dodatni dio ovisi o prekidu - nastavku je prikazano za prekid brojila
; kod kojeg je potrebno ponovo omogućiti prekidanje s tim brojilom.
      LDR         r0, =IRQ_Enable_Set
      LDR         r0, [r0]
      MOV         r2, #IRQ_Enable_Set_Value
      STR         r2, [r0]

; Propusti sljedeću dretvu u monitor.
J_uvrstiti_u_red_uvjeta_kraj
      B           J_izaci_iz_monitora_sljedeca
```

7.2.6. Jezgrina funkcija `Osloboditi_iz_reda_uvjeta()`

Rad funkcije može se opisati u nekoliko koraka. Najprije se pozivajuća dretva premješta u red monitora. Potom se prva iz reda uvjeta premješta u isti ulazni red monitora. Iz tog se reda tada odabire dretva najvećeg prioriteta, dodjeljuje joj se monitor i pomiče ju se u red pripravnih. Funkcija završava aktiviranjem prve iz reda pripravnih.

```
J_osloboditi_iz_reda_uvjeta
; smanjiti_prioritet(Aktivna, M) - prot. nasljeđivanja/vršnih pri.
;     STMFD sp!, {lr}
;     BL          Smanjiti_prioritet          ; r0=M, r2=Aktivna
;     LDMFD sp!, {lr}

; S korisničkog stoga potrebno je još dohvatiti indeks reda uvjeta.
      LDR         r4, [sp]
      LDR         r5, [r4, #KONTEKST_SP]
      LDMFD       r5!, {r3}
      STR         r5, [r4, #KONTEKST_SP]

; Pozivajuća se dretva pomiče u red monitora
      LDR         r4, [r1, #MONITOR_RED]
      STR         r4, [r2, #DRETVA_IDUCA]
      STR         r2, [r1, #MONITOR_RED]
      STR         r0, [r2, #DRETVA_RED]

; Dohvaća se adresa reda uvjeta
      ADD         r4, r1, #MONITOR_RED_UVJETA
      ADD         r4, r4, r3, LSL #2
      LDR         r5, [r4]
      CMP         r5, #KRAJ_LISTE
      BEQ         J_osloboditi_iz_reda_uvjeta__nema

; Prva iz reda (po prioritetu) vadi se iz reda
      STMFD       sp!, {r0,r1,lr}
      STMFD       sp!, {r4}
```

```
      BL          j_izvaditi_prvu_iz_reda
      LDMFD       sp!, {r4}
      LDMFD       sp!, {r0,r1,lr}

; Izvađena se dretva pomiče u red monitora
      LDR         r5, [r1, #MONITOR_RED]
      STR         r5, [r4, #DRETVA_IDUCA]
      STR         r4, [r1, #MONITOR_RED]
      STR         r0, [r4, #DRETVA_RED]

J_osloboditi_iz_reda_uvjeta__nema
; Prva iz reda monitora prebacuje se u monitor, tj. daje joj se monitor i
; prebacuje se u red pripravnih.
      B           J_izaci_iz_monitora_sljedeca
```

7.2.7. Prihvat prekida sata

Ukoliko se radi o prekidu uzrokovanim nekim sklopovljem, npr. brojilom, procesor prelazi u prekidni način rada te izvodi instrukciju na definiranoj lokaciji. Ta instrukcija mora biti instrukcija grananja na dio kôda za obradu prekida. Budući je prekid vrlo specifičan događaj za procesor, pored samih funkcija za rad s monitorima i njegov prihvat i obrada detaljnije su prikazani na primjeru prihvata prekida sata.

```
IRQ_prihvat
; Pohrana konteksta te zabrana danjeg prekidanja
      STMFD       sp!, {r0,r1,lr}
      MOV         lr, #Mode_IRQ|I_Bit|F_Bit
      MSR         cpsr_cf, lr

; Čitanje statusa IRQ sustava
      LDR         r0, =IRQ_Status
      LDR         r0, [r0]                ; adresa statusa IRQ u r0
      LDR         r0, [r0]                ; status IRQ u r0

; Uzrok prekida određuje funkciju koja će se pozvati
      TST         r0, #0x0010             ; brojilo 1
      BNE         Brojilo_1_prekid
      TST         r0, #0x0020             ; brojilo 2
      BNE         Brojilo_2_prekid
      TST         r0, #0x0001             ; FIQ
      BNE         FIQ_Obrada
      TST         r0, #0x0002             ; Programski prekid
      BNE         SWI_obrada
      TST         r0, #0x0004             ; COMM Rx
      BNE         COMM_Rx
      TST         r0, #0x0008             ; COMM Tx
      BNE         COMM_Tx

; Drugog razloga za prekid ne bi smjelo biti
```

```
      LDMFD       sp!, {r0,r1,lr}         ; kontekst sa stoga
      SUBS        pc, lr, #4

Brojilo_1_prekid
; Najprije treba onemogućiti daljnje prekidanje tim brojilom.
      LDR         r0, =IRQ_Enable_Set
      LDR         r0, [r0]
      MOV         r1, #IRQ_Disable_Set_Value
      STR         r1, [r0]

; Treba provjeriti je li možda obrada u tijeku
      LDR         r0, =Prekid_aktivan
      LDR         r1, [r0]
      CMP         r1, #1
; Ako je obrada u tijeku treba se samo vratiti natrag
      LDMEQFD     sp!, {r0,r1,pc}^

; Označiti da se ide u obradu prekida
      MOV         r1, #1
      STR         r1, [r0]

; Obrisati prekid brojila 1
      LDR         r0, =Timer1_Clear
      LDR         r0, [r0]
      MOV         r1, #0
      STR         r1, [r0]

; Pohraniti kontekst aktivne dretve
      LDMFD       sp!, {r0,r1,lr}
      SUB         lr, lr, #4        ; zbog prekida treba podesiti „pc“
      STMFD       sp!, {lr}
      LDR         lr, =Aktivna
      LDR         lr, [lr]
      ADD         lr, lr, #KONTEKST
      STMEA       lr, {r0-r14}^
      MRS         r0, spsr
      STR         r0, [lr, #KONTEKST_SPSR]
      LDMFD       sp!, {r3}
      STR         r3, [lr, #KONTEKST_PC]

; Premjestiti aktivnu u red pripravnih
      LDR         r3, =Aktivna
      LDR         r0, [r3]
      MOV         r1, #PRIPRAVNE
      STR         r1, [r0, #DRETVA_RED]
      LDR         r1, =Pripravne
      LDR         r2, [r1]
      STR         r2, [r0, #DRETVA_IDUCA]
```

```
        STR         r0, [r1]

; Izbaciti dretvu Dretva_IRQ iz reda uvjeta ili u red monitora
; ako nije slobodan, ili odmah u monitor ako je slobodan.
        LDR         r0, =Dretva_IRQ
        LDR         r1, =Monitor
        MOV         r2, #Monitor_IRQ
        LDR         r1, [r1, r2, LSL #2]

; IRQ dretva je sama u tom redu uvjeta, a sad se miče
        MOV         r2, #KRAJ_LISTE
        STR         r2, [r1, #MONITOR_RED_UVJETA]

; Je li monitor slobodan?
        LDR         r2, [r1, #MONITOR_VLASNIK]
        CMP         r2, #MONITOR_NITKO
        BNE         Prekid_netko_u_monitoru

; Ukoliko jest slobodan, IRQ dretva ga dobiva
        LDR         r2, [r0, #DRETVA_ID]
        STR         r2, [r1, #MONITOR_VLASNIK]

; Metoda vršnog prioriteta - Povećati prioritet
;       MOV         r2, r0
;       MOV         r0, #Monitor_IRQ
;       BL          Povecati_prioritet
;       MOV         r0, r2

; Ukoliko IRQ dretva ima veći prioritet od prve iz reda pripravnih
; može se odmah i proglasiti aktivnom, inače se prva iz reda pripravnih
; proglašava aktivnom a ova se stavlja u red.
        LDR         r2, =Pripravne
        LDR         r4, [r2]
        LDR         r5, [r4, #DRETVA_PRI]
        LDR         r6, [r0, #DRETVA_PRI]
        CMP         r6, r5
        STRLE       r0, [r3]
        BLE         Aktiviraj_aktivnu
; Potrebno je prvu iz reda pripravnih pomaknuti u Aktivnu, a IRQ dretvu
; postaviti u taj red
        LDR         r5, [r4, #DRETVA_IDUCA]
        STR         r5, [r0, #DRETVA_IDUCA]
        STR         r0, [r2]
        STR         r4, [r3]
        MOV         r5, #PRIPRAVNE
        STR         r5, [r0, #DRETVA_RED]
        MOV         r5, #AKTIVNA
        STR         r5, [r4, #DRETVA_RED]
```

```
      B           Aktiviraj_aktivnu

Prekid_netko_u_monitoru
; IRQ dretvu treba prebaciti u red monitora
      LDR         r2, [r1, #MONITOR_RED]
      STR         r2, [r0, #DRETVA_IDUCA]
      STR         r0, [r1, #MONITOR_RED]

; Nasljeđivanje prioriteta
;     MOV         r2, r0
;     MOV         r0, #Monitor_IRQ
;     BL          Nasljedni_prioriteti_povecati

; Vratiti prvu pripravnu u aktivnu: prva = fizički prva, jer je bila
; zadnja aktivna.

      LDR         r2, =Pripravne
      LDR         r4, [r2]
      STR         r4, [r3]
      LDR         r4, [r4, #DRETVA_IDUCA]
      STR         r4, [r2]
      B           Aktiviraj_aktivnu
```

U prikazanom rješenju prihvata i obrade prekida sata pretpostavljeno je da samo jedna dretva obrađuje prekide. U slučaju potrebe obrade više sličnih prekida navedeni kôd bi trebalo modificirati na način da se za svaku vrstu prekida doda po jedna dretva uz ostale potrebne varijable.

7.3. Veličina jezgrinih funkcija

S obzirom da je korišten normalni skup instrukcija svaka instrukcija zauzima 32 bita – jednu riječ. Zauzeće spremničkog prostora lako se izračuna kao broj instrukcija pomnožen s veličinom instrukcije. Tome treba dodati spremnički prostor potreban za pohranu podataka o dretvama i monitorima kao i za stogove. Neke se konstante, kao što su adrese UI naprava i upravljačkih sklopova također moraju pohraniti u spremnički prostor.

Analizom kôda može se doći do potrebne veličine sustavskog stoga. Budući se on koristi samo u jezgrinim funkcijama te pri obradi prekida potrebno je vrlo malo spremničkog prostora, manje od desetak riječi.

Veličina stoga dretvi ovisi o tome što dretve rade. Ukoliko nema rekurzivnih poziva funkcija te ukoliko funkcije nemaju veći brojem parametara i stog za dretve može biti prilično malen.

Podaci jezgre, u koje spadaju podaci o dretvama i podaci o monitorima također ovise o broju dretvi i monitorima u sustavu. Za svaki opisnik dretve potrebno je 25 riječi spremničkog prostora. Monitori zahtijevaju mnogo manje prostora. Minimalno svaki monitor zauzima 3 riječi te za svaki red uvjeta po jednu riječ više. Iz navedenog se jednostavno izračuna veličina potrebnog spremničkog prostora. Za neke podatke dretvi i monitora, kao što su redni broj, prioritet i oznaka reda u kome se nalazi, moglo bi se

koristiti i manje od jedne riječi, ali je ušteda na spremničkom prostoru neznatna.

Npr. ukoliko u sustavu postoji pet dretvi te tri monitora od kojih svaki ima po dva reda uvjeta za njih će biti dovoljno:

$$5 \times 25 + 3 \times 5 = 140 \text{ riječi.} \qquad (7.1)$$

Spremnički prostora za kôd sastoji se od nekoliko dijelova. Kôd samih funkcija za rad s monitorima zauzima najviše prostora. Uz te funkcije najčešće je potreban i sustav za prihvat i obradu prekida te za ostvarivanje kašnjenja. Zauzeće pojedinih dijelova kôda prikazano je u tablici 7.2.

Tablica 7.2 Zauzeće spremničkog prostora

Dijelovi kôda	**Veličina (riječi)**
Jezgrine funkcije (bez nasljeđivanja)	175
Nasljeđivanje	75
Ostvarivanje kašnjenja	210
Inicijalizacija	60
Ukupno:	520

Uz korištenje *Thumb* skupa instrukcija veličina se može znatno smanjiti. Ne baš dvostruko, jer se neke instrukcije ne mogu doslovno prenijeti u *Thumb* kao jedna instrukcija već kao dvije ili više njih.

Ukupno zauzeće spremničkog prostora jezgre dobiva se zbrajanjem prostora potrebnog za kôd te za podatke. Npr. za spomenuti sustav s pet dretvi i tri monitora za jezgru sustava potrebno je:

$$140 \text{ (dretve i monitori)} + 520 \text{ (kod)} + 40 \text{ (stog)} = 605 \text{ riječi} = 2420 \text{ okteta.} \qquad (7.2)$$

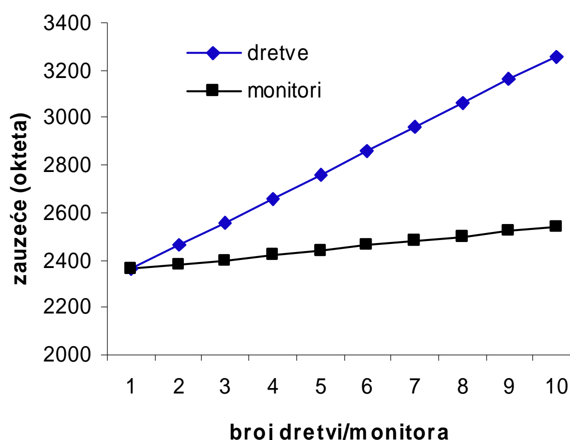
Ako se poveća broj dretvi na 10 ukupno zauzeće jednako je 2920 okteta. Za izračun veličine potrebnog spremničkog prostora za kôd i podatke jezgre mogle bi se koristiti formule:

$$V = 560 + 25 \times \text{broj_dretvi} + 5 \times \text{broj_monitora [riječi]},$$

$$V = 2240 + 100 \times \text{broj_dretvi} + 20 \times \text{broj_monitora [okteta].} \qquad (7.3)$$

Slika 7.2 prikazuje zauzeće spremničkog prostora jezgre za sustave s različitim brojem dretvi, odnosno, različitim brojem monitora. Može se primijetiti da monitori vrlo malo utječu na veličinu, dok povećanje zauzeća jezgre zbog dretvi nije zanemarivo. Podaci za deset dretvi zauzimaju gotovo kao polovica prostora za kôd.

Za bilo kakav posao dretvi, spremnički prostor potreban za njihov kôd biti će daleko veći od navedenog prostora potrebnog za jezgru. To je vidljivo i iz kôda za ostvarivanje kašnjenja. Njegova veličina usporediva je s funkcijama za ostvarenje monitora (gotovo 3/4), a obavlja relativno jednostavan posao (prikazan u potpoglavlju 6.2).



Slika 7.2 Zauzeće jezgre u ovisnosti o broju dretvi/monitora u sustavu

Jedna od ideja rada bila je upravo smanjenje veličine jezgre kako bi se ona mogla upotrijebiti na uređajima s vrlo ograničenim spremničkim prostorom. Vidljivo je dakle da se koncept monitora može upotrijebiti u ugrađenim sustavima kao temelj izgradnje programskog rješenja koje značajno smanjuje potrebe za spremničkim prostorom.

7.4. Trajanje izvođenja jezgrinih funkcija

U idealnom slučaju procesor obavlja jednu instrukciju u jednom ciklusu. Međutim, zbog instrukcija grananja i zavisnosti susjednih instrukcija u stvarnim programima dolazi do odlaganja izvođenja instrukcija u trodijelnom instrukcijskom cjevovodu kojeg koristi ARM7. Zbog toga se trajanje izvođenja produljuje i broj instrukcija se ne može uzeti kao mjerilo trajanja. Brzina izvođenja funkcija za rad s monitorima kao i dijela za prihvat i obradu prekida stoga je najlakše izmjerljiva koristeći brojila koja broje u sinkronizaciji s taktom procesora. Budući je korišten simulator ARM7 procesora, jezgrine su funkcije mogle biti analizirane u svakoj instrukciji te su vremena izvođenja očitavana izravno iz registara brojača vremena.

Trajanje pojedinih funkcija može biti različito u ovisnosti o stanju u kome se nalaze ostale dretve u sustavu. Tako, npr. trajanje poziva za ulaz u monitor biti će kraće ukoliko je monitor slobodan te nije potrebno dretvu postavljati u red, a neku drugu izvaditi.

Svaka je funkcija analizirana s obzirom na stanje u kojem se monitori mogu nalaziti i broj dretvi u pojedinim redovima. S obzirom da redovi nisu složeni, pretraživanje redova kreće od prve do posljednje dretve u redu.

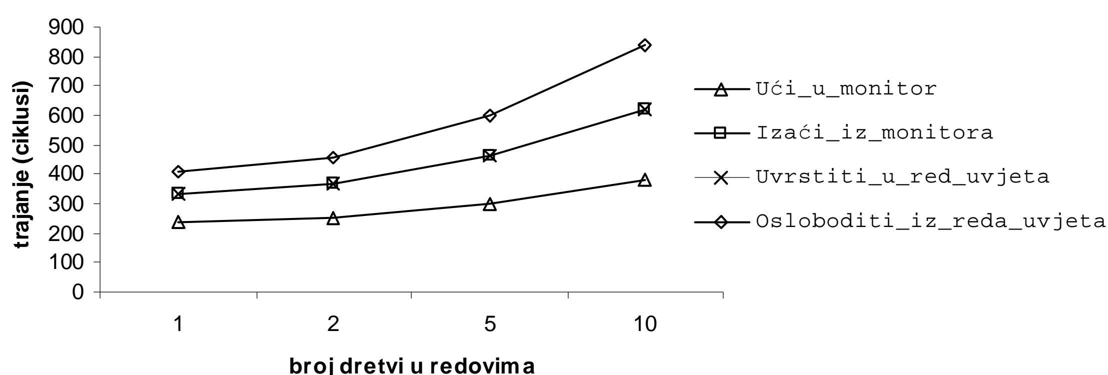
Tablica 7.3 prikazuje trajanje funkcija za rad s monitorima u jedinicama radnog takta procesora. Zadnja četiri stupca pokazuju ta vremena izračunata za 1, 2, 5 i 10 dretvi u dotičnim redovima. Ukoliko u redovima nema dretvi trajanje funkcija je značajno kraće jer se dio u kojem se red pretražuje preskače. Porastom broja dretvi u redovima trajanje izvođenja linearno raste. Funkcije koje rukuju s više redova uvjeta traju proporcionalno dulje. Trajanja se kreću od 149 ciklusa radnog takta procesora za funkciju ulaska u monitor do 410 za funkciju oslobađanja iz reda uvjeta ukoliko u redovima ima po jedna dretva. Za 10 dretvi u redovima vremena trajanja funkcija se za neke funkcije i udvostručuju. Međutim, 10 dretvi u svakom redu nije reprezentativni primjer za sustave koji se promatraju, u kojima je često i ukupan broj dretvi u sustavu manji od deset.

Tablica 7.3 Trajanje izvođenja funkcija

Funkcija	Trajanje u ciklusima procesora	*1	*2	*5	*10
`Ući_u_monitor`					
monitor slobodan	149	149	149	149	149
monitor zauzet	220 + 16 × broj_dretvi_u_redu_pripravnih	236	252	300	380
`Izaći_iz_monitora`					
red monitora prazan	252 + 16 × broj_dretvi_u_redu_pripravnih	268	284	332	412
postoje dretve u redu monitora	302 + 16 × broj_dretvi_u_redu_monitora + 16 × broj_dretvi_u_redu_pripravnih	334	366	462	622
`Uvrstiti_u_red_uvjeta`					
	303 + 16 × broj_dretvi_u_redu_monitora + 16 × broj_dretvi_u_redu_pripravnih	335	367	463	623
`Osloboditi_iz_reda_uvjeta`					
red uvjeta prazan	321 + 16 × broj_dretvi_u_redu_monitora + 16 × broj_dretvi_u_redu_pripravnih	353	385	481	641
postoje dretve u redu uvjeta	362 + 16 × broj_dretvi_u_redu_uvjeta + 16 × broj_dretvi_u_redu_monitora + 16 × broj_dretvi_u_redu_pripravnih	410	458	602	842

* označava broj dretvi u odgovarajućem redu

Slika 7.3 prikazuje trajanja funkcija za slučajeve različitog broja dretvi u redovima. Iz slike je vidljivo da se tek za pet i više dretvi u redovima trajanje funkcija primjetno produlji.



Slika 7.3 Trajanja jezgrinih funkcija

Navedena vremena ne uključuju funkcije za ostvarivanje protokola nasljeđivanja prioriteta ili protokola vršnih prioriteta. Uključivanjem tih funkcija trajanje se nešto produljuje. Protokol vršnih prioriteta koristi funkciju `povećati_prioritet()` za povećavanje prioriteta te funkciju `smanjiti_prioritet()` za smanjivanje prioriteta. Protokol nasljeđivanja prioriteta koristi funkciju `nasljedni_prioriteti_povećati()` za povećavanje prioriteta te istu funkciju `smanjiti_prioritet()` pri smanjivanju prioriteta. Tablica 7.4 sadrži vremena koja su potrebna za izvođenje navedenih funkcija.

Tablica 7.4 Trajanja funkcija za protokole rješavanja inverzija prioriteta

Funkcija	Trajanje u ciklusima procesora
`povećati_prioritet`	35 - kada se prioritet ne mijenja, 49 - kad se mijenja prioritet
`smanjiti_prioritet`	45 - kada se prioritet ne mijenja, +31 po smanjivanju, tj. ukupno 76 za jedno smanjivanje
`nasljedni_prioriteti_povećati`	51 - kada se prioritet ne mijenja, +21 za svako tranzitivno povećavanje, ukupno 72 za jedno povećavanje

Navedene vrijednosti treba dodati odgovarajućim jezgrinim funkcijama koje ih pozivaju, a koje su navedene u potpoglavlju 4.2. Iz tablice je vidljivo da su trajanja tih funkcija mnogo manja od trajanja početnih jezgrinih funkcija (Tablica 7.3) te se njihovim dodavanjem neće primjetno usporiti njihov rad osim u slučaju mnogostrukog povećavanja ili smanjivanja prioriteta.

Prihvat prekida sata korištenjem dodatne dretve, kako je prikazano u potpoglavlju 6.2 i odjeljku 7.2.7 traje oko 190 ciklusa procesora. Sličan vremena treba očekivati i za ostale prekide koji bi se prihvaćali na sličan način. Prekidi koji ne uzrokuju promjene stanja dretvi u sustavu mogu se prihvatiti i početi obrađivati puno prije, sa svega 30-tak ciklusa procesora kašnjenja koliko je potrebno da se odredi uzrok prekida.

Iako je brzina izvođenja bila jedna od prioriteta pri pisanju kôda, uz veličinu i jednostavnost kôda, vjerojatno se on još može optimirati. Jedna od ideja daljnje optimizacije bila bi korištenje registara za prijenos podataka u jezgrine funkcije, a ne stoga. Također bi se mogao izbaciti mehanizam povratka vrijednosti jer povratna vrijednost za ove funkcije zapravo i nije potrebna. Ona bi bila potrebna u slučaju kada bi se u funkcije dodao kôd koji provjerava ispravnost parametara te je zato i korištena iako provjera nije ugrađena. Druga optimizacija mogla bi biti u optimalnijem korištenju registara, bez suvišnih pohranjivanja podataka na stog. Optimizacija kôda s obzirom na protočnu strukturu procesora također bi mogla dati povoljne rezultate.

7.5. Sklopovska podrška ostvarenju monitora

Razmatranjem ostvarenja monitora za ARM7TDMI arhitekturu uviđa se da su postojeći mehanizmi dovoljni za ostvarenje svih funkcija. Ostaje međutim pitanje učinkovitosti te kako bi se ona mogla popraviti nekim izmjenama u arhitekturi.

Nedostatak koji je uočen u ARM7TDMI arhitekturi je nemogućnost zabrane prekidanja iz korisničkog načina rada. To je zapravo prednost ukoliko bi se dotični procesor koristio u svrhu izgradnje operacijskog sustava općenite namjene. Kod ugrađenih sustava, međutim, to postaje nedostatak jer se za zabranu prekida mora koristiti programski prekid, iz kojeg se onda može promijeniti statusni registar i time zabraniti prekide. Programi koji se izvode na ugrađenim sustavima trebali bi biti temeljito provjereni te ne bi trebala postojati opasnost od neprimjerenog korištenja zabrane prekida. Rješenje problema moglo bi biti u korištenju nadglednog rada procesora (npr. sustavski način rada) i za obavljanje poslova dretvi. Tada bi se i jezgrine funkcije mogle izravno pozvati korištenjem instrukcija za grananje bez korištenja programskih prekida. U slučaju potrebe ugradnje

složenijih funkcija (npr. složenija raspoređivanja dretvi) odabir nadglednog načina rada za dretve mnogo je bolji jer se zabrana prekida, koja je u takvim slučajevima često potrebna, mnogo brže ostvaruje.

Ukoliko bi se procesor proširio dodatnim skrivenim registrima koji se mogu koristiti isključivo u prekidnom načinu rada, tada bi se spremanje konteksta tekuće dretve moglo značajno reducirati. Pogotovo stoga što često dretva koja poziva funkcije za rad s monitorima bude i dalje aktivna nakon završetka poziva. Operacije koje se obavljaju unutar tih funkcija nisu složene te bi bile dovoljne i samo najosnovnije operacije za rad s tim dodatnim registrima, ako bi to pridonijelo pojednostavljenju sklopovlja. ARM7 za prekidne načine rada (osim za brzi prekidni način rada) ima na raspolaganju samo dva posebna registra koji i tako već imaju pridijeljenu ulogu: registar stoga za taj način rada i registar s vrijednošću povratne adrese nakon obrade prekida.

S obzirom na zauzeće spremničkog prostora za kôd jezgrinih funkcija možda bi procesori sa složenijim instrukcijama bili bolji izbor. Svojim bi složenim instrukcijama mogli pridonijeti smanjenju ukupnog broja instrukcija potrebnih za obavljanje određenih funkcija i tako smanjiti ukupnu veličinu jezgre. U sustavima s mikroprogramirljivom procesorskom jezgrom moglo bi se čak ići i do toga da se cijela funkcija sinkronizacije ugradi u jednu instrukciju.

Što se tiče korištenja navedenih funkcija na višeprocesorskim sustavima s zajedničkim dijeljenim spremničkim prostorom bilo bi potrebno osigurati da se jezgrine funkcije zaštite od istovremenog izvođenja na različitim procesorima. To znači da se prije početka izvođenja jezgrine funkcije mora osigurati isključivi pristup jezgri. U takvim se sustavima to može napraviti s instrukcijama *pročitaj_i_postavi* (engl. *Test And Set* – TAS) koje u dva uzastopna sabirnička ciklusa najprije dohvaćaju vrijednost s navedene spremničke lokacije te potom na tu lokaciju postavljaju odgovarajuću vrijednost. Na ovaj se način ulazi i u jezgre standardnih operacijskih sustava koji se koriste na višeprocesorskim računalima. U tom je slučaju potrebno dodati sljedeći dio kôda na početak svake jezgrine funkcije [Bud03]:

```
TAS OGRADA_JEZGRE
dok je (OGRADA_JEZGRE ≠ 0) {
   TAS OGRADA_JEZGRE
}
```

S obzirom da bi tada bilo i više aktivnih dretvi bilo bi potrebno i ponešto promijeniti pojedine dijelove jezgrinih funkcija i strukturu podataka. Procesore bi trebalo na neki način numerirati ili korištenjem sklopovskih mogućnosti ili programskim rješenjima zbog spremanja i obnavljanja konteksta dretvi prije i poslije jezgrinih funkcija.

U raspodijeljenom sustavu sinkronizaciju ulaska u jezgru trebalo bi obaviti korištenjem drugih komunikacijskih mehanizama (najčešće su to poruke). Osim problema kod ulaska u jezgru, pojavljuje se i problem smještaja podataka jezgre koji se mogu nalaziti u jednom čvoru ili mogu biti raspodijeljeni po svim čvorovima sustava. Promjenu tih podataka u jednom čvoru potrebno je odgovarajućim protokolima objaviti i ostalim čvorovima. Trajanje jezgrinih funkcija se u raspodijeljenim sustavima zbog toga znatno produljuje.

8. UPORABA MONITORA

Za prikaz korištenja monitora odabrana su dva primjera različite složenosti. Prvi primjer može se koristiti u sustavu upravljanja ugrađenim ručnim uređajem, npr. multimedijalni uređaj, telefon i slično. Složeniji primjer opisuje moguću uporabu monitora u ostvarenju sustava upravljanja i nadzora skupa objekata koji se kreću kroz područje u kome se očekuju povremena pogoršanja uvjeta rada.

8.1. Ručni uređaj

8.1.1. Opis sustava

Sustav koji se razmatra sastoji se od glavnog posla i nekoliko sporednih poslova. Pri pojavi glavnog posla sve ostale poslove treba zaustaviti i omogućiti glavnome nesmetan rad. Ukoliko glavni posao nije aktivan, ostali poslovi mogu raditi i istovremeno. U nastavku je zbog jednostavnosti pretpostavljeno da se sustav sastoji od samo dva sporedna posla.

8.1.2. Ostvarenje sustava

Sustav se sastoji tri dretve: dvije za sporedne poslove i treća za glavni posao. Prve dvije dretve obavljaju po strukturi sličan posao. U početku te dretve čekaju na signal za početak rada svog posla. Nakon što dretve prime signal one svoj posao obavljaju u dijelovima, obrađujući dio po dio u pravilnim vremenskim razmacima. Ove dretve obavljaju posao koji nije primarna namjena uređaja, npr. na mobilnom telefonu to bi mogao biti multimedijalni program za slušanje glazbe. Svojstvo takvih poslova jest da se obavljaju kroz duže vremensko razdoblje, zahtijevajući procesor samo povremeno, najčešće periodički. U slučaju glazbe, ulazni niz podataka treba po dijelovima dekodirati i proslijediti zvučnom podsustavu i to u pravilnim vremenskim razmacima.

Algoritmi koji predstavljaju obavljanje navedenih dviju dretvi mogu se opisati sljedećim recima:

```
Dretva posao_N() {
   ponavljati {
      čekati na signal za početak

      inicijalizacija posla N

      ponavljati {
         obaviti dio posla N
         zakasniti (ponavljanje[N])
      } do kraja posla N
```

```
      završetak posla N

   } do zauvijek
}
```

Razlika među dretvama iskazana je jedino u brojci `N`. Međutim, za svaku dretvu dio označen s `inicijalizacija posla N`, `obaviti dio posla N` i `završetak posla N` najčešće je potpuno različit.

Glavna dretva u početku čeka na signal za početak rada. Signal joj se može poslati iz obrade prekida kada se pojavi zahtjev za obradu hitnog posla. Nakon što obavi svoj posao dretva ponovo ulazi u stanje čekanja na sljedeći zahtjev.

```
Dretva hitno() {
   ponavljati {
      čekati na signal za početak
      obaviti hitan posao
   } do zauvijek
}
```

Neka se uvedu dodatne pretpostavke na rad sustava. Posao koji obavlja glavna dretva neka ne treba procesor tijekom obavljanja hitne radnje već je dovoljno prije pokretanja obrade obaviti određene predradnje, npr. podešavanje upravljačkih međusklopova. Ipak, za vrijeme trajanja hitnog posla sve ostale dretve u sustavu trebaju zaustaviti svoje izvođenje. To zaustavljanje ne mora biti naglo, npr. jednostavnim oduzimanjem procesora na vrijeme trajanja obrade, već može biti ostvareno i kroz kraći vremenski period. Npr. ukoliko se pojavi telefonski poziv za vrijeme slušanja glazbe moguće je kroz sljedeću sekundu postupno utišati glazbu i tek onda stati s reprodukcijom. Nakon što hitan posao završi tim istim dretvama treba signalizirati da mogu nastaviti s radom.

Detaljniji opisi rada dretvi koji uključuje navedene dijelove prikazani su u nastavku:

```
Dretva posao_N() {
   ponavljati {
      čekati na signal za početak

      inicijalizacija posla N

      ponavljati {
         ako je (hitan_posao = AKTIVAN) {
            zaustaviti izvođenje posla N
            pričekati završetak hitnog posla
            obnoviti izvođenje posla N
         } inače {
            obaviti dio posla N
            zakasniti (ponavljanje[N])
         }
      } do kraja posla N

      završetak posla N
   } do zauvijek
}
```

```
Dretva hitno() {
   ponavljati {
      čekati na signal za početak
      objaviti pojavu hitnog posla
      poduzeti akcije za početak rada hitnog posla
      //obaviti hitan posao
      pričekati završetak hitnog posla
      poduzeti akcije za kraj rada hitnog posla
      objaviti kraj rada hitnog posla
      omogućiti nastavak rada ostalim poslovima
   } do zauvijek
}
```

Za sinkronizaciju dretvi korištenjem monitora potreban je jedan monitora s dva reda uvjeta za svaku od dretvi u sustavu, osim hitne dretva za koju je dovoljan jedan red uvjeta. Jedno ostvarenje gornjih dretvi korištenjem monitora prikazano je u nastavku.

```
Dretva posao_N() {
   Ući_u_monitor(M)
   ponavljati {
      Uvrstiti_u_red_uvjeta(M, id_dretva_N)
      inicijalizacija posla N
      //posao može uključivati kašnjenje i čitanje vanjskih događaja
      ponavljati {
         ako je (hitan_posao = AKTIVAN ) {                        **
            zaustaviti izvođenje posla N                           **
            Uvrstiti_u_red_uvjeta(M, id_dretva_N_pauza)            **
            obnoviti izvođenje posla N                             **
         }                                                         **
         inače {                                                   **
            Izaći_iz_monitora(M)
            napraviti periodički posao dretve N
            Zakasniti (ponavljanje[N])
            Ući_u_monitor(M)
         }
      } do kraja posla N
      završetak posla N
   } do zauvijek
   Izaći_iz_monitora(M)
}
```

Čekanje na signal za početak rada, kao i čekanje na završetak obrade hitnog posla, ostvaruje se smještanjem dretve u red uvjeta monitora. Provjera stanja sustava obavlja se unutar monitora dok se sama obrada posla obavlja izvan. Na taj je način osigurano da su stanja varijabli sustava (`hitan_posao`) konzistentna i ne mijenjaju se dok ih dretva provjerava.

```
Dretva hitno() {
   Ući_u_monitor(M)
   ponavljati {
      Uvrstiti_u_red_uvjeta(M, id_dretva_3) //čekati na događaj
```

```
      hitan_posao = AKTIVAN                                              **
      poduzeti akcije za početak rada hitnog posla
      //obavljati posao akcije //ne mora se zahtijevati procesor        ##
      Uvrstiti_u_red_uvjeta(M, id_dretva_3)//čekati na završetak posla **
      hitan_posao = NEAKTIVAN
      poduzeti akcije za kraj rada hitnog posla
      Osloboditi_iz_reda_uvjeta(M, id_dretva1_pauza)                    **
      Osloboditi_iz_reda_uvjeta(M, id_dretva2_pauza)                    **
   } do zauvijek
   Izaći_iz_monitora(M)
}
```

U slučaju da hitan posao zahtjeva procesor tijekom svog izvođenja tada se redci koji završavaju zvjezdicama (**) izbacuju, a redak označen sa ## predstavlja hitan posao. Dovoljno je da dretva koja obavlja hitan posao nakon ulaska u monitor isti ne napušta do završetka posla. Ostale će dretve tada čekati na ulazu u monitor prije obavljanja svog posla.

Ukoliko hitni posao ne smije početi s izvođenjem dok ostale dretve ne stanu s izvođenjem potrebno je dodati nekoliko poziva za rad s monitorima. Tj. nakon što se označi da se hitan posao pojavio u sustavu postavljanjem varijable `dretva_hitno`, dretva `posao_hitno()` ulazi u red uvjeta te joj ostale dretve prije nego što uđu u red uvjeta `id_dretva_N_pauza` signaliziraju svoje zaustavljanje.

Dretva koja se pokreće pri pojavi vanjskih događaja nazvana je `upravljanje`. Ona utvrđuje uzrok prekida te oslobađa pojedine dretve iz reda uvjeta.

```
Dretva upravljanje() {
   Ući_u_monitor(M_IRQ)
   ponavljati {
      //čekati na neku funkcijsku tipku
      Uvrstiti_u_red_uvjeta(M_IRQ, id_upravljanje)

      Ući_u_monitor(M)
      pročitati status
      ako je (uzrok prekida = zahtjev za hitnim poslom) ∨
                        (uzrok prekida = kraj hitnog posla) {
         Osloboditi_iz_reda_uvjeta(M, id_dretva_3)
      }
      inače ako je (pritisnuta tipka = POKRENI_DRETVA_1) {
         Osloboditi_iz_reda_uvjeta(M, id_dretva_1)
      }
      inače ako je (pritisnuta tipka = POKRENI_DRETVA_2) {
         Osloboditi_iz_reda_uvjeta(M, id_dretva_s2)
      }
      Izaći_iz_monitora(M)
   }
   Izaći_iz_monitora(M_IRQ)
}
```

Dretva se početno nalazi u redu uvjeta monitora `M_IRQ`. Isti monitor koristi i dretva za

kašnjenje te eventualno ostale dretve koje bi se koristile za prihvat prekida.

Prihvat prekida potrebno je ostvariti kao i prihvat prekida sata kako je prikazano u potpoglavlju 6.2 tj. korištenjem dodatne dretve. Ukupno se u sustavu nalaze dretve: `posao_N` ($N \in \{1,2\}$), `posao_hitno`, dretva `upravljanje`, dretva za upravljanja kašnjenjem te dodatna dretva za slučaj kada niti jedna od prethodnih nije spremna za izvođenje.

Način ostvarenja navedenog primjera u strojnom jeziku ARM7 procesora prikazano je za prvu dretvu. Za razliku od opisnog jezika strojni je nešto dulji ali ga dosljedno prati.

```
; Posao dretve 1
; Ući_u_monitor(M)
      MOV   r0, #Monitor_1
      STMFD sp!, {r0}
      SWI   Uci_u_monitor
      LDMFD sp!, {r0}

Posao_1
; Uvrstiti_u_red_uvjeta(M, id_dretva_N)
      MOV   r0, #Monitor_1
      MOV   r1, #id_dretva_1
      STMFD sp!, {r1}
      STMFD sp!, {r0}
      SWI   Uvrstiti_u_red_uvjeta
      LDMFD sp!, {r0}

; inicijalizacija posla N
      BL    inicijalizacija_posla_1

Posao_1_ponavljati
; ako je (hitan_posao = AKTIVAN)
      LDR   r0, =hitan_posao
      LDR   r0, [r0]
      CMP   r0, #AKTIVAN
      BNE   Posao_1_dio_posla

      BL    zaustaviti_izvodjenje_posla_1
; Uvrstiti_u_red_uvjeta(M, id_dretva_N_pauza)
      MOV   r0, #Monitor_1
      MOV   r1, #id_dretva_1_pauza
      STMFD sp!, {r1}
      STMFD sp!, {r0}
      SWI   Uvrstiti_u_red_uvjeta
      LDMFD sp!, {r3}
      BL    obnoviti_izvodjenje_posla_1
      B     Posao_1_ponavljati

Posao_1_dio_posla
; Izaći_iz_monitora(M)
```

```
      MOV   r0, #Monitor_1
      STMFD sp!, {r0}
      SWI   Izaci_iz_monitora
      LDMFD sp!, {r0}

      BL    napraviti_periodicki_posao_dretve_1

; Zakasniti (ponavljanje[N])
      LDR   r0, =ponavljanje_1
      LDR   r0, [r0]
      STMFD sp!, {r0}
      BL    Zakasniti

; Ući_u_monitor(M)
      MOV   r0, #Monitor_1
      STMFD sp!, {r0}
      SWI   Uci_u_monitor
      LDMFD sp!, {r0}

; do kraja posla N
      BL    provjera_kraja_posla_1  ;povratna vrijednost je u registru r0
      CMP   r0, #KRAJ_POSLA
      BNE   Posao_1_ponavljati

; završetak posla N
      BL    zavrsetak_posla_1

      B     Posao_1

; Izaći_iz_monitora(M)
      MOV   r0, #Monitor_1
      STMFD sp!, {r0}
      SWI   Izaci_iz_monitora
      LDMFD sp!, {r0}
```

Sve korištene simboličke oznake varijabli i konstanti u gornjem kôdu trebaju biti prethodno definirane. Specifične radnje dretve obavljaju se u funkcijama koje se ovdje samo pozivaju, ali nisu ostvarene jer nisu definirane.

8.2. Upravljanje raspodijeljenim objektima

8.2.1. Opis sustava

Sustav koji se u ovom potpoglavlju razmatra sastoji se od središnjeg mjesta s kojeg se nadzire položaj i stanje određenog skupa pokretnih objekata. Objekti se kreću po području u kome obavljaju zadane poslove. Objekti međusobno kao i sa središtem komuniciraju

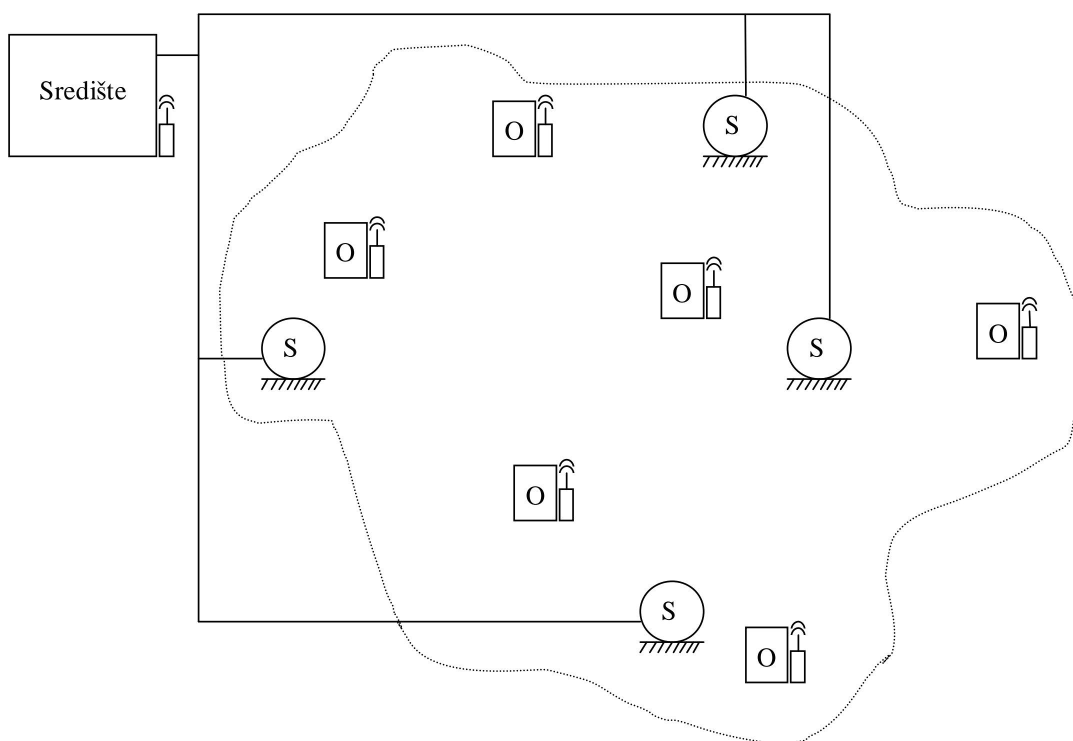
bežično koristeći kratke poruke.

Uvjeti u području u kojem objekti obavljaju poslove mogu se i promijeniti te su za nadzor uvjeta u tom području raspoređeni i dodatni senzori. Očitavanje vrijednosti senzora obavlja se sa središnjeg mjesta. Kada uvjeti u određenom području prijeđu dozvoljenu granicu objekti moraju privremeno napustiti područje. Pri povratku normalnog stanja objektima treba signalizirati povratak. U nekim slučajevima može biti i dovoljno samo pokrenuti dodatne uređaje kojima je zadaća normalizacija stanja, bez da objekti napuštaju to područje.

Zbog specifičnosti područja može se dogoditi i prekid veze između središnjeg mjesta i određenog senzora. U tom je slučaju potrebno prikazati upozorenje na zaslonu nadzornika koji odlučuje o potrebnoj akciji.

Središte periodički provjerava stanje objekata slanjem poruka. Ukoliko neki od objekata ne odgovori na zahtjev moguće je da se objekt nalazi van dometa središta ili da je u kvaru. U tom se slučaju s problematičnim objektom pokušava komunicirati posredno, koristeći ostale objekte. Ako se ni tada objekt ne javi potrebno je prikazati odgovarajuće upozorenje na zaslonu nadzornika sustava.

Nadzorniku treba omogućiti slanje različitih poruka objektima. Sličnu funkcionalnost treba omogućiti i objektima. Osim upravljačkih poruka objektima se može omogućiti i međusobno slanje proizvoljnih poruka.



Slika 8.1 Skica sustava

Grafički prikaz sustava prikazan na slici 8.1 uključuje središte, senzore označene slovom S koji su sa središtem povezani fizičkim vezama te objekte označene slovom O koji koriste bežičnu komunikaciju.

8.2.2. Ostvarenje upravljanja

8.2.2.1. Upravljanje središtem

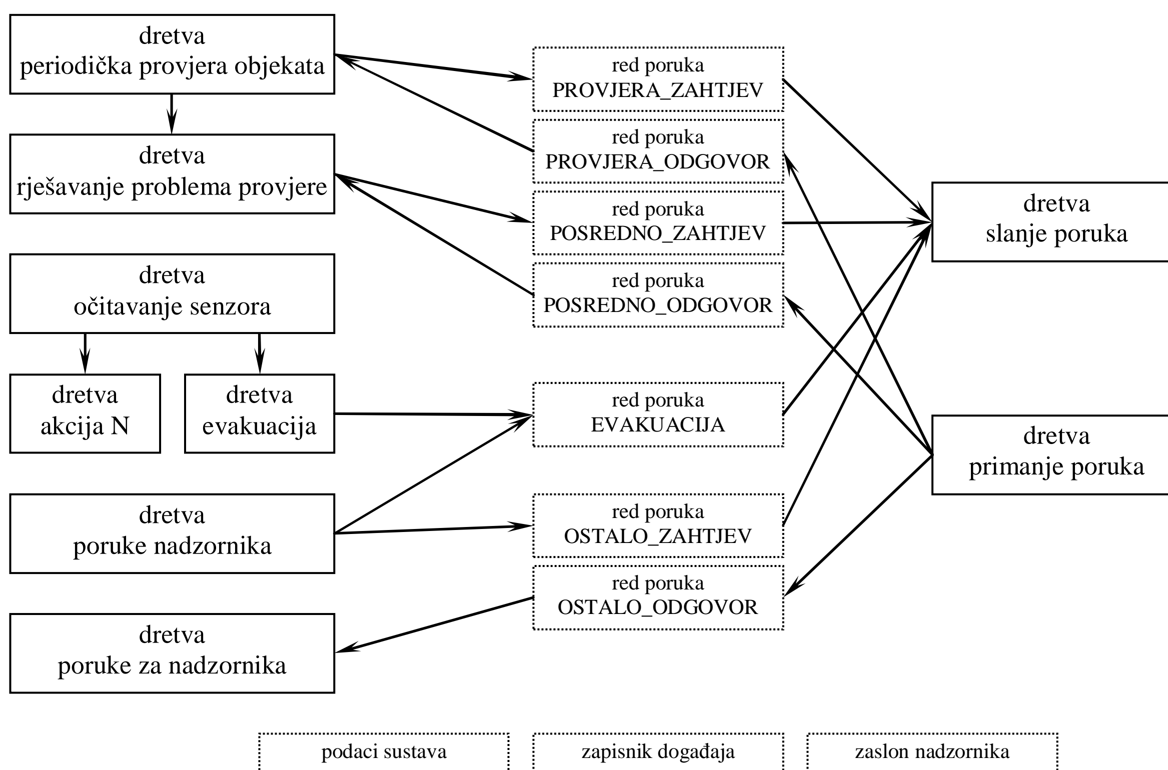
Periodičko provjeravanje objekata ostvareno je s dvije dretve. Prva dretva periodički šalje poruke svim objektima te potom prima odgovore. U slučaju da se neki objekt ne odazove aktivira se druga dretva koja pokušava posredno komunicirati s tim objektom. Ako se ni tada objekti ne jave uključuje se odgovarajuće upozorenje na zaslonu nadzornika.

Očitavanje senzora obavlja se u posebnoj dretvi. Ukoliko se neki od senzora ne mogu očitati prikazuje se odgovarajuće upozorenje. U slučaju da očitana vrijednost ukazuje na pogoršanje uvjeta u pripadajućem području tada treba pokrenuti procese koji vode normalizaciji uvjeta, odnosno, aktivirati posebne dretve koje to rade i nadziru. Ako su uvjeti van granica radnog područja tada treba evakuirati dotično područje. Slanje poruka o evakuaciji obavlja posebna dretva.

Za prihvat i prosljeđivanje poruka nadzornika objektima koristi se jedna dretva dok se za prihvat i ispis poruka za nadzornika koristi druga.

Slanje i primanje poruka od središta ka objektima i obratno obavlja se korištenjem dviju dretvi koje za tu zadaću koriste odgovarajuće sklopovlje. Za primanje i prosljeđivanje poruka među dretvama koristi se nekoliko redova poruka.

Slika 8.2 prikazuje dretve i sredstva sustava te njihovu povezanost.



Slika 8.2 Dretve i redovi poruka sustava

Poslovi koje obavljaju pojedine dretve opisani su u nastavku. Dijelovi koji koriste dijeljena sredstva sustava i dijelovi koji obavljaju sinkronizaciju dretvi izdvojeni su u posebne funkcije. Na ovaj način opis poslova u opisnom jeziku je mnogo jednostavniji i manji te ne uključuje jezgrine funkcije sinkronizacije.

Poruke koje se među dretvama razmjenjuju sastoje se od nekoliko elemenata:

- `tip` – opis sadržaja: `PROVJERA_ZAHTJEV, PROVJERA_ODGOVOR, POSREDNO_ZAHTJEV, POSREDNO_ODGOVOR, EVAKUACIJA, OSTALO_ZAHTJEV, OSTALO_ODGOVOR,`
- `sadržaj` – informacija koju poruka nosi,
- `prima` – adresa odredišta – može biti fizička adresa nekog objekta ili središta i
- `šalje` – adresa pošiljatelja.

Dretva `periodička_provjera()` šalje poruke svim objektima te potom čita njihove odgovore.

```
Dretva periodička_provjera_objekata() {
   poruka.tip = PROVJERA_ZAHTJEV
   poruka.sadržaj = ZAHTJEV_PERIODIČKE_PROVJERE_OBJEKTA
   ponavljati {
      za i=1 do broj_objekata {
         poruka.prima = adresa(objekt[i].id)
         Poslati_poruku(RED_PROVJERA_ZAHTJEV, poruka)
      }
      Zakasniti(DOPUŠTENO_VRIJEME_ODGOVORA)
      dok je (ima_poruka_u_redu(RED_PROVJERA_ODGOVOR)) {
         primljeno = Primiti_poruku(RED_PROVJERA_ODGOVOR)
         ako je (primljeno.tip = PROVJERA_ODGOVOR) {
            prijava[primljeno.objekt_id] = PRIMLJEN_ODGOVOR
            zabilježiti(REDOVNA_PROVJERA, primljeno)
            ažurirati(objekt[i].id, primljeno)
      }  }
      za i=1 do broj_objekata {
         ako je (prijava[objekt[i].id] ≠ PRIMLJEN_ODGOVOR) {
            zabilježiti(NEUSPJEŠNA_PROVJERA, objekt[i].id)
            aktivirati_dretvu_za_rješavanje_problema_provjere()
      }  }
      Zakasniti(SLJEDEĆI_TRENUTAK_PROVJERE)
   } do zauvijek
}
```

U slučaju problema pokreće se dretva `rješavanje_problema_provjere()`. Iako se u gornjem tekstu aktiviranje dretve za rješavanje problema može i više puta uzastopce pozvati, ona se aktivira samo jednom jer ako je red uvjeta prazan funkcija `Osloboditi_iz_reda_monitora()` ne radi ništa.

```
m-funkcija aktivirati_dretvu_za_rješavanje_problema_provjere () {
   Ući_u_monitor(M_problem_provjere)
   Osloboditi_iz_reda_monitora(M_problem_provjere, 1)
   Izaći_iz_monitora(M_problem_provjere)
}
```

Za slanje i primanje poruka koriste se funkcije slične onima definiranim u potpoglavlju 6.5 uz neke izmjene i dodatnu funkciju `ima_poruka_u_redu()` definiranu u nastavku.

```
m-funkcija Poslati_poruku(red, P) {
   Ući_u_monitor(M_poruke_slanje)
   Poruke[red].čeka_na_slanje++
   dok je (Poruke[red].broj_praznih = 0) {
      Uvrstiti_u_red_uvjeta(M_poruke_slanje, red)
   }
   Poruke[red].čeka_na_slanje--
   Poruke[red].MS[Poruke[red].UL] = P
   Poruke[red].UL = (Poruke[red].UL + 1) mod Poruke[red].N
   Poruke[red].broj_praznih--
   Osloboditi_iz_reda_uvjeta(M_poruke_slanje, DRETVA_ŠALJE)
   Izaći_iz_monitora(M_poruke_slanje)
}
```

Sve poruke koje se šalju funkcijom `Poslati_poruku()` idu najprije prema dretvi `slanje_poruka()` te tek potom prema objektima.

```
m-funkcija Primiti_poruku(red) {
   Ući_u_monitor(M_poruke_primanje)
   Poruke[red].čeka_na_poruku++
   dok je (Poruke[red].broj_praznih = Poruke[red].N) {
      Uvrstiti_u_red_uvjeta(M_poruke_primanje, red)
   }
   Poruke[red].čeka_na_poruku--
   R = Poruke[red].MS[Poruke[red].IZ]
   Poruke[red].IZ =(Poruke[red].IZ + 1) mod Poruke[red].N
   Poruke[red].broj_praznih++
   ako je (Poruke[red].čeka_na_slanje > 0) {
      Osloboditi_iz_reda_uvjeta(M_poruke_primanje, DRETVA_PRIMA)
   }
   Izaći_iz_monitora(M_poruke_primanje)
   vratiti(R)
}
```

Poruke koje se čitaju funkcijom `Primiti_poruku()` dolaze od objekata i najprije ih čita dretva `primanje_poruka()` koja ih prosljeđuje u odgovarajuće redove.

```
m-funkcija ima_poruka_u_redu(red) {
   ima_poruka = 1
   Ući_u_monitor(M_poruke_primanje)
   ako je (Poruke[red].broj_praznih = Poruke[red].N){
      ima_poruka = 0
   }
   Izaći_iz_monitora(M_poruke_primanje)
   vratiti(ima_poruka)
}
```

Funkcije `zabilježiti()` i `ažurirati()` pristupaju zajedničkom spremničkom prostoru te zapisuju događaje i stanje sustava. Istovremeni pristup onemogućen je korištenjem monitora kao sredstva međusobnog isključivanja.

```
m-funkcija zabilježiti(događaj, opis) {
   Ući_u_monitor(M_zapisnik)
   zapisati(događaj, opis)
   Izaći_iz_monitora(M_zapisnik)
}

m-funkcija ažurirati(objekt_id, poruka) {
   Ući_u_monitor(M_ažuriranje)
   ažuriranje(objekt_id, poruka)
   Izaći_iz_monitora(M_ažuriranje)
}
```

Kašnjenje je ostvareno na način definiran u potpoglavlju 6.2, odnosno poziva se funkcija `Zakasniti()`.

Dretva za rješavanje problema provjere slična je prethodnoj dretvi, tj. šalje poruke i čeka na odgovore. Međutim, ova dretva nije periodička već se aktivira na zahtjev prethodne dretve.

```
Dretva rješavanje_problema_provjere () {
   poruka.tip = POSREDNA_PROVJERA
   ponavljati {
      čekati_na_pojavu_problema_provjere()

      za i=1 do broj_objekata {
         ako je (prijava[objekt[i].id] ≠ PRIMLJEN_ODGOVOR) {
            za j=1 do broj_objekata {
               poruka.prima = adresa(objekt[j].id)
               poruka.sadržaj = zahtjev_provjere_objekta(objekt[i].id)
               Poslati_poruku(RED_POSREDNO_ZAHTJEV, poruka)
      }  }  }

      pričekati(DOPUŠTENO_VRIJEME_POSREDNOG_ODGOVORA)

      dok je (ima_poruka_u_redu(RED_POSREDNO_ODGOVOR)) {
         primljeno = Primiti_poruku(RED_POSREDNO_ODGOVOR)
         id = id_dretve(primljeno.sadržaj)
         prijava[id]=PRIMLJEN_ODGOVOR
         zabilježiti(PONOVLJENA_PROVJERA, primljeno)
         ažurirati(id, primljeno)
      }
      za i=1 do broj_objekata {
         ako je (prijava[objekt[i].id] ≠ PRIMLJEN_ODGOVOR) {
            zabilježiti(NEUSPJEŠNA_PONOVLJENA_PROVJERA, objekt[i].id)
            aktivirati_upozorenje(NEUSPJEŠNA_PROVJERA, objekt[i].id)
      }  }
   } do zauvijek
}
```

Aktiviranje dretve zbiva se u funkciji `čekati_na_pojavu_problema_provjere()`.

```
m-funkcija čekati_na_pojavu_problema_provjere () {
   Ući_u_monitor(M_problem_provjere)
   Uvrstiti_u_red_uvjeta(M_problem_provjere, 1)
   Izaći_iz_monitora(M_problem_provjere)
}
```

Pomoćne funkcije `zahtjev_provjere_objekta()` i `id_dretve()` oblikuju sadržaj poruke te iz njega čitaju određene podatke.

Periodičko očitavanje senzora obavlja se dretvom `očitavanje_senzora()`.

```
Dretva očitavanje_senzora () {
   ponavljati {
      za i=1 do broj_senzora {
         stanje = očitati_senzor(senzor[i].id)
         ako je (stanje.očitanje = NEUSPJEŠNO) {
            zabilježiti(OČITANJE_SENZORA_NEUSPJEŠNO, senzor[i].id)
            aktivirati_upozorenje(PROBLEM_SENZORA, senzor[i].id)
         }
         inače {
            status = područje_rada(senzor[i].id, stanje)
            zabilježiti(OČITANJE_SENZORA, stanje)
            ako je (status = KRITIČNO_STANJE){
               aktivirati_upozorenje(KRITIČNO_STANJE, stanje)
               područje = područje_senzora(senzor[i].id)
               aktivirati_dretvu_za_evakuaciju_područja(područje, stanje)
               aktivirati_dretvu_za_akciju(stanje)
            }
            ako je (status = POKRENUTI_AKCIJU){
               aktivirati_upozorenje(POKRENUTI_AKCIJU, stanje)
               aktivirati_dretvu_za_akciju(stanje)
      }  }  }
      Zakasniti(SLJEDEĆI_TRENUTAK_PROVJERE_SENZORA)
   } do zauvijek
}
```

Funkcija `očitaj_senzor()` pristupa zadanom senzoru i očitava vrijednosti koje on mjeri. Kada su vrijednosti izvan dozvoljenih granica potrebno je pokrenuti odgovarajuće akcije, tj. aktivirati odgovarajuće dretve.

```
m-funkcija aktivirati_dretvu_za_evakuaciju_područja(područje, stanje) {
   Ući_u_monitor(M_problem_senzori)
   Osloboditi_iz_reda_monitora(M_problem_senzori, DRETVA_EVAKUACIJA)
   Izaći_iz_monitora(M_problem_senzori)
}
m-funkcija aktivirati_dretvu_za_akciju(stanje) {
   Ući_u_monitor(M_problem_senzori)
   dretva_id = odredi_dretvu_za_akciju(stanje)
   Osloboditi_iz_reda_monitora(M_problem_senzori, dretva_id)
   Izaći_iz_monitora(M_problem_senzori)
}
```

Dretva za evakuaciju šalje poruke svim objektima koji se nalaze u području evakuacije.

```
Dretva evakuacija (područje, stanje) {
   poruka.tip = EVAKUACIJA
   poruka.sadržaj = ZAHTJEV_ZA_EVAKUACIJU
   ponavljati {
      čekati_na_pojavu_signala_za_evakuaciju()
      za i=1 do broj_objekata {
         ako je (evakuacija_potrebna(objekt[i].id, područje, stanje)) {
            poruka.prima = adresa(objekt[i].id)
            Poslati_poruku(RED_EVAKUACIJA, poruka)
      }  }
   } do zauvijek
}
```

Čekanje na signal evakuacije identično je kao i kod dretve za rješavanje problema provjere objekata. Razlika je jedino u korištenim monitorima i redovima uvjeta.

```
m-funkcija čekati_na_pojavu_signala_za_evakuaciju() {
   Ući_u_monitor(M_problem_senzori)
   Uvrstiti_u_red_uvjeta(M_problem_senzori, DRETVA_EVAKUACIJA)
   Izaći_iz_monitora(M_problem_senzori)
}
```

Funkcijom `evakuacija_potrebna()` određuje se treba li zadani objekt napustiti područje. To može ovisiti o tome da li se objekt nalazi u tom području te da li trenutno stanje utječe na izvođenje njegova posla.

Akcije koje treba provesti u slučaju kada su očitanja senzora van dozvoljenog područja definiraju se u posebnim dretvama, u nastavku označenim varijablom `n`.

```
Dretva akcija(n) {
   ponavljati {
      čekati_na_pojavu_signala_za_akciju(n)
      provoditi_akciju_za_normalizaciju_stanja(n)
      novo_stanje = provjera_stanja(n)
      ako je (novo_stanje.status = STANJE_NORMALNO) {
         poruka.tip = STANJE_NORMALNO
         poruka.sadržaj = STANJE_NORMALNO
         za i=1 do broj_objekata {
            ako je (obavijest_potrebna(objekt[i].id, n)) {
               poruka.prima = adresa(objekt[i].id)
               Poslati_poruku(RED_EVAKUACIJA, poruka)
      }  }  }
      inače {
         aktivirati_upozorenje(STANJE_KRITIČNO, novo_stanje)
      }
   } do zauvijek
}
```

```
m-funkcija čekati_na_pojavu_signala_za_akciju(n) {
   Ući_u_monitor(M_problem_senzori)
   Uvrstiti_u_red_uvjeta(M_problem_senzori, n)
   Izaći_iz_monitora(M_problem_senzori)
}
```

Funkcija `provoditi_akciju_za_normalizaciju_stanja()` definira provođenje odgovarajuće akcije, `provjera_stanja()` očitava odgovarajuće senzore te `obavijest_potrebna()` provjerava za svaki objekt da li mu je potrebno slati obavijest o normalizaciji stanja.

Čitanje i slanje poruka nadzornika obavljaju dretve `poruke_nadzornika()` i `poruke_za_nadzornika()`.

```
Dretva poruke_nadzornika() {
   ponavljati {
      čekati_na_zahtjev_za_slanje_poruke()
      poruka = pročitati_poruku()
      Poslati_poruku(RED_OSTALO_ZAHTJEV, poruka)
   } do zauvijek
}

Dretva poruke_za_nadzornika () {
   ponavljati {
      poruka = Primiti_poruku(RED_OSTALO_ODGOVOR)
      ispisati_poruku(poruka)
   } do zauvijek
}
```

Funkcija `čekati_na_zahtjev_za_slanje_poruke()` mora čekati na vanjski zahtjev za novom porukom. Nakon pojave zahtjeva funkcija `pročitati_poruku()` s nekog vanjskog uređaja učitava sadržaj i adresu poruke. Funkcija `ispisati_poruku()` ispisuje primljene poruke na zaslon nadzornika.

Dretva za slanje poruka prema objektima čita poruke iz redova poruka te ih šalje preko odgovarajućeg sklopovlja prema objektima. Čitanje poruke iz više redova izvodi se u monitorskoj funkciji `uzeti_poruku_iz_redova()`. Ukoliko nema poruka dretva će se zaustaviti u redu uvjeta. Slanje poruka prema objektima izvodi se u funkciji `oblikovati_i_poslati_poruku()`.

```
Dretva slanje_poruka() {
   ponavljati {
      poruka=uzeti_poruku_iz_redova()
      oblikovati_i_poslati_poruku(poruka)
   }
}

m-funkcija uzeti_poruku_iz_redova() {
   redovi [] = { RED_EVAKUACIJA, RED_POSREDNO_ZAHTJEV,
                 RED_PROVJERA_ZAHTJEV, RED_OSTALO_ZAHTJEV }
   Ući_u_monitor(M_poruke_slanje)
```

```
   ponavljati {
      i = 1
      ima_poruka = 0
      ponavljati {
         red = redovi[i]
         ako je (Poruke[red].broj_praznih < Poruke[red].N) {
            poruka = Poruke[red].MS[Poruke[red].IZ]
            Poruke[red].IZ = (Poruke[red].IZ + 1) mod Poruke[red].N
            Poruke[red].broj_praznih++
            ako je (Poruke[red].čeka_na_slanje > 0) {
               Osloboditi_iz_reda_uvjeta(M_poruke_slanje, red)
            }
            ima_poruka = 1
         }
         i++
      } dok je (ima_poruka = 0) ∧ (i < 5)
      ako je (ima_poruka = 0){
         Uvrstiti_u_red_uvjeta(M_poruke_slanje, DRETVA_ŠALJE)
      }
   } dok je (ima_poruka = 0)
   Izaći_iz_monitora(M_poruke_slanje)
   Vratiti(poruka)
}
```

Redovima su pridijeljeni prioriteti da bi hitne poruke bile prve poslane. Nakon svakog slanja poruka ponovno se pretražuju svi redovi, počevši s najprioritetnijim.

Primanje poruka koje šalju objekti prema središtu te njihovo prosljeđivanje u odgovarajuće redove obavlja dretva `primanje_poruka()`.

```
Dretva primanje_poruka() {
   ponavljati {
      pričekati_pojavu_nove_poruke()
      primljeno = prihvatiti_poruku()
      poruka = oblikovati_poruku(primljeno)
      red = odredi_red(primljeno)
      staviti_poruku_u_red(red, poruka)
   } do zauvijek
}
```

Funkcije `pričekati_pojavu_nove_poruke()` i `prihvatiti_poruku()` koriste sklopovlje za čekanje na pojavu nove poruke i njeno prihvaćanje. Format primljene poruke ne mora odgovarati porukama koje se šalju unutar središta. Oblikovanje u zadani format obavlja funkcija `oblikovati_poruku()`, a određivanje ciljnog reda u koji poruku treba smjestiti funkcija `odredi_red()`. Monitorska funkcija `staviti_poruku_u_red()` obavlja sam postupak stavljanja poruke u red.

```
m-funkcija staviti_poruku_u_red(red, P) {
   Ući_u_monitor(M_poruke_primanje)
   Poruke[red].čeka_na_slanje = 1
```

```
   dok je (Poruke[red].broj_praznih = 0) {
      Uvrstiti_u_red_uvjeta(M_poruke_primanje, DRETVA_PRIMA)
   }
   Poruke[red].čeka_na_slanje = 0
   Poruke[red].MS[Poruke[red].UL] = P
   Poruke[red].UL = (Poruke[red].UL + 1) mod Poruke[red].N
   Poruke[red].broj_praznih--
   ako je (Poruke[red].čeka_na_slanje > 0) {
      Osloboditi_iz_reda_uvjeta(M_poruke_primanje, red)
   }
   Izaći_iz_monitora(M_poruke_primanje)
}
```

Prioritete dretvama treba pridijeliti prema važnosti posla koji obavljaju. Očito je evakuacija najkritičniji dio sustava, međutim, nije dovoljno samo dretvi `evakuacija()` pridijeliti najveći prioritet. Dretva `očitanje_senzora()` ustanovljava da je stanje kritično te i ona mora imati visoki prioritet. Isto tako poruke o evakuaciji treba što hitnije proslijediti objektima te i prioritet dretve za slanje poruka treba biti visok. U skupinu dretvi visokog prioriteta treba ubrojiti i dretve koje obavljaju određenu akciju na očitanje senzora. Ostale dretve sustava mogu imati manji prioritet.

Moguće pridjeljivanje prioriteta dretvama od najznačajnije do manje značajnih:

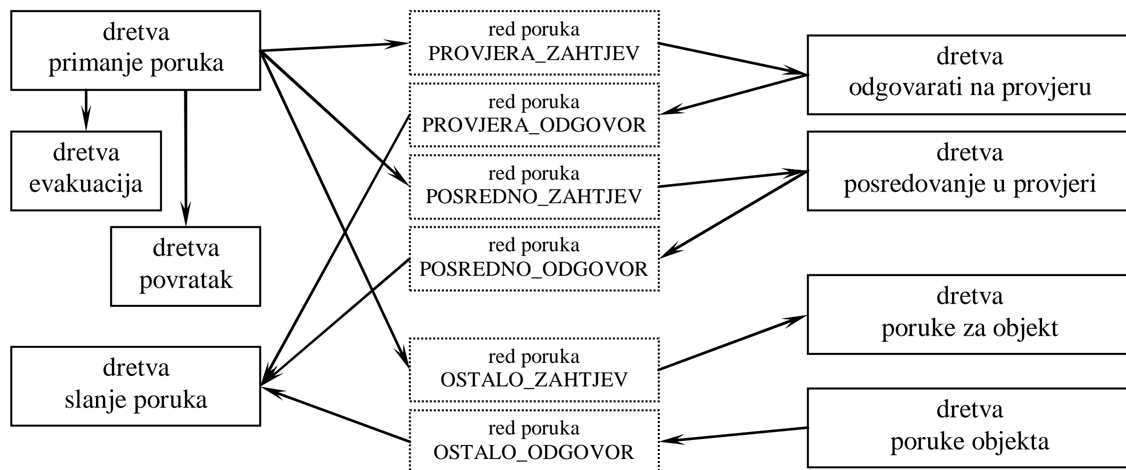
1. `evakuacija()`,
2. `očitavanje_senzora()`,
3. `slanje_poruka()`,
4. `akcija(n)`,
5. `primanje_poruka()`,
6. `rješavanje_problema_provjere()`,
7. `periodička_provjera_objekata()`,
8. `poruke_nadzornika()` i
9. `poruke_za_nadzornika()`.

Broj dretvi u sustavu ograničen je na deset do dvadeset, ovisno o broju dretvi za određene akcije. Tom broju dodaje se dretva za ostvarenje kašnjenja te eventualno još neka dretva koja prihvaća i obrađuje prekide na isti način kao što se i prekid kašnjenja prihvaća. Uz navedeni broj dretvi prostor za podatke i kôd jezgrinih funkcija, ako se oni ostvaruju za ARM7, može se izračunati prema tablicama potpoglavlja 7.3 te će on iznosit nešto više od 3000 okteta. Da bi se dobila ukupna potrebna veličina spremničkog prostora potrebno je dodati i kôd samih dretvi i upravljačke programe.

8.2.2.2. Upravljanje objektom

Za upravljanje objektom, primanje i odgovaranje na poruke koristi se osam dretvi i šest redova poruka, prema slici 8.3.

Primanje i slanje poruka slično je kao i kod središta. Ukoliko je primljena poruka o evakuaciji ili povratku s evakuacije ta se poruka ne prosljeđuje u red već se izravno aktiviraju dretve koje obavljaju zadane operacije.



Slika 8.3 Dretve i redovi poruke objekta

```
Dretva primanje_poruka() {
   ponavljati {
      pričekati_pojavu_nove_poruke()
      primljeno = prihvatiti_poruku()
      poruka = oblikovati_poruku(primljeno)
      ako je (poruka.tip = EVAKUACIJA) {
         ako je (poruka.sadržaj = ZAHTJEV_ZA_EVAKUACIJU) {
            aktivirati_dretvu_za_evakuaciju()
         }
         inače {
            aktivirati_dretvu_za_povratak()
         }
      }
      inače {
         red = odredi_red(primljeno)
         staviti_poruku_u_red(red, poruka)
      }
   } do zauvijek
}
```

Aktiviranje dretvi obavlja se na isti način kao i kod dretvi središta.

```
m-funkcija aktivirati_dretvu_za_evakuaciju() {
   Ući_u_monitor(M_evakuacija)
   Osloboditi_iz_reda_monitora(M_evakuacija, DRETVA_EVAKUACIJA)
   Izaći_iz_monitora(M_evakuacija)
}
m-funkcija aktivirati_dretvu_za_povratak() {
   Ući_u_monitor(M_evakuacija)
   Osloboditi_iz_reda_monitora(M_evakuacija, DRETVA_POVRATAK)
   Izaći_iz_monitora(M_evakuacija)
}
```

Dretve za evakuaciju i povratak nakon evakuacije obavljaju unaprijed definirane radnje. U najjednostavnijem je slučaju to signalizacija putem određenih žaruljica ili zvukova dok u složenijim slučajevima to može biti navođenje objekta do sigurne zone. U

nastavku se te radnje obavljaju u posebnim funkcijama.

```
Dretva evakuacija() {
   ponavljati {
      čekati_na_zahtjev_za_evakuaciju()
      poduzeti_odgovarajuće_akcije_za_evakuaciju()
   } do zauvijek
}
Dretva povratak() {
   ponavljati {
      čekati_na_zahtjev_za_povratak()
      poduzeti_odgovarajuće_akcije_za_povratak()
   } do zauvijek
}
```

Prethodne dretve na događaje čekaju u redovima uvjeta koristeći funkcije:

```
m-funkcija čekati_na_zahtjev_za_evakuaciju() {
   Ući_u_monitor(M_evakuacija)
   Uvrstiti_u_red_uvjeta(M_evakuacija, DRETVA_EVAKUACIJA)
   Izaći_iz_monitora(M_evakuacija)
}
m-funkcija čekati_na_zahtjev_za_povratak() {
   Ući_u_monitor(M_evakuacija)
   Uvrstiti_u_red_uvjeta(M_evakuacija, DRETVA_POVRATAK)
   Izaći_iz_monitora(M_evakuacija)
}
```

Monitorska funkcija `staviti_poruku_u_red()` identična je istoimenoj funkciji kao i kod središta.

Dretva `slanje_poruka()` koja čita poruke iz redova i šalje ih središtu i drugim objektima ista je kao i istoimena dretva kod središta. Razlika se nalazi jedino u redovima poruka koji se koriste u funkciji `uzeti_poruku_iz_redova()`, odnosno polje `redovi[]` treba zamijeniti sa:

```
   redovi [] = { RED_POSREDNO_ODGOVOR, RED_PROVJERA_ODGOVOR,
                 RED_OSTALO_ODGOVOR }
```

Čitanje poruka o zahtjevima periodičke provjere i odgovaranja na njih obavlja se u dretvi `odgovarati_na_provjeru()`.

```
Dretva odgovarati_na_provjeru() {
   ponavljati {
      poruka = Primiti_poruku(RED_PROVJERA_ZAHTJEV)
      odgovor = stanje_objekta()
      odgovor.prima = poruka.šalje
      Poslati_poruku(RED_PROVJERA_ODGOVOR, odgovor)
   } do zauvijek
}
```

Monitorske funkcije `Primiti_poruku()` i `Poslati_poruku()` identične su istoimenim funkcijama na središnjem računalu.

Dretva `posredovanje_u_provjeri()` prihvaća i obrađuje zahtjeve za posrednu

provjeru objekata. Poruka može stizati ili od središta ili od drugog objekta. Ako je odredišna adresa poruke upravo adresa objekta koji prima poruku na nju odgovara porukom u koju uključuje stanje objekta. Ukoliko je poruka odgovor na posrednu provjeru ta se poruka prosljeđuje središtu, inače se poruka prosljeđuje drugom objektu.

```
Dretva posredovanje_u_provjeri() {
   ponavljati {
      poruka = Primiti_poruku(RED_POSREDNO_ZAHTJEV)
      adresa_odredišta = odredi_adresu_odredišta(poruka.sadržaj)
      ako je (adresa_odredišta = ADRESA_OBJEKTA) {
         stanje = stanje_objekta()
         odgovor.sadržaj = oblikovati_odgovor(stanje, ADRESA_CENTRA)
         odgovor.prima = poruka.šalje
         odgovor.tip = POSREDNO_ODGOVOR
         Poslati_poruku(RED_POSREDNO_ODGOVOR, odgovor)
      }
      inače {// drugi objekt
         ako je (poruka.tip = POSREDNO_ODGOVOR){
            poruka.prima = ADRESA_CENTRA
            poruka.šalje = ADRESA_OBJEKTA
            Poslati_poruku(RED_POSREDNO_ODGOVOR, poruka)
         }
         inače { //proslijediti zahtjev prema „adresa_odredišta"
            poruka.prima = adresa_odredišta
            poruka.šalje = ADRESA_OBJEKTA
            Poslati_poruku(RED_POSREDNO_ODGOVOR, poruka)
      } }
   } do zauvijek
}
```

Slanje i primanje ostalih poruka ostvareno je isto kao i kod poruka nadzornika. Razlika jest jedino u imenima dretvi i redovima koje koriste za čitanje i slanje poruka.

```
Dretva poruke_objekta() {
   ponavljati {
      čekati_na_zahtjev_za_slanje_poruke()
      poruka = pročitati_poruku()
      Poslati_poruku(RED_OSTALO_ODGOVOR, poruka)
   } do zauvijek
}
Dretva poruke_za_objekt () {
   ponavljati {
      poruka = Primiti_poruku(RED_OSTALO_ZAHTJEV)
      ispisati_poruku(poruka)
   } do zauvijek
}
```

Prioriteti dretvi koje upravljaju objektom određuju se prema važnosti posla koji obavljaju. Proces evakuacije je najčešće najhitniji te bi dretva `evakuacija` trebala imati najveći prioritet. Po prioritetu bi trebala slijediti dretva `primanje_poruka()` koja prihvaća poruke od središta i drugih objekata zbog mogućnosti pojave poruke o evakuaciji. Prioriteti

ostalih dretvi mogu ovisiti o uporabi, ali bi najlogičnije bilo da dretve `povratak()` i `slanje_poruka()` slijede prije ostalih.

Prihvatljiv redoslijed prioriteta dretvi, od najznačajnije prema manje značajnim može biti:

1. `evakuacija()`,
2. `primanje_poruka()`,
3. `povratak()`,
4. `slanje_poruka()`,
5. `odgovarati_na_provjeru()`,
6. `posredovanje_u_provjeri()`,
7. `poruke_za_objekt()` i
8. `poruke_objekta()`.

Navedeni redoslijed ne mora značiti da dvije ili više susjednih dretvi ne mogu imati i isti prioritet. Međutim, isti prioritet nije poželjan jer pridonosi nedeterminizmu.

8.2.3. Mogućnosti proširenja

Prikazani način ostvarenja sustava dretvi središta i objekata treba smatrati kao temelje nad kojim je moguće ostvariti stvarni sustav. Veći broj funkcija u predloženom rješenju je nedefinirana jer njihov posao određuje stvarna uporaba. Neke od prikazanih funkcionalnosti negdje ne moraju biti potrebne, dok u drugim sustavima biti će potrebno dodati nove.

Na primjer, i senzori mogu sa središtem komunicirati bežično. Nadalje, moguće je da sami senzori i objekti tvore komunikacijsku mrežu kojom se prenose poruke, tj. komunikacija među objektima, senzorima i središnjim upravljačkim uređajem može se obavljati posredno, korištenjem te mreže koja tada postaje senzorska mreža. Moguće je i da sami senzori šalju upozoravajuće poruke ostalim objektima, odnosno, da se poruke o opasnosti šire zadanim područjem automatski [Li05]. Senzori mogu biti i pokretni, ugrađeni u objekte koji obavljaju zadani posao u promatranom području. Radi duljeg rada senzora neki se od njih mogu i povremeno isključivati ukoliko je trenutna pokrivenost njihova područja dostatna [Xin05]. Sve navedene dodatne funkcionalnosti se mogu ostvariti korištenjem osnovnih operacija: očitavanje vrijednosti senzora, slanje i primanje poruka među dretvama, sinkronizacija dretvi te slanje i primanje poruka među različitim objektima i senzorima. Sve ove operacije sadržane su u navedenom primjeru.

Iz prikazanog je rješenja vidljivo da se korištenjem monitora mogu ostvariti različiti sustavi, bez dodatnih mehanizama sinkronizacije i komunikacije. Složeniji oblici sinkronizacije i komunikacije mogu se ostvariti korištenjem monitora kako je to i prikazano u ovom primjeru.

9. ZAKLJUČAK

U radu je predložen monitor kao temelj izgradnje višedretvenog ugrađenog sustava. Među raznim modelima monitora odabran je najprikladniji za uporabu u ugrađenim sustavima te koji svojom strukturom smanjuje mogućnost pojave problema pri raspoređivanju kao što je to problem inverzije prioriteta.

Funkcije za ostvarenje odabranog modela monitora jednostavne su za izgradnju što je prikazano detaljnim opisom koji se može prepisati u neki programski jezik. Za ostvarenje monitora dovoljne su četiri funkcije. U funkcije je ugrađen prioritetni raspoređivač koji može biti dostatan u mnogim primjenama, dovoljno je dretvama pridijeliti odgovarajuće prioritete.

Nadogradnja monitora naprednim mogućnostima prikazana je pomoću dva protokola: protokol nasljeđivanja prioriteta i protokol vršnih prioriteta. Zadaća ovih protokola jest sprječavanje pojave inverzije prioriteta, odnosno, njeno rješavanje ukoliko se takva situacija dogodi.

Korištenje odabranog modela monitora prikazano je na nekoliko klasičnih problema sinkronizacije i komunikacije. Usporedbom rješenja tih problema s monitorima i sa semaforima vidljiva je prednost uporabe monitora u složenijim problemima. Štoviše, neki se problemi s klasičnim semaforima ne mogu niti riješiti.

Načini ostvarenja funkcija operacijskog sustava korištenjem monitora prikazani su na nekoliko primjera, od zaštite pri pristupu ulazno izlaznim napravama, ostvarivanja kašnjenja, složenijih metoda raspoređivanja dretvi, komunikacije porukama te ostvarivanje semafora.

Zasnivanje ugrađenih sustava prikazano je na dva primjera: ručnom uređaju i upravljanje raspodijeljenim objektima. Oba ostvarenja potvrđuju mogućnosti monitora kao osnovnog i jedinog potrebnog sinkronizacijskog mehanizma u sustavu.

Ostvarenjem predloženih funkcija u strojnom jeziku procesora ARM7 dobivena je kvantitativna ocjena o zauzeću spremničkog prostora i trajanju pojedinih funkcija. Uz prostor za podatke, koji uključuje sustavski stog te podatke o dretvama i monitorima, za sustav s manjim brojem dretvi, potrebno je oko 3000 okteta spremničkog prostora. To je značajno manje od drugih rješenja koja koriste jezgru, bila ona ostvarena samo i sa skupom osnovnih funkcija. Trajanje funkcija monitora ovisi o broju dretvi u sustavu, odnosno veličini redova s kojima se rukuje. Za manji broj dretvi vremena se kreću od 150 do 400 ciklusa procesora. Povećanjem veličine redova vremena linearno rastu te bi za sustav u kojem se očekuju veći redovi (više od 10 dretvi u svakom) ta vremena mogla biti i višestruko veća. Sustavi s velikim brojem dretvi izlaze iz okvira sustava razmatranih u ovom radu koji je optimiran za manji broj dretvi.

Nedostatak zasnivanja ugrađenog sustava na monitorima pojavljuje se kada je sustavu potrebno dodati naprednije algoritme raspoređivanja. Iako se jednostavniji algoritmi mogu relativno jednostavno ugraditi, kao što je to pokazano na algoritmu raspoređivanja prema trenutku nužnog završetka, složeniji algoritmi zahtijevaju dinamičko rukovanje dretvama,

tj. njihovim stanjima i prioritetima. Već se na primjeru ostvarenja pojednostavljenog algoritma raspoređivanja kružnim posluživanjem prioriteti dretvi moraju mijenjati izvan funkcija monitora. Ugradnjom raspoređivača u jezgru navedeni bi se nedostatak mogao ukloniti. Neki se raspoređivači mogu dodati i u funkcije za ostvarenje monitora, tj. zamjenom trenutnog raspoređivača koji radi isključivo prema prioritetu dretvi. Takvo bi uplitanje u osnovne funkcije ipak značajno promijenilo i sam monitor te potrebnu podatkovnu strukturu i za monitor i za dretve. Uključivanje složenijih raspoređivača u funkcije za ostvarenje monitora vodi prema izgradnji jezgre, koja više ne bi bila niti malena niti jednostavna. Sustavi koji traže takve raspoređivače su složeni sustavi u kojima prikazano rješenje neće biti dovoljno, već će biti potrebno koristiti neka druga rješenja.

Zasnivanje ugrađenih sustava korištenjem koncepta monitora na način prikazan u ovom radu ima svojih prednosti i nedostataka. Jednostavniji sustavi, bez potrebe složenih algoritama raspoređivanja dretvi, korištenjem monitora dobivaju moćne sinkronizacijske mehanizme uz istovremeno vrlo male zahtjeve na spremnički prostor.

DODATAK A – OSTVARENJE MONITORA U STROJNOM JEZIKU ARM7TDMI PROCESORA

Funkcije za ostvarenje monitora izgrađene u strojnom jeziku procesora ARM7TDMI uz kôd za inicijalizaciju prekidnog sustava i brojila, funkcije za ostvarivanje kašnjenja, zajedno sa svim korištenim konstantama i podatkovnim strukturama prikazane su u ovom dodatku. Uz navedeni kôd i podatke, koji na neki način mogu predstavljati jezgru ugrađenog sustava, naveden je i kôd za tri vrlo jednostavne dretve tako da se sve skupa čini zaokruženu cjelinu koja se može prevesti i pokrenuti (korištenjem *RealView Developer Suite v2.1* razvojnog alata i simulatora).

```
    PRESERVE8

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Konstante
    AREA    Data, DATA, READWRITE, ALIGN=3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Jezgrine funkcije
Uci_u_monitor               EQU 1
Izaci_iz_monitora           EQU 2
Uvrstiti_u_red_uvjeta       EQU 3
Osloboditi_iz_reda_uvjeta   EQU 4

MUTEX       EQU     -1
UVJET       EQU     -2
KRAJ_LISTE  EQU     -4

PRIPRAVNE   EQU     -8
AKTIVNA     EQU     -16
RED_UVJETA  EQU     -32

DRETVA_ID           EQU     0
DRETVA_PRI          EQU     4
DRETVA_RED          EQU     8
DRETVA_IDUCA        EQU     12
DRETVA_VRIJEME      EQU     16
DRETVA_DP_ZADNJI    EQU     20
DRETVA_DP_PRI       EQU     24

MONITOR_VLASNIK     EQU     0
MONITOR_RED         EQU     4
MONITOR_RED_UVJETA  EQU     8

MONITOR_NITKO       EQU     0
NITKO               EQU     0

KONTEKST        EQU     7*4
KONTEKST_SP     EQU     13*4
KONTEKST_PC     EQU     15*4
KONTEKST_SPSR   EQU     16*4

VELICINA_DRETVE EQU     24*4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Konstante specifične za ARM

; Statusni registar
Mode_USR        EQU 0x10
Mode_FIQ        EQU 0x11
Mode_IRQ        EQU 0x12
```

```
Mode_SVC         EQU 0x13
Mode_ABT         EQU 0x17
Mode_UND         EQU 0x1B
Mode_SYS         EQU 0x1F
I_Bit            EQU 0x80
F_Bit            EQU 0x40

; Brojila - adrese i vrijednosti
Timer_base       EQU 0x0a800000
Timer1_Load      DCD Timer_base
Timer1_Value     DCD Timer_base+4
Timer1_Control   DCD Timer_base+8
Timer1_Clear     DCD Timer_base+12
Timer2_Load      DCD Timer_base+32
Timer2_Value     DCD Timer_base+36
Timer2_Control   DCD Timer_base+40
Timer2_Clear     DCD Timer_base+44

Timer_vrijeme    DCD 0x0000ffff
Clock_start      EQU 0x000000c0

; Konstante prekidnog sustava (IRQ)
IRQ_base                 EQU 0x0a000000
IRQ_Enable_Clear_Value   EQU 0x003FFFFF   ; zabraniti sve prekide
IRQ_Enable_Set_Value     EQU 0x30         ; dozvoliti prekide brojila 1 i 2
IRQ_Disable_Set_Value    EQU 0x20         ; ne dozvoliti prekide brojila 1

IRQ_Status               DCD IRQ_base
IRQ_RawStatus            DCD IRQ_base+4
IRQ_Enable_Set           DCD IRQ_base+8
IRQ_Enable_Clear         DCD IRQ_base+12

IRQ_Enable_Clear_adr     DCD IRQ_Enable_Clear_Value

; Vrijeme sustava - brojilo vremena - broj isteka brojila 2
Sat     DCD 0


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inicijalizacija

    AREA Init_code, CODE, READONLY, ALIGN=3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inicijalizacija sustava - počinje u SVC načinu

    ENTRY       ; Oznaka početka

Init

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Postavljanje vektora za obradu programskih prekida (SWI)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    MOV     r0, #0x00000008     ; adresa SWI vektora
    LDR     r1, =SWI_obrada
    SUB     r1, r1, r0
    SUB     r1,r1,#0x00000008
    MOV     r1, r1, LSR #2
    MOV     r2, #0xea000000
    ADD     r1, r1, r2          ; r1 = kôd instrukcije "B SWI_obrada"
    STR     r1, [r0]            ; na 0x08 -> B SWI_obrada

; Inicijalizacija kazaljki SVC
    MOV     r0, #Mode_SVC|I_Bit|F_Bit       ; SVC mod, uz zabranjene prekide
    MSR     cpsr_cf, r0
    LDR     sp, =Stog_sustav


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Postavljanje vektora za obradu prekida (IRQ)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    MOV     r0, #0x00000018     ; adresa IRQ vektora
    LDR     r1, =IRQ_prihvat
    SUB     r1, r1, r0
    SUB     r1,r1,#0x00000008
    MOV     r1, r1, LSR #2
```

```
    MOV     r2, #0xea000000
    ADD     r1, r1, r2          ; r1 = kôd instrukcije "B IRQ_prihvat"
    STR     r1, [r0]            ; na 0x08 -> B IRQ_prihvat

; Inicijalizacija kazaljki IRQ
    MOV     r0, #Mode_IRQ|I_Bit|F_Bit       ; IRQ mod, uz zabranjene prekide
    MSR     cpsr_cf, r0
    LDR     sp, =Stog_sustav

; Inicijalizacija IRQ sustava
    LDR     r0, =IRQ_Enable_Clear
    LDR     r0, [r0]
    LDR     r1, =IRQ_Enable_Clear_adr
    LDR     r1, [r1]
    STR     r1, [r0]

    LDR     r0, =IRQ_Enable_Set
    LDR     r0, [r0]
    MOV     r1, #IRQ_Enable_Set_Value
    STR     r1, [r0]

; Inicijalizacija brojila
; Brojilo 1 - za ostvarivanje kašnjenja
; Brojilo 2 - za ostvarivanje logičkog sata

; Brojilo 1: u početku nebitna početna vrijednost
    LDR     r0, =Timer1_Clear
    LDR     r0, [r0]
    MOV     r1, #0
    STR     r1, [r0]

    LDR     r0, =Timer1_Control
    LDR     r0, [r0]
    MOV     r1, #Clock_start
    STR     r1, [r0]

    LDR     r0, =Timer1_Load
    LDR     r0, [r0]
    LDR     r1, =Timer_vrijeme
    LDR     r1, [r1]
    STR     r1, [r0]


; Brojilo 2: kreće od nule (0 -> ffff -> fffe ...)
    LDR     r0, =Timer2_Clear
    LDR     r0, [r0]
    MOV     r1, #0
    STR     r1, [r0]

    LDR     r0, =Timer2_Control
    LDR     r0, [r0]
    MOV     r1, #Clock_start
    STR     r1, [r0]

    LDR     r0, =Timer2_Load
    LDR     r0, [r0]
    LDR     r1, =Timer_vrijeme
    LDR     r1, [r1]
    STR     r1, [r0]

; Pokretanje dretve - iz aktivne obnoviti kontekst ...

Aktiviraj_aktivnu
    LDR     lr, =Aktivna
    LDR     lr, [lr]

    MOV     r0, #AKTIVNA                ; označiti da je aktivna
    STR     r0, [lr, #DRETVA_RED]

    ADD     lr, lr, #KONTEKST
    LDR     r1, [lr, #KONTEKST_SPSR]    ; statusni registar 16*4
    MSR     SPSR_cf, r1
    LDMFD   lr, {r0-r14}^               ; učitati sve osim pc-a
    ADD     lr, lr,#KONTEKST_PC         ; r15=pc
    LDMFD   lr, {pc}^                   ; učitati kontekst i početi s radom Aktivne
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Prihvat prekida (IRQ)

    AREA IRQ_prihvat_code, CODE, READONLY, ALIGN=3

IRQ_prihvat
    STMFD   sp!, {r0,r1,lr}     ; kontekst na stog
    MOV     lr, #Mode_IRQ|I_Bit|F_Bit       ; IRQ mod, uz zabranjene prekide
    MSR     cpsr_cf, lr

    ; pogledati u status sklopa za prihvat prekida da se vidi koji uzrok
    LDR     r0, =IRQ_Status
    LDR     r0, [r0]            ; adresa statusa IRQ u r0
    LDR     r0, [r0]            ; status IRQ u r0

    TST     r0, #0x0010         ; brojilo 1
    BNE     Brojilo_1_prekid
    TST     r0, #0x0020         ; brojilo 2
    BNE     Brojilo_2_prekid
    TST     r0, #0x0001         ; FIQ
    BNE     FIQ_Obrada
    TST     r0, #0x0002         ; Programski prekid
    BNE     SWI_obrada
    TST     r0, #0x0004         ; COMM Rx
    BNE     COMM_Rx
    TST     r0, #0x0008         ; COMM Tx
    BNE     COMM_Tx

FIQ_Obrada
COMM_Rx
COMM_Tx
; Nepoznati razlog
    LDMFD   sp!, {r0,r1,lr} ; kontekst sa stoga+povratak
    SUBS    pc, lr, #4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Brojilo_1_prekid
; Prekid brojila 1 - koristi se ostvarivanje kašnjenja

    ; onemogućiti daljnje prekidanje s tim brojilom
    LDR     r0, =IRQ_Enable_Clear
    LDR     r0, [r0]
    LDR     r1, =IRQ_Enable_Clear_adr
    LDR     r1, [r1]
    STR     r1, [r0]

    LDR     r0, =IRQ_Enable_Set
    LDR     r0, [r0]
    MOV     r1, #IRQ_Disable_Set_Value
    STR     r1, [r0]

; Treba provjeriti je li možda obrada u tijeku
    LDR     r0, =Prekid_aktivan
    LDR     r1, [r0]
    CMP     r1, #1
; Ako je obrada u tijeku samo se vrati natrag
    LDMEQFD sp!, {r0,r1,pc}^

    ; označiti da se ide u obradu
    MOV     r1, #1
    STR     r1, [r0]

    ; obrisati prekid brojila 1
    LDR     r0, =Timer1_Clear
    LDR     r0, [r0]
    MOV     r1, #0
    STR     r1, [r0]

; Pohraniti kontekst aktivne i premjestiti ju u pripravne

    ; spremiti kontekst Aktivne
    LDMFD   sp!, {r0,r1,lr}
    SUB     lr, lr, #4              ; zbog prekida treba podesit pc
    STMFD   sp!, {lr}
    LDR     lr, =Aktivna            ; lr - adr. kazaljke na Aktivnu (&Aktivna)
    LDR     lr, [lr]                ; lr - adr. Aktivne (Aktivna)
```

```
    ADD     lr, lr, #KONTEKST       ; lr - adr. pok. na mjesto za kontekst
    STMEA   lr, {r0-r14}^           ; spremiti korisničke registre (prvih 60 okteta)

    MRS     r0, spsr                ; spsr - kopija cpsr-a prije prekida
    STR     r0, [lr, #KONTEKST_SPSR]    ; spsr u kontekst (64)
    LDMFD   sp!, {r3}               ; lr sa stoga (povratna adresa)
    STR     r3, [lr, #KONTEKST_PC]  ; lr u kontekst (na r15)

    ; premjestiti ju u Pripravne
    LDR     r3, =Aktivna
    LDR     r0, [r3]
    MOV     r1, #PRIPRAVNE
    STR     r1, [r0, #DRETVA_RED]   ; Aktivna->red=PRIPRAVNE
    LDR     r1, =Pripravne
    LDR     r2, [r1]                ; Pripravne
    STR     r2, [r0, #DRETVA_IDUCA] ; Aktivna->iduća=Pripravne
    STR     r0, [r1]                ; Pripravne=Aktivna

; Izbaciti Dretvu IRQ iz reda uvjeta u ili u red za monitor ako on nije slobodan,
; ili odmah u monitor ako je slobodan
    LDR     r0, =Dretva_IRQ

    LDR     r1, =Monitor
    MOV     r2, #Monitor_IRQ        ; IRQ dretva za ovaj primjer je dretva za kašnjenje
    LDR     r1, [r1, r2, LSL #2]    ; r1 = Monitor[Monitor_IRQ]

    ; IRQ dretva je sama u tom redu uvjeta, a sad je se miče
    MOV     r2, #KRAJ_LISTE
    STR     r2, [r1, #MONITOR_RED_UVJETA] ; prvi red uvjeta je za ovu dretvu

    ; je li monitor slobodan?
    LDR     r2, [r1, #MONITOR_VLASNIK]
    CMP     r2, #MONITOR_NITKO
    BNE     Prekid_netko_u_monitoru

    LDR     r2, [r0, #DRETVA_ID]
    STR     r2, [r1, #MONITOR_VLASNIK]

; Metoda vršnog prioriteta - Povećati prioritet
;   MOV     r2, r0
;   MOV     r0, #Monitor_IRQ
;   BL      Povecati_prioritet
;   MOV     r0, r2

; Ukoliko IRQ dretva ima veći prioritet od prve iz reda pripravnih
; može se odmah i proglasiti aktivnom, inače se prva iz reda pripravnih
; proglašava aktivnom a ova se stavlja u red.
    LDR     r2, =Pripravne
    LDR     r4, [r2]
    LDR     r5, [r4, #DRETVA_PRI]
    LDR     r6, [r0, #DRETVA_PRI]
    CMP     r6, r5
    STRLE   r0, [r3]
    BLE     Aktiviraj_aktivnu

    ; Potrebno je prvu iz reda pripravnih pomaknuti u Aktivnu, a IRQ dretvu
    ; postaviti u taj red
    LDR     r5, [r4, #DRETVA_IDUCA]
    STR     r5, [r0, #DRETVA_IDUCA]
    STR     r0, [r2]
    STR     r4, [r3]
    MOV     r5, #PRIPRAVNE
    STR     r5, [r0, #DRETVA_RED]
    MOV     r5, #AKTIVNA
    STR     r5, [r4, #DRETVA_RED]
    B       Aktiviraj_aktivnu

Prekid_netko_u_monitoru
    ; IRQ dretvu prebaci u red monitora
    LDR     r2, [r1, #MONITOR_RED]
    STR     r2, [r0, #DRETVA_IDUCA]
    STR     r0, [r1, #MONITOR_RED]

; Naslijeđivanje prioriteta - ukoliko ova dretva ima veći prioritet od one u monitoru
    MOV     r2, r0
    MOV     r0, #Monitor_IRQ
```

```
    BL      Nasljedni_prioriteti_povecati

    ; vrati prvu pripravnu u aktivnu
    ; prva == fizički prva, jer je zadnja aktivna
    ; r3=Aktivna

    LDR     r2, =Pripravne
    LDR     r4, [r2]                ; r4=prva pripravna
    STR     r4, [r3]                ; Aktivna = r4
    LDR     r4, [r4, #DRETVA_IDUCA] ; r4=r4->iduca
    STR     r4, [r2]                ; Pripravna=r4
    B Aktiviraj_aktivnu

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Brojilo_2_prekid
; Ažurirati vrijeme = varijabla "Sat"
    LDR     r0, =Sat
    LDR     r1, [r0]
    ADD     r1, r1, #1
    STR     r1, [r0]

; Obrisati prekid brojila 2
    LDR     r0, =Timer2_Clear
    LDR     r0, [r0]
    MOV     r1, #0
    STR     r1, [r0]

    LDMFD   sp!, {r0,r1,lr} ; kontekst sa stoga+povratak
    SUBS    pc, lr, #4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Dohvatiti_sat() - "realno" vrijeme - brojilo 2
;
Dohvatiti_sat
    SUB     sp, sp, #4      ; mjesto za pohranu rezultata
    STMFD   sp!, {r0, r1}
    LDR     r1, =Timer2_Value
    LDR     r1, [r1]
    LDR     r1, [r1]
    LDR     r0, =Timer_vrijeme
    LDR     r0, [r0]
    SUB     r1, r0, r1          ; r1 = 0x0000ffff - brojilo

    LDR     r0, =Sat
    LDR     r0, [r0]
    ADD     r0, r1, r0, LSL #16
    STR     r0, [sp, #+8]       ; na stog
    LDMFD   sp!, {r0,r1}
    MOV     pc, lr


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Dohvatiti_brojilo() - stanje brojila, koliko je odbrojilo ili koliko ima za brojiti

; 16 bita, ako treba više onda staviti i dijelitelj na frekv., ili dodati još 1 brojač

Dohvatiti_brojilo
    SUB     sp, sp, #4      ; mjesto za pohranu rezultata
    STMFD   sp!, {r0}
    LDR     r0, =Timer1_Value
    LDR     r0, [r0]
    LDR     r0, [r0]
    STR     r0, [sp, #4]
    LDMFD   sp!, {r0}
    MOV     pc, lr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Postaviti_brojilo() - postaviti koliko ima za brojiti

; Na stogu se nalazi vrijednost na koju treba postaviti
Postaviti_brojilo
    STMFD   sp!, {r0, r1}
    LDR     r0, =Timer1_Clear
    LDR     r0, [r0]
    MOV     r1, #0
    STR     r1, [r0]
```

```
    LDR     r0, =Timer1_Control
    LDR     r0, [r0]
    MOV     r1, #Clock_start
    STR     r1, [r0]

    LDR     r0, =Timer1_Load
    LDR     r0, [r0]
    LDR     r1, [sp, #+8]           ; sa stoga vrijeme
    STR     r1, [r0]
    LDMFD   sp!, {r0, r1}
    ADD     sp, sp, #4              ; "brisanje" parametra sa stoga
    MOV     pc, lr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    AREA SWI_obrada_code, CODE, READONLY, ALIGN=3

SWI_obrada
; Spremiti adresu povratka i zabraniti prekide
    STMFD   sp!, {lr}                   ; adresa povratka
    MOV     lr, #Mode_SVC|I_Bit|F_Bit   ; SVC mod, uz zabranjene prekide
    MSR     cpsr_cf, lr

    LDR     lr, =Aktivna            ; lr - adr. kazaljke na Aktivnu (&Aktivna)
    LDR     lr, [lr]                ; lr - adr. Aktivne (Aktivna)
    ADD     lr, lr, #KONTEKST       ; lr - adr. pok. na mjesto za kontekst
    STMEA   lr, {r0-r14}^           ; spremi korisničke registre (prvih 60 okteta)

    MRS     r0, spsr                ; spsr - kopija cpsr-a prije prekida
    STR     r0, [lr, #KONTEKST_SPSR]; spsr u kontekst (64)
    LDMFD   sp!, {r3}               ; lr sa stoga (povratna adresa)
    STR     r3, [lr, #KONTEKST_PC]  ; lr u kontekst (na r15)

    STMFD   sp!, {lr}               ; postaviti adresu konteksta na stog
                                    ; radi parametara u/iz j-funkcije

; Odrediti koja se J-funkcija zove
    LDR     r3, [r3,#-4]            ; dohvatiti instrukciju SWI
    BIC     r3, r3, #0xFF000000     ; obrisati gornjih 8 bitova - ostaje broj f-je

; Sve 4 funkcije imaju parametar broj monitora pa se neke stvari mogu napraviti ovdje
    LDR     r1, [lr, #KONTEKST_SP]  ; "sp" dretve koja je pozvala fju u r0 (iz kont.)
    LDMFD   r1!, {r0}               ; parametar u r0 = M
    STR     r1, [lr, #KONTEKST_SP]  ; vrati ažurirani sp (u kontekst dretve)

    LDR     r1, =Monitor
    LDR     r1, [r1, r0, LSL #2]    ; r1 - adresa podataka traženog Monitora
    LDR     r2, =Aktivna            ; r2 - adr. kazaljke na Aktivnu (&Aktivna)
    LDR     r2, [r2]                ; r2 - adr. Aktivne (Aktivna)

; r0 = M
; r1 - Monitor[M]
; r2 - Aktivna

; r3 sadrži broj J-funkcije - pozovi ju preko tablice
    BL      pozovi_j_funkciju

; nakon povratka iz f-je
; prvo: staviti povratnu vrijednost na korisnički stog dretve koja je funkciju pozvala
    LDMFD   sp!, {r0}               ; skinuti povratnu vrijednost sa stoga sustava
    LDMFD   sp!, {r1}               ; skinuti adresu konteksta dretve
    LDR     r2, [r1, #KONTEKST_SP]  ; učitati kazaljku na vrh korisničkog stoga
    STMFD   r2!, {r0}               ; postaviti povratnu vrijednost na stog
    STR     r2, [r1, #KONTEKST_SP]  ; ažurirati "sp" u kontekstu

; drugo: učitati kontekst aktivne dretve
    B       Aktiviraj_aktivnu

pozovi_j_funkciju
    LDR     r4, =adresa_j_funkcija  ; adresa gdje su adrese J-funkcija
    LDR     pc, [r4, r3, LSL #2]    ; pc=[ADRESA_J_FUNKCIJA+r3] - LSL #2 = *4 (4 okteta)
```

```
adresa_j_funkcija
    DCD     0   ; greška 1-4
    DCD     J_uci_u_monitor
    DCD     J_izaci_iz_monitora
    DCD     J_uvrstiti_u_red_uvjeta
    DCD     J_osloboditi_iz_reda_uvjeta

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    AREA J_uci_u_monitor_code, CODE, READONLY, ALIGN=3

J_uci_u_monitor
; r0 = M
; r1 - Monitor[M]
; r2 - Aktivna
    LDR     r3, [r2, #DRETVA_ID]
; r3 - Aktivna->id

    LDR     r4, [r1, #MONITOR_VLASNIK]  ; r4=Monitor[M].vlasnik
    CMP     r4, #MONITOR_NITKO          ; je li slobodan?
    BNE     J_uci_u_monitor_u_red
    STR     r3, [r1, #MONITOR_VLASNIK]  ; Slobodan je! -> Monitor[M].vlasnik=Aktivna->id

; Povećati_prioritet(Aktivna, M) - MVP
;   STMFD   sp!, {lr}
;   BL      Povecati_prioritet  ; r0=M, r2=Aktivna
;   LDMFD   sp!, {lr}

    B       vrati_ok__J_uci_u_monitor

; treba staviti u red
J_uci_u_monitor_u_red
    LDR     r3, [r1, #MONITOR_RED]  ; r3 = Monitor[M].red
    STR     r3, [r2, #DRETVA_IDUCA] ; Aktivna->iduca=Monitor[M].red
    STR     r2, [r1, #MONITOR_RED]  ; Monitor[M].red=Aktivna
    STR     r0, [r2, #DRETVA_RED]   ; Aktivna->red=M

; Nasljedni_prioriteti_povecati(Aktivna, M) - protokol nasljeđivanja prioriteta
    STMFD   sp!, {lr}
    BL      Nasljedni_prioriteti_povecati   ; r0=M, r1=Monitor[M], r2=Aktivna
    LDMFD   sp!, {lr}

; prebaciti prvu iz reda pripravnih u red aktivne
    STMFD   sp!, {lr}               ; lr na stog
    BL      j_prebaci_prvu_pripravnu_u_aktivne
    LDMFD   sp!, {lr}               ; lr sa stoga

vrati_ok__J_uci_u_monitor
    MOV     r0, #0
    STMFD   sp!, {r0}               ; 0 na stog - povratna vrijednost
    MOV     pc, lr                  ; vratiti se nazad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    AREA J_izaci_iz_monitora_code, CODE, READONLY, ALIGN=3

J_izaci_iz_monitora
; r0 = M
; r1 - Monitor[M]
; r2 - Aktivna

; smanjiti_prioritet(Aktivna, M) - protokol nasljeđivanja/vršnih prioriteta
    STMFD   sp!, {lr}
    BL      Smanjiti_prioritet      ; r0=M, r2=Aktivna
    LDMFD   sp!, {lr}

; Aktivna --> Pripravne
    LDR     r3, =Pripravne
    LDR     r4, [r3]                ; r4=Pripravne
    STR     r4, [r2, #DRETVA_IDUCA] ; Aktivna->iduca = Pripravne
    STR     r2, [r3]                ; Pripravne = Aktivna
    MOV     r4, #PRIPRAVNE
    STR     r4, [r2, #DRETVA_RED]   ; Aktivna->red=PRIPRAVNE

J_izaci_iz_monitora_sljedeca
    STMFD   sp!, {lr}

    ; osloboditi prvu iz reda monitora
```

```
    STMFD   sp!, {r1}               ; Monitor[M] na stog (adresa)
    BL      j_osloboditi_prvu_iz_reda_monitora

    ; prebaciti prvu iz reda pripravnih u red aktivne
    BL      j_prebaci_prvu_pripravnu_u_aktivne

    LDMFD   sp!, {lr}               ; lr sa stoga

    MOV     r0, #0
    STMFD   sp!, {r0}               ; 0 na stog
    MOV     pc, lr                  ; vratiti se nazad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    AREA J_uvrstiti_u_red_uvjeta_code, CODE, READONLY, ALIGN=3

J_uvrstiti_u_red_uvjeta         ; parametri (M,K)
; r0 = M -- nije potreban u ovoj f-ji
; r1 - Monitor[M]
; r2 - Aktivna

; smanjiti_prioritet(Aktivna, M) - protokol nasljeđivanja prioriteta i MVP
    STMFD   sp!, {lr}
    BL      Smanjiti_prioritet      ; r0=M, r2=Aktivna - pohrani one koji koristiš
    LDMFD   sp!, {lr}

; parametar K sa stoga
    LDR     r4, [sp]                ; adresa konteksta s vrha sustavskog stoga
    LDR     r6, [r4, #KONTEKST_SP]  ; "sp" dretve koja je pozvala fju (iz kont.)
    LDMFD   r6!, {r3}               ; parametar 2 u r3 (K)
    STR     r6, [r4, #KONTEKST_SP]  ; vratiti ažurirani sp (u kontekst dretve)
; r3 = K

; Aktivna -> red uvjeta Monitor[M].red_uvjeta[K], M=r1, K=r3
    MOV     r6, #RED_UVJETA
    STR     r6, [r2, #DRETVA_RED]           ; Aktivna->red=RED_UVJETA

    ADD     r6, r1, #MONITOR_RED_UVJETA     ; r6=Monitor[M].red_uvjeta[0]
    ADD     r6, r6, r3, LSL #2              ; r6=r6+r3*4 (*4 zbog 32 bita=4 okteta)

    LDR     r3, [r6]                        ; r3=Monitor[M].red_uvjeta[K]
    STR     r3, [r2, #DRETVA_IDUCA]         ; Aktivna->iduca=r6
    STR     r2, [r6]                        ; Monitor[M].red_uvjeta[K]=Aktivna

; r1 ne dirati - koristi se pri prebacivanju prve iz reda monitora u red pripravnih

; dodatno za prekide (kašnjenje): ako je dretva koja ide u red uvjeta IRQ dretva,
; onda označiti da je IRQ obrada završena
    ; r2=Aktivna
    LDR     r3, =Dretva_IRQ
    CMP     r2, r3      ; Aktivna == IRQ ??
    BNE     J_izaci_iz_monitora_sljedeca

    ; radi se o IRQ dretvi - dodatni posao za ovaj prekid
    ; ponovo omogućiti prekidanje s brojilom 1
    LDR     r6, =IRQ_Enable_Set
    LDR     r6, [r6]
    MOV     r2, #IRQ_Enable_Set_Value
    STR     r2, [r6]

    ; označiti kraj obrade prekida
    LDR     r6, =Prekid_aktivan
    MOV     r2, #0
    STR     r2, [r6]

; prebaciti prvu iz reda monitora u red pripravnih
; isto kao i u funkciji Izaci_iz_monitora
    B   J_izaci_iz_monitora_sljedeca

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    AREA J_osloboditi_iz_reda_uvjeta_code, CODE, READONLY, ALIGN=3

J_osloboditi_iz_reda_uvjeta     ; parametri (M,K)
```

```
; r0 = M
; r1 - Monitor[M]
; r2 - Aktivna

; smanjiti_prioritet(Aktivna, M) - protokol nasljeđivanja/vršnih prioriteta
    STMFD   sp!, {lr}
    BL      Smanjiti_prioritet
    LDMFD   sp!, {lr}

; parametar K sa stoga
    LDR     r4, [sp]                ; adresa konteksta s vrha sustavskog stoga
    LDR     r5, [r4, #KONTEKST_SP]  ; "sp" dretve koja je pozvala fju (iz kont.)
    LDMFD   r5!, {r3}               ; parametar 2 u r3 (K)
    STR     r5, [r4, #KONTEKST_SP]  ; vrati ažurirani sp (u kontekst dretve)
; r3 - K

; aktivnu u red monitora
    LDR     r4, [r1, #MONITOR_RED]  ; r4 = kazaljka na dretvu u redu (ili KRAJ_LISTE)
    STR     r4, [r2, #DRETVA_IDUCA] ; Aktivna->iduca=Monitor[M].red
    STR     r2, [r1, #MONITOR_RED]  ; Monitor[M].red=Aktivna
    STR     r0, [r2, #DRETVA_RED]   ; Aktivna->red=r1

; prvu iz reda uvjeta u red monitora
    ADD     r4, r1, #MONITOR_RED_UVJETA     ; r4=Monitor[M].red_uvjeta[0]
    ADD     r4, r4, r3, LSL #2              ; r4=r4+r3*4 (*4 zbog 32 bita=4 okteta)
; r4=Monitor[M].red_uvjeta[K]
    LDR     r5, [r4]
    CMP     r5, #KRAJ_LISTE                 ; Monitor[M].red_uvjeta[K] == KRAJ_LISTE ?
    BEQ     J_osloboditi_iz_reda_uvjeta__nema

; izvaditi prvu iz reda uvjeta
    STMFD   sp!, {r0,r1,lr}
    STMFD   sp!, {r4}                       ; r4 = red
    BL      j_izvaditi_prvu_iz_reda
    LDMFD   sp!, {r4}                       ; r4 = izbačena dretva
    LDMFD   sp!, {r0,r1,lr}

    LDR     r5, [r1, #MONITOR_RED]  ; r5=Monitor[M].red
    STR     r5, [r4, #DRETVA_IDUCA] ; r4->iduca=Monitor[M].red
    STR     r4, [r1, #MONITOR_RED]  ; Monitor[M].red=r4
    STR     r0, [r4, #DRETVA_RED]   ; r4->red = M

J_osloboditi_iz_reda_uvjeta__nema

; prebaciti prvu iz reda monitora u red pripravnih
; isto kao i u funkciji Izaci_iz_monitora
    B   J_izaci_iz_monitora_sljedeca

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    AREA j_pomocne_code, CODE, READONLY, ALIGN=3


j_prebaci_prvu_pripravnu_u_aktivne

; treba pronaći dretvu najvećeg prioriteta i nju izbaciti iz liste
; u redu se uvijek nalazi BAR jedna dretva !!! (idle dretva)
; ne pohranjuju se registri jer je to zadnji poziv
    LDR     r0, =Pripravne

    STMFD   sp!, {lr}               ; lr na stog
    STMFD   sp!, {r0}               ; r0 na stog
    BL      j_izvaditi_prvu_iz_reda
    LDMFD   sp!, {r0}               ; r0 sa stoga - dretva koja je izvađena iz reda
    LDMFD   sp!, {lr}               ; lr sa stoga

    LDR     r1, =Aktivna                ; r1 - adr. kazaljke na Aktivnu (&Aktivna)
    STR     r0, [r1]                ; Aktivna=r0
```

```
    MOV     pc, lr          ; vrati se nazad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
j_izvaditi_prvu_iz_reda
; ne pohranjuju se registri, ako treba to se napravi prije poziva ove f-je

    LDMFD   sp!, {r10}              ; r10 sa stoga - red iz kojeg se vadi
    LDR     r4, [r10]               ; r4 == prva

    CMP     r4, #KRAJ_LISTE         ; lista prazna?
    STMEQFD sp!, {r4}               ; r4 na stog
    MOVEQ   pc, lr                  ; vratiti se nazad

    ;lista nije prazna
    MOV     r5, r4                  ; r5 == prije
    MOV     r7, r5                  ; r7=r5 == prije_pom
    LDR     r2, [r4, #DRETVA_PRI]   ; r2=r4->pri

j_izvaditi_prvu_iz_reda__petlja_pocetak
    LDR     r6, [r7, #DRETVA_IDUCA] ; r6 == pom (=pom->iduca)
    CMP     r6, #KRAJ_LISTE
    BEQ     j_izvaditi_prvu_iz_reda__kraj_liste
    LDR     r3, [r6, #DRETVA_PRI]   ; r3=r6->pri
    CMP     r2, r3                  ; r2 ? r3
    MOVGE   r5, r7                  ; prije=pom_prije
    MOVGE   r4, r6                  ; prva=pom
    MOVGE   r2, r3

    MOV     r7, r6                  ; prije_pom=pom
    B       j_izvaditi_prvu_iz_reda__petlja_pocetak

j_izvaditi_prvu_iz_reda__kraj_liste
    LDR     r6, [r4, #DRETVA_IDUCA] ; r6=prva->iduca
    CMP     r4, r5                  ; vadi se prva (fizički) iz reda?
    STREQ   r6, [r10]

    ; ne prva iz reda, treba premostit
    STRNE   r6, [r5, #DRETVA_IDUCA] ; r5->iduca = r4->iduca = r6

    STMFD   sp!, {r4}               ; r4 na stog - prva iz reda
    MOV     pc, lr                  ; vrati se nazad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
j_osloboditi_prvu_iz_reda_monitora
    LDMFD   sp, {r6}                ; Monitor[M] , ostaje na stogu !!!
    LDR     r1, [r6, #MONITOR_RED]  ; r1=Monitor[M].red

    CMP     r1, #KRAJ_LISTE
    MOVEQ   r2, #NITKO
    STREQ   r2, [r6, #MONITOR_VLASNIK]
    LDMEQFD sp!, {r6}               ; makni r0 sa stoga
    MOVEQ   pc, lr                  ; vrati se nazad

    STMFD   sp!, {lr}
    ADD     r6, r6, #MONITOR_RED
    STMFD   sp!, {r6}
    BL      j_izvaditi_prvu_iz_reda
    LDMFD   sp!, {r1}                   ; r1 = prva
    LDMFD   sp!, {lr}
    LDMFD   sp!, {r6}                   ; r0 = Monitor[M]

    LDR     r2, [r1, #DRETVA_ID]        ; r2=prva->id
    STR     r2, [r6, #MONITOR_VLASNIK]  ; Monitor[M].vlasnik=prva->id

    MOV     r3, #PRIPRAVNE
    STR     r3, [r1, #DRETVA_RED]   ; prva->red=PRIPRAVNE

    LDR     r4, =Pripravne
    LDR     r5, [r4]                ; r5=&Pripravne

    STR     r5, [r1, #DRETVA_IDUCA] ; prva->iduca=Pripravne
    STR     r1, [r4]                ; Pripravne=prva

; Povecati_prioritet(Aktivna, M) - protokol vršnih prioriteta
;   STMFD   sp!, {lr}
;   MOV     r2, r1
```

```
;   BL      Povecati_prioritet          ; r0=M, r2=Aktivna - pohrani one koji koristiš
;   LDMFD   sp!, {lr}

    MOV pc, lr                  ; vrati se nazad


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    AREA j_prioriteti_code, CODE, READONLY, ALIGN=3

; Protokoli vršnih prioriteta te nasljeđivanja prioriteta
; r0 sadrži M, r1=Monitor[M], r2 adresu Aktivne dretve

Nasljedni_prioriteti_povecati
    STMFD   sp!, {r3, r4, r5, r6, r7, r8, r9, r10}

    MOV     r10, r0                     ; r10=red=M
    LDR     r3, [r2, #DRETVA_PRI]       ; r3=Aktivna->pri
    LDR     r4, [r1, #MONITOR_VLASNIK]
    SUB     r4, r4, #1
    LDR     r5, =Dretva
    LDR     r6, [r5, r4, LSL #2]        ; r6=Dretva[Monitor[M].vlasnik]

Povecavaj_rekurzivno
    LDR     r7, [r6, #DRETVA_PRI]
    CMP     r7, r3
    BLE     Nasljedni_prioriteti_povecati_kraj

    LDR     r8, [r6, #DRETVA_DP_ZADNJI]
    ADD     r8, r8, #1
    STR     r8, [r6, #DRETVA_DP_ZADNJI]
    LDR     r9, [r6, #DRETVA_DP_PRI]
    ADD     r9, r9, r8, LSL #3          ; r9 = povecana->din_pri[kraj]
    STR     r7, [r9]                    ; prethodni prioritet
    STR     r10, [r9, #4]               ; uzrok povećavanja
    STR     r3, [r6, #DRETVA_PRI]       ; novi prioritet

; Za rekurzivno
    LDR     r10, [r6, #DRETVA_RED]      ; r10 = povecana->red
    CMP     r10, #0                     ; red > 0 ?
    BLT     Nasljedni_prioriteti_povecati_kraj

    LDR     r4, =Monitor
    LDR     r6, [r4, r10, LSL #2]       ; r6=Monitor[red]
    LDR     r6, [r6, #MONITOR_VLASNIK]  ; r6=Monitor[red].vlasnik
    SUB     r6, r6, #1
    LDR     r6, [r5, r6, LSL #2]        ; r6=Dretva[Monitor[red].vlasnik]

    B       Povecavaj_rekurzivno

Nasljedni_prioriteti_povecati_kraj
    LDMFD   sp!, {r3, r4, r5, r6, r7, r8, r9, r10}
    MOV     pc, lr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Povecati_prioritet(dretva, M)
; dretva=Aktivna=r2, M=r0

Povecati_prioritet
    STMFD   sp!, {r3, r4, r5, r6}

    LDR     r3, [r2, #DRETVA_PRI]       ; r3=Aktivna->pri

    LDR     r6, =Prioriteti
    LDR     r6, [r6, r0, LSL #2]        ; r6=Prioriteti[M]

    CMP     r3, r6
    BLE     Povecati_prioritet_povratak

    LDR     r4, [r2, #DRETVA_DP_ZADNJI]
    ADD     r4, r4, #1
    STR     r4, [r2, #DRETVA_DP_ZADNJI]
```

```
    LDR     r5, [r2, #DRETVA_DP_PRI]
    ADD     r5, r5, r4, LSL #3          ; r5 = adresa(povecana->din_pri[kraj])
    STR     r3, [r5]                    ; prethodni prioritet
    STR     r0, [r5, #4]                ; uzrok povećavanja

    STR     r6, [r2, #DRETVA_PRI]       ; novi prioritet

Povecati_prioritet_povratak
    LDMFD   sp!, {r3, r4, r5, r6}
    MOV     pc, lr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Smanjiti_prioritet(dretva, M)
; dretva=Aktivna=r2, M=r0
Smanjiti_prioritet
    STMFD   sp!, {r3, r4, r5, r6, r7, r8}

    LDR     r3, [r2, #DRETVA_DP_ZADNJI]
    LDR     r4, [r2, #DRETVA_DP_PRI]    ; r4 = adresa(povecana->din_pri[0])

    MOV     r5, #1
    MOV     r7, #-1

Smanjiti_prioritet_petlja1
    CMP     r5, r3
    BGT     Smanjiti_prioritet_petlja1_kraj
    ADD     r6, r4, r5, LSL #3
    LDR     r8, [r6, #4]                ; r8=uzork
    CMP     r8, r0
    STREQ   r7, [r6, #4]                ; brišemo
    ADD     r5, r5, #1
    B       Smanjiti_prioritet_petlja1

Smanjiti_prioritet_petlja1_kraj
    MOV     r5, r3

Smanjiti_prioritet_petlja2
    CMP     r5, #0
    BLE     Smanjiti_prioritet_petlja2_kraj
    ADD     r6, r4, r5, LSL #3
    LDR     r8, [r6, #4]                ; r8=uzork
    CMP     r8, #-1
    BNE     Smanjiti_prioritet_petlja2_kraj
    LDR     r7, [r4, r5, LSL #3]
    STR     r7, [r2, #DRETVA_PRI]       ; dretva->pri=prijašnji prioritet
    SUB     r5, r5, #1
    MOV     r3, r5
    B       Smanjiti_prioritet_petlja2

Smanjiti_prioritet_petlja2_kraj
    STR     r3, [r2, #DRETVA_DP_ZADNJI]

    LDMFD   sp!, {r3, r4, r5, r6, r7, r8}
    MOV     pc, lr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


    AREA j_kasnjenje, CODE, READONLY, ALIGN=3

; Funkcije za ostvarivanje kašnjenja

Zakasniti
    ; vrijednost za koju treba zakasniti je na stogu - 16 bitna vrijednost (u 32 bita)

    ; Uci_u_monitor
    STMFD   sp!, {r0-r4, lr}
    MOV     r4, #Monitor_Zakasniti
    STMFD   sp!, {r4}
```

```
    SWI     Uci_u_monitor
    LDMFD   sp!, {r0}

    LDR     r0, [sp, #+24]      ; vrijeme
    CMP     r0, #0              ; ako je nula vrati se
    BEQ     Zakasniti_izaci

    BL      Dohvatiti_sat
    LDMFD   sp!, {r1}           ; dohvatiti sat
    ADD     r1, r1, r0          ; Aktivna->vrijeme

    LDR     r2, =Aktivna
    LDR     r2, [r2]
    STR     r1, [r2, #DRETVA_VRIJEME]

    BL      Dohvatiti_brojilo
    LDMFD   sp!, {r3}           ; do_prekida

    CMP     r3, r0              ; do_prekida > vrijeme ?
    STMGTFD sp!, {r0}
    BLGT        Postaviti_brojilo   ; Postavit_brojilo(vrijeme)

    ; Uvrstiti_u_red_uvjeta
    LDR     r3, [r2, #DRETVA_ID]
    STMFD   sp!, {r3}
    STMFD   sp!, {r4}
    SWI     Uvrstiti_u_red_uvjeta
    LDMFD   sp!, {r3}

    ; Izaci_iz_monitora
Zakasniti_izaci
    STMFD   sp!, {r4}
    SWI     Izaci_iz_monitora
    LDMFD   sp!, {r4}

    LDMFD   sp!, {r0-r4, lr}

    ADD     sp, sp, #4          ; vrijeme sa stoga
    MOV     pc, lr


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Istek_vremena
    ; izvodi ju dretva Dretva_IRQ, ona je unutar monitora ...


    MOV     r0, #Monitor_Zakasniti
    STMFD   sp!, {r0}
    SWI     Uci_u_monitor
    LDMFD   sp!, {r1}   ; povratna vrijednost

    BL      Dohvatiti_sat
    LDMFD   sp!, {r1}           ; dohvatiti sat == sada

    LDR     r2, =Timer_vrijeme
    LDR     r2, [r2]            ; MAX_VRIJEME=brojač
    MOV     r12, r2

    MOV     r3, #BROJ_DRETVI
    MOV     r4, #-1             ; i=-1
    LDR     r5, =Dretva         ; na polje dretvi

Istek_vremena_petlja1
    ADD     r4, r4, #1
    CMP     r4, r3
    BGE     Istek_vremena_petlja1_kraj

    LDR     r6, [r5, r4, LSL #2]    ; r6=Dretva[i] == [r5+r4*4]
    LDR     r7, [r6, #DRETVA_VRIJEME]
    CMP     r7, #0                  ; Dretva[i].vrijeme==0 ?
    BEQ     Istek_vremena_petlja1

    CMP     r7, r1                  ; Dretva[i].vrijeme < sada ?
    STRLE   r12, [r6, #DRETVA_VRIJEME]  ; Dretva[i].vrijeme=MAX_VRIJEME --> treba ju osl.
```

```
    BLE     Istek_vremena_petlja1

    SUB     r7, r7, r1
    CMP     r7, r2                  ; Dretva[i].vrijeme-sada < brojač ?
    MOVLT   r2, r7                  ; broja = Dretva[i].vrijeme-sada
    B       Istek_vremena_petlja1

Istek_vremena_petlja1_kraj
    CMP     r2, r12                 ; postaviti brojač na manju vrijednost?
    STMLTFD sp!, {r2}
    BLLT    Postaviti_brojilo       ; Postavit_brojilo(brojač)

    MOV     r4, #-1                 ; i=-1

Istek_vremena_petlja2
    MOV     r8, #0
    ADD     r4, r4, #1
    CMP     r4, r3
    BGE     Istek_vremena_petlja2_kraj

    LDR     r6, [r5, r4, LSL #2]    ; r6=Dretva[i] == [r5+r4*4]
    LDR     r7, [r6, #DRETVA_VRIJEME]
    CMP     r7, r12                 ; Dretva[i].vrijeme==0 ?
    BNE     Istek_vremena_petlja2

    STR     r8, [r6, #DRETVA_VRIJEME]   ; Dretva[i].vrijeme=0
    ADD     r8, r4, #1
    STMFD   sp!, {r8}               ; K=i
    STMFD   sp!, {r0}               ; M=Monitor_Zakasniti
    SWI     Osloboditi_iz_reda_uvjeta
    LDMFD   sp!, {r6}
    B       Istek_vremena_petlja2

Istek_vremena_petlja2_kraj

; Izaci_iz_monitora
    STMFD   sp!, {r0}
    SWI     Izaci_iz_monitora
    LDMFD   sp!, {r1}   ; povratna vrijednost

; kraj obrade prekida brojila - Uvrstiti_u_red_uvjeta
    MOV     r0, #Monitor_IRQ
    MOV     r1, #0
    STMFD   sp!, {r1}   ; K=0
    STMFD   sp!, {r0}
    SWI     Uvrstiti_u_red_uvjeta
    LDMFD   sp!, {r0}

    B       Istek_vremena

    ; Izaci_iz_monitora
    ; nikad ne dolazi ovdje


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;; DIO SPECIFIČAN ZA PROBLEM (podaci)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    AREA    Data_problem, DATA, READWRITE, ALIGN=3

BROJ_DRETVI EQU     3   ; broj dretvi u sustavu, uključujući "Idle" dretvu, ali ne
                        ; uključuje IRQ dretve

STOG_SYS_VEL    EQU 256 ; veličina stoga za SVC & IRQ modove
STOG_USR1_VEL   EQU 512 ; veličina stoga za user dretvu
STOG_USR2_VEL   EQU 512 ; veličina stoga za user dretvu
STOG_USR3_VEL   EQU 512 ; veličina stoga za user dretvu


; Podaci o aktivnoj dretvi te prvoj iz reda pripravnih

Aktivna     DCD     Dretva_2    ; adresa trenutno aktivne dretve
Pripravne   DCD     Dretva_3    ; kazaljka na prvu dretvu iz reda pripravnih - idle

Prekid_aktivan  DCD 0           ; je li aktivan prekid, ako jest ne prihvaćati nove
```

```
; memorija gdje su spremljeni podaci o dretvama:
;Dretva
; struktura:
;   id, pri, red, iduca, vrijeme, naslj_pri_top, naslj_data (ako se koristi)
;   r0-r12 - sadržaj registara
;   sp, lr, pc, cpsr

Dretva  DCD Dretva_1, Dretva_2, Dretva_3, Dretva_IRQ
; redom podaci o dretvama
Dretva_1 ; IDLE DRETVA
    DCD 1, 255, PRIPRAVNE, KRAJ_LISTE, 0, 0, KRAJ_LISTE
    SPACE 13*4
    DCD 0, 0, Dretva_1_start, 0x00000010

Dretva_2
    DCD 2, 100, AKTIVNA, KRAJ_LISTE, 0, 0, NP_Dretva2
    DCD 0,1,2,3,4,5,6,7,8,9,10,11,12
    DCD Stog_Dretva2, 14, Dretva_2_start, 0x00000010

Dretva_3
    DCD 3, 50, PRIPRAVNE, Dretva_1, 0, 0, NP_Dretva3
    DCD 1,1,2,3,4,5,6,7,8,9,10,11,12
    DCD Stog_Dretva3, 14, Dretva_3_start, 0x00000010

Dretva_IRQ ; za obradu prekida
    DCD 4, 1, RED_UVJETA, KRAJ_LISTE, 0, 0, KRAJ_LISTE
    DCD 1,1,2,3,4,5,6,7,8,9,10,11,12                  ; r0=..=r12=0
    DCD Stog_DretvaIRQ, 14, Istek_vremena, 0x00000010 ; počinje s f-jom "Istek_vremena"


;Monitori s podacima (i inicijalnim vrijednostima)
Monitor
    DCD Mon1
    DCD Mon2
    DCD Mon3

; Monitor za IRQ dretve
Mon1
    DCD MONITOR_NITKO
    DCD KRAJ_LISTE      ; red
    DCD Dretva_IRQ      ; red uvjeta 1 - za IRQ dretvu

; Monitor za kašnjenje
Mon2
    DCD MONITOR_NITKO   ; vlasnik
    DCD KRAJ_LISTE      ; red
    DCD KRAJ_LISTE      ; ne koristi se
    DCD KRAJ_LISTE      ; red uvjeta 1 - za 1. dretvu
    DCD KRAJ_LISTE      ; red uvjeta 2 - za 2. dretvu
    DCD KRAJ_LISTE      ; red uvjeta 3 - za 3. dretvu

; Monitor za korisničke dretve
Mon3
    DCD MONITOR_NITKO   ; vlasnik
    DCD KRAJ_LISTE      ; red
    DCD KRAJ_LISTE      ; red uvjeta 1

Monitor_IRQ         EQU 0
Monitor_Zakasniti   EQU 1
Monitor_1           EQU 2

Prioriteti
    DCD 5
    DCD 10
    DCD 20

;;;;;;;;;;;;;;; Stog ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Stog sustava
Stog_start
    SPACE   STOG_SYS_VEL-4
Stog_sustav                     ; STOG_SVC_VEL
    SPACE   STOG_USR1_VEL
Stog_Dretva1
NP_Dretva2
    SPACE   STOG_USR1_VEL
```

```
Stog_Dretva2
NP_Dretva3
    SPACE   STOG_USR1_VEL
Stog_Dretva3
    SPACE   STOG_USR1_VEL
Stog_DretvaIRQ  DCD 0


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;   kôd dretvi

    AREA dretve_code, CODE, READONLY, ALIGN=3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; IDLE dretva - Dretva 1

Dretva_1_start
    NOP
    B Dretva_1_start

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Dretva 2

Dretva_2_start
; Ući_u_monitor
    MOV     r0, #Monitor_1
    STMFD   sp!, {r0}
    SWI     Uci_u_monitor
    LDMFD   sp!, {r0}   ; povratna vrijednost

; Uvrstiti_u_red_uvjeta
    MOV     r0, #Monitor_1
    MOV     r1, #0      ; K=0
    STMFD   sp!, {r1}
    STMFD   sp!, {r0}
    SWI     Uvrstiti_u_red_uvjeta
    LDMFD   sp!, {r0}   ; povratna vrijednost

; Izaći_iz_monitora
    MOV     r0, #Monitor_1
    STMFD   sp!, {r0}
    SWI     Izaci_iz_monitora
    LDMFD   sp!, {r0}   ; povratna vrijednost

    B Dretva_2_start

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Dretva 3

Dretva_3_start
; Ući_u_monitor
    MOV     r0, #Monitor_1
    STMFD   sp!, {r0}
    SWI     Uci_u_monitor
    LDMFD   sp!, {r0}   ; povratna vrijednost

; Zakasniti
    LDR     r5, =Timer_vrijeme
    LDR     r5, [r5]
    MOV     r5, r5, LSR #2
    STMFD   sp!, {r5}
    BL      Zakasniti

Dretva_3_u_mon
; Osloboditi_iz_reda_uvjeta
    MOV     r0, #Monitor_1
    MOV     r1, #0      ; K=0
    STMFD   sp!, {r1}
    STMFD   sp!, {r0}
```

```
    SWI     Osloboditi_iz_reda_uvjeta
    LDMFD   sp!, {r0}    ; povratna vrijednost

; Izaći_iz_monitora
    MOV     r0, #Monitor_1
    STMFD   sp!, {r0}
    SWI     Izaci_iz_monitora
    LDMFD   sp!, {r0}    ; povratna vrijednost

    B       Dretva_3_start

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    END
```

POPIS LITERATURE

(Dostupnost dokumenata s Internet adresa provjerena u rujnu 2005.)

[Abd04] T.F. Abdelzaher, V. Sharma, C. Lu, *A Utilization Bound for Aperiodic tasks and Priority Driven Scheduling*, IEEE Trans. on Computers, Vol. 53, No 3, March 2004.

[Bar97] S.K. Baruah, J.R. Haritsa, *Scheduling for Overload in Real-Time*, IEEE Trans. on Computers, Vol. 46, No 9, Septemeber 1997.

[Bar99] M. Barr, *Programming Embedded Systems in C and C++*, O'Reilly & Associates, 1999.

[Bar03] S.K. Baruah, J. Goosens, *Rate-Monotonic Scheduling on Uniform Multiprocessors*, IEEE Trans. on Computers, Vol. 52, No 7, July 2003.

[Ber01] G. Bernat, A. Burns, A. Llamosi, *Weakly Hard Real-Time Systems*, IEEE Trans. on Computers, Vol. 50, No 4, April 1990.

[Bha03] N. Bhargava, *Linux as a RTOS*, Linux Gazette, Issue 96, 2003, http://www.linuxgazette.com/issue96/bhargava.html.

[Bin03] E. Bini, G.C. Buttazzo, G.M. Muttazzo, *Rate Monotonic Analysis: The Hyperbolic Bound*, IEEE Trans. on Computers, Vol. 52, No 7, July 2003.

[Bom92] A. Bomberger et al., *The KeyKOS NanoKernel Architecture*, Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, USENIX Association, April 1992.

[Bri72] P. Brinch Hansen, *Structured Multiprogramming*, Communications of the ACM, Vol. 15, Num. 7, July 1972.

[Bri73] P. Brinch Hansen, *Concurrent Programming Concepts*, Computing Surveys, Vol 5, No 4, December 1973.

[Bud99] L. Budin, L. Jelenkovic, *Time-Constrained Programming in Windows NT Environment*, IEEE International Symposium on Industrial Electronics ISIE'99, Vol. 1, pp. 90-94, Bled, 1999.

[Bud03] L. Budin, *Predavanja iz predmeta: Operacijski sustavi I*, skripta s predavanja, Zagreb, 2003.

[Buh95a] P.A. Buhr, M. Fortier, *Monitor Classification*, ACM Computer Surveys, 27(1):63-107, March 1995.

[Buh95b] P.A. Buhr, *Are safe concurrency Libraries Possible?*, Comm. of the ACM, 38(2), February 95, pp 117-120.

[Buh99] P.A. Buhr, *Advanced Exception Handling Mechanisms*, IEEE Trans. on Software Eng., Vol 26, No 9, October 2000.

[Bur01] A. Burns, A. Wellings, *Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX (3rd Edition)*, Addison Wesley, 3

edition, 2001.

[But99] G.C. Buttazzo, F. Sensini, *Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Real-Time Environments*, IEEE Trans. on Computers, Vol. 48, No 10, October 1999.

[Che94] D.R. Cheriton, K.J. Duda, *A Caching Model of Operating System Kernel Functionality*, Proceedings of the „First Symposium on Operating Systems Design and Implementation“, Usenix Association, November 1994.

[Chu90] J.Y. Chung, J.W.S. Liu, K.J. Lin, *Scheduling Periodic Jobs That Allow Imprecise Results*, IEEE Trans. on Computers, Vol. 39, No 9, Septemeber 1990.

[Cou71] P.J. Courtois, F.Heymans, D.L. Parnas, *Concurrent Control with „Readers“ and „Writers“*, Communications of the ACM, Vol 14, No 10, 1971.

[Cre04] T.L. Crenshaw, *CS423, Advanced Operating Systems – Research in Real-Time System Kernels*, http://netfiles.uiuc.edu/tcrensha/www/cs423.html, 2004.

[Dij65] E.W. Dijkstra, *Cooperating sequential processes*, Technical Report EWD-123, Technical University, Eindhoven, the Netherlands, 1965.

[Dij68] E.W. Dijkstra, *The Structure of the 'THE'-Multiprogramming System*, Communications of the ACM, Vol 11, No 5, May 1968.

[Eng95a] D.R. Engler, M.F. Kaashoek, *Exterminate all operating system abstractions*, Proc. of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V), Washington, 1995.

[Eng95b] D.R. Engler, M.F. Kaashoek, J. O'Toole, *Exokernel: an operating system architecture for application-level resource management*, In the Proc. of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, December 1995, pages 251-266.

[Eng98] D. Engler, *The Exokernel Operating System Architecture*, PhD thesis, Massachusetts Institute of Technology, 1998.

[Fur00] S. Furber *ARM – System-on-Chip Architecture*, Addison-Wesley, Harlow, 2000.

[Fur91] B. Furht, D. Grostick, D. Gluch, G. Rabbat, J. Parker, M. McRoberts, *Real-Time UNIX Systems: Design and Application Guide*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[Gol05] M. Golub, L. Jelenković, *Operacijski sustavi I - upute za laboratorijske vježbe*, http://www.zemris.fer.hr/predmeti/os1/, 2005.

[Hak91] W.A. Hakang, A.D. Stoyenko, *Constructing predictable real time systems*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[Ham95] M. Hamdaoui, P. Ramanathan, *A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines*, IEEE Trans. on Computers, Vol. 44, No 12, December 1990.

[Han96] C.C. Han, K.J Lin, C.J. Hou, *Distance-Constrained Scheduling and Its Applications to Real-Time Systems*, IEEE Trans. on Computers, Vol. 45, No 7, July 1996.

[Han03] C.C. Han, K.G. Shin, J. Wu, *A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults*, IEEE Trans. on Computers, Vol. 52, No 3, March 2003.

[Hoa74] C.A.R. Hoare, *Monitors: An Operating System Concept*, Communications of the ACM, Vol 17, No 10, October 1974.

[Hon89] J. Hong, X. Tan, D. Towsley, *A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System*, IEEE Trans. on Computers, Vol. 38, No 12, December 1989.

[How76] J.H. Howard, *Signaling in monitors*, Int. Conf. on Software Eng., Proc. of the 2nd Int. conf. on Software eng., San Francisco, 1976, pp. 47-52.

[Jak02] D. Jakobović, L. Jelenković, *The Forward and Inverse Kinematics Problems for Stewart Parallel Mechanisms*, 8th Int. Scientific Conference on Production Engineering, Brijuni, Croatia, 2002, pp. II-001 - II-012.

[Jak03] D. Jakobović, L. Jelenković, *Kinematic Evaluation and Forward Kinematic Problem for Stewart Platform Based Manipulators*, Proc. 1st Int. conf. on Computational Cybernetics, Siofok, Hungary, August 2003.

[Jel97] L. Jelenković, G. Omrčen-Čeko, *Experiments with Multithreading in Parallel Computing*, Proc. 19th Int. Conference ITI'97, Pula, Croatia, June 1997, pp.357-362.

[Jel98] L. Jelenković, J. Poljak, *Multithreaded Simulated Annealing,*, Proc. 20th Int. Conference ITI'98, Pula, Croatia, June 14-17 1998, pp.525-530.

[Jel02] L. Jelenković, L. Budin, *Error Analysis of a Stewart Platform Based Manipulators*, 6th Int. conf. on Intelligent Engineering Systems 2002, Opatija, Croatia, 2002.

[Jel04] L. Jelenković, D. Jakobović, L. Budin, *Hexapod Structure Evaluation as Web Service*, Proc. 1st Int. conf. on Informatics in Control, Automation and Robotics, Setubal, Portugal, August 2004.

[Jon97] M. Jones, *What really happened on Mars?*, http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

[Jos96] M. Joseph, *Real-time Systems, Specification, Verification and Analysis*, Prentice Hall International, London, 1996. (dostupna i na: http://www.onesmartclick.com/rtos/rtos.html)

[Kat95] D.I. Katcher, S.S. Sathaye, J.K. Strsnider, *Fixed Priority Scheduling with Limited Priority Levels*, IEEE Trans. on Computers, Vol 44, No 9, September 1995.

[Kee78] J.L. Keedy, *On Structuring Operating Systems With Monitors*, The Australian Computer Journal, Vol 10, No 1, February 1978.

[Kli05] R.M. Kline, *Producer/Consumer Algorithms*, 2005, http://www.cs.wcupa.edu/rkline/OS/PC-algorithms.html

[Kos73] S.R. Kosaraju, *Limitations of Dijkstra's semaphore primitives and Petri nets*, proc. of the 4th ACM Symposium on Operating Systems Principles, 1973, pp. 122-126.

[Kuo02] T.W. Kuo, W.R. Yang, K.J. Lin, *A Class of Rate-Based Real-Time Scheduling*

Algorithms, IEEE Trans. on Computers, Vol. 51, No 6, June 2002.

[Lab02] J.J. Labrosse, *MicroC/OS-II: The Real Time Kernel, 2nd edition*, CMP Books, San Francisco, 2002.

[Lau03] S. Lauzac, R. Melhem, D. Mosse, *An Improved Rate-Monotonic Admission Control and Its Applications*, IEEE Trans. on Computers, Vol. 52, No 3, March 2003.

[Li05] Q. Li, D. Rus, *Navigation Protocols in Sensor Networks*, ACM Trans. on Sensor Networks, Vol. 1, Num. 1, August 2005.

[Lie01] J. Liedtke, U. Dannowski, K. Elphinstone, G. Liefländer, E. Skoglund, V. Uhlig, C. Ceelen, A. Haeberlen, M. Völp, *The L4Ka Vision*, White Paper, April 2001, http://i30www.ira.uka.de/aboutus/people/personal/liedtke/

[Lie93] J. Liedtke, *Improving IPC by Kernel Design*, 14th ACM Symposium on Operating System Principles (SOSP), Asheville, NC, December 1993, (dostupno na http://i30www.ira.uka.de/aboutus/people/personal/liedtke/)

[Lie95] J. Liedtke, *On μ-Kernel Construction*, 15th ACM Symposium on Operating System Principles (SOSP), Copper Mountain Resort, CO, December 1995.

[Lie96a] J. Liedtke, *Toward Real μ-kernels*, Communications of the ACM, 39(9), October 1996, pp. 70-77.

[Lie96b] J. Liedtke, *Micro-Kernel Must and Can be Small*, IEEE International Workshop on Object-Orientation in Operating Systems, Seattle, WA, USA, October 1996.

[Lie97] J. Liedtke, *The Performance of μ-Kernel-Based Systems*, 16th ACM Symposium on Operating Systems Principles (SOSP '97), Saint-Malo, France, October 5–8, 1997.

[Lim03] G.M. de A. Lima, A. Burns, *An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems*, IEEE Trans. on Computers, Vol. 52, No 10, October 2003.

[Liu73] C.L. Liu, J.W. Layland, *Scheduling algorithms for multiprogramming in hard-real-time environment*, Journal of the ACM, 20(1), January 1973, pp. 46-61.

[Loc88] D. Locke, L. Sha, R. Rajkumar, J. Lehoczky, G. Burns, *Priority Inversion and Its Control: An Experimental Investigation*, Ada Letters, Vol. 8, No. 7, Special Edition, 1988.

[Loe92] K. Loepere, *Mach 3 Kernel Principles*, Open Software Foundation and Carnegie Mellon University, http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/doc/osf.html, 1992.

[Mar94] P. Martineau, M. Silly, *Scheduling in a hard real-time system with shared resources*, Proceedings of the Sixth Euromicro Workshop on Real-Time Systems, Real-Time Systems, Vaesteraas, Jun 1994.

[Ni02] N. Ni, L.N. Bhuyan, *Fair Scheduling in Internet Routers*, IEEE Trans. on Computers, Vol. 51, No 6, June 2002.

[Nic96] D.M. Nicol, R. Simha, D. Towsley, *Static Assigment of Stochastic Tasks Using Majorization*, IEEE Trans. on Computers, Vol. 45, No 6, June 1996.

[Nut00] G. Nutt, *Operating Systems: A Modern Perspective*, Addison-Wesley, Reading, 2000.

[Par75] D.L. Parnas, *On a Solution to the Cigarette Smoker's Problem (without conditional statements)*, Communications of the ACM, Vol 18, No 3, March 1975.

[Pro91] D. Probert, J.L. Bruno, M. Karaorman, *SPACE: A New Approach to Operating System Abstraction*, In proceedings of the International workshop on object-orientation in operating systems, Palo Alto, 1991.

[Raj91] R. Rajkumar, *Synchronization in real-time systems: A Priority Inheritance Approach*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[Rob00] S. Robbins, *Experimentation with bounded buffer synchronization*, Proc. of the thirty-first SIGCSE technical symposium on Computer science education, Austin, Texas, 2000, pp. 330-334.

[Rob01] S. Robbins, *Starving philosophers: experimentation with monitor synchronization*, Proc. of the 32nd SIGCSE technical symposium on Computer science education, Charlotte, North Carolina, 2001, pp. 317-321.

[Sha90] L. Sha, R. Rajkumar, J.P. Lehoczky, *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*, IEEE Trans. on Computers, Vol. 39, No 9, Septemeber 1990.

[Shi95] K.G. Shin, Y.C. Chang, *A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks*, IEEE Trans. on Computers, Vol. 44, No 12, December 1995.

[Sil94] A. Silberschatz, P.B. Galvin, *Operating system concepts*, Addison-Wesley, Reading, 1994.

[Ste94] D.B. Stewart, *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems*, PhD thesis, Carnegie Mellon University, 1994.

[Ste97] D.B. Stewart, *An I/O Device Driver Model and Framework for Embedded Systems*, In Proc. of Workshop on Middleware for Distributed Real-Time Systems, San Francisco, December 1997.

[Str95] J.K. Strosnider, J.P. Lehoczky, L. Sha, *The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments*, IEEE Trans. on Computers, Vol. 44, No 1, January 1995.

[Tan95] S. Tan, D. Raila, R. Campbell, *An Object-Oriented Nano-Kernel for Operating System Hardware Support*, Proc. of the International Workshop on Object-Orientation in Operating Systems, IWOOOS'95. IEEE, Coputer Society Press, 1995.

[Til91a] A.M. Tilborg, G.M. Koob, *Foundations of Real-Time Computing: Scheduling and Resource Management*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[Til91b] A.M. Tilborg, G.M. Koob, *Foundations of Real-Time Computing: Formal Specifications and Methods*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1991.

[Voe00] G. Voelker, *OS Modules, Interfaces, and Structure*, Principles of Computer

Operating Systems, http://www.cs.ucsd.edu/classes/fa00/cse120/, 2000.

[Voy02] R. Voyles, *Notes on Kernels*, CSci 5980 - Real-Time Systems Laboratory, http://www-users.itlabs.umn.edu/classes/Spring-2002/csci5980/, 2002.

[Whi03] B. White, *Linux 2.6: A Breakthrough for Embedded Systems*, LinuxDevices.com, 2003, http://www.linuxdevices.com/articles/AT7751365763.html

[Wol01] W. Wolf, *Computers as Components – Principles of Embedded Computing System Design*, Academic Press, 2001.

[Xin05] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, C. Gill, *Integrated Coverage and Connectivity Configuration for Energy Conservation in Sensor Networks*, ACM Trans. on Sensor Networks, Vol. 1, Num. 1, August 2005

[Zub01] K.M. Zuberi, *EMERALDS: A Small-Memory Real-Time Microkernel*, IEEE Trans.on Software Engineering, Vol. 27, No. 10, October 2001.

Popis Internet adresa

[1] *Embedded System*, http://en.wikipedia.org/wiki/Embedded_system

[2] *Embedded Systems Dictionary*, http://www.netrino.com/Books/Dictionary

[3] *The L4 μ-Kernel Family*, http://os.inf.tu-dresden.de/L4/

[4] *The Mach project*, http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

[5] *The Mach project* – objavljeni i neobjavljeni radovi, tehničke dokumentacije, knjige, http://www.cs.cmu.edu/afs/cs/project/mach/public/www/doc/documents_top.html

[6] *GNU Hurd*, http://www.gnu.org/software/hurd/hurd.html

[7] *OSEK/VDX Operating system specification 2.1*, http://www.osek-vdx.org/

[8] *The Single UNIX Specification*, Version 3 (POSIX), http://www.unix.org/version3/

[9] *Neutrino Realtime Operating System*, http://www.qnx.com/products/rtos/

[10] *The Micro-Choices Operating System*, http://choices.cs.uiuc.edu/uChoices.htmls

[11] *Windows CE .NET*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/cmconAboutWindowsCE.asp

[12] *ARM Documentation*, http://www.arm.com/documentation/

[13] *ARM Instruction Set Information* http://www.arm.com/documentation/Instruction_Set/

[14] *RealView ARMulator ISS – User Guide*, Version 1.4, ARM, 2002, http://www.arm.com/documentation/Software_Development_Tools/

[15] *Wind River VxWorks*, http://www.windriver.com

[16] *VxWorks 5.4 Programmer's Guide*, Wind River, http://www.windriver.com

[17] *The Tanenbaum-Torvalds Debate*, http://www.oreilly.com/catalog/opensources/book/appa.html

[18] *Linux*, http://www.linux.org/

Popis korištenih oznaka i kratica

D_i oznaka dretve s indeksom „i"
D_Q dretva koja čeka u redu uvjeta
D_E dretva koja čeka na ulaz u monitor
D_A aktivna dretva unutar monitora
E ulazni red monitora
E_P prioritet ulaznog reda monitora
J oznaka zadatka
J_H blokirani zadatak s najvećim prioritetom
J_L oznaka zadatka koji se blokira
$p(J_H)$ prioritet od J_H
S oznaka binarnog semafora
S red signalnih dretvi monitora
S^* oznaka binarnog semafora najvećeg prioriteta koji je neprolazan
S_P prioritet reda signalnih dretvi monitora
T period ponavljanja
t_d trenutak dolaska (*arrival time*)
t_{mp} trenutak mogućeg početka (*ready time*)
t_p trenutak početka (*start time*)
t_z trenutak završetka (*completion time*)
t_{nz} trenutak nužnog završetka (*deadline*)
W red spremnih dretvi monitora
W_P prioritet reda spremnih dretvi monitora

EDF *Earliest Deadline First*
RMS *Rate Monotonic Scheduling*

SAŽETAK

U radu su razmatrani višedretveni ugrađeni sustavi s minimalnim sklopovljem, tj. s ograničenim spremničkim prostorom i jednostavnijim procesorima. Razmatran je način ostvarenja takvih sustava uporabom monitora kao jedinog sinkronizacijskog mehanizma. Među prikazanim modelima monitora odabran je i opisan onaj koji svojim svojstvima najbolje odgovara uporabi u ugrađenim sustavima. Uz sam monitor prikazane su dvije metode za rješavanje problema inverzije prioriteta koje se jednostavno ugrađuju u funkcije monitora. Korištenje odabranog monitora prikazano je na nekoliko klasičnih problema sinkronizacije dretvi. Prikazane su mogućnosti izgradnje bitnih funkcija operacijskog sustava korištenjem monitora. Funkcije za ostvarenje monitora osim u detaljnom opisnom jeziku napravljene su i u strojnom jeziku ARM7 procesora. Zasnivanje ugrađenog sustava na monitorima prikazano je na dva primjera. Analizom funkcija i njihovog ostvarenja u strojnom jeziku dobivena je kvantitativna ocjena predloženog rješenja, kako u zauzeću spremničkog prostora, tako i u trajanju funkcija. Rezultati istraživanja ukazuju da je rješenje s monitorima pogodno za sustave kojima nisu potrebni složeni raspoređivači dretvi. Pored toga, monitori značajno smanjuju potrebni spremnički prostor u usporedbi s rješenjima u obliku operacijskog sustava ili samo njegove osnovne jezgre, istovremeno nudeći vrlo moćno sredstvo sinkronizacije.

Ključne riječi: ugrađeni sustavi, višedretvenost, monitori, sinkronizacija, jezgra operacijskog sustava, minimalno sklopovlje, ARM7.

ABSTRACT

Monitor-based multithreaded embedded systems

Multithreaded embedded systems with limited memory and simpler processors were examined and methods of building such systems using only one synchronization mechanism – monitors are discussed. Different monitor concepts are shown and regarding their usage in embedded system the appropriate one is chosen. The selected monitor with and all necessary data structures is described along with two simple methods for solving priority inversions that can be simply built-in. Usage of the selected monitor is shown on several classical synchronization problems. Methods of implementing some core operating system functions with monitors are also presented. For analysis purposes monitors are implemented in assembly language for ARM7 processor. Building multithreaded embedded system based on monitors is shown on two examples. Quantitative analysis concerning needed memory, execution time of its functions is performed using monitor model and its implementation in assembly language. Results of analysis show that monitor concept is appropriate for systems that don’t need advanced task schedulers but need a powerful synchronization mechanism and that systems built only on monitors require significantly less memory in comparison to systems using an operating system or just some minimal kernel.

Keywords: embedded systems, multithreading, monitors, synchronization, kernel, minimal hardware, ARM7.

ŽIVOTOPIS

Leonardo Jelenković je rođen 1973. godine u Pazinu. Osnovnu školu pohađao je u Karojbi, a srednju školu u Pazinu. Godine 1992. upisuje se na Fakultet elektrotehnike i računarstva u Zagrebu, gdje je i diplomirao u prosincu 1996. godine s temom *Zbirka potprograma za praćenje izvođenja višedretvenog programa*.

U travnju 1997. primljen je kao znanstveni novak na Fakultet elektrotehnike i računarstva, Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave na radno mjesto mlađeg asistenta. U nastavi je sudjelovao na predmetima: Operacijski sustavi I i II, Analiza i projektiranje računalom, Automati, formalni jezici i jezični procesori I i II, Mreže računala i Digitalna elektronika.

Paralelno sa zaposlenjem upisao je poslijediplomski studij na istom fakultetu te je magistrirao u prosincu 2001. godine s temom *Analiza i vrednovanje Stewartovih paralelnih mehanizama*.

U svom znanstvenom radu bavi se područjem paralelnog računanja, višedretvenošću, operacijskim sustavima, ugrađenim sustava, metodama optimiranja, analizom paralelnih kinematskih struktura. Kao suradnik sudjelovao je na projektima *Računalna potpora rješavanju inženjerskih zadataka* (036014) i *Raspodijeljeni ugrađeni računalni sustavi* (0036051) te kao sunositelj na projektu *Programska podrška modeliranju Stewartovih paralelnih mehanizama* (IP 2002-067).

U tijeku svog znanstvenog rada objavio je nekoliko članaka na domaćim i inozemnim znanstvenim skupovima.

BIOGRAPHY

Leonardo Jelenkovic was born in Pazin, 1973. He went to elementary school in Karojba and the high school in Pazin. In 1992 he enrolled the Faculty of Electrical Engineering of the University of Zagreb and received BS degree in December 1996 in Electrical Engineering with thesis *Functions for tracking multithreaded programs*.

In April 1997 he joined Department of Electronics, Microelectronics, Computer and Intelligent Systems of Faculty of Electrical Engineering and Computing, Zagreb as research assistant. In undergraduate program he was involved with courses: Operating systems I and II, Computer-Aided Analysis and Design, Automata Theory, Formal Languages and Compiler Design I and II, Computer Networks and Digital Electronics.

He received MS degree in December 2001 in Computer science with thesis *The evaluation and analysis of Stewart parallel mechanisms*.

His research interests include areas of parallel computing, multithreading, operating systems, embedded systems, optimization methods, parallel kinematic structures. He was involved in several research projects supported by the Ministry of Science and Technology of the Republic of Croatia: *Problem-Solving Environments in Engineering* (036014), *Distributed embedded systems* (0036051) and *Computer Aided Modeling of Stewart Parallel Mechanisms* (IP 2002-067).

His publications include several conference papers from mentioned research areas.