Operacijski sustavi - uvod

Dijelovi teksta iz knjige:

Sustavna programska potpora - RADNI MATERIJAL,
L. Jelenković i ostali, Algebra, 2010

(pripremljeno za srednje tehničke škole – računarstvo)

Zagreb, 2010

1. Operacijski sustavi

1.1. Računalni sustav

Računala znatno pomažu čovjeku jer neke poslove za koje bi čovjeku trebalo mnogo vremena računala mogu napraviti gotovo trenutačno. Jednako tako, računala podižu i kvalitetu života, upravljajući jednostavnijim poslovima (gdje su znatno preciznija od čovjeka), dodajući u svakodnevni život sadržaje kao što su glazba, video, igre i slično.

Iako računala mogu znatno brže obavljati matematičke i logičke operacije od čovjeka, u mnogim mu područjima još nisu dorasla. Npr. u području razumijevanja govora i teksta, analiza slika i ostaloga što spada u tzv. područje *umjetne inteligencije*.

Način rada računala znatno se razlikuje od načina rada i razmišljanja čovjeka. Da bi čovjek mogao iskoristiti računalo, mora mu znati određenu naredbu ili zadatak naložiti onako kako će računalo to moći izvesti. Znamo da se taj način ostvaruje *instrukcijama* (uputama) koje računalo izvodi jednu za drugom. Čovjek koji koristi računalo ipak ne komunicira s računalom izravno zadajući takve instrukcije jer bi i za gotovo najmanji posao svaki put trebao dati mnogo instrukcija. Uobičajeni se poslovi stoga prethodno pripreme u obliku *slijeda instrukcija* koje nazivamo *programima*. Programi se pohranjuju u računalu te ih čovjek po potrebi *pokreće*.

Instrukcija je niz bitova koje *procesor* zna prepoznati i izvesti. Primjeri instrukcija uključuju:

- *prijenos podataka* između procesora (njegovih *registara*) i *glavnog spremnika*,
- *obavljanje matematičkih operacija* nad ulaznim operandima (registrima),
- ispitivanje uvjeta i skok na zadanu instrukciju (npr. ostvarenje petlji, potprograma).

Programi mogu biti razni: od uređivača teksta (npr. za pisanje ovog teksta korišten je jedan), preglednika *web*-sadržaja, programa za reprodukciju i stvaranje multimedijalnih sadržaja, igara, sredstva za komunikaciju i dr.

Čovjek s računalom komunicira preko vanjskih elemenata računala, kao što su tipkovnica, miš i zaslon kod uobičajenog *osobnog računala*. Računalo raspoznaje *naredbe* koje mu čovjek zadaje preko tog *sučelja* te pokreće odgovarajuće aktivnosti. Primjerice, pritiskom na *ikonu* nekog programa taj će se program pokrenuti, tipkanjem po tipkovnici u odabranom će se prozoru pojaviti zadani tekst i sl. Povratnu informaciju čovjek dobiva preko *izlaznih jedinica* kao što su zaslon, zvučnici i pisač.

Programi za svoj rad trebaju i odgovarajuće podatke, a ne samo instrukcije. Podatci mogu biti pohranjeni dijelom uz sam program (kao jedna cjelina) ili u dodatnim zasebnim cjelinama (datotekama), ili ih korisnici trebaju unijeti pri pokretanju. Uobičajeno se i program i podatci s kojima programi rade *trajno* zapisuju u prikladne spremnike podataka (*memorija*, engl. *storage*, *memory*).

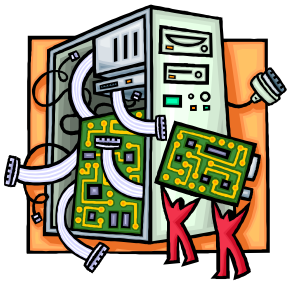
Osobno računalo (engl. *personal computer*) je računalo namijenjeno radu (ili razonodi) jednog korisnika.

Uobičajeni spremnik podataka, koji zadržava sadržaj i nakon *isključivanja* računala, jest *disk* (*tvrdi disk* od engl. *hard disk*). Osnovna jedinica podataka na disku (s motrišta korisnika i njegovih programa) je *datoteka* (engl. *file*). Na primjer, jedan program može biti smješten u jednoj datoteci. Jedan zvučni zapis (jedna pjesma) smješten je u jednoj datoteci. Cijeli album smješten je u nekoliko datoteka.

Poslužitelj (engl. *server*) je računalo koje istodobno koristi više korisnika (npr. *Web*-poslužitelj).

Organizacija datoteka, koju čovjek može upamtiti i jednostavnije koristiti, sastoji se od niza hijerarhijski povezanih *direktorija* (*kazala*, engl. *directory, folder*). Jedan direktorij može sadržavati više datoteka, ali i drugih *poddirektorija*. Na taj se način datoteke logički mogu smjestiti u čovjeku razumljivim mjestima u hijerarhijskom stablu datoteka i direktorija. Na primjer, programi se mogu smjestiti u direktorij *programi*, glazbene datoteke

u *glazba*, videosadržaji u *video*, igre u *igre* itd.



U prethodnim su razmatranjima dotaknuta samo neka motrišta računala, njegovih programa i njegove interakcije s korisnikom koji zajedno čine ***računalni sustav***. Računalni sustav je vrlo složen sustav, s mnogo elemenata, načina njihova rada i sl. Mnoga se pitanja mogu postaviti, primjerice poput: kako izgleda i radi sklopovlje (procesor, spremnik, sabirnice, ulazno-izlazne naprave, disk...), kako se ostvaruje sučelje prema čovjeku (tipkovnica, miš, zaslon, način pokretanja programa i sl.), kako se datoteke stvarno smještaju na disk (njihovu logičku organizaciju i prezentaciju korisniku znamo), kako više programa može raditi istodobno, samo su neka od njih. U nastavku se pokušava objasniti dio tih pitanja.

Osim programa koji obavljaju zadane operacije, u računalnom sustavu moraju postojati mehanizmi za pokretanje takvih programa što minimalno uključuje:

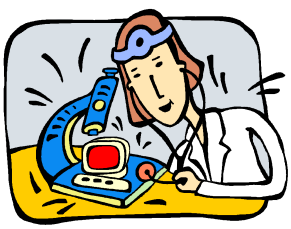
- rezervaciju spremničkog prostora za program,
- učitavanje programa s diska u spremnik,
- pokretanje programa,
- omogućavanje interakcije korisnika s programom,
- zaštitu pokrenutog programa od već prisutnih programa u sustavu i obrnuto,
- i sl.

Takvi mehanizmi ostvaruju se *pomoćnim programima* koji omogućuju izvođenje operacija na računalu, a koji objedinjeni čine ***operacijski sustav***.

Zadaća operacijskog sustava jest *upravljanje sustavom*, od upravljanja datotekama na disku, spremnikom i ostalim elementima sustava do upravljanja programima, od njihova pokretanja, komunikacije s korisnikom, drugim programima te sklopovljem. Osim samog upravljanja sustavom, zadaća je operacijskog sustava i *olakšati korištenje računala* tako da se i korisnicima i programerima sakrije složenost samog sklopovlja korištenjem standardiziranog sučelja kojim se ono može jednostavnije iskoristiti za obavljanje uobičajenih operacija.

Da bi mogao upravljati takvim složenim sustavom, operacijski je sustav vrlo složen. No ta se složenost ne mora savladati za njegovo korištenje.

Za *običnog korisnika* bitno je da zna koristiti računalni sustav, tj. da zna koristiti njegovo sučelje, kako tipkovnicu i miša, tako i njihovu povezanost s *grafičkim* korisničkim sučeljem koje mu sustav prikazuje, da može pokretati programe, koristiti programe i sl. Takav se korisnik, međutim, pri odabiru računalnog sustava koji će zadovoljiti njegove potrebe mora osloniti na mišljenje i preporuke stručnjaka iz tog područja koji znaju procijeniti koji su sklopovski i programski elementi korisniku potrebni (dovoljni).



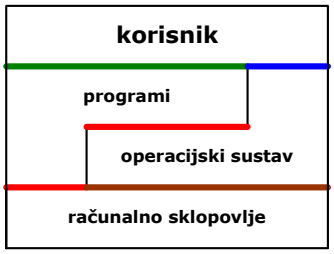
S druge strane, za osoblje koje održava računalne sustave potrebno je dublje poznavanje. *Programeri* koji razvijaju nove programe moraju znati još više detalja. Inženjeri koji osmišljavaju i unaprjeđuju računalne sustave moraju znati gotovo sve detalje, posebnosti i mogućnosti svih elemenata sustava.

1.2. Slojevi i sastavnice

Složeni se sustavi, kao što je i operacijski sustav, grade u sastavnicama (komponentama), a također i u slojevima (i s više motrišta).

Na primjer, *računalni sustav* može se podijeliti na četiri sloja (ili sastavnice):

- *korisnik* – bez njega sustav nema smisla,
- *primjenski programi* – definiraju kako obaviti određene operacije,
- *operacijski sustav* – upravlja sklopovljem i programima i pojednostavljuje uporabu sklopova nudeći korisniku i programima standardizirano sučelje te
- *računalno sklopovlje* – obavlja sve zadane instrukcije.



Slika 1.1
Slojevi računalnog sustava

Između svaka dva sloja nalazi se prikladno *sučelje*. Primjerice, korisnik programe koristi preko korisničkog sučelja, operacijski sustav preko njegovog sučelja (grafičkog korisničkog sučelja, engl. *graphical user interface – GUI*). Programi većim dijelom koriste sučelje sklopovlja (izvođenje instrukcija programa), a tek dijelom koriste sučelje operacijskog sustava (engl. *application programming interface – API*) kada je potrebno obaviti neke posebne (privilegirane) operacije.

S druge strane, operacijski sustav upravlja svim elementima sustava te za gotovo svaki element ima zasebnu sastavnicu, tj. Podsustav pa tako postoje podsustavi za:

- upravljanje datotekama (*datotečni podsustav*),
- upravljanje ulazno-izlaznim napravama (*UI podsustav)*,
- upravljanje programima (*procesima* i *dretvama*),
- upravljanje spremnikom (*spremnički podsustav*),
- upravljanje povezivanjem i komunikaciju s drugim sličnim sustavima (*mrežni podsustav*),
- upravljanje sigurnošću (zaštita korisnika i njegovih podataka),
- korisničko sučelje.

Slika 1.2
Elementi operacijskog sustava

Svaki se podsustav može ostvariti na razne načine, korištenjem raznih algoritama i struktura. Današnji operacijski sustavi većinu zadaća rade slično jer su se s vremenom istaknuli i usvojili oni postupci koji pokazuju bolja svojstva u pogledu učinkovitosti (npr. s motrišta brzine rada, zauzeća spremničkog prostora, lakoće korištenja i sl.). Ipak, i dalje postoje neke razlike koje mogu biti izravno vidljive korisniku kroz različita korisnička sučelja, ali i mogu biti skrivene od njega, s obzirom da su ugrađene u neke postupke.

Najkorišteniji operacijski sustavi za *osobna računala* i *poslužitelje* (u ovim prostorima) su razne inačice *Microsoft Windows* operacijskih sustava, *Mac OS X*, tvrtke *Apple Inc.* te besplatni operacijski sustavi zasnovani na *Linux* jezgri, npr. *Ubuntu*. Drugi tipovi računala imaju druge operacijske sustave. Primjerice, novije generacije mobilnih telefona i ručnih računala pokretani su *Windows Mobile*, *Android*, *Symbian*, *BlackBerry*, *MeeGo* i sličnim operacijskim sustavima.

Bez obzira na namjenu, u svim operacijskim sustavima osnovna je funkcionalnost istoimenih podsustava slična. U nastavku slijedi kratak opis podsustava, a u sljedećim će se poglavljima

Među ***Windows*** operacijske sustave koji se danas koriste ubrajamo *Windows 7*, *Windows Vista* i *Windows XP* za osobna računala te *Windows 2003* i *Windows 2008* za poslužitelje.

Osim navedenog *Ubuntua*, danas postoji mnoštvo operacijskih sustava zasnovanih na ***Linux*** jezgri (i sličnim jezgrama s slobodno dostupnim izvornim kodom). Na primjer, tu se mogu ubrojiti: *Fedora*, *CentOS*, *openSUSE*, *Debian*, *Knoppix*, *Slackware*, *BSD* i sl.

neki od njih prikazati detaljnije.

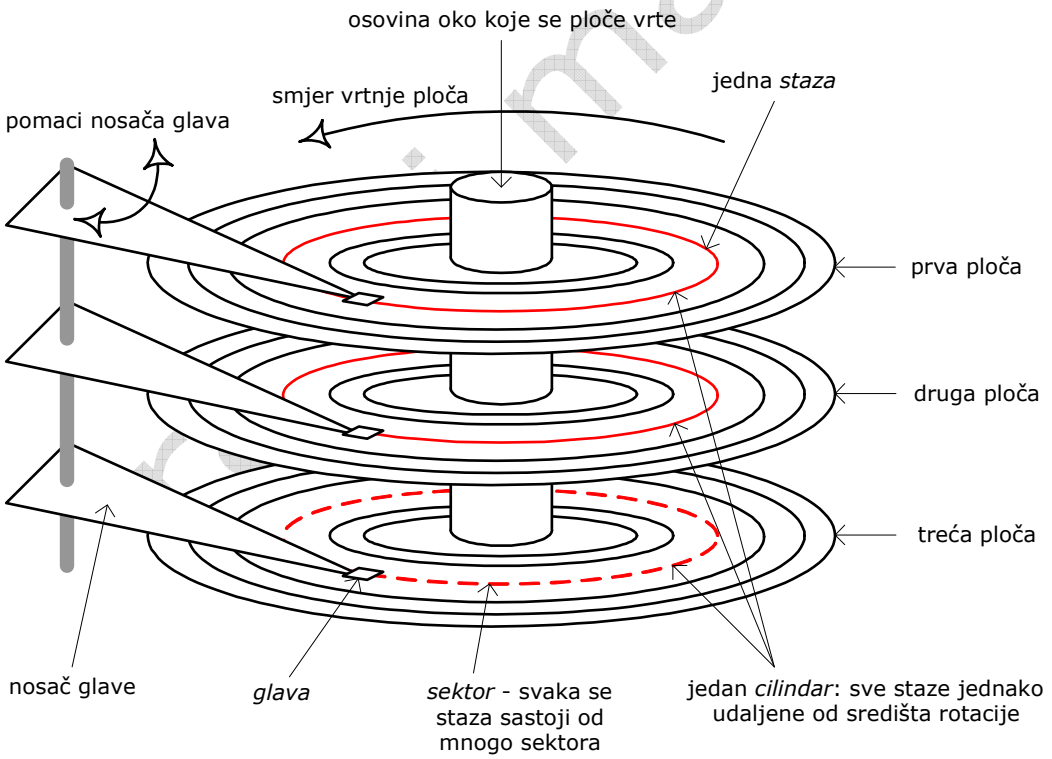
1.3. Datotečni podsustav

Upravljanje datotečnim sustavima vrlo je bitna sastavnica sustava jer su gotovo sve informacije zapisane u datotekama: programi, njihovi ulazni podatci, a i rezultati rada najčešće se zapisuju u datoteke. Programi koji sačinjavaju operacijski sustav također su pohranjeni kao datoteke u datotečnom sustavu.

Zadaća *datotečnog podsustava operacijskog sustava* jest da upravlja i omogući korištenje raspoloživih *datotečnih sustava* (npr. oni koji se nalaze na disku, DVD mediju i sl.).

Datotečni sustav određuje način organizacije i pohrane datoteka na disk. Zato je potrebno razumjeti osnovna svojstva samog *diska*.

Disk je elektromehanička naprava koja se sastoji od elektroničkog i mehaničkog dijela. Mehanički dio sastoji se od nekoliko magnetskih ploča na koje se zapisuju podatci. Postupak zapisa temelji se na postavljanju magnetske orijentacije malim elementima površine ploča, koji se tada mogu i detektirati i interpretirati kao nule ili jedinice (jedan ili drugi smjer magnetske orijentacije). S obzirom da se uređaj koji čita ili piše po magnetskoj ploči mora fizički postaviti iznad odgovarajućeg dijela ploče, svojstva diska su značajno lošija od ostalih tipova spremnika koji su isključivo elektronički. Razlika u vremenima pristupa podacima je i do milijun puta!



Slika 1.3 Osnovni elementi pomičnog dijela diska

Tipična organizacija diska sastoji se od nekoliko ploča s magnetskim materijalom na jednoj ili obje strane, koje su učvršćene središnjom osovinom oko koje se vrte. Za čitanje se koristi

glava (zasebna za svaku površinu), koja radijalno obilazi ploču (koja se vrti neprestanom brzinom). S obzirom na fizičku organizaciju, i podatci su radijalno organizirani. Osnovna jedinica podataka naziva se *sektor* i smještena je na jednoj *stazi* jedne *površine*. Na istoj se stazi nalaze i drugi sektori, a na istoj površini i druge staze sa sektorima. Staze koje su isto udaljene od osi vrtnje, a nalaze se na različitim površinama, nazivaju se *cilindri*. Jedan cilindar sadržava sve staze koje se mogu dohvatiti bez pomicanja glave (sve su glave učvršćene za isti nosač).

Adresa pojedinog sektora određuje se površinom na kojoj se nalazi, rednim brojem staze na toj površini te rednim brojem sektora na toj stazi. Današnji se diskovi ipak prema ostatku sustava predstavljaju i na jednostavniji način, kao linearni poredak sektora, počevši od prvog do zadnjeg. Pretvaranje tog rednog broja u broj površine, staze i sektora radi upravljačka elektronika diska.

Ipak, ono što je bitno zapamtiti jest da je disk bitnim dijelom mehanička naprava te kada se želi učinkovito koristiti, podatke koji su povezani (čine jednu datoteku) treba pohraniti na susjednim sektorima i stazama, minimizirajući mehaničke pokrete pri njihovom čitanju i pisanju. To načelo koriste i datotečni sustavi i operacijski sustavi koji njima upravljaju.

Način ostvarenja datotečnog sustava, tj. odabir struktura podataka i njihove organizacije i smještaja na disku uvelike određuje njegova svojstva. Danas se može izdvojiti nekoliko popularnijih datotečnih sustava: *FAT32*, *NTFS*, *ext4*.

Tablica 1.1
Najčešći datotečni sustavi

Tip	Primjena
FAT, FAT32	stariji sustavi zasnovani na nekom *Windows* OS-u, spremničke kartice
NTFS	noviji sustavi zasnovani na *Windows* OS-u
ext2, ext4	sustavi zasnovani na *Linux* jezgri
ISO 9660, UDF	CD i DVD mediji

Nekad glavni datotečni sustav *Windows* operacijskih sustava – FAT32 (engl. *File Allocation Table*) danas se rjeđe koristi kao glavni datotečni sustav na disku, ali zbog svoje jednostavnosti češće se koristi za pomoćne spremnike manjih kapaciteta (npr. spremničke kartice za mobilne uređaje, telefone, fotoaparate i sl.). Osnovne mane današnje primjene u osobnim i poslužiteljskim računalima su nedostatak podrške za višekorisnički rad (i prikladne zaštite podataka), nepostojanje naprednih metoda zaštite i upravljanja sadržajem (npr. dnevnički način rada, engl. *journaling*).

NTFS (engl. *New Technology File System*) pojavljuje se s operacijskim sustavom *Windows NT*, a koriste ga svi noviji operacijski sustavi iz porodice *Microsoft Windows* (inačice od NT, 2000, XP, 2003, Vista, 2008, 7). On ispravlja nedostatke *FAT32* i donosi potrebne funkcionalnosti u datotečni sustav. Zbog rasprostranjenosti operacijskih sustava koji ga koriste, *NTFS* je danas najčešće korišteni datotečni sustav.

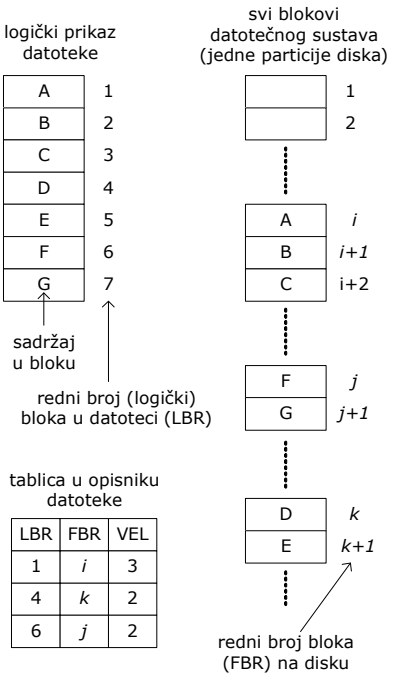
Linux operacijski sustavi koriste razne datotečne sustave, ali osnovni prepoznatljivi u njihovoj domeni svakako je *ext4* (engl. *Fourth Extended File System*) i njegove preteče *ext3* i *ext2*. Iako drukčije ostvarena, svojstva tog sustava slična su već spomenutom *NTFS*-u (kako u podržanim operacijama, tako i u performansama).

Datoteka je u datotečnom sustavu identificirana svojim *imenom* i položajem, tj. *direktorijem* u kojem se nalaze. Osim ovog osnovnog podatka, datotečni sustav pohranjuje i mnoge druge koji su mu potrebni za upravljanje. Neki od njih su:

- ime datoteke,

- veličina (zauzeće prostora na disku),
- vrijeme zadnje promjene sadržaja datoteke,
- podatci o pravima pristupa (vlasnik, tko joj smije pristupiti),
- tip datoteke (običan tekst, slika, video, ...),
- opis smještaja na disku.

Navedeni podatci smješteni su u *opisniku datoteke* koji je smješten u posebno područje na disku (npr. za *NTFS* to je glavna tablica datoteka, *MFT – Master File Table*). Svaka datoteke ima svoj opisnik.



Slika 1.4
Opis smještaja blokova datoteke za *NTFS* na primjeru

Sadržaj datoteke je, dakle, smješten na disku u nekim sektorima. Uobičajeno je da datotečni sustav ima svoju jedinku podataka koju nazivamo *blokom* (engl. *cluster*), a koja je jednaka veličini jednog, dva, četiri ili više sektora. Datoteka se tako pohranjuje u jedan ili više blokova. Optimalno bi bilo da su ti blokovi i na disku smješteni jedan do drugog, istim redoslijedom koji se oni nalaze i u datoteci (logičko uređenje). No to nije uvijek moguće. Zato dijelovi – blokovi datoteke mogu biti i razbacani po raznima stazama i na raznim površinama (različitih ploča). Čitanje takve datoteke bit će sporije od čitanja kompaktno smještene datoteke jer će biti potrebno znatno više pomicati nosač na kojemu se nalaze glave koje čitaju podatke s magnetskih površina. Taj problem nazivamo problemom *fragmentacije*. Operacijski sustavi nastoje umanjiti taj problem tako da datoteke smještaju kompaktno (zbijeno, čvrsto povezano) ili barem po dijelovima kompaktno. Npr. prvih *N* blokova smjeste kompaktno na jednu stazu, drugih *M* na drugu i tako dalje. Neki datotečni sustavi koriste navedeno svojstvo da učinkovitije opišu smještaj blokova koji čine jednu datoteku (npr. *NTFS*).

Korisnik se s datotečnim sustavom susreće kroz korisničko sučelje programa za upravljanje datotekama, koji mu prikazuje logičku organizaciju datotečnog sustava i omogućuje mu upravljanje nad njim (stvaranje, premještanje, kopiranje datoteka i direktorija).

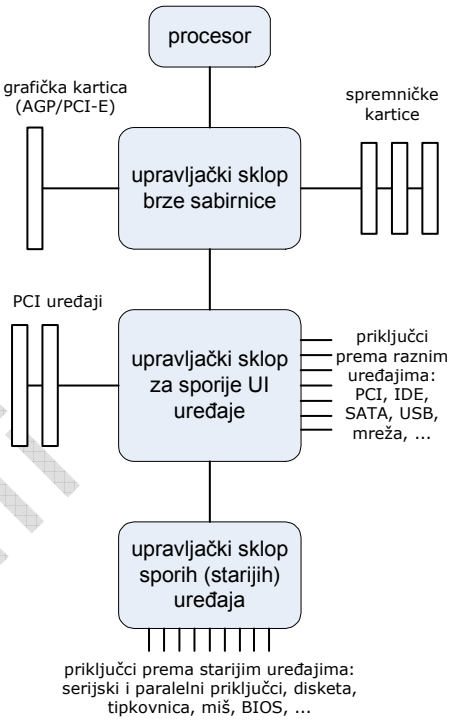
Drugi je način korištenja sustava, danas sve češći, korištenje *tražilice*. Umjesto pamćenja u kojem je direktoriju neki dokument, dovoljno je poznavati ime datoteke, ili čak i samo neke ključne riječi, a sustav pretraživanja će vrlo brzo ponuditi tražene datoteke.

Održavanje datotečnih sustava danas je gotovo automatizirano. Operacijski sustavi sami pokreću potrebne operacije, kao što je izrada sigurnosne kopije ključnih datoteka sustava, preslagivanje sadržaja na disku radi ubrzavanja pristupa podatcima, osiguravanje ravnomjernog korištenja svih dijelova diska (magnetske ploče imaju veliki, ali ipak ograničeni broj čitanja/pisanja) i sl. Ipak, od korisnika ili administratora očekuje se povremeno praćenje zapisa u dnevniku sustava, kako bi se na vrijeme otkrili problemi, poput pojave grešaka (npr. greške na određenim dijelovima – kada se podatci ne mogu ispravno pročitati).

1.4. Ulazno-izlazne naprave

Ulazno-izlazne naprave spajaju se na računalo preko vanjskih ili

unutarnjih priključaka. Primjer takvih naprava su tipkovnica, miš, zaslon, USB-naprave i pisač. U ulazno-izlazne naprave ubrajamo i računalne sastavnice koje se nalaze u računalu, npr. grafička kartica, disk, zvučni podsustav, mrežna kartica ili sklopovi na *matičnoj ploči* i sl. Da bi te naprave, koje se mogu i naknadno ugrađivati u računalo, ispravno radile, potrebno je poznavati način njihova rada, odnosno kako od njih nešto postići. Za svaku takvu napravu operacijski sustav ima *upravljački program* (u žargonu *drajver,* od engl. *device driver*). On definira sučelje za komunikaciju s napravom, potrebne strukture podataka, spremničke lokacije, vremenski slijed naredbi i sl. Upravljačke programe najčešće stvaraju proizvođači naprava (uz napravu dolaze i potrebni upravljački programi, ako uobičajeni, već prisutni s operacijskim sustavom nisu odgovarajući).



Slika 1.5
Uobičajeni način spajanja sastavnica računala

S obzirom da samo procesor izvodi instrukcije nekog programa, ulazno-izlazne naprave koriste mehanizam *prekida* da prekinu rad procesora i zahtijevaju obradu aktivnosti kojima su one uzrok. Na primjer, nakon što je disk dobio zahtjeve za nekim podatcima, trebat će mu neko vrijeme da ih dohvati. U međuvremenu procesor neće čekati, već će izvoditi neki drugi posao koji je spreman i ne ovisi o zahtijevanim podatcima. Kada disk konačno dohvati tražene podatke i učita ih u svoju priručni spremnik (ili izravno u glavni spremnik), on signalom prekida obavještava procesor da je njegov posao gotov. U obradi tog prekida omogućuje se nastavak poslova koji su te podatke zahtijevali, a disku se pošalju novi zahtjevi, ako takvi postoje.

Upravljanje napravama mehanizmom prekida podrobnije je objašnjeno u trećem poglavlju.

1.5. Upravljanje programima/procesima

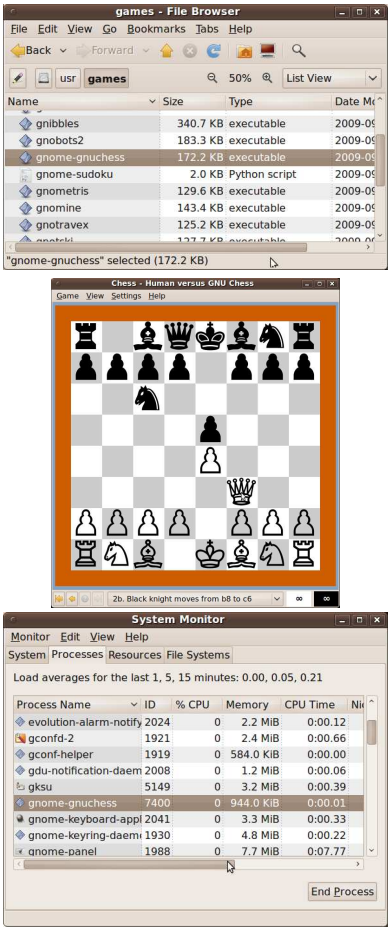
Programi koji su korisniku potrebni trebaju se najprije postaviti na sustav (*instalirati*). Sve se potrebne datoteke stave na odgovarajuća mjesta u datotečni sustav, registriraju za odgovarajući tip datoteka, postave prečicu za njihovo pokretanje i sl.

Pri pokretanju programa, operacijski sustav mora u spremnik istodobno smjestiti i instrukcije i podatke programa te započeti s njegovim izvođenjem. Od tog trenutka zapravo govorimo o *procesu* koji se izvodi, a koji pritom zauzima određena sredstva računalnog sustava: spremnik, procesorsko vrijeme, naprave (dok je program samo niz instrukcija).

Operacijski sustav mora osigurati da se svi procesi nesmetano izvode (da jedan ne smeta drugome i obrnuto). Problemi koji se zbog toga pojavljuju su u pristupu sredstvima sustava (osim spremnika).

Osnovno sredstvo sustava je procesor. Kako omogućiti da istodobno više procesa/programa dobije svoj dio procesorskog vremena (da se i on izvodi)? Uobičajeno rješenje koje koristi *raspoređivač poslova* koji upravlja dodjelom procesora jest metoda kružne podjele vremena (engl. *Round Robin*): svi procesi programi dobiju po djelić vremena, slijedno, jedan po jedan, kružnim slijedom.

Osim procesora, razni procesi programi mogu istodobno tražiti korištenje nekog drugog sredstva. Takvi se problemi rješavaju



Slika 1.6
Program* i *proces

korištenjem *sinkronizacijskih mehanizama*, kojima se sinkronizira pristup tim sredstvima (npr. jedan po jedan).

Procesi mogu međusobno željeti komunicirati radi ostvarenja svojih zadaća te im operacijski sustav i to mora omogućiti. Uobičajeni načini komunikacije su korištenjem poruka, cjevovoda, zajedničkih spremničkih lokacija, datotečnog sustava te mrežnog sustava (kada mogu komunicirati i programi s različitih računala).

Sve operacije koje mogu kompromitirati sustav ili druge programe zaštićene su tako da su ostvarene u *jezgri operacijskog sustava*. Kada neki program želi pozvati takvu operaciju, on pozove *funkciju jezgre* korištenjem mehanizma *programskog prekida*, tj. posebnom instrukcijom izaziva prekid. S obzirom da se obrada prekida obavlja u privilegiranom načinu rada, u jezgrinoj se funkciji obavlja zadana operacija, ako je dopuštena za pozivajući program.

Detaljniji opis dijela operacijskog sustava koji upravlja programima, tj. procesima (i dretvama) opisan je u četvrtom poglavlju.

1.6. Upravljanje spremnikom

Spremnici podataka mogu se podijeliti u nekoliko skupina:
- *glavni spremnik* (*memorija*) – kroz njega sve prolazi; drugi je po veličini (iza diska),
- *disk* – trajni spremnik podataka znatno većeg kapaciteta od glavnog spremnika, na kojem podatci ostaju sačuvani i nakon gašenja računala
- *priručni spremnici* – (engl. *cache*) služe za ubrzavanje rada pri korištenju sporijih spremnika:
  - uz procesor (L1, L2, L3),
  - uz naprave,
  - kao dio glavnog spremnika namijenjenog ubrzanju rada s nekom napravom (npr. diskom).

Gotovo sve aktivnosti računalnog sustava izvodi procesor. Zato se svi programi i njihovi podatci prije izvođenja moraju prvo dohvatiti i smjestiti u glavni spremnik računala. U spremniku se stoga u istom trenutku nalaze instrukcije i podatci više programa (procesa), podatci i programi operacijskog sustava, priručni spremnici za sporije naprave (npr. za disk) i dr.

Odgovor na pitanje: „Kako se sve smješta u zajednički spremnik, a da svi procesi obavljaju svoje poslove, da ne smetaju jedan drugome, da su zaštićeni jedni od drugih, da se učinkovito koristi spremnički prostor?“ daje podsustav za upravljanje spremnikom.

Upravljanje spremnikom u osobnim računalima i poslužiteljima obavlja se konceptom straničenja (engl. *paging*). Ukratko, u ovome načinu upravljanja spremnikom, procesi i njihovi podatci dijele se na *stranice*. Spremnik je također podijeljen na dijelove – *okvire* istih veličina (uobičajeno 4 KB). Sustav upravljanja mora znati povezati pojedinu stranicu programa s pripadnim okvirom gdje se ona nalazi. Sve stranice programa ne moraju istodobno biti prisutne u glavnom spremniku, već samo one koje su zaista potrebne. Time se postiže mogućnost da se u sustavima pokreću i programi koji su veći od raspoloživog spremničkog prostora (npr. računalo s 1 GB prostora u spremniku može izvoditi proces koji zahtijeva 2 GB spremničkog prostora).

Koji su ostali načini upravljanja spremnikom i zašto se baš straničenje koristi za upravljanje prikazano je u petom poglavlju.

1.7. Mrežni podsustav

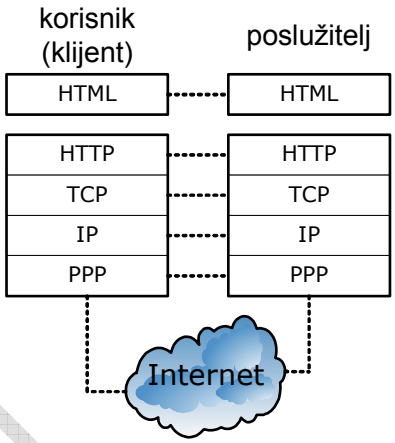
Današnja računala, pa i ona najmanja, gotovo su nezamisliva bez mogućnosti spajanja na Internet. Ako to ne mogu napraviti,

korisniku uskraćuju značajno velik broj usluga koje bi mogle pružiti.

Mrežni podsustav iznutra je gotovo složen kao i sam operacijski sustav. No njegovo korištenje nije potrebno posebno znanje – današnji operacijski sustavi sami otkrivaju moguće načine pristupa prema Internetu te se sami prilagođavaju toj vezi, a od korisnika jedino traže podatke za spajanje (korisničko ime i lozinku, ako je potrebno).

Korisnici koji žele sami podesiti mrežne postavke ili uspostaviti priključak na Internet, gdje automatski postupci operacijskih sustava nisu uspjeli, ipak trebaju znati nešto o konceptima mrežnog podsustava te načinu njegova podešavanja na operacijskom sustavu te možebitno i ostalim napravama (npr. preklopnicima i usmjerivačima).

Mrežni podsustav detaljnije je prikazan u šestom poglavlju.



Slika 1.7
Primjer korištenja protokola pri komunikaciji klijenta i poslužitelja

1.8. Sigurnost i privatnost

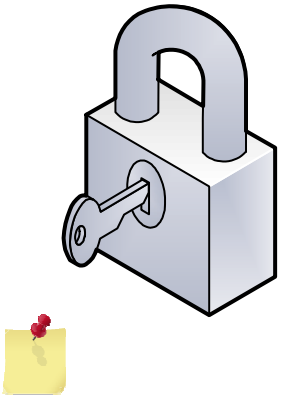
Današnji su računalni sustavi međusobno povezani infrastrukturom koju nazivamo Internetom. Zbog toga se u računalnim sustavima pojavljuje i problem sigurnosti i privatnosti.

Kako se zaštititi od zlonamjernih korisnika koji žele doći do naših podataka: pročitati ih, izmijeniti ili obrisati?

Opće načelo koje koriste operacijski sustavi je uskraćivanje pristupa podatcima onim korisnicima, tj. njihovim programima, koji za te podatke nemaju odgovarajuće ovlasti.

Za osiguravanje sigurnosti i privatnosti potrebno je u gotovo sve sastavnice operacijskog sustava ugraditi odgovarajuće mehanizme, počevši s datotečnim sustavom, upravljanjem spremnikom i procesima te zatim mrežnim podsustavom.

Iako u području sigurnosti ima znatnih napredaka u operacijskim sustavima, problem njegove složenosti i složenosti njegovih sastavnica onemogućuju potpuno siguran rad. Poneka je greška u sastavnicama neizbježna, a njih često pronalaze upravo zlonamjerni napadači koji tada prikladno oblikovanim programima iskorištavaju te otkrivene propuste radi stjecanja pristupa do informacija koje bi im sustav inače uskratio.



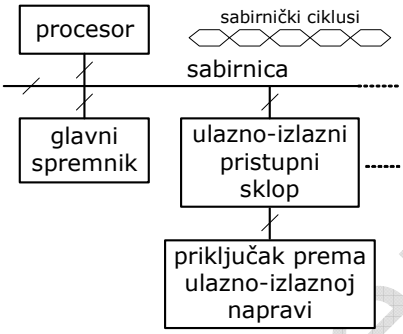
Operacijski sustavi, kao i drugi sustavi i programi, za zaštitu od neovlaštenog pristupa najčešće koriste *korisničko ime* i *lozinku*. Napredniji oblici zaštite uključuju korištenje *digitalnih certifikata* (koji se nalaze na zasebnim karticama) ili nekih biometrijskih značajki (npr. otisak prsta).

1.9. Korisničko sučelje

Računalni sustav korisnici uglavnom vide preko njegova korisnička sučelja. Uobičajena korisnička sučelja danas su grafička (*GUI*), gdje se pomoću miša (ili slične naprave) pomiče značka po zaslonu te pokreću i upravljaju potrebni programi. Zadaća korisničkih sučelja je da olakšaju rad korisniku, da on bude učinkovitiji pri obavljanju svog svakodnevnog posla ili da mu pruže više zadovoljstva pri korištenju računala za razonodu.

2. Prekidi

Današnja su računala *sabirnički* orijentirana. Umjesto da se svaka sastavnica izravno povezuje sa svakom drugom, računalo ima nekoliko sabirnica na koje su spojene razne naprave, prema njihovim svojstvima. Na taj se način znatno pojednostavljuje samo sklopovlje koje treba biti prilagođeno samo jednoj sabirnici na koju se priključuje. Standardizacija protokola koji se koriste za prijenos podataka preko sabirnice omogućuje modularnu izgradnju računalnog sustava, prilagođavanje računalnog sustava potrebama, odnosno, dodavanje samo potrebnih sklopovskih sastavnica.

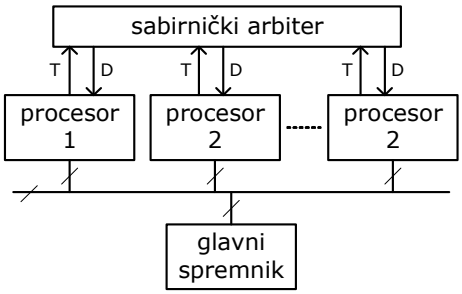


Slika 2.1 Konceptualna shema povezivanja sastavnica računala

Prijenos podataka preko sabirnice ide u vremenski sinkroniziranim sabirničkim ciklusima. Duljina sabirničkih ciklusa može biti određena signalom takta (*sinkrona sabirnica*) ili može biti varijabilne duljine, ovisno o uređajima koji ju trenutačno koriste, koji tada označavaju završetak sabirničke operacije (*asinkrona sabirnica*).

Upravljanje sabirnicom uobičajeno je dodijeljeno procesoru kao glavnoj sastavnici. Procesor započinje operaciju postavljanjem adrese i upravljačkih signala na sabirnicu, a naprava koja je adresirana dovrši operaciju. Na primjer, pri slanju podatka u glavni spremnik, procesor na sabirnicu postavi adresu spremničke lokacije, sam podatak te upravljački signal koji spremnik razumije kao *pohrani*. U nastavku te jedinične sabirničke operacije spremnik, koji cijelo vrijeme osluškuje što se na sabirnici događa, utvrđuje da je adresirana jedna njegova spremnička lokacija te prihvaća podatak, privremeno ga pohranjuje u svoje registre, a potom i na zadanu adresu.

U višeprocesorskim sustavima postoji dodatni sklop – sabirnički arbitar, koji omogućuje dijeljenje sabirnice među procesorima, tako da se ne dogodi konflikt pri korištenju sabirnice.



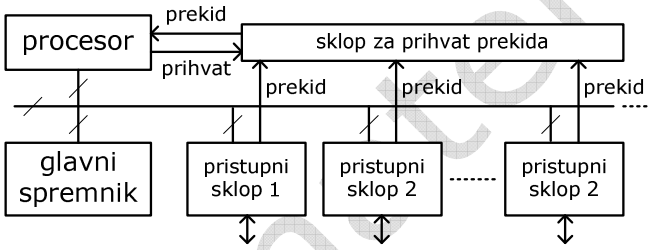
Slika 2.2 Sabirnički povezani višeprocesorski sustav

Sabirnički arbitar upravlja sabirnicom po načelu zahtjev-odgovor. Kada procesor treba koristiti sabirnicu, on najprije traži dozvolu od arbitra signalom *traži* (T, engl. *bus request*). Kada na njega dođe red, arbitar mu dodjeljuje sabirnicu i o tome ga obavještava signalom *dodjela* (D, engl. *bus acknowledge*).

Osim procesora, i drugi sklopovi koriste sabirnicu. U nekim im operacijama procesor izravno šalje podatke (kao što to radi sa spremnikom), u drugim operacijama procesor preuzima podatke i sl. Nekim napravama procesor šalje podatke koje naprava interpretira kao naredbe. Primjerice, pri radu s diskom procesor šalje disku zahtjeve za određenim blokovima podataka. S obzi-

rom da je disk znatno sporiji od glavnog spremnika, dok se podatci ne dohvate, procesor obrađuje one poslove (procese) koji ne zahtijevaju te podatke, tj. koji imaju sve što im je potrebno. *Kako će procesor saznati je li disk dohvatio tražene podatke u svoj međuspremnik ili u glavni spremnik?*

Jedan od načina kojim procesor može upravljati vanjskim napravama jest da periodički provjerava njihove *statusne registre*. Neke zastavice u tim registrima mijenjaju stanje kad se s napravom nešto dogodi. Na primjer, u statusnom bi se registru diska postavila zastavica „zahtjev obrađen" kada bi se s diska dohvatio traženi podatak. Pri otkrivanju (detekciji) te zastavice, procesor bi mogao nastaviti s procesom koji je te podatke i zahtijevao, a disku možda proslijedio sljedeći zahtjev. Periodička provjera mora biti programirana u sustav što značajno komplicira programsku izvedbu upravljanja sustavom. S druge strane, vrlo će rijetko takve provjere dati povod nekoj radnji s obzirom da je procesor znatno brži od ulazno-izlaznih naprava. Bolje bi rješenje bilo kada bi naprava sama dojavila da se s njom nešto dogodilo, a da procesor, dok se tako nešto ne dogodi, radi neki drugi koristan posao. Mehanizam koji se za to koristi jest mehanizam *prekida*.



Slika 2.3 Upravljanje UI napravama mehanizmom prekida

Svaka je naprava u takvom sustavu spojena na sklop koji upravlja prekidima, a koji takve prekide propušta do procesora. Procesor će, ako trenutačno ne obavlja neki kritični posao koji se ne smije prekidati (npr. obrađuje prekid neke druge naprave), privremeno prekinuti s onim što je do sada radio i *obraditi* pristigli prekid. Za ostvarenje navedenog, najčešće se koristi *stog* na koji se pohranjuje *kontekst* posla koji se privremeno prekida. Nakon obrade prekida s tog stoga obnavlja se kontekst i nastavlja se s prekinutim poslom.

Kako procesor doznaje koja je naprava izazvala prekid?

Uobičajeno se uz sam zahtjev za prekid procesoru šalje i *prekidni broj*. Korištenjem tog broja procesor (uz odgovarajuću inicijalizaciju sustava) može doznati koja je naprava generirala prekid i kako da ju obradi. Pri inicijalizaciji operacijskog sustava za sve je naprave pripremljen odgovarajuću upravljački program (engl. *device driver*). Pri obradi prekida koristi se (poziva se) upravljački program one naprave koja je prekid izazvala.

Pohrana konteksta, utvrđivanje koja je naprava izazvala prekid te naknadni povratak u prekinuti program (obnova konteksta) često se nazivaju *kućanskim poslovima* koji se pojavljuju pri svakom prekidu. U sustavima gdje se prekidi vrlo često pojavljuju, sami kućanski poslovi mogu značajno utjecati na učinkovitost sustava. Zato se za takve naprave koriste i drugi postupci.

Prihvat prekida kojeg izaziva neka naprava obavlja se u nekoliko koraka.
Procesor prije prihvata prekida završava trenutačnu instrukciju (ako ona nije uzrok prekidu).
Po završetku instrukcije procesor provjerava jesu li dopušteni prekidi i je li zahtjev za prekid postavljen. Tek ako su oba uvjeta ispunjena, kreće se u prihvat prekida.
Koraci u prihvatu prekida:
- zabrani daljnje prekidanje,
- ako je podržano, postavi procesor u prekidni način rada (privilegirani način rada),
- pohrani *programsko brojilo* na stog (adresa instrukcije na koju se treba vratiti nakon obrade prekida),
- u programsko brojilo stavi adresu procedure za obradu prihvaćenog prekida.

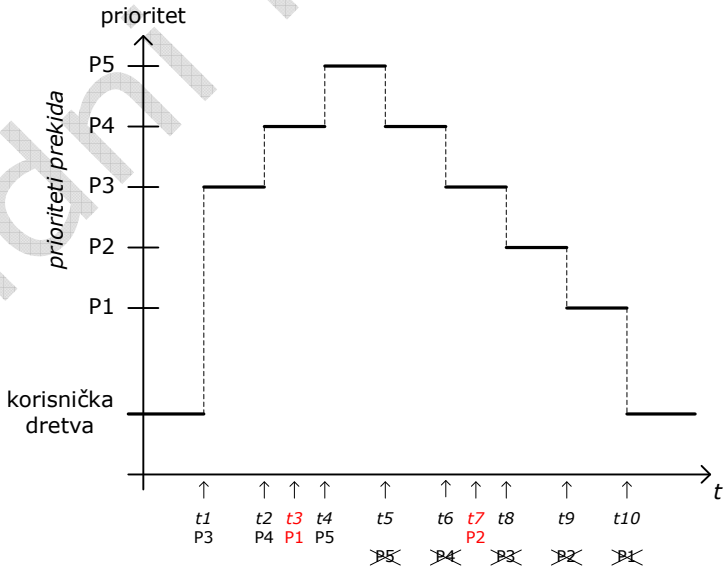
Po završetku obrade prekida potrebno je vratiti se u prekinutu dretvu.
Povratak iz obrade prekida obavlja se obrnutim redoslijedom od prihvata prekida:
- obnovi programsko brojilo sa stoga,
- prebaci procesor u prijašnji način rada,
- dopusti prekide.

Postupak prihvata prekida ugrađen je u ponašanje procesora – nije ga potrebno programski ostvariti (osim same obrade).
Povratak iz prekida zahtijeva izvođenje posebne instrukcije procesora koja je vrlo slična povratku iz potprograma uz dodatne funkcionalnosti prebacivanja procesora u prijašnji način rada.
Prije same obrade prekida potrebno je *pohraniti kontekst* prekinute dretve (sadržaje ostalih registara procesora, osim programskog brojila koje je automatski pohranjeno), a prije poziva instrukcije za povratak iz prekida potrebno je *obnoviti kontekst* prekinute dretve.

Neke naprave mogu povremeno i samostalno koristiti sabirnicu. Primjerice, pri radu s diskom s njega se učitavaju veće količine podataka. Kada bi upravljanje prenošenjem podataka s diska (njegove upravljačke elektronike, tj. njegovih međuspremnika) u glavni spremnik obavljao procesor, tada bi bilo mnogo neučinkovitog korištenja sabirnice: procesor bi svaki podatak najprije morao dohvatiti u svoj registar, a tek ga potom pohraniti u glavni spremnik. Uobičajeno je da se takvi poslovi mogu povjeriti i samoj napravi: ona će, kada želi koristiti sabirnicu, dati zahtjev sabirničkom arbitru (što u nekim sustavima može biti i dio procesora). Arbitar će odlučiti kad će toj napravi dodijeliti sabirničke cikluse, kada će naprava samostalno prenijeti podatke u glavni spremnik ili pročitati podatke iz njega. Takvim se operacijama glavnom procesoru uskraćuju samo poneki sabirnički ciklusi, ali se sustav učinkovitije iskorišćuje: osim dvostruko manjeg korištenja sabirnice za prijenos istih podataka s naprave u glavni spremnik, procesor se ne prekida u svom radu (prihvat prekida unosi dosta kućanskog posla!). U slučaju diska, on može generirati prekid tek kad prenese sve podatke koji čine jedan zahtjev i tek tada procesor može nastaviti s drugim procesom koji je te podatke zatražio.

2.1. Prioriteti prekida

Obrada nekog prekida može potrajati i poprilično dugo (u računalnom smislu). Za vrijeme obrade prekida, procesor može ili zabraniti prihvat drugih prekida ili ih dopustiti. Kada bi procesor sve prekide obrađivao slijedno, uz zabranu prihvata drugih prekida za vrijeme obrade prvog, tada bi neki prekidi mogli i poprilično dugo čekati na obradu. Zbog toga se koriste razni postupci, sklopovski ili programski, kojima se nastoji te probleme umanjiti.



Slika 2.4 Obrada prekida prema prioritetima

Sklopovski se raznim napravama mogu dodijeliti različiti prioriteti te se u sklopu koji upravlja prekidima (ako takav postoji) do procesora mogu propustiti samo oni prekidi koji imaju veći prioritet od onog koji se trenutačno obrađuje.

Prekidi koji se dogode za vrijeme obrade prioritetnijih prekida ili za za vrijeme kada su prekidi bili zabranjeni samo se zabilježe u sklopu koji upravlja prekidima. Ti se prekidi propuste do procesora onda kada on dovrši obradu prioritetnijih prekida ili onda kada on ponovno dopusti prihvat prekida.
U *registru stanja* procesora postoji (barem) jedna zastavica (engl. *Interrupt Flag*) koja definira prihvaćaju li se prekidi trenutačno ili ne.

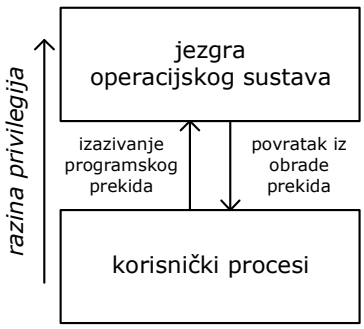
Obrada prekida u dva odvojena dijela na raznim se sustavima različito naziva.
Za *Windows* sustave nazivi su: prekidna procedura i prekidna dretva (engl. *Interrupt Service Routine – ISR* i *Interrupt Service Thread - IST*).
Za *Linux* sustave nazivi su: gornja polovica i donja polovica (engl. *Top Half* i *Bottom Half*: *Tasklet, Softirq, Workqueue*).

Programski se nastoji obrada prekida učiniti što kraćom, ako je to moguće. Ako nije, onda se obrada dijeli na nekoliko dijelova između kojih se obrada može prekidati. Na primjer, obrada se može podijeliti na dva dijela: prvi dio koji će samo pohraniti potrebne podatke za obradu (dohvatiti ih od naprave), a u drugom dijelu (koji se može i prekidati) dovršiti obradu. Na ovaj će se način prioritetniji prekidi moći prije obaviti, ako su zahtjevi za obradom (drugi dio obrade) složeni prema prioritetima.

Podjela prekidnih brojeva (engl. *Interrupt Number*) zadaća je operacijskog sustava i upravljačkih programa. Najčešće svaka naprava koristi vlastiti prekidni broj. No većina naprava može i dijeliti prekidni broj s drugom napravom, ako zbog sklopovskih okolnosti ne mogu dobiti zasebne (kada bi sustav bio učinkovitiji). Ponekad (u starijim sustavima) se ipak može dogoditi takozvani *konflikt*, kada dvije različite naprave žele koristiti isti prekidni broj, i to bez dijeljenja. Tada korisnik, korištenjem sučelja operacijskog sustava ili zasebnih programa koji su isporučeni uz naprave ili korištenjem BIOS-a prije pokretanja operacijskog sustava, treba pronaći alternativno rješenje za jednu od njih, u skladu s mogućnošću naprava.

2.2. Programski prekidi

Posebna vrsta prekida jesu *programski prekidi*. Operacijski sustav svoje operacije nudi korisniku kroz *korisničko grafičko sučelje*, dok ih programima nudi kroz *programsko sučelje* (*API*).



Slika 2.5
Privilegije jezgre i procesa

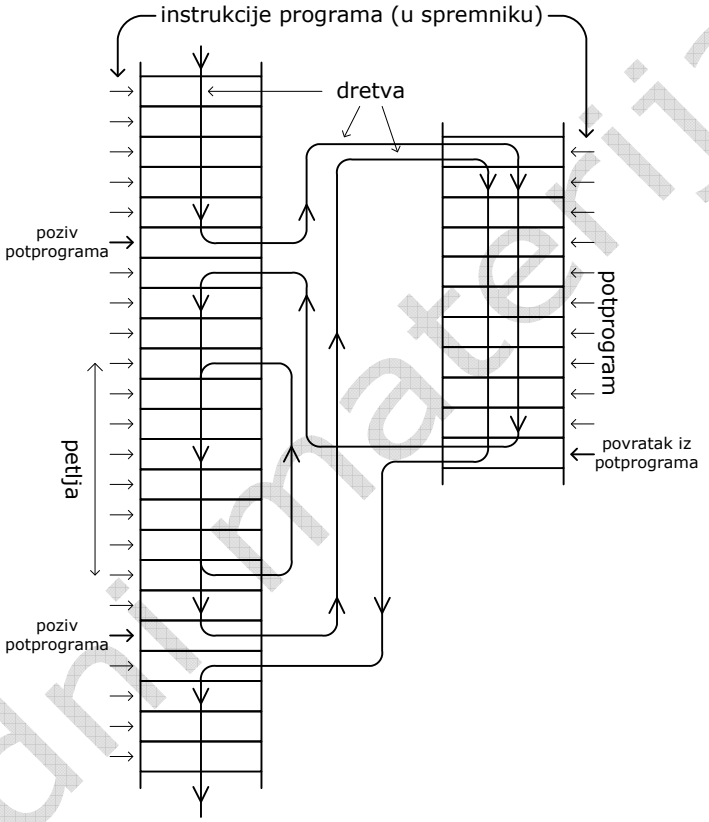
U izvođenju operacija koje se nude kroz programsko sučelje mogu biti i neki privilegirani ili zaštićeni podzadaci. U takvim se slučajevima za njihovo obavljanje koristi mehanizam *programskog prekida*.

Programski prekid, kao što i sam naziv govori, izaziva se programski, posebnim instrukcijama procesora. Prije izazivanja programskog prekida program mora u unaprijed definirane registre (ili na stog) postaviti parametre za funkciju operacijskog sustava koju poziva tim prekidom. Programskim prekidom, kao i svim ostalim prekidima, procesor prelazi u privilegirani način rada u kojem može obaviti i operacije koje su inače nedostupne. Pri pokretanju operacijskog sustava postavljaju se funkcije koje obrađuju prekide i koje se mogu pozvati isključivo na taj način. Ovako se istodobno postiže i zaštita operacijskog sustava od programa koji sve kritične operacije mogu provesti jedino preko unaprijed definiranih funkcija operacijskog sustava.

3. Procesi

3.1. Program, proces, dretva

Svaki će *program* svoje zadaće obaviti tek kada ga se *pokrene*. U tom postupku operacijski sustav, prema uputama u datoteci s programom, priprema za njegovo izvođenje potrebnu okolinu koju nazivamo *procesom* (pokretanjem *programa* nastaje *proces*). Navedena se okolina sastoji od spremničkih segmenata s učitanim instrukcijama, segmenta koji će se koristiti kao stog, segmenta za druge podatke i sl. Sve navedene spremničke lokacije opisane su u *opisniku procesa*. Osim samih spremničkih lokacija, operacijski sustav stvara i jednu *dretvu* koja će započeti s izvođenjem instrukcija.



Slika 3.1 Instrukcijska dretva

Dretva

Niz procesorskih instrukcija koje se nalaze u spremniku (ili programu) samo je *niz instrukcija*. S obzirom da su neke od tih instrukcija *instrukcije skoka*, pri izvođenju programa procesor neće samo slijedno izvoditi te instrukcije (jednu iza druge) – neke će segmente instrukcija višestruko obilaziti (*petlje*), neke će segmente povremeno pozivati (*potprogrami*, *rutine*, *procedure*), a neke će preskočiti jer uvjeti za njihovo izvođenje nisu bili zadovoljeni (grananja tipa *ako*). Ako usporedimo početni *niz instrukcija* s nizom onih koje su se izvele, vidimo veliku razliku. Zato se u kontekstu operacijskih sustava (i općenito programiranja) uvodi novi pojam: *instrukcijska dretva* ili kraće samo *dretva* (engl. *thread*) koja opisuje vezu instrukcija prema vremenu njihova izvođenja. Drugim riječima, dretva opisuje

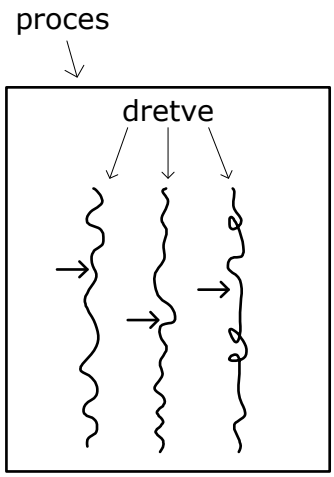
Svaki program izvodi neki *zadatak* ili više njih. Sustav koji može istodobno (paralelno) izvoditi više zadataka naziva se *višezadaćnim*.
Današnji operacijski sustavi za osobna računala, poslužitelje pa čak i ručna računala su višezadaćni, omogućuju pokretanje i rad više od jednog programa paralelno (npr. reprodukciju glazbe *u pozadini* dok se koristi internetski preglednik).
Svaki se zadatak (uglavnom) obavlja u zasebnom procesu (koji je nastao pokretanjem programa).

Pojedini se zadatak može sastojati od više *podzadataka*. Npr. reprodukcija sadržaja s Interneta sastoji se od podzadatka koji dohvaća sadržaj, podzadatka koji sadržaj prikazuje te podzadatka koji očitava naredbe korisnika.
Uobičajen način ostvarenja podzadataka jest korištenjem zasebne dretve za svaki podzadatak.
Operacijski sustav koji omogućuje postojanje više dretvi unutar istog procesa naziva se i *višedretvenim*.

Neki jednostavniji sustavi (npr. *ugrađeni sustavi*) za obavljanje različitih zadataka koriste različite dretve, ali ne koriste mehanizam procesa – sve dretve dijele isti spremnički prostor sustava.

izvođenje jednog programa – kojim su se redoslijedom njegove instrukcije izvodile u konkretnom pokretanju tog programa. Isti termin koristimo i pri analizi rada programa (npr. „kada dretva dođe do te linije koda").

Proces



Slika 3.2
Svaki proces ima bar jednu dretvu

Proces opisuje adresni prostor (i ostala sredstva) u kojem se program odvija, u što se uključuje i dretva. No osim jedne dretve koja izvodi program, unutar istog procesa može biti i više dretvi koje to rade, a koje mora stvoriti postojeća dretva odgovarajućim pozivom operacijskog sustava. Razlozi zašto bismo željeli imati više dretvi su razni.

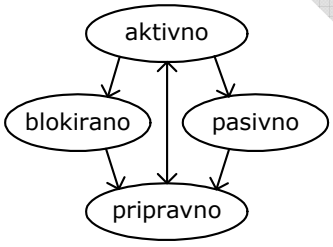
Najočitiji razlog jest želja za iskorištavanjem višeprocesorskog sustava pri rješavanju jednog problema. Primjerice, neke se operacije nad velikim skupom podataka mogu napraviti i paralelno: podijeliti skup podataka na jednake dijelove koje će obrađivati dretve na različitim procesorima. S obzirom da se posao paralelno izvodi, prije će biti gotov!

Drugi je primjer u aplikacijama koje koriste grafičko korisničko sučelje. Zasebna dretva može upravljati sučeljem, a druge mogu izvoditi potrebne proračune. Na taj će način sučelje reagirati na korisnikove naredbe i za vrijeme trajanja proračuna.

Treći primjer može biti zbog smanjenja složenosti izgradnje nekog sustava. Razdvajanjem (djelomično) nezavisnih cjelina u zasebne dretve složenost se može značajno smanjiti, a time i ubrzati razvoj i umanjiti broj grešaka (engl. *bug*).

Različiti programi rade različite poslove. Zato su oni pri pokretanju razdvojeni u različitim procesima. Greška u jednome, a koju uzrokuje jedna njegova dretva, ne utječe na drugi, već samo na taj u kojem se greška dogodila. Mehanizam upravljanja spremnikom (opisan u poglavlju 5) odvaja adresne prostore različitih procesa i sprječava da jedan drugome mijenjaju podatke. Ponekad se i iz jednog programa stvori više procesa – početni stvara nove. Razlozi za to mogu biti u potrebi odvajanja zbog sigurnosnih razloga. Primjerice, različiti prozori internetskog preglednika mogu sadržavati povjerljive podatke i treba ih zaštiti od mogućeg zlonamjernog koda iz nekog drugog prozora.

3.2. Raspoređivanje



Slika 3.3
Stanja dretvi u sustavu

U svakom trenutku u operacijskom sustavu postoji barem nekoliko procesa od kojih svaki ima barem po jednu dretvu. Ukupan broj dretvi svih procesa zajedno je gotovo uvijek veći od broja procesora.

Kako je onda moguće da svi ti procesi sa svojim dretvama mogu postojati u sustavu i u njemu se istodobno izvoditi? Kako operacijski sustav upravlja sustavom da se istodobno može raditi s npr. uređivačem teksta dok se u pozadini također reproducira neki album?

Da bi operacijski sustav mogao upravljati dretvama svih procesa, on mora imati popis svih tih dretvi i znati u kojim su one *stanjima*.

Dretva koja se trenutačno izvodi na procesoru naziva se *aktivna dretva*. U jednoprocesorskim sustavima samo jedna dretva istodobno može biti aktivna, dok u višeprocesorskim sustavima svaki procesor ima svoju aktivnu dretvu.

Dretva koja čeka na nekakav događaj naziva se *blokirana dretva*. Razlozi blokiranja mogu biti razni: od čekanja na događaj neke naprave, do događaja koje izazivaju druge dretve u sustavu ili sam operacijski sustav.

Dretve mogu biti i spremne za izvođenje, ali se ipak ne izvode jer procesor u tom trenutku izvodi instrukcije neke druge dretve. Takve dretve nazivamo *pripravnim dretvama*. Kada trenutačno aktivna dretva završi s radom (i odlazi u tzv. *pasivno stanje*) ili se u svom radu blokira, jedna od pripravnih dretvi postat će aktivna.

Dio operacijskog sustava koji upravlja dretvama i koji će odrediti koja će od pripravnih dretvi biti promovirana u aktivnu, naziva se *raspoređivač* (engl. *scheduler*). Uobičajeno načelo raspoređivanja jest u pravednoj podjeli procesorskog vremena svim dretvama. To se načelo modificira možebitnim različitim prioritetima koje dretve mogu imati. Što dretva ima veći prioritet, prije će doći do procesora. U nekim će se načinima rada raspoređivača dretva većeg prioriteta uvijek izvoditi prije dretve manjeg prioriteta, dok će se u drugim načinima rada prioritet samo odraziti na dio procesorskog vremena koji će dretva dobiti.

Uobičajeni teorijski načini raspoređivanja dretvi su:
- raspoređivanje *prema redu prispijeća*,
- raspoređivanje *prema prioritetu*,
- raspoređivanje *podjelom vremena*.

Raspoređivač koji radi po redu prispijeća (engl. *First-Come, First-Served – FCFS*, *First-In, First-Out – FIFO*) izvodi dretve onim redoslijedom kojim su se one pojavile u sustavu.

Raspoređivač koji koristi prioritete uvijek odabire dretvu najvećeg prioriteta. Na primjer, ako se za vrijeme izvođenja jedne dretve – dretve A u sustavu pojavi druga dretva – dretva B većeg prioriteta (npr. aktivira se nekim događajem), tada raspoređivač prekine izvođenje dretve A i nastavlja s dretvom B, dretvom većeg prioriteta. Tek pošto dretva B završi s radom, raspoređivač će odabrati dretvu A i nastaviti s njezinim izvođenjem.

Raspoređivanje podjelom vremena koristi kružnu dodjelu jednog intervala vremena pojedinoj dretvi (engl. *Round-Robin*). Primjerice, ako se u sustavu nalaze četiri dretve A, B, C i D, tada će raspoređivač najprije dretvi A dodijeliti jedan interval vremena – *kvant* vremena, potom dretvi B, potom dretvi C, potom dretvi D, potom opet dretvi A, pa B i tako kružno dok se dretve ne obave do kraja ili dok se neka od njih ne blokira i izađe iz kruga dretvi koje dobivaju kvant vremena.

Operacijski sustavi koriste kombinacije navedenih načina raspoređivanja. Tako se običnim sustavima koristi raspoređivanje podjelom vremena uz prilagodbu za dretve različita prioriteta pa dretve većeg prioriteta dobiju nešto više procesorskog vremena od dretvi nižeg prioriteta.

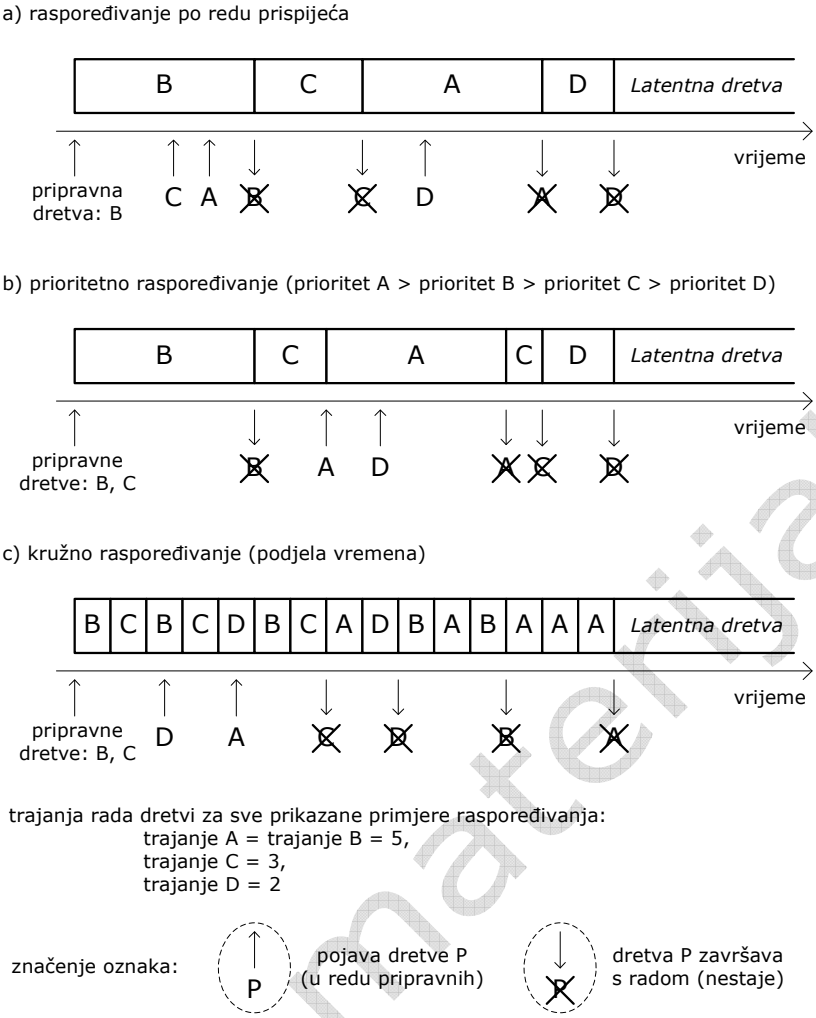
Latentna dretva
Kada nema niti jedne pripravne dretve (nema procesa koji može raditi neki koristan posao), tada se aktivira tzv. *latentna dretva* (engl. *idle thread, idle process*). Ona ne radi nikakav koristan posao, već najčešće izvodi instrukcije koje postavljaju procesor u stanje niske potrošnje energije (*pauziraju procesor*).
U kontekstu raspoređivanja, takva dretva ima najmanji prioritet. Čim se u sustavu pojavi neka druga dretva, ona nastavlja s izvođenjem (a latentna se uklanja s procesora u red pripravnih dretvi).

Windows operacijski sustavi prioritet dretve određuju kombinacijom *prioritetne klase procesa* te *prioritetnom razinom dretve* čime se dobiva broj od 0 do 31 (veći broj označava veći prioritet). Prioriteti od 1 do 15 namijenjeni su za obične dretve, a prioriteti od 16-31 za RT dretve. Raspoređivač radi tako da odabire dretve najvećeg prioriteta za aktivne (uz male iznimke).

UNIX operacijski sustavi dretvama pridjeljuju brojčani prioritet. Dio prioritetnog raspona pripada RT dretvama, a dio običnim dretvama (engl. *nice level*). Obične dretve dijele procesorsko vrijeme u skladu s njihovim prioritetom.

Raspoređivanje dretvi obavlja se iz jezgre. Ono se izvodi (poziva):
a) kada se pojavi nova dretva u sustavu (stvori nova dretva),
b) kada trenutačno aktivna dretva završava s radom,
c) kada se trenutačno aktivna dretva blokira,
d) kada se neka blokirana dretva propusti (*odblokira*),
e) kada se dogodi prekid sata sustava (za kružno raspoređivanje).

Sve navedene situaciju događaju se prekidom, programskim (a, b, c i d) i sklopovskim (d i e).



Slika 3.4 Primjeri raspoređivanja raznim načinima

Sustavi za rad u stvarnom vremenu (engl. *Real Time Systems*) prioritet dretvi uzimaju kao primarni kriterij raspoređivanja. Svaka dretva u takvom sustavu (*RT dretva*) ima unaprijed određeni prioritet i raspoređivanje se obavlja poštovanjem tih prioriteta – uvijek se odabire pripravna dretva najvećeg prioriteta. Ako više dretvi ima isti prioritet, tada se kao drugi kriterij raspoređivanja koristi ili raspoređivanje po redu prispijeća (FIFO) ili kružno posluživanje (RR).

Kružno posluživanje zahtijeva vanjski izvor prekida koji će prekinuti dretvu kojoj je istekao kvant vremena. U obradi tog prekida poziva se raspoređivač koji odabere sljedeću dretvu iz niza i promovira ju u aktivnu.

3.3. Sinkronizacija

Kada u jednom procesu ima više dretvi, one koriste zajednička sredstva kojima proces raspolaže. Istodobno korištenje tih sredstava često se ne smije dopustiti jer bi se mogla dogoditi greška. Isto vrijedi i na razini operacijskog sustava: dva različita procesa ne smiju istodobno koristiti isto sredstvo ili napravu. Na primjer, kada se tipka na tipkovnici, samo se jednom procesu (onom u fokusu, engl. *foreground*) šalju dotični znakovi – drugi će dobiti znakove kada oni budu u fokusu. Da bi se sinkronizira-

lo rad dretvi, u takvim se situacijama koriste sinkronizacijski mehanizmi.

Jedan od najjednostavnijih sinkronizacijskih mehanizama je *semafor*.

Semafor može biti u *prolaznom stanju* kada ne blokira dretvu koja ga pokušava proći, ili u *neprolaznom stanju* kada će blokirati dretvu koja ga pokušava proći. Stanje se može utvrditi očitanjem *vrijednosti semafora*. Ako je vrijednost semafora veća do nule, on je u prolaznom stanju, inače, ako je vrijednost semafora jednaka nuli, on je neprolazan i dretve koje ga pokušaju proći bit će blokirane na tom semaforu.

Ostvarivanje *međusobnog isključivanja* vrlo je jednostavno korištenjem semafora. Njegova se vrijednost pri inicijalizaciji postavi u jedan, a prije korištenja zajedničkog sredstva mora se proći kroz semafor. Prva dretva koja dođe do semafora uspjet će proći kroz njega, ali će pritom postaviti njegovu vrijednost u nulu (i sve ostale dretve koje pokušavaju proći bit će blokirane). Kada dretva završi korištenje zajedničkog sredstva, ona *otpušta* semafor. U operaciji otpuštanja vrijednost semafora povećava se za jedan te se jedna druga dretva može propustiti kroz semafor, koja će tada moći koristiti zajedničko sredstvo (međusobno isključivo s ostalim dretvama).

Osim *semafora*, operacijski sustavi pružaju mnoge druge sinkronizacijske mehanizme kroz svoje sučelje (API).

Jedan od jednostavnijih je mehanizam za *međusobno isključivanje* (engl. *mutual exclusion* – *mutex*). Načelo mehanizma je sličan semaforu koji ne može poprimiti vrijednost veću od jedan (*binarni semafor*).

U novije vrijeme sve se češće koriste mehanizmi koji sami ne sadržavaju opis stanja (kao što to radi semafor), već samo nude metode za zaštićen rad s varijablama koje opisuju stanje sustava i na temelju čega se donose odluke o propuštanju dretvi (ili njihovu blokiranju). Najčešće korišteni takav mehanizam jest *monitor* [Budin, 2010].

3.4. Opisnik procesa

Radi upravljanja procesima i dretvama svaki proces mora imati opisnik. Dio podataka koje opisnik sadržava može se vidjeti i iz sučelja operacijskog sustava (sučelja za upravljanje procesima).

Uobičajeno su ti elementi:

- identifikacijski broj procesa (engl. *Process Identifier* – PID),
- ime programa iz kojeg je proces nastao,
- identifikacija korisnika kome taj proces pripada,
- prioritet procesa,
- postotak korištenja procesora,
- ukupno potrošeno procesorsko vrijeme,
- veličina korištenog spremničkog prostora,
- broj dretvi u procesu.

Elementi koji nisu vidljivi sadržavaju detaljniji opis nekih od navedenih elemenata koji se koriste za upravljanje. Na primjer, za upravljanje dretvama treba postojati zasebni opisnik za svaku dretvu, koji sadržava podatke o njezinom prioritetu, načinu raspoređivanja, potrošenom procesorskom vremenu, mjesto za pohranu konteksta i sl. Za upravljanje spremnikom potrebno je mnogo više podataka o procesu i njegovim dretvama.

4. Spremnički prostor

Glavni je spremnik središnje mjesto računala kroz koje prolaze svi podatci i instrukcije (pri njegovu radu). Organizacija i upravljanje spremničkim prostorom stoga su vrlo bitni i osnovne su zadaće operacijskog sustava.

Za vrijeme rada računala, u spremniku se nalaze:

- podatci i instrukcije operacijskog sustava potrebni za upravljanje sustavom:
  - opisnici svih procesa i dretvi,
  - upravljački programi i potrebni podatci za sve prisutne naprave,
  - podatci svih podsustava,
  - podatci za upravljanje svim sredstvima sustava (kao što je i sam spremnik!),
- instrukcije i podatci za svaki proces, zasebni stog za svaku dretvu.

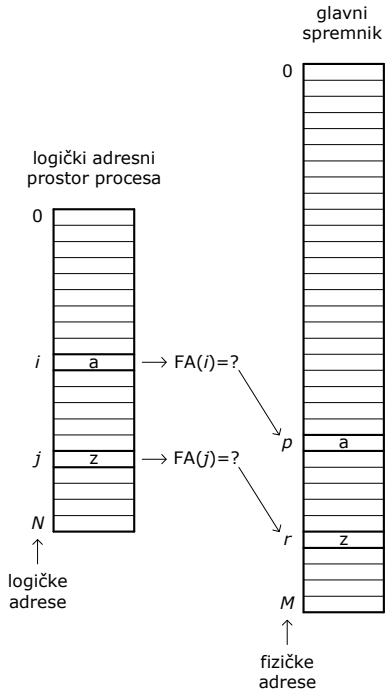
Kako sve navedeno smjestiti u spremnik? Kako zaštititi jedan proces od drugog? Kako zaštititi podatke operacijskog sustava od procesa? Kako učinkovito iskoristiti spremnički prostor? Što s programima koji su veći od raspoloživog spremničkog prostora? Kako treba pripremiti programe prije učitavanja u spremnik?

Navedena su osnovna pitanja na koja mora odgovore dati podsustav za upravljanje spremnikom.

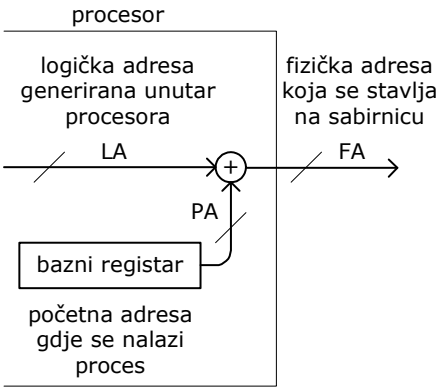
U različitim tipovima računalnih sustava susrećemo različite načine upravljanja spremnikom. U jednostavnijim se sustavima koriste metode koje se mogu ostvariti bez dodatnog sklopovlja namijenjenog upravljanju spremnikom. U drugim, koji imaju nekakvo sklopovlje za upravljanje spremnikom (npr. adresno zbrajalo spojeno između procesora i sabirnice), mogu se koristiti prikladne metode koje će iskoristiti to sklopovlje i poboljšati upravljanje spremnikom.

Složeni računalni sustavi, koji zahvaljujući razvoju računalne tehnologije danas uključuju i većinu ručnih računala, koriste postupak upravljanja spremnikom koji se naziva *straničenje* (engl. *paging*), a koji je omogućen složenim sklopovskim podsustavom za upravljanje spremnikom koje se nalazi na samom procesoru (engl. *memory management unit* – MMU).

Osnovna ideja koja olakšava upravljanje spremnikom jest da se programi mogu učitati u bilo koji slobodni dio spremnika. S obzirom da program u svom izvođenju generira zahtjeve prema spremniku, te adrese koje on generira ne smiju ovisiti o mjestu gdje će se program učitati u spremnik. Adrese moraju biti nezavisne, *relativne* samo u odnosu na sam program (tj. proces), npr. u odnosu na njegov početak (koji ima relativnu adresu jednaku nuli). Relativna se adresa još naziva i *logičkom*, s obzirom da ju generira sam program. Sustavi za upravljanje spremnikom (uz njegovu sklopovsku potporu) moraju *logičku adresu* koju generira program pretvoriti u *stvarnu adresu* spremničke lokacije koju je program zapravo tražio, prije stavljanja adrese na sabirnicu. Tu ćemo adresu još nazvati i *apsolutnom* ili *fizičkom adresom*. Zadatak je sustava za upravljanje spremnikom, dakle, pretvorba zahtjeva koji je iskazan u



Slika 4.1
Logičke i fizičke adrese

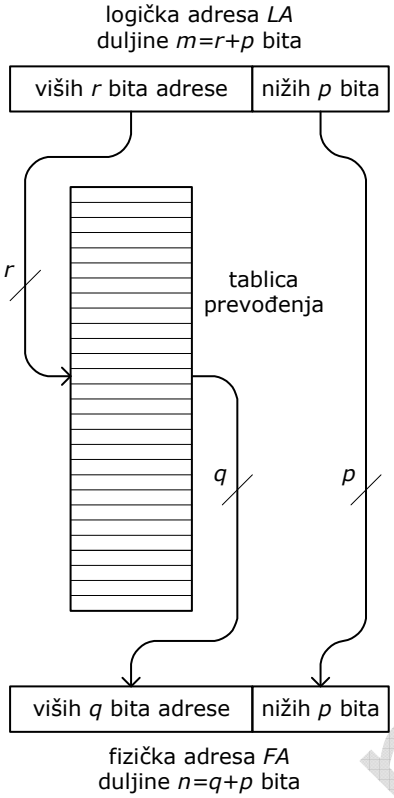


Slika 4.2
Pretvorba adresa kod dinamičkog upravljanja spremnikom

logičkoj adresi programa u *fizičku adresu spremničke lokacije* koja joj odgovara.

Najjednostavnije takvo upravljanje, koje se često naziva i *dinamičko upravljanje spremnikom*, koristi jednostavno zbrajalo adresa. Ako se program učita na lokaciju *početne adrese* (PA), tada se svaka adresa koju generira program (logička adresa – LA) mora zbrojiti s početnom adresom (PA) prije prosljeđivanja na sabirnicu. Sklop koji omogućuje ovakav način upravljanja spremnikom sastoji se samo od dodatnog registra za pohranu početne adrese procesa (koji postaje dio konteksta procesa) te dodatnog zbrajala.

Dinamičko upravljanje spremnikom, međutim, ima suviše nedostataka da bi se koristio u složenijim sustavima (budući da postoji bolji). Neki od nedostataka su: nema zaštite između procesa, dinamički zahtjevi za spremnikom ne mogu se ostvariti (npr. povećanje već dodijeljenog segmenta), problem fragmentacije spremnika, nemogućnost pokretanja programa koji cijeli ne stanu u spremnik.



Slika 4.3
Pretvorba adresa kod straničenja

Upravljanje *straničenjem*, u osnovi algoritma, također koristi zbrajanje adresa. No u tom načinu upravljanja proces je podijeljen u dijelove istih veličina, koje nazivamo *stranicama* (engl. *page*). Svaka je stranica nezavisno spremljena u neki dio spremnika. Sustav za upravljanje će pri svakom zahtjevu morati odrediti kojoj stranici zahtjev pripada te gdje se ta stranica nalazi u spremniku. Nakon što se to utvrdi, iz drugog se dijela adrese određuje koliko je zahtjev udaljen od početka stranice te se taj broj zbraja na početnu adresu stranice u spremniku. Sve navedeno automatski obavlja sklopovlje pa je pretvorba adresa iznimno brza i ne uvodi kašnjenja u radu (osim iznimno, što je objašnjeno kasnije).

Radi jednostavnosti sklopovlja, veličina stranice je višekratnik broja dva: uobičajeno je to 4 kB (4096 okteta). Jedna će se takva stranica u spremnik spremiti na adresu koja je također isti višekratnik broja dva, tj. na adrese koje su višekratnik od 4 kB. Takve adrese imaju nižih p bitova (12 bitova, 2^{12} = 4096) jednake nuli. Ovakav odabir omogućuje da se pri pretvorbi, nakon što se odredi početak spremnika gdje je stranica pohranjena (a što se naziva *okvir*, engl. *frame*), kao odmak od početka stranice jednostavno može iskoristiti nižih p bitova relativne adrese, tj. tih nižih p bitova ostaje nepromijenjeno, samo se *redni broj stranice procesa* mora pretvoriti u adresu početka te stranice, odnosno mjesta spremničkog prostora gdje je ona spremljena, što ćemo nazvati *rednim brojem okvira spremnika*.

Za pretvorbu između rednog broja stranice procesa i rednog broja okvira spremnika koristi se *tablica prevođenja* dotičnog procesa u kojoj svaki redak opisuje po jednu stranicu. Prije pokretanja programa operacijski sustav mora pripremiti tablicu prevođenja za njega (i učitati dijelove programa u spremnik). Pri izvođenju procesa pretvorba adresa koje on generira obavlja sklopovlje korištenjem pripremljene tablice.

U stvarnim se sklopovskim ostvarenjima često koristi hijerarhijska organizacija tablice prevođenja prvenstveno radi smanjenja potrebnih dijelova tablice (npr. za manje procese).

Straničenje zahtijeva složeno sklopovlje, ali zato ima i mnoga dobra svojstva. Zaštita između procesa je besprijekorna, fragmentacija ne postoji, procesi mogu dinamički tražiti dodatne spremničke lokacije, čak se i program koji se cijeli ne može smjestiti u spremnik može izvoditi tako da se učitavaju samo oni dijelovi koji su trenutačno potrebni, a ostali su spremljeni na *pomoćnom spremniku* (disku).

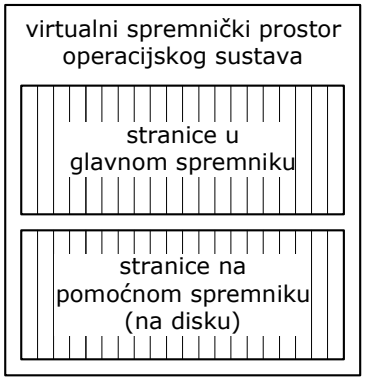
Sustav koji nema dovoljno spremničkog prostora (glavnog spremnika) može koristiti pomoćni spremnik da privremeno makne neke stranice koje trenutačno nisu potrebne, a da bi na njihovo mjesto smjestio one koje se traže. Naravno da će takve operacije u određenoj mjeri usporiti rad cijelog sustava jer je pomoćni spremnik znatno sporiji od glavnog, ali će ipak omogućiti pokretanje svih željenih programa. Zbog toga je pri odabiru potrebne veličine spremničkog prostora (pri kupovini računala) dobro poznavati zahtjeve koji će se na sustav postavljati i s motrišta spremničkog prostora. Premali glavni spremnik intenzivno će morati koristiti pomoćni spremnik i time znatno smanjiti učinkovitost sustava. S druge strane, preveliki će spremnik ostati neiskorišten (uzalud potrošen novac).

Sustavi za rad u stvarnom vremenu, koji očekuju vrlo brzu reakciju na događaje u sustavu, ne koriste pomoćne spremnike jer oni unose značajna kašnjenja u rad sustava. Na primjer, ako se u obradi kritičnog događaja trebaju koristiti podatci koji se trenutačno nalaze u stranici koja je na pomoćnom spremniku, tada će proteći barem nekoliko milisekundi dok se ta stranica najprije ne dohvati s diska. Takva odgoda u nekim sustavima nije prihvatljiva te se u takvim sustavima ili straničenje uopće ne koristi, ili se koristi, ali bez pomoćnog spremnika, ili se koristi i pomoćni spremnik, ali samo za procese koji nisu vremenski kritični.

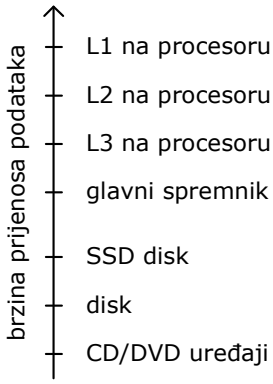
Spremnik je znatno sporiji od procesora (barem nekoliko puta). Zato se koriste razne tehnološke metode da bi se taj nedostatak ublažio. Među najznačajnije spada korištenje priručnog spremnika samog procesora koji, iako je znatno manjeg kapaciteta, znatno ubrzava rad sustava. Brzina priručnog spremnika procesora bitno je veća od samog spremnika. Brži spremnici prilično su skuplji te se i sam priručni spremnik na procesoru dijeli na nekoliko razina različitih brzina, a da se takvom organizacijom postignu prihvatljive performanse uz prihvatljivu cijenu. Uobičajena podjela priručnih spremnika procesora je na tri razine (engl. *cache levels*): prva (najmanja) razina (L1) koja radi na brzini samog procesora, druga (L2) koja radi nešto sporije, ali je većeg kapaciteta te treća (L3) koja je značajno većeg kapaciteta, ali je još sporija, međutim, ipak brža od glavnog spremnika.

Zahtjevi za podatcima u spremniku vrlo su često lokalizirani (vremenski bliski zahtjevi su i prostorno bliski – zahtjevi za susjednim lokacijama), procesor će vrlo često moći i samostalno raditi, korištenjem već dohvaćenih podataka u priručnim spremnicima, bez komunikacije s glavnim spremnikom.

Veličina priručnog spremnika procesora prilagođena je njegovu korištenju. Dok će za zahtjevne poslužitelje trebati odabrati procesore sa znatnom količinom priručnog spremnika (zato i značajno skuplje), za obična su računala (kućna ili uredska) dovoljni i jednostavniji (jeftiniji) procesori s manjom količinom



Slika 4.4
Spremnički prostor kojim upravlja operacijski sustav



Slika 4.5
Poredak tipova spremnika prema brzinama prijenosa podataka

SSD diskovi (engl. *Solid State Disk*) noviji su uređaji za trajnu pohranu podataka, po priključcima i protokolima slični običnom disku. U izvedbi se potpuno razlikuju od običnih diskova - oni koriste samo elektroničke elemente za trajnu pohranu podataka. Kako nemaju mehaničkih dijelova, znatno su brži od običnih diskova (pristup podatcima i brzina prijenosa značajno je veća). S druge strane, poprilično manji kapacitet i značajno veća cijena glavni su razlozi njihovog još uvijek rijetkog korištenja.

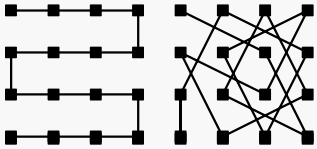
priručnog spremnika.

Korištenje priručnih spremnika mijenja i pristup spremniku. Iako program može zahtijevati samo jedan podatak, sklopovlje će dohvatiti cijeli blok podataka jer mu je to znatno isplativije – iako će početno trajati nešto više od samo jednog zahtjeva (ne mnogo!), to će se višestruko isplatiti ako se i nekoliko kasnijih zahtjeva odnosi da dohvaćeni blok.

Imajući na umu navedena svojstva glavnog i priručnih spremnika, pristup u izgradnji novih programa, koji su procesorski i spremnički zahtjevni, mora biti prikladan želi li se sklopovlje maksimalno iskoristiti. Korištenje priručnih spremnika je (najčešće) u potpunosti sklopovski riješeno. Poznavanjem svojstava i načina rada tog sklopovlja može se povećati učinkovitost programa, ali to nije nužno pri izradi nezahtjevnih programa.

Matične ploče na koje se stavljaju sastavnice računalnog sustava obično i same podržavaju neke dodatne mogućnosti ubrzavanja rada spremnika. S obzirom na ograničenja spremničkih modula koji se ugrađuju, a koji su određeni širinom sabirnice (brojem priključaka na ploču – brojem kontakata) i brzinom rada, matične ploče (tj. upravljački čipovi na njima), omogućuju uparivanje takvih modula radi postizanja efektivno šire sabirnice koja u istom razdoblju može prenijeti više podataka (npr. dvokanalno, trokanalno, engl. *dual-channel, triple-channel*).

Učinkovitost korištenja spremnika za obavljanje nekog posla (vrijeme potrošeno za rad sa spremnikom) može se pokušati usporediti s duljinom puta potrebnog za obilazak pravilno raspoređenih točaka.



Kada se točke slijedno obilaze, duljina puta višestruko je kraća od obilaska kojemu uzastopne točke na putu nisu susjedne.

Vježbe iz poglavlja 1 – Informacije o sustavu

1) **Uvod:**

Prvi korak pri bilo kakvoj promjeni u računalnom sustavu jest njegovo detaljno upoznavanje. Osim određivanja samog operacijskog sustava (i naknadno primijenjena ažuriranja), potrebno je poznavati i samo sklopovlje, barem osnovne elemente u koje spadaju:

- tip i izvedba procesora,
- matična ploča,
- tip i veličina glavnog spremnika,
- disk, njegova veličina i ostala svojstva (particije, datotečni sustavi),
- grafička kartica,
- zaslon i njegova svojstva,
- priključak na Internet.

Većinu osnovnih podataka za navedene elemente moguće je dobiti odgovarajućim programima koji su dio samog operacijskog sustava. Više se detalja može dobiti ili izravnim uvidom u sklopovlje (otvaranjem računala i izravnim očitanjem parametara – ili nakon ustanovljavanja modela uvidom u dokumentaciju istih), ili korištenjem zasebnih programa čija je namjena dohvat svih detalja o pojedinim sklopovskim sastavnicama.
Uz operacijski se sustav obično isporučuju i osnovni programi za upravljanje osnovnim tipovima podataka (datotekama). No da bi računalni sustav obavljao neku korisnu operaciju, najčešće je potrebno naknadno instalirati dodatne programe. Pri održavanju računala potrebno je poznavati način pregleda svih tih programa (instaliranih uobičajenom procedurom), gdje su oni smješteni, koliko spremničkog prostora (na disku) zauzimaju, koliko se često koriste i sl.

2) **Temeljni pojmovi:**

- procesor,
- matična ploča,
- glavni spremnik,
- grafička kartica,
- disk,
- zaslon (*monitor*),
- priključak na Internet,
- programi

3) **Cilj vježbe:**

Cilj ove vježbe je upoznati se s metodama prikupljanja podataka o računalnom sustavu.

4) **Priprema za vježbe:**

Upoznati se (teoretski) s osnovnim elementima računala i njihovim parametrima.

5) **Vježbe: [...]**

6) **Upute i objašnjenja uz vježbe**

Vježba 1 a)
Ako je računalo zatvoreno, traženi se podatci mogu doznati jedino preko možebitnih naljepnica s podatcima (uobičajeno može biti prisutna markica s operacijskim sustavnom i posebnim ključem za to računalo) i preko oznake modela računala (i uvidom tehničke dokumentacije tog model).

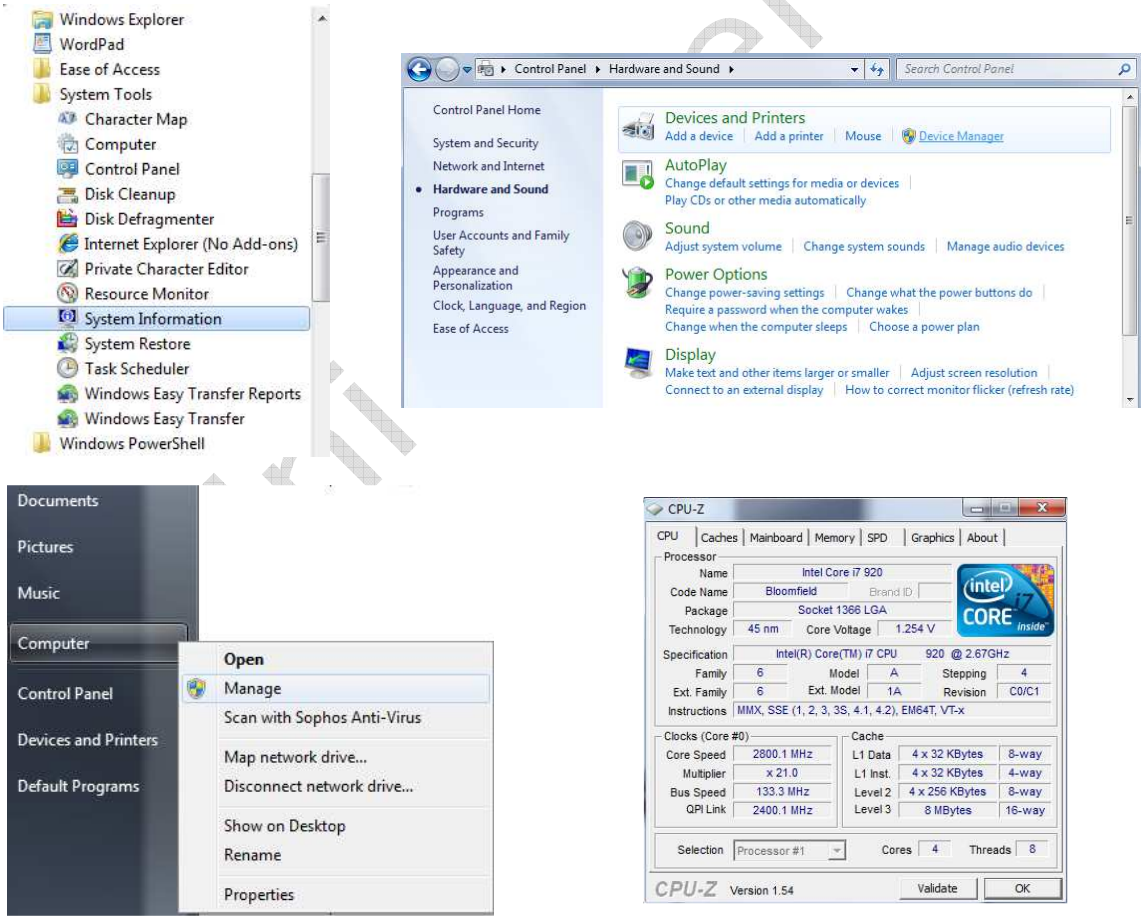
Vježba 1 b)

Otvaranje i izravan rad s elementima računala (postavljanje/vađenje/prilagodba) trebaju raditi samo osobe koje su dobro upućene u način rada s tim elementima. Ipak, ako se i SAMO PROMATRA unutrašnjost, mogu se dobiti potrebni podatci o pojedinim elementima računala. Neki će elementi biti označeni samo kraticom, dok će za neke druge biti dostupno i više podataka. Ponekad je nužno otvoriti računalo i pogledati koje su sastavnice unutra kako bi se moglo pripremiti odgovarajuće upravljačke programe i za sam postupak instalacije operacijskog sustava. Najčešće to nije potrebno kada je riječ o uobičajenom sklopovlju, ali kada je neko sklopovlje manje uobičajeno ili se tek pojavilo na tržištu, moguće je da operacijski sustav nema odgovarajuće upravljačke programe za tu napravu (npr. RAID kartica).

Vježba 1 c)

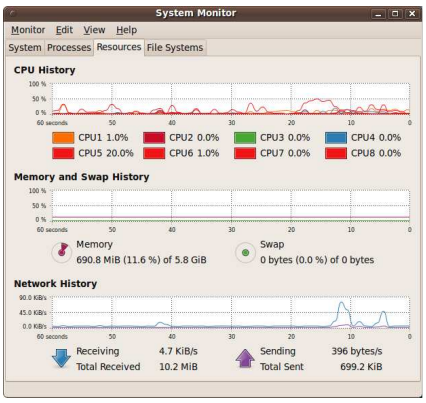
Različiti operacijski sustavi nude različita sučelja za njihovo podešavanje i pregled elemenata i njihovih postavki.

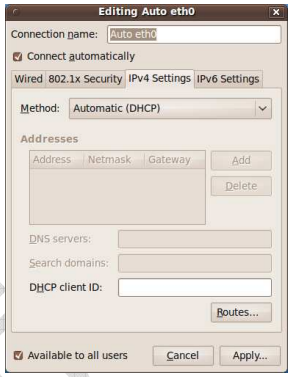
Windows porodica operacijskih sustava gotovo sve navedeno nudi kroz *Control Panel*. Osim kroz njega, neki se alati (npr. *System Information*) nalaze i u okviru izbornika (*Start → Programs → Accessories → System Tools*). Za pregled sklopovlja i instaliranih programa, najbolje će poslužiti programi: *System Information* i *Device Manager* te specijalizirani dodatni programi (npr. *CPU-Z*).

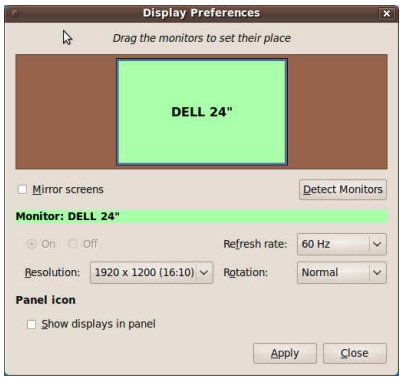


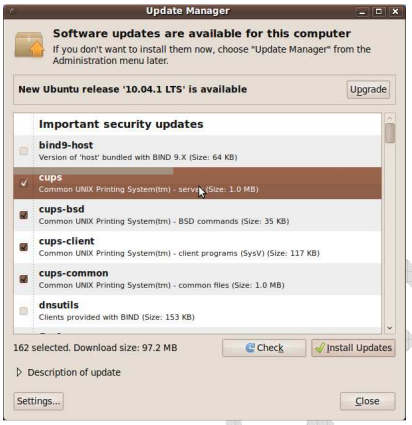
Na *Linux* i sličnim *UNIX* sustavima, većina podešavanja i informacija o sustavu dobiva se naredbama u komandnoj liniji. Informacije o trenutačnom stanju sustava mogu se dobiti i pregledom virtualnog direktorija */proc*. Na primjer, podatci o procesorima mogu se pogledati u datoteci */proc/cpuinfo* (naredbom `$ cat /proc/cpuinfo`). Približavanjem *Linux* sustava i običnim korisnicima pojavljuje se i sve više alata koji omogućuju podešavanje sustava i iz grafičkog okruženja. Primjeri takvih programa iz operacijskog sustava *Ubuntu* (inačice 9.10) prikazani su u nastavku pomoću slika njihova rada.

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 26
model name      : Intel(R) Core(TM)
i7 CPU, 920  @ 2.67GHz
stepping        : 4
cpu MHz         : 1596.000
cache size      : 8192 KB
physical id     : 0
...

processor       : 1
...

$ cat /proc/meminfo
MemTotal:        6113956 kB
MemFree:         4700024 kB
Buffers:          114420 kB
Cached:           604248 kB
...
```









Vježbe iz poglavlja 2 – Prekidi

1) **Uvod:**

Mehanizam prekida koristi se za upravljanje mnogih sklopovskih elemenata računalnog sustava (uređaja, kartica, ulazno-izlaznih naprava), ali je i jedan od bitnih mehanizama koji se koriste radi ostvarenja *višezadaćnih* sustava. Broj prekida (prekidnih brojeva) koje sustav podržava ograničen je (broj prekida koji naprave mogu generirati), ali najčešće ipak dovoljan. Neke naprave (sklopovi) mogu čak i dijeliti isti prekid (prekidni broj), ali neke nisu tako projektirane i mogu izazvati konflikt ako se i neki druga naprava koristi istim prekidom. Ponekad se (vrlo rijetko) ipak od korisnika očekuje ručno rješavanje nekih konfliktnih situacija, dodjelom prekidnih brojeva napravama koje su u konfliktu.

2) **Temeljni pojmovi:**

- procesor,
- (sklopovski) prekidi,
- konflikti,
- programski prekidi.

3) **Cilj vježbe:**

Prekidni podsustav operacijskog sustava mora svim napravama koje to zahtijevaju osigurati korištenje odgovarajućih prekidnih brojeva. Cilj ove vježbe jest utvrditi koje sve naprave i sklopovi generiraju zahtjeve za prekid te s kojim brojem. Jednako je tako moguće odrediti dijele li neke naprave isti prekid te jesu li u konfliktu.

4) **Priprema za vježbe:**

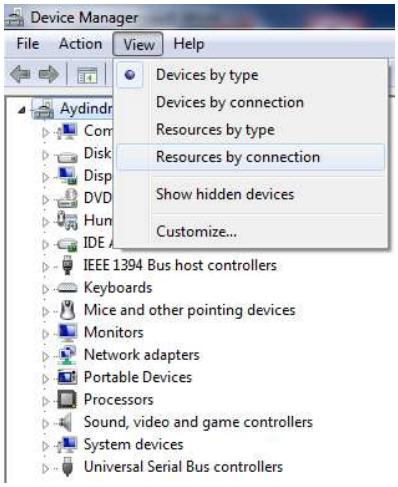
Upoznati se (teoretski) s osnovnim elementima računala koji koriste prekide u svom radu.

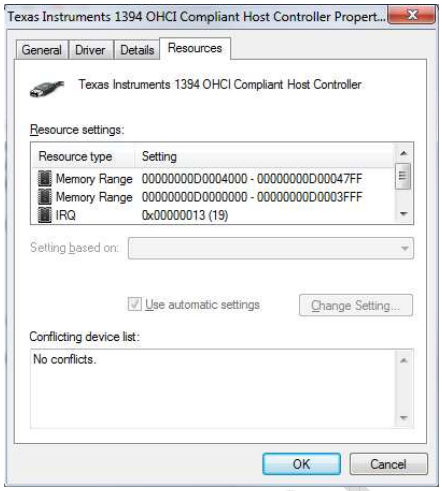
5) **Vježbe: [...]**

6) **Upute i objašnjenja uz vježbe**

Uvid i postavke prekidnog podsustava obavljaju se kroz različito sučelje u različitim sustavima, bilo kroz grafičko sučelje ili preko jednostavnih naredbi u komandnoj liniji. Često je moguće (potrebno) pogledati (ili i promijeniti) te postavke i prije podizanja operacijskog sustava, i to korištenjem BIOS-a.

Na *Windows* sustavima uvid se može dobiti preko *System Information* programa, dok se promjene mogu raditi korištenjem *Device Manager* programa (uz druge načine).





Na *Linux* sustavima, uvid u prekide može se dobiti pregledom */proc* direktorija. Broj dosad pristiglih prekida može se pogledati u datoteci */proc/interrupts*.

```
$ cat /proc/interrupts
           CPU0       CPU1
  0:         47          0   IO-APIC-edge      timer
  1:       3830          2   IO-APIC-edge      i8042
  3:          1          1   IO-APIC-edge
  4:          0          2   IO-APIC-edge
  6:          5          0   IO-APIC-edge      floppy
  7:          0          0   IO-APIC-edge      parport0
  8:          0          0   IO-APIC-edge      rtc0
  9:          0          0   IO-APIC-fasteoi   acpi
 12:     122314          2   IO-APIC-edge      i8042
 14:          0          0   IO-APIC-edge      ata_piix
 15:     326709     211866   IO-APIC-edge      ata_piix
 17:      28326        158   IO-APIC-fasteoi   ioc0
 18:       5316       5199   IO-APIC-fasteoi   vmxnet ether
 19:        304          0   IO-APIC-fasteoi   uhci_hcd:usb1, Ensoniq AudioPCI
NMI:          0          0   Non-maskable interrupts
LOC:    2313781    2038823   Local timer interrupts
SPU:          0          0   Spurious interrupts
CNT:          0          0   Performance counter interrupts
PND:          0          0   Performance pending work
RES:     412108     436113   Rescheduling interrupts
CAL:         56        115   Function call interrupts
TLB:      18454      16546   TLB shootdowns
TRM:          0          0   Thermal event interrupts
THR:          0          0   Threshold APIC interrupts
MCE:          0          0   Machine check exceptions
MCP:        355        355   Machine check polls
ERR:          0
MIS:          0
```

Vježbe iz poglavlja 3 – Procesi

1) **Uvod:**

Pokretanjem programa nastaju procesi. Procesi od operacijskog sustava traže neke resurse (spremnički prostor, procesorsko vrijeme, datoteke, ...). Sustav treba biti prikladno projektiran za poslove, tj. procese koji se u njemu izvode. Ako se previše procesa pokrene, može se dogoditi da se svi resursi potroše te programi zbog toga ne uspiju obaviti do kraja ili dolazi do znatne degradacije performansi (što se očituje u sporom reagiranju pojedinih programa ili općenito korisničkog sučelja). Ponekad je uzrok zagušenosti neki program koji je greškom još prisutan i kojeg treba zaustaviti.

Prioritet procesa odražava njihovu važnost naspram ostalih. Programi su najčešće tako projektirani da na početku svog rada prilagode svoj prioritet prema svojim potrebama. Ipak, ponekad i sam korisnik može poželjeti promijeniti te prioritete i procesu za koji smatra da je manje bitan smanjiti prioritet i, obratno, nekim drugim povećati prioritet.

Kada se neki problem rješava korištenjem više dretvi, treba uzeti u obzir da te dretve mogu paralelno mijenjati zajedničke podatke, što može, ako se ne napravi u zaštićenom načinu, dovesti do grešaka. Mehanizmi zaštite uključuju razne sinkronizacijske mehanizme (semafore, monitore i sl.).

2) **Temeljni pojmovi:**

- proces,
- dretva,
- zaustavljanje procesa,
- promjena prioriteta,
- opterećenje procesora,
- sinkronizacija.

3) **Cilj vježbe:**

Prvi cilj jest utvrditi uobičajeni skup procesa koji se izvodi u sustavu s njihovim svojstvima (opterećenje za sustav). Nadalje, treba se upoznati s alatima za pregled trenutačnih procesa, alatima za zaustavljanje procesa, promjenu njihova prioriteta iz sučelja operacijskog sustava.

Korištenjem priloženih programa (u C-u i kao već prevedene) ispitat će se utjecaji novih programa na rad sustava.

4) **Priprema za vježbe:**

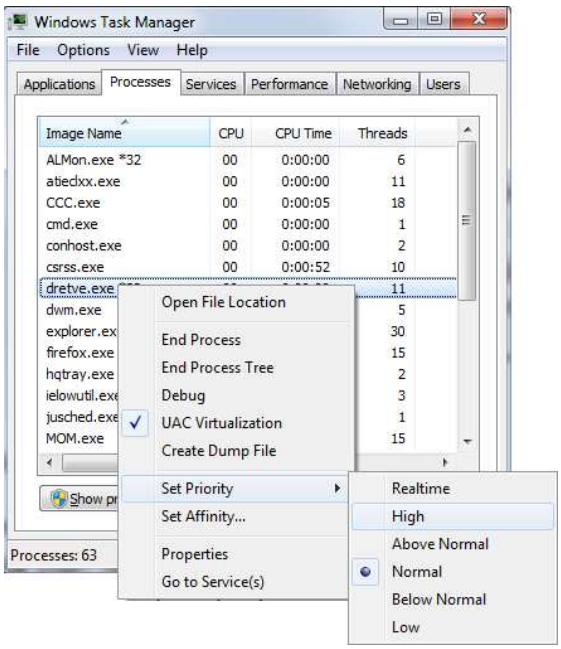
Upoznati se (teoretski) s procesima i dretvama te njihovim parametrima.

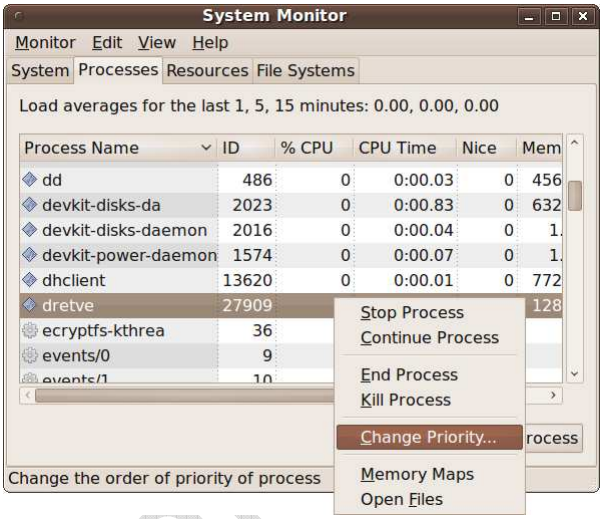
5) **Vježbe: [...]**

6) **Upute i objašnjenja uz vježbe**

Uvid i upravljanje procesima može biti ostvareno kroz jedinstveno sučelje (npr. *Task Manager*) ili kroz odvojena sučelja (npr. naredbe `tasklist` i `taskkill`, odnosno, `ps`, `kill`, `nice` na UNIX-u). Uz razne će zastavice ili opcije u izborniku ti programi ispisati više ili manje svojstava pojedinih procesa (npr. za *Task Manager* iz izbornika *View → Select Columns…*). Potrebni podatci za ovu vježbu uključuju: ime procesa, trenutačno zauzeće procesora, do sada potrošeno procesorsko vrijeme, prioritet.

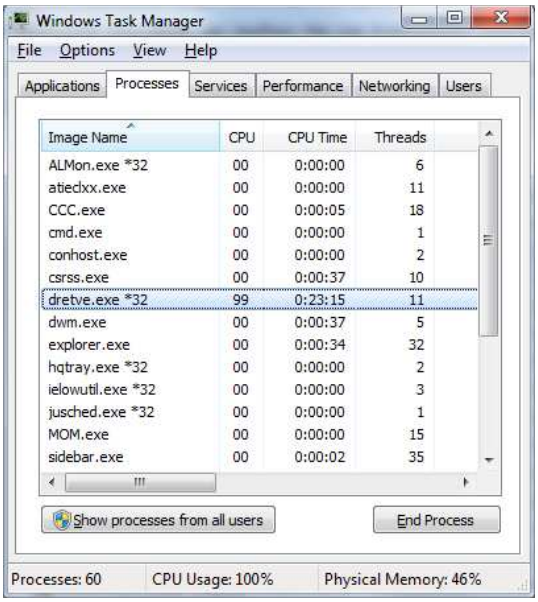
Promjena prioriteta dretvi/procesu može se obaviti programski, kada to dretva sama napravi korištenjem odgovarajućih funkcija (kako je to prikazano u *dretve.c*) ili korištenjem sučelja operacijskog sustava. Grafičko sučelje *Windows Task Managera* ili *System Monitora* (na *Linuxu*) omogućuje promjenu prioriteta prema sljedećim primjerima:

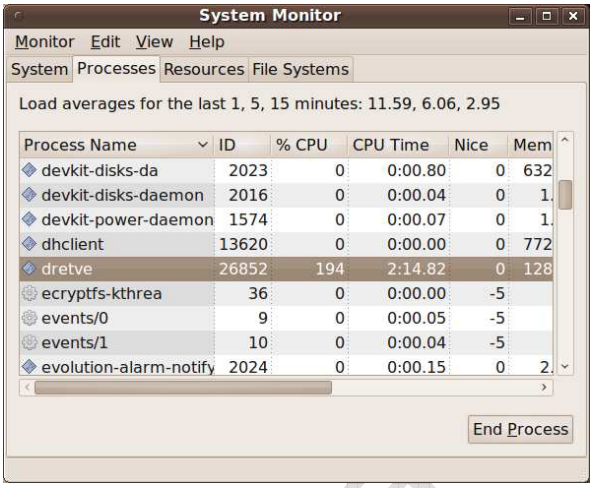




Pri stvaranju većeg broja dretvi, a pogotovo procesa, treba biti oprezan jer se može ugroziti stabilnost sustava. Programi se uobičajeno mogu prekinuti s Ctrl+C, ali u slučaju procesa može se dogoditi da se ne prekinu baš svi! U tom se slučaju može koristiti naredba `taskkill` (npr. `taskkill /f /im procesi.exe`) koja će zaustaviti i maknuti sve zaostale procese istog imena.

Program *sinkro.c* prikazuje probleme koji nastaju kada se koristi više dretvi. S obzirom da one mogu paralelno raditi, mogu paralelno mijenjati i zajedničke varijable. U ovom je primjeru samo jedna zajednička varijabla *broj* nad kojom dretve obavljaju jednostavnu operaciju povećanja vrijednosti za jedan. No i ta se jednostavna operacija (koja može biti ostvarena i jednom instrukcijom) zapravo sastoji od tri koraka: u prvom se varijabla dohvaća iz glavnog spremnika u registar procesora, potom se ta vrijednost poveća za jedan te se konačno nova vrijednost pohrani ponovno u spremnik. S obzirom da se u radu procesora taj slijed operacija može prekinuti raspoređivačem poslova (ako nije izvedena jednom instrukcijom), pogotovo se korištenje sabirnice između više procesora može poklopiti tako da više njih pročitaju istu vrijednost i nad njome nezavisno obavljaju zbrajanja čime se gube pojedinačna zbrajanja. Rezultat toga jest manja očekivana vrijednost varijable *broj*. Na primjer, ako se program pokrene sa *sinkro 10 100000*, očekuje se da će konačna vrijednost biti 10*100000 = 1000000. Ipak, zbog prethodno navedenih razloga konačna vrijednost može biti i znatno manja. Problem se može riješiti zaključavanjem *kritičnog odsječka* – promjene varijable *broj*. U tom je slučaju konačna vrijednost očekivana (10*100000 za navedeni primjer). Ipak, sinkronizacija ima svoju cijenu. U navedenom je primjeru izračunavanje koje koristi zaključavanje znatno sporije od onog koje to ne koristi (sa ili bez zadnjeg argumenta *s* pri pokretanju).





Izvorni kod datoteke *procesi.c* za *Windows*:

```c
/* kod "procesi.c" za Windows sustave */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <process.h>

#define NAJPROCESA 10000

void upotreba ( char *param[] );
void radi ();

int main ( int brarg, char *arg[] )
{
	STARTUPINFO si;
	PROCESS_INFORMATION *pi;
	int i, broj_procesa, stvoreno;
	char proces[] = "procesi.exe";

	if ( brarg == 1 )
	{
		radi (); /* kada se pokrece kao novi process iz ovog programa */
	}
	else if ( brarg > 2)
	{
		upotreba ( arg );
	}
	else {
		broj_procesa = atoi ( arg[1] );

		if ( broj_procesa < 1 || broj_procesa > NAJPROCESA )
			upotreba ( arg );

		GetStartupInfo ( &si );

		/* stvori zadani broj procesa */
		stvoreno = broj procesa;
		pi = malloc ( broj_procesa * sizeof (PROCESS_INFORMATION) );
		for ( i = 0; i < broj_procesa; i++ )
		{
			if( !CreateProcess ( NULL, proces, NULL, NULL, FALSE, 0,
					NULL, NULL, &si, &pi[i] ) )
			{
				stvoreno = i;
				printf ( "Stvoreno %d procesa (od zadanih %d)\n",
					stvoreno, broj_procesa );
				break;
			}
		}

		printf ( "Svi procesi stvoreni, cekam kraj (Ctrl+C)\n" );

		/* cekaj da svi procesi zavrse */
		for ( i = 0; i < broj_procesa; i++ )
			WaitForSingleObject ( pi[i].hProcess, INFINITE );
	}

	return 0;
}
void upotreba ( char *param[] )
{
	printf ( "Upotreba: %s [broj procesa]\n\tbroj procesa iz intervala "
		 "[1-%d]\n", param[0], NAJPROCESA);
	exit ( 1 );
}
void radi ()
{
	while ( 1 )
		Sleep ( 10000 );
}
```

Izvorni kod datoteke *procesi.c* za *Linux*:

```c
/* kod "procesi.c" za Linux sustave */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define NAJPROCESA 10000

void upotreba ( char *param[] );
void radi ();

int main ( int brarg, char *arg[] )
{
	int i, broj_procesa, stvoreno;

	if ( brarg != 2)
		upotreba ( arg );

	broj_procesa = atoi ( arg[1] );

	if ( broj_procesa < 1 || broj_procesa > NAJPROCESA )
		upotreba ( arg );

	/* stvori zadani broj procesa */
	stvoreno = broj_procesa;
	for ( i = 0; i < broj_procesa; i++ )
	{
		switch ( fork() )
		{
		case 0:
			radi ();
			exit (0);
		case -1:
			stvoreno = i;
			printf ( "Stvoreno %d procesa (od zadanih %d)\n",
					stvoreno, broj_procesa );
			break;
		}
	}

	printf ( "Svi procesi stvoreni, cekam kraj (Ctrl+C)\n" );

	/* cekaj da svi procesi zavrse */
	for ( i = 0; i < broj_procesa; i++ )
		wait ( NULL );

	return 0;
}

void upotreba ( char *param[] )
{
	printf ( "Upotreba: %s broj procesa\n\tbroj procesa iz intervala "
		 "[1-%d]\n", param[0], NAJPROCESA);
	exit ( 1 );
}

void radi ()
{
	while ( 1 )
		sleep ( 10 );
}

/* prevodjenje: gcc procesi.c -o proc
   pokretanje: ./proc 100 (za 100 novih procesa)
*/
```

Izvorni kod datoteke *dretve.c* za *Windows*:

```
/* kod "dretve.c" za Windows sustave */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

#define NAJDRETVI  10000
DWORD WINAPI dretva ( void *param );
void upotreba ( char *param[] );

int main ( int brarg, char *arg[] )
{
          int i, broj dretvi, stvoreno, postavi prioritet;
          DWORD idd;
          HANDLE *opisnik;
          void *param;

          /* pretpostavljene vrijednosti */
          param = NULL; postavi prioritet = 0; broj dretvi = 10;

          for ( i = 1; i < brarg; i++ ) /* arg[0] se preskace */
          {
                    if ( arg[i][0] == '-' ) /* zastavica */
                    {
                              if ( arg[i][1] == 'o' )
                                        param = (void *) 1;
                              else if ( arg[i][1] == 'i' )
                                        param = NULL;
                              else if ( arg[i][1] == 'p' )
                                        postavi prioritet = 1;
                              /* else ... - ostale zastavice se ignoriraju */
                    }
                    else { /* ocekuje se broj dretvi */
                              broj_dretvi = atoi ( arg[i] );
                              if ( broj_dretvi < 1 || broj_dretvi > NAJDRETVI )
                                        upotreba ( arg );
                    }
          }
          /* postavi prioritet pocetnoj dretvi */
          if ( postavi_prioritet )
          {
                    if ( !SetPriorityClass ( GetCurrentProcess(), HIGH_PRIORITY_CLASS ) )
                    {
                              printf ( "Prioritetna klasa nije postavljena!!!\n");
                              exit (1);
                    }
                    if ( !SetThreadPriority ( GetCurrentThread(),THREAD_PRIORITY_HIGHEST ) )
                    {
                              printf ( "Prioritet nije postavljen!!!\n");
                              exit (1);
                    }
          }
          /* stvori zadani broj dretvi */
          stvoreno = broj dretvi;
          opisnik = malloc ( broj_dretvi * sizeof (HANDLE) );
          for ( i = 0; i < broj_dretvi; i++ )
          {
                    opisnik[i] = CreateThread ( NULL, 0, dretva, param, 0, &idd );

                    if ( opisnik[i] == NULL )
                    {
                              stvoreno = i;
                              printf ( "Stvoreno %d dretvi (od zadanih %d)\n",
                                        stvoreno, broj_dretvi );
                              break;
                    }
                    else if ( postavi_prioritet )
                    {
                              if ( !SetThreadPriority ( opisnik[i], THREAD_PRIORITY_HIGHEST ) )
                              {
                                        printf ( "Prioritet nije postavljen!!!\n");
                                        exit (1);
                              }
                    }
          }
```

```
        printf ( "Sve dretve stvorene, cekam kraj (Ctrl+C)\n" );

        /* cekaj da sve dretve zavrse */
        for ( i = 0; i < stvoreno; i++ )
                WaitForSingleObject ( opisnik[i], INFINITE );

        free ( opisnik );

        return 0;
}

/* pocetna funkcija za nove dretve */
DWORD WINAPI dretva ( void *param )
{
        while ( 1 )
        {
                if ( param == NULL )
                        Sleep ( 10000 );
        }
        return 0;
}
void upotreba ( char *param[] )
{
        printf ( "Upotreba: %s [zastavice][broj dretvi]\n\tzastavice:"
                 "\n\t\t -i dretve ne rade nista"
                 "\n\t\t -o dretve rade"
                 "\n\t\t -p povecaj prioritet"
                 "\n\tbroj dretvi iz intervala [1-%d]\n", param[0], NAJDRETVI);
        exit ( 1 );
}
```

Izvorni kod datoteke *dretve.c* za *Linux*:

```
/* kod "dretve.c" za Linux sustave */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/resource.h>

#define NAJDRETVI  10000
#define PRIOMAX    -20
void *dretva ( void *param );
void upotreba ( char *param[] );

int main ( int brarg, char *arg[] )
{
        int i, broj_dretvi, stvoreno, postavi_prioritet;
        pthread_t *opisnik;
        void *param;

        /* pretpostavljene vrijednosti */
        param = NULL; postavi prioritet = 0; broj dretvi = 10;

        for ( i = 1; i < brarg; i++ ) /* arg[0] (ime programa) se preskace */
        {
                if ( arg[i][0] == '-' ) /* zastavica */
                {
                        if ( arg[i][1] == 'o' )
                                param = (void *) 1;
                        else if ( arg[i][1] == 'i' )
                                param = NULL;
                        else if ( arg[i][1] == 'p' )
                                postavi prioritet = 1;
                        /* else ... - ostale zastavice se ignoriraju */
                }
                else { /* ocekuje se broj dretvi */
                        broj_dretvi = atoi ( arg[i] );
                        if ( broj_dretvi < 1 || broj_dretvi > NAJDRETVI )
                                upotreba ( arg );
                }
        }

        /* postavi prioritet pocetnoj dretvi */
        if ( postavi_prioritet )
```

```
        {
                if ( setpriority ( PRIO_PROCESS, getpid(), PRIOMAX ) )
                        fprintf ( stderr, "Nije postavljen prioritet (ovlasti?!)\n" );
                /* Sve stvorene dretve naslijediti će prioritet ove dretve.
                   Koristenjem pthread_setschedparam funkcije mogu se postaviti i veci
                   prioriteti (Real-Time), ali to je opasno ako nije kontrolirano! */
        }

        /* stvori zadani broj dretvi */
        stvoreno = broj_dretvi;
        opisnik = malloc ( broj_dretvi * sizeof (pthread_t) );

        for ( i = 0; i < broj_dretvi; i++ )
                if ( pthread_create ( &opisnik[i], NULL, dretva, param ) )
                {
                        stvoreno = i;
                        printf ( "Stvoreno %d dretvi (od zadanih %d)\n",
                                stvoreno, broj_dretvi );
                        break;
                }

        printf ( "Sve dretve stvorene, cekam kraj (Ctrl+C)\n" );

        /* cekaj da sve dretve zavrse */
        for ( i = 0; i < stvoreno; i++ )
                pthread_join ( opisnik[i], NULL );

        free ( opisnik );

        return 0;
}

/* pocetna funkcija za nove dretve */
void *dretva ( void *param )
{
        while ( 1 )
                if ( param == NULL )
                        sleep ( 10 );
        return NULL;
}
void upotreba ( char *param[] )
{
        printf ( "Upotreba: %s [zastavice][broj dretvi]\n\tzastavice:"
                 "\n\t\t -i dretve ne rade nista"
                 "\n\t\t -o dretve rade"
                 "\n\t\t -p povecaj prioritet"
                 "\n\tbroj dretvi iz intervala [1-%d]\n", param[0], NAJDRETVI);
        exit ( 1 );
}
/* prevodjenje: gcc dretve.c -lpthread -o dretve
   pokretanje: ./dretve 10 ili ./dretve -i 10 ili ./dretve -o 10 ili
               sudo ./dretve -p 10  (10 zamijeniti s zeljenim broje dretvi)

   u drugom terminalu se mogu naredbom 'ps' vidjeti te dretve. Npr. sa:
      ps -C dretve -mLly   ili   ps -C dretve -m -o uid,pid,ppid,lwp,ni,sz,wchan,time,cmd

Primjer pokretanja i ispisa (masno su otisnute naredbe koje su zadane):
$ ./dretve -o 5 &
[1] 26751
$ Sve dretve stvorene, cekam kraj (Ctrl+C)    (enter)
$ ps -C dretve -m -o uid,pid,ppid,lwp,ni,sz,wchan,time,cmd
  UID   PID  PPID   LWP  NI    SZ WCHAN      TIME CMD
 1000 26751 14237     -   - 10687 -      00:02:43 ./dretve -o 5
 1000     -     - 26751   0     - futex_ 00:00:00 -
 1000     -     - 26752   0     - -      00:00:32 -
 1000     -     - 26753   0     - -      00:00:30 -
 1000     -     - 26754   0     - -      00:00:33 -
 1000     -     - 26755   0     - -      00:00:30 -
 1000     -     - 26756   0     - -      00:00:33 -
$ fg
./dretve -o 5
^C (Ctrl+C)
$
*/
```

Izvorni kod datoteke *sinkro.c* za *Windows*:

```c
/* kod "sinkro.c" za Windows sustave */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

#define NAJDRETVI  1000
#define NAJITER    1000000000L

DWORD WINAPI dretva ( void *param );
void upotreba ( char *param[] );

int br iteracija;
volatile int broj;
HANDLE brava;

int main ( int brarg, char *arg[] )
{
          int i, broj dretvi;
          DWORD idd;
          HANDLE *opisnik;
          void *param;

          if ( brarg < 3 )
                    upotreba ( arg );

          broj_dretvi = atoi ( arg[1] );
          if ( broj_dretvi < 1 || broj_dretvi > NAJDRETVI )
                    upotreba ( arg );

          br iteracija = atoi ( arg[2] );
          if ( br_iteracija < 1 || br_iteracija > NAJITER )
                    upotreba ( arg );

          if ( brarg > 3 && arg[3][0] == 's' )
          {
                    param = (void *) 1;
                    brava = CreateMutex ( NULL, TRUE, NULL );
          }
          else {
                    param = NULL;
          }

          broj = 0;

          /* stvori zadani broj dretvi */
          opisnik = malloc ( broj_dretvi * sizeof (HANDLE) );
          for ( i = 0; i < broj_dretvi; i++ )
          {
                    opisnik[i] = CreateThread ( NULL, 0, dretva, param, 0, &idd );

                    if ( opisnik[i] == NULL )
                    {
                              printf ( "Greska pri stvaranju dretvi!!!\n" );
                              exit (1);
                    }
          }

          printf ( "Sve dretve stvorene, cekam kraj (ili Ctrl+C)\n" );

          /* ako se koristi sinkronizacija, otkljucaj bravu */
          if ( brarg > 3 && arg[3][0] == 's' )
                    ReleaseMutex ( brava );

          /* cekaj da sve dretve zavrse */
          for ( i = 0; i < broj_dretvi; i++ )
                    WaitForSingleObject ( opisnik[i], INFINITE );

          free ( opisnik );
          CloseHandle ( brava );

          printf ( "Konacna vrijednost: broj = %d\n", broj );

          return 0;
}
```

```
/* pocetna funkcija za nove dretve */
DWORD WINAPI dretva ( void *param )
{
	int i;

	if ( param == NULL )
	{
		for ( i = 0; i < br_iteracija; i++ )
			broj++;
	}
	else {
		for ( i = 0; i < br_iteracija; i++ )
		{
			WaitForSingleObject ( brava, INFINITE );
			broj++;
			ReleaseMutex ( brava );
		}
	}

	return 0;
}

void upotreba ( char *param[] )
{
	printf ( "Upotreba: %s broj_dretvi broj_iteracija [s]\n", param[0] );
	printf ( "\topcionalni 's' oznacava da se koristi zakljucavanje\n" );
	printf ( "\tnajveci broj dretvi ogranicen na %d\n", NAJDRETVI );
	printf ( "\tnajveci broj iteracija ogranicen na %d\n", NAJITER );

	exit ( 1 );
}
```

Izvorni kod datoteke *sinkro.c* za *Linux*:

```
/* kod "sinkro.c" za Linux sustave */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NAJDRETVI  1000
#define NAJITER             1000000000L
void *dretva ( void *param );
void upotreba ( char *param[] );
int br_iteracija;
volatile int broj;
pthread_mutex_t brava;

int main ( int brarg, char *arg[] )
{
	int i, broj_dretvi;
	pthread_t *opisnik;
	void *param;

	if ( brarg < 3 )
		upotreba ( arg );

	broj_dretvi = atoi ( arg[1] );
	if ( broj_dretvi < 1 || broj_dretvi > NAJDRETVI )
		upotreba ( arg );

	br_iteracija = atoi ( arg[2] );
	if ( br_iteracija < 1 || br_iteracija > NAJITER )
		upotreba ( arg );

	if ( brarg > 3 && arg[3][0] == 's' )
	{
		param = (void *) 1;
		pthread_mutex_init ( &brava, NULL );
		pthread_mutex_lock ( &brava );
	}
	else {
		param = NULL;
	}

	broj = 0;
```

```
        /* stvori zadani broj dretvi */
        opisnik = malloc ( broj_dretvi * sizeof (pthread_t) );

        for ( i = 0; i < broj_dretvi; i++ )
        {
                if ( pthread_create ( &opisnik[i], NULL, dretva, param ) )
                {
                        printf ( "Greska pri stvaranju dretvi!!!\n" );
                        exit (1);
                }
        }

        printf ( "Sve dretve stvorene, cekam kraj (ili Ctrl+C)\n" );

        /* ako se koristi sinkronizacija, otkljucaj bravu */
        if ( brarg > 3 && arg[3][0] == 's' )
                pthread_mutex_unlock ( &brava );

        /* cekaj da sve dretve zavrse */
        for ( i = 0; i < broj_dretvi; i++ )
                pthread join ( opisnik[i], NULL );

        free ( opisnik );
        pthread_mutex_destroy ( &brava );

        printf ( "Konacna vrijednost: broj = %d\n", broj );

        return 0;
}

/* pocetna funkcija za nove dretve */
void *dretva ( void *param )
{
        int i;

        if ( param == NULL )
        {
                for ( i = 0; i < br_iteracija; i++ )
                        broj++;
        }
        else {
                for ( i = 0; i < br_iteracija; i++ )
                {
                        pthread mutex lock ( &brava );
                        broj++;
                        pthread_mutex_unlock ( &brava );
                }
        }

        return NULL;
}

void upotreba ( char *param[] )
{
        printf ( "Upotreba: %s broj dretvi broj iteracija [s]\n", param[0] );
        printf ( "\topcionalni 's' oznacava da se koristi zakljucavanje\n" );
        printf ( "\tnajveci broj dretvi ogranicen na %d\n", NAJDRETVI );
        printf ( "\tnajveci broj iteracija ogranicen na %ld\n", NAJITER );

        exit ( 1 );
}
```

Vježbe iz poglavlja 4 – Spremnički prostor

1) **Uvod:**

Veličina i svojstva spremničkog prostora uvelike određuju mogućnosti računala (uz procesor). Sami operacijski sustavi, kao i programi, postavljaju zahtjeve za potrebnu veličinu spremničkog prostora. Ako spremničkog prostora ima premalo, računalo može biti i znatno sporije od očekivanog (zbog intenzivnog korištenja diska kao pomoćnog spremnika). Ako spremničkog prostora ima previše, ono ostaje neiskorišteno (uzalud potrošen novac). Svojstva spremnika treba odabrati prema potrebi. Nije potrebno kupovati vrlo skupe spremničke module za obično uredsko računalo. S druge je strane za poslužitelje koji rade bez prestanka, za računala koji obavljaju kritične poslove (upravljanja) i sl. vrlo bitno odabrati spremnik prikladnih svojstava jer će ili greške u radu, ili (prerani) kvar, ili loše performanse u usporedbi s ostalim sastavnicama i sl. degradirati računalo ili čak onemogućiti njegovo korištenje u takvim okolinama.

Svojstva spremničkog prostora koji se nalazi u računalu mogu se utvrditi izravnim očitanjem sa spremničkih modula, posebnom opremom za njihovo ispitivanje, korištenjem programa koji su dio operacijskog sustava ili korištenjem dodatnih programa.

Poznavanje svojstava spremničkog prostora računala omogućuje i procjenu njegovih mogućnosti.

2) **Temeljni pojmovi:**

- glavni spremnik,
- pomoćni spremnik,
- priručni spremnik.

3) **Cilj vježbe:**

Korištenjem raznih alata potrebno je utvrditi svojstva spremničkog prostora, potrebe operacijskog sustava i programa za spremničkim prostorom. Korištenjem priloženog programa ispitati moguća zauzeća spremničkog prostora za programe te učinkovitost korištenja priručnog spremnika (s obzirom na način korištenja spremnika: slijedno ili nasumično).

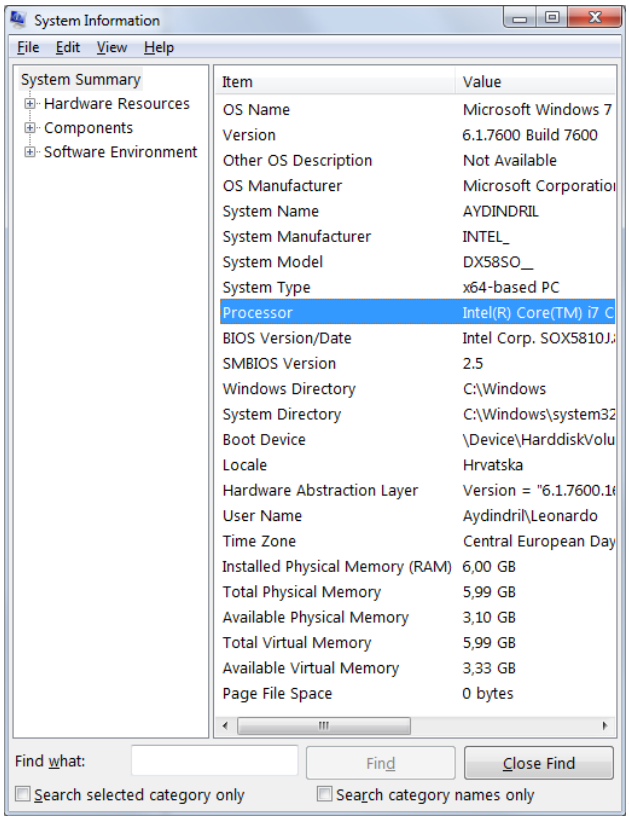
4) **Priprema za vježbe:**

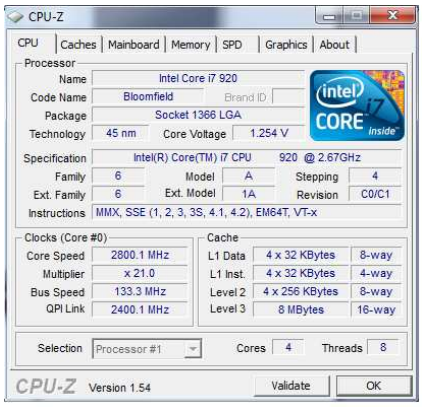
Upoznati se (teoretski) s načinom rada spremnika, njegovim parametrima, načinima njegovim upravljanjem, zahtjevima operacijskog sustava i programa prema spremniku.

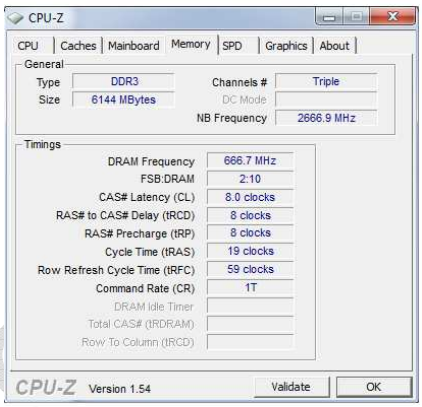
5) **Vježbe: [...]**

6) **Upute i objašnjenja uz vježbe**

Svojstva spremničkog prostora kao i njegovo trenutačno stanje mogu se doznati korištenjem programa isporučenih s operacijskim sustavom (npr. *System Information* i *Task Manager* na *Windows* sustavima te *System Monitor*, naredba *ps*, datoteka */proc/meminfo* i sl. na *Linuxu*), ali i zasebnim alatima (npr. *CPU-Z*).







S obzirom da većina operacijskih sustava koristi straničenje, bitno je znati interpretirati brojke koje programi prikazuju. Zato i priloženi program ima opciju samog zauzeća spremnika te zauzeća i korištenja. Naime, pri samom se zauzeću zauzeti segment samo opiše (u opisniku procesa), ali ne i rezervira u stvarnom spremniku. Tek se njegovim korištenjem (pisanjem i čitanjem) dio fizičkog spremnika zaista i rezervira i koristi. Na primjer, u programu *Task Manager*, uključivanjem stupaca *Working Set* (dio fizičkog spremnika koji proces koristi) te *Commit Size* (samo rezervirani dio spremnika) vidi se da se samo pri rezervaciji spremnika (priloženim programom) mijenja samo drugi stupac. No njegovim se korištenjem mijenja i prvi.

Pri većim zahtjevima, kada se raspoloživi spremnički prostor skoro do kraja iskoristi, neki operacijski sustavi daju korisnicima upozorenje da se sustav nalazi na granici. Tada se novi zahtjevi za spremnikom neće uvažavati (programi neće moći raditi). Navedeno se može i isprobati korištenjem priloženog programa, tako da se on paralelno pokreće (barem dvije instance) – veće zauzeće jednog procesa onemogućiti će drugome da zauzme veći dio (ako je spremnički prostor ograničen).

Izvorni kod datoteke *spremnik.c* (isti kod i za *Windows* i za *Linux* sustave):

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <sys/timeb.h>
#include <time.h>
#include <string.h>

#define POCETNO             10000UL
#define NAJVEL              3221225472UL       /* 3 GB */
#define NAJZAHJTEVA         32
#define RESETIRAJ           1
#define ISPISI              2

#define UPUTE                                                               \
"Zadaj naredbu: naredba velicina [velicina bloka]\n"                        \
"naredba: s - samo rezerviraj zadano\n"                                     \
"         k - koristi rezervirani spremnik\n"                               \
"         t - ispituj brzinu spremnika stavljanjem nasumicnih vrijednosti\n"\
"             u njega, ali ipak po blokovima zadanih velicina\n\n"

/*
   generator pseudo slucajnih brojeva
   (zbog mogucih ogranicenja nekih postojecih generatora, koristi se vlastiti)
*/
static unsigned long int sl_broj ()
{
          static unsigned long int slbroj = 1;

          slbroj = slbroj * 1103515245L + 12345;

          return slbroj;
}

static void mjeri_vrijeme ( int sto );

int main ()
{
          unsigned long int velicina, broj, vbloka;
          unsigned long int i, j, ni, nj;
          long int *zauzeto = NULL;
          char redak[80], naredba;

          while ( 1 )
          {
                    printf ( "\n\n%s", UPUTE );
                    fflush ( stdin );
                    (void) fgets ( redak, 80, stdin );

                    sscanf ( redak, "%c %lu", &naredba, &velicina );

                    if ( naredba != 's' && naredba != 'k' && naredba != 't' )
                    {
                              printf ( "Kriva naredba\n" );
                              continue;
                    }
                    if ( velicina < POCETNO || velicina > NAJVEL )
                    {
                              printf ( "Kriva velicina (od %lu do %lu)\n", POCETNO, NAJVEL );
                              continue;
                    }

                    if ( zauzeto )
                              free ( zauzeto );

                    zauzeto = malloc ( velicina );
                    if ( zauzeto == NULL )
                    {
                              printf ( "Nema dovoljno spremnika za zahtjev (%lu)\n", velicina);
                              continue;
                    }

                    if ( naredba == 'k' ) /* "koristi" taj segment */
                    {
                              memset ( zauzeto, 5, velicina );
                    }
```

```
                    else if (naredba == 't') /* testiraj brzinu koristenja */
                    {
                              vbloka = 0;
                              sscanf ( redak, "%c %lu %lu", &naredba, &velicina, &vbloka );

                              if ( vbloka == 0 || vbloka > velicina )
                                        vbloka = velicina;
                              else if ( vbloka < 4096 )
                                        vbloka = 4096;

                              ni = velicina / vbloka;
                              nj = vbloka / sizeof (int);

                              memset ( zauzeto, 5, velicina );

                              printf ( "\nSlijedno postavljanje (%lux%lu)!\n", ni,
                                        nj * sizeof (int) );

                              mjeri_vrijeme(RESETIRAJ);
                              for ( i = 0; i < ni; i++ )
                              {
                                        for ( j = 0; j < nj; j++ )
                                        {
                                                  broj = sl_broj() % nj;
                                                  zauzeto[i * nj + j] = broj;
                                        }
                              }
                              mjeri_vrijeme(ISPISI);

                              printf ( "\nPostavljam nasumicno (%lux%lu)!\n", ni,
                                        nj * sizeof (int) );

                              mjeri_vrijeme(RESETIRAJ);
                              for ( i = 0; i < ni; i++ )
                              {
                                        for ( j = 0; j < nj; j++ )
                                        {
                                                  broj = sl broj() % nj;
                                                  zauzeto[i * nj + broj] = broj;
                                        }
                              }
                              mjeri_vrijeme(ISPISI);
                    }
          }

          return 0;
}

/* pojednostavljeno mjerenje protoka vremena */
static void mjeri vrijeme ( int sto )
{
          static struct timeb t1, t2;
          unsigned int sek, milisek;

          if ( sto == RESETIRAJ )
          {
                    ftime( &t1 );
          }
          else if ( sto == ISPISI )
          {
                    ftime( &t2 );

                    sek = t2.time - t1.time;
                    if ( t2.millitm >= t1.millitm )
                    {
                              milisek = t2.millitm - t1.millitm;
                    }
                    else {
                              milisek = 1000 + t2.millitm - t1.millitm;
                              sek--;
                    }
                    printf ( "Proteklo: %u:%03u [s]\n", sek, milisek);
          }
}
```