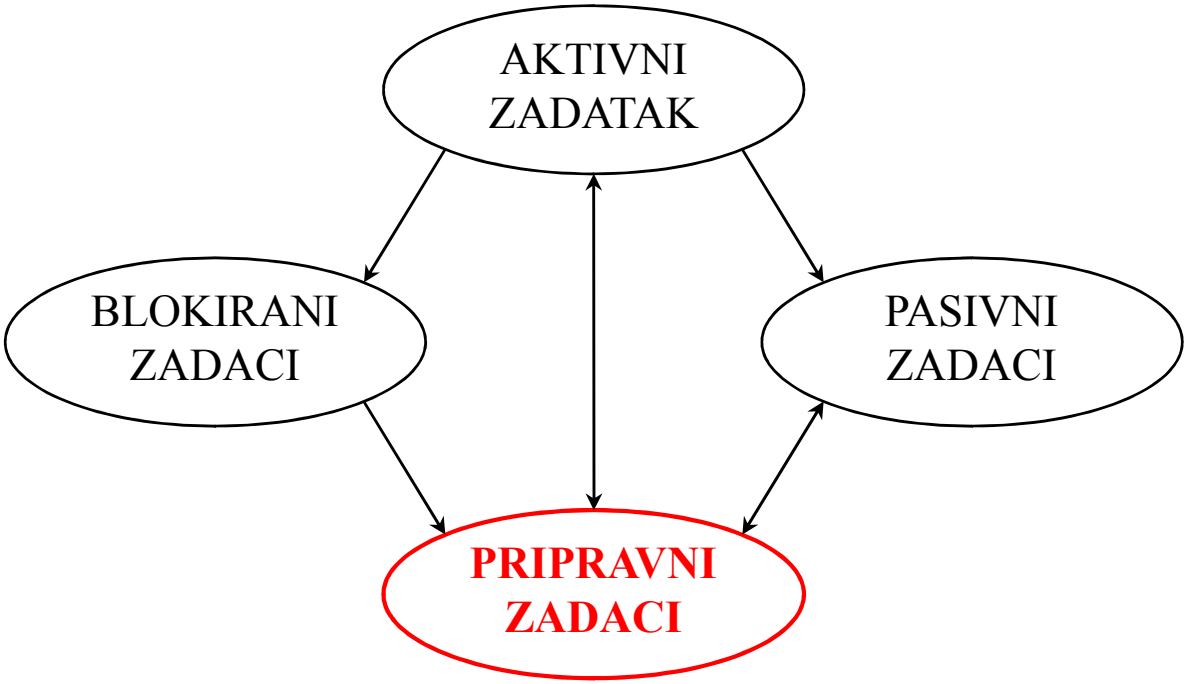
Osnovni koncepti operacijskih sustava

Raspoređivanje zadataka

Raspoređivanje zadataka (dretvi, *task scheduling*)

- Raspoređuju se pripravni zadaci



- Aktivni zadatak – trenutno se izvodi na procesoru
- Pripravni zadaci – spremni za izvođenje
- Blokirani zadaci – čekaju na nekom sredstvu (UI, sem.)
- Pasivni zadaci – zadaci koji su završili ili nisu još ni pokrenuti

Problem raspoređivanja zadataka

- Sustav ima više zadataka nego procesorskih jedinki
- Osnovni problem:
  - kako podijeliti procesorsko vrijeme među pripravnim zadacima
- Kako raspoređivati različite zadatke?
  - zadaci imaju različita svojstva i *očekivanje* od raspoređivača
  - da li koristiti upravljanje prema tipovima zadatake ili koristiti iste globalne principe raspoređivanja?
  - kako mjeriti kvalitetu raspoređivača (njegovog algoritma)? koju metriku korisititi?

Okoline u kojima se koriste raspoređivači

- U različitim se okolinama nalaze različiti zadaci, različitih zahtjeva i osnovnih ciljeva
  - **Ugrađeni sustavi** *(Embedded Systems)***, sustavi za rad u stvarnom vremenu** (*Real-Time Systems, "RT sustavi"*),
    - nadzor i upravljanje "stvarnim procesima" – vremenom usklađeno upravljanje je od ključnog značaja
  - **Osobna računala, radne stanice**
    - orijentirana (građena) prema korisničkim potrebama (izravna interakcija s korisnikom)
  - **Poslužitelji**
    - uslužno orijentirani, obrađuju zahtjeve različitih klijenata (klijenti nisu izravno korisnici, već njihove aplikacije)
  - **Mobilni uređaji** (dlanovnici, telefoni, …)
    - koristi ih jedan korisnik
    - najčešće koristi samo jednu aplikaciju (istovremeno)

Kvaliteta raspoređivanja

- U RT sustavima, loše raspoređivanje može imati i vrlo **ozbiljne posljedice**!

- Primjer *ocjena* raspoređivača za "obične" sustave
  - *multimedijalna aplikacija* mora vremenski usklađeno slati podatke audio i video podsustavu, inače će doći do degradacije kvalitete **koju će korisnik primijetiti**

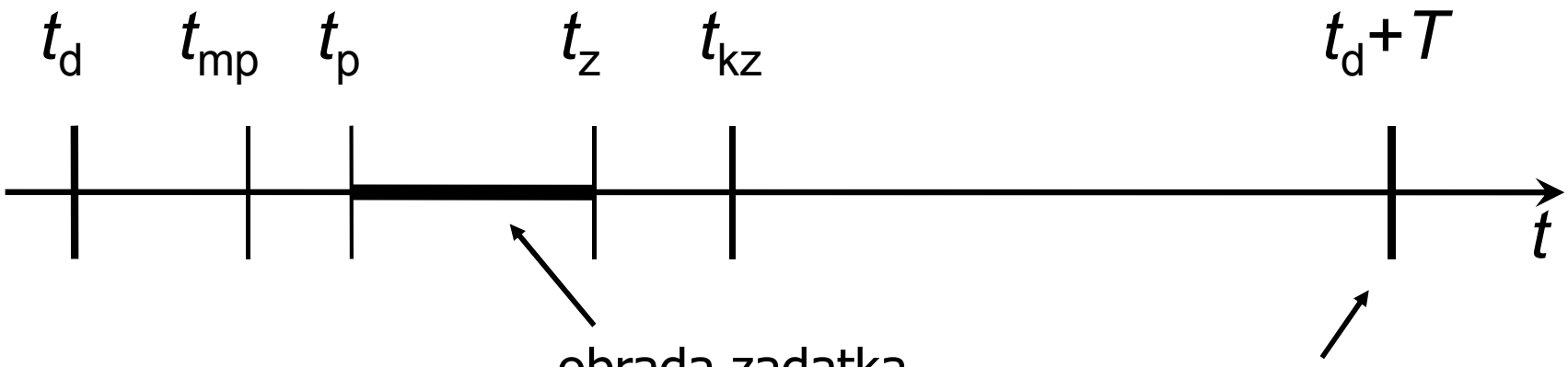
  - *korisničko sučelje* mora reagirati na korisničke naredbe
    - veća kašnjenja u odzivu na unos naredbi, operacije s mišem i sl. degradiraju kvalitetu sustava (s korisničkog stajališta)

  - *matematički proračuni, kompresija podataka, prijenos datoteka* i slične dugotrajne aktivnosti ***mogu se odgoditi*** ako je potrebno, bez primjetne degradacije kvalitete sustava

Tipovi zadataka

- Tipovi zadataka prema kriteriju “kako raspoređivati”:
  - “obični zadaci” – korisnički programi
    - izvode *standardne* operacije
    - nisu im potrebne posebne ovlasti
    - nemaju stroga vremenska ograničenja
    - koriste sredstva sustava kroz pozive operacijskog sustava
  - “zadaci sustava” – izvode operacije sustava (usluge)
    - traže povlaštene ovlasti za rad
    - mogu imati (ublažena) vremenska ograničenja
  - zadaci s vremenskim ograničenjima – RT zadaci (i multimedijalne aplikacije imaju slična ograničenja, ali …)
    - “trenutak krajnjeg završetka” mora biti poštivan za zadatak ili se događa kritična greška sustava!
    - ako je zadatak periodički, mora se obaviti barem prije idućeg pojavljivanja (“implicitnog trenutka krajnjeg završetka”)

Periodički RT zadatak – karakteristična vremena



Legenda:

t_d – trenutak dolaska
t_{mp} – trenutak mogućeg početka
t_p – trenutak početka
t_z – trenutak završetka
t_{kz} – trenutak krajnjeg završetka
T – period ponavljanja

Donošenje odluka o raspoređivanju

- **Statičko raspoređivanje**
  - **odluke su donesene prije pokretanja sustava**
    - odluke se mogu ugraditi u sustav (“hardkodirati”)
    - statički se može donijeti odluka o redoslijedu izvođenja
    - zadacima se mogu pridijeliti prioriteti te raspoređivati prema njima (u svakom trenutku odabrati zadatak najvećeg prioriteta)
- **Dinamičko raspoređivanje**
  - **odluke se donose u tijeku rada**
  - stanje sustava (pripravni zadaci) se analizira tijekom rada (*online*), te se na osnovu njega donose odluke
  - primjeri:
    - uspoređuju se dobivena procesorska vremena zadataka te zadatak s najmanjim vremenom se aktivira (da bi sustav pravedno raspodjeljivao raspoloživo vrijeme svim zadacima)
    - trenuci krajnjih završetaka (*deadline*) se uspoređuju i odabire se zadatak kojemu je to vrijeme najbliže

Opći principi raspoređivanja

- **Raspoređivanje po redu prispijeća**
  - *First Come First Served; First In First Out – FIFO*
  - prihvatljivi za poslužitelje – zahtjevi se obrađuju prema vremenima pojave (prema redu prispijeća u sustav)

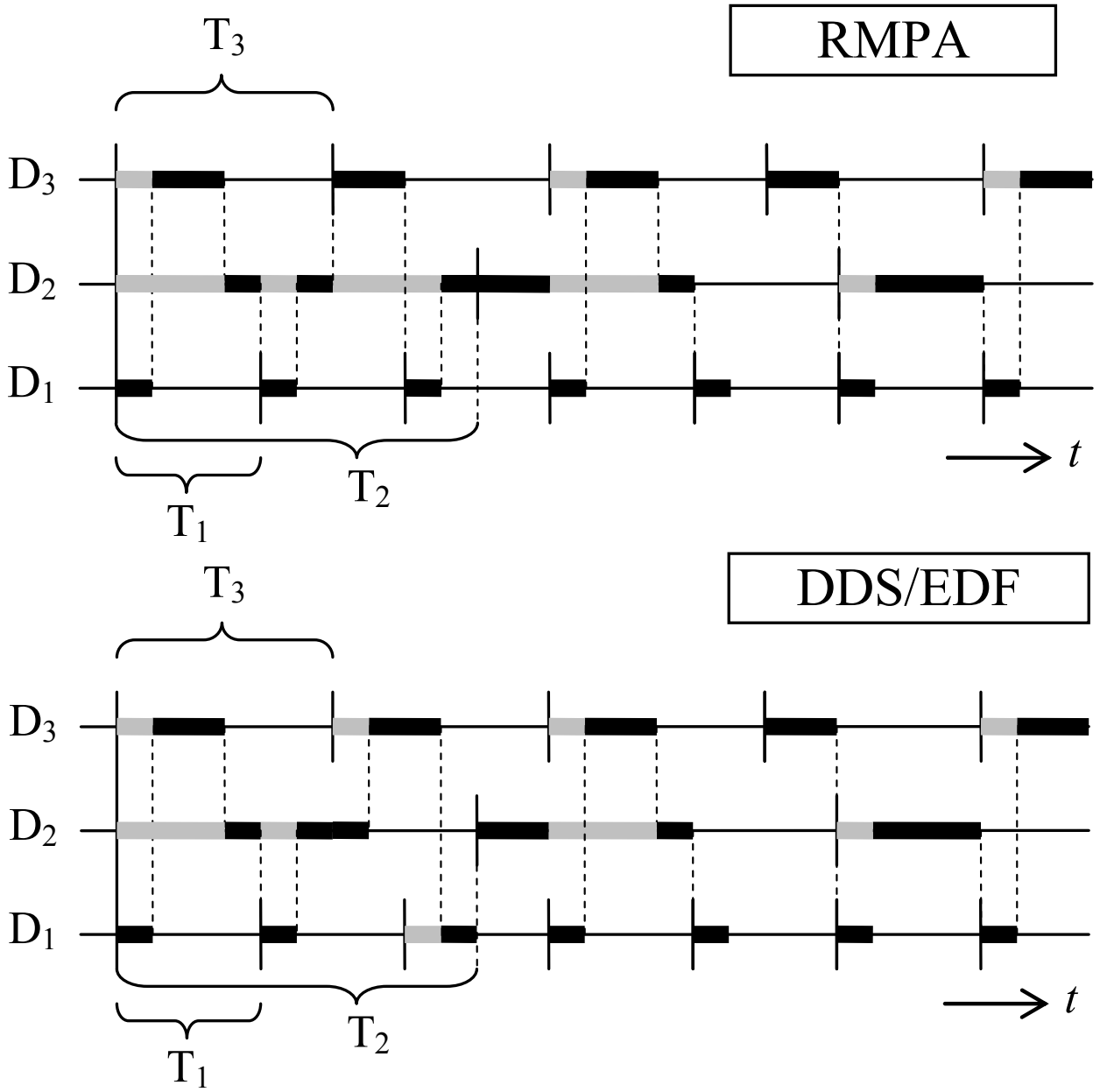
- **Raspoređivanje prema prioritetu**
  - zadaci višeg prioriteta imaju prednost i uvijek će istisnuti zadatak nižeg prioriteta
  - princip je prikladan za RT sustave – prioritet označuje važnost zadatka

- **Raspoređivanje podjelom vremena**
  - kružno posluživanje (*Round Robin*) i slična
  - zadaci (pravedno) dijele procesorsko vrijeme
  - prikladno za višekorisničke sustave
    - npr. udaljeni rad na poslužiteljima (preko tekstualnih ili grafičkih terminala)

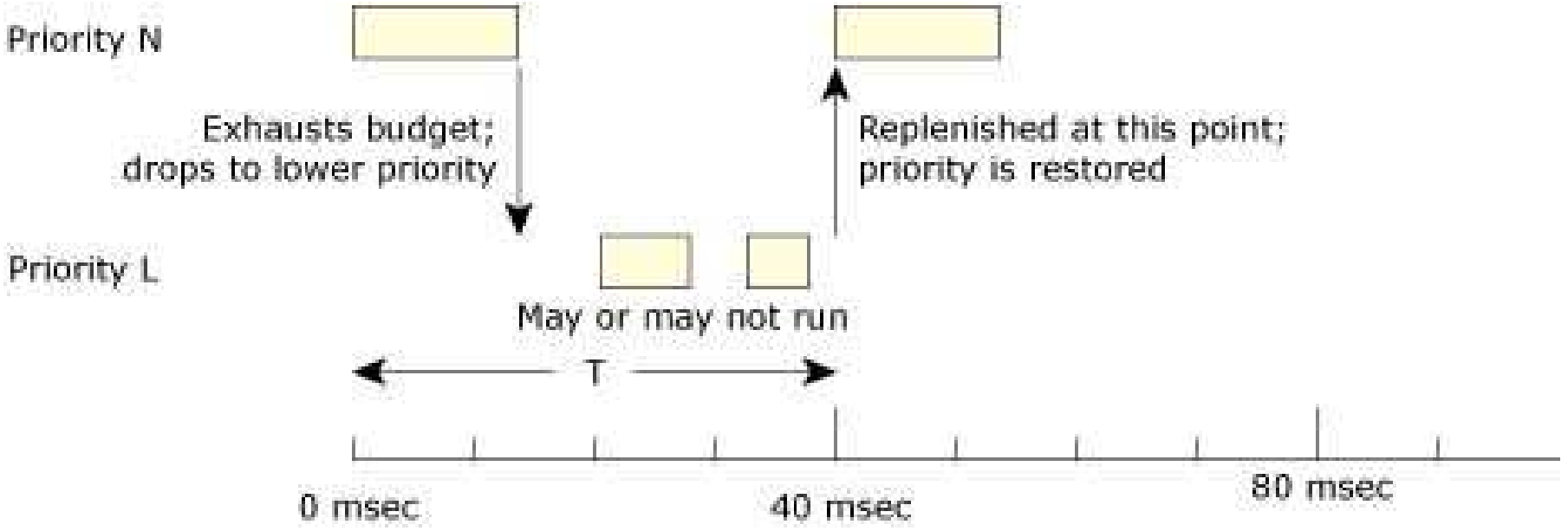
Osnovni principi raspoređivanja u RT sustavima

- **Princip mjere ponavljanja (RMS, RMPA)**
  - ***R**ate **M**onotonic **S**cheduling*, ***R**ate **M**onotonic **P**riority **A**ssignment*
  - to je najčešće korišteni (najjednostavniji) princip
  - zadacima se pridjeljuju prioriteti u skladu s njihovom učestalošću pojavljivanja:
    - zadaci koji se češće javljaju (kraće periode) dobivaju veći prioritet
  - raspoređivač koristi dodijeljene prioritete pri svom radu
- **Rasp. prema trenucima krajnjih završetaka (DDS)**
  - *Deadline Driven Scheduling (DDS), Earlier Deadline First (EDF)*
  - zadatak s najbližim trenutkom krajnjeg završetka se uvijek odabire prvi za izvođenje
- **Raspoređivanje sporadičnih zadataka**
  - zadatak tijekom jedne svoje periode, dobiva kratko vrijeme izvođenja s višim prioritetom – ako u tom vremenu ne obavi svoj periodički posao, prioritet mu se smanjuje

Primjeri RMPA i DDS raspoređivanja



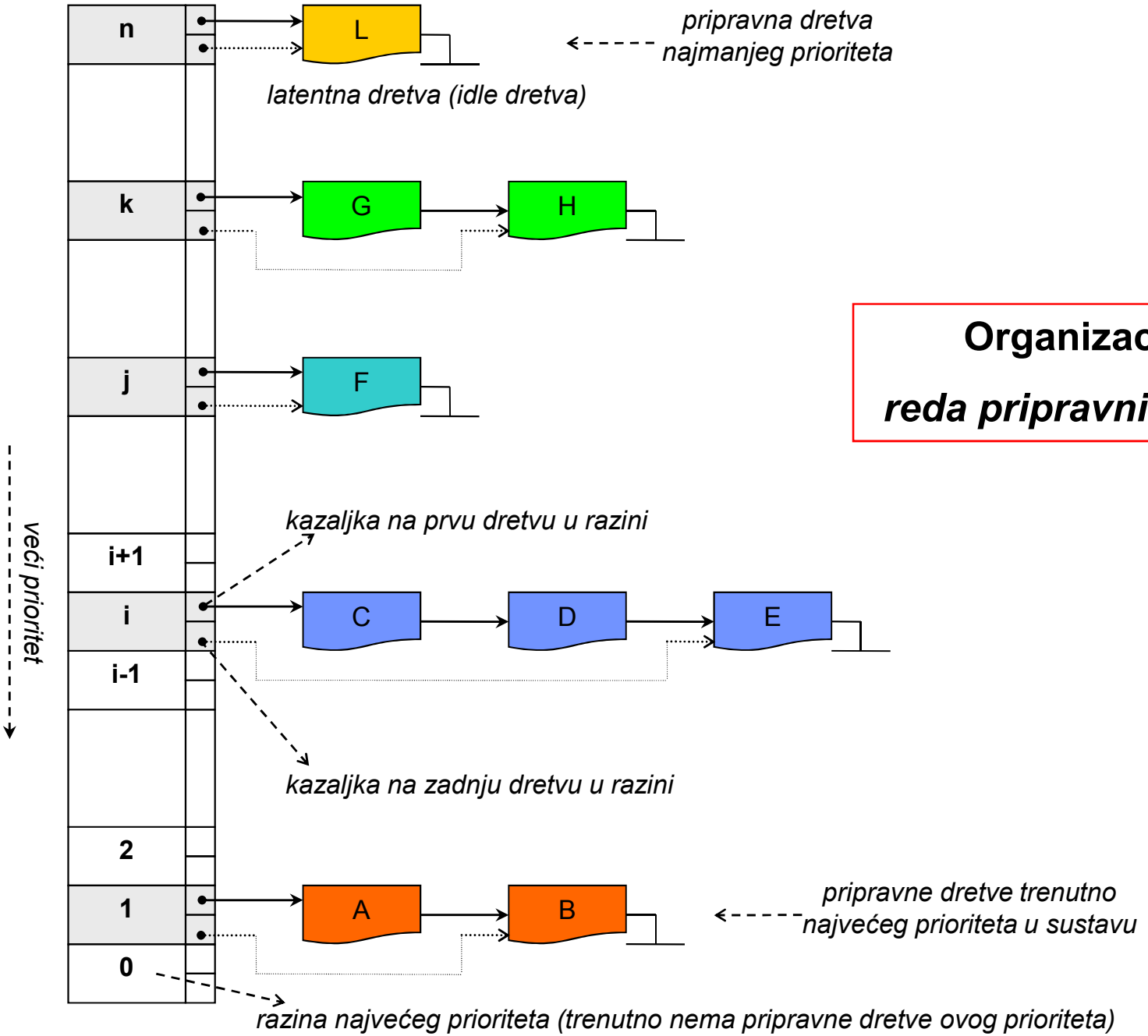
Primjer sporadičnog raspoređivanja



Izvor slike: http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/sys_arch/kernel.html

Najčešće korišteni principi raspoređivanja

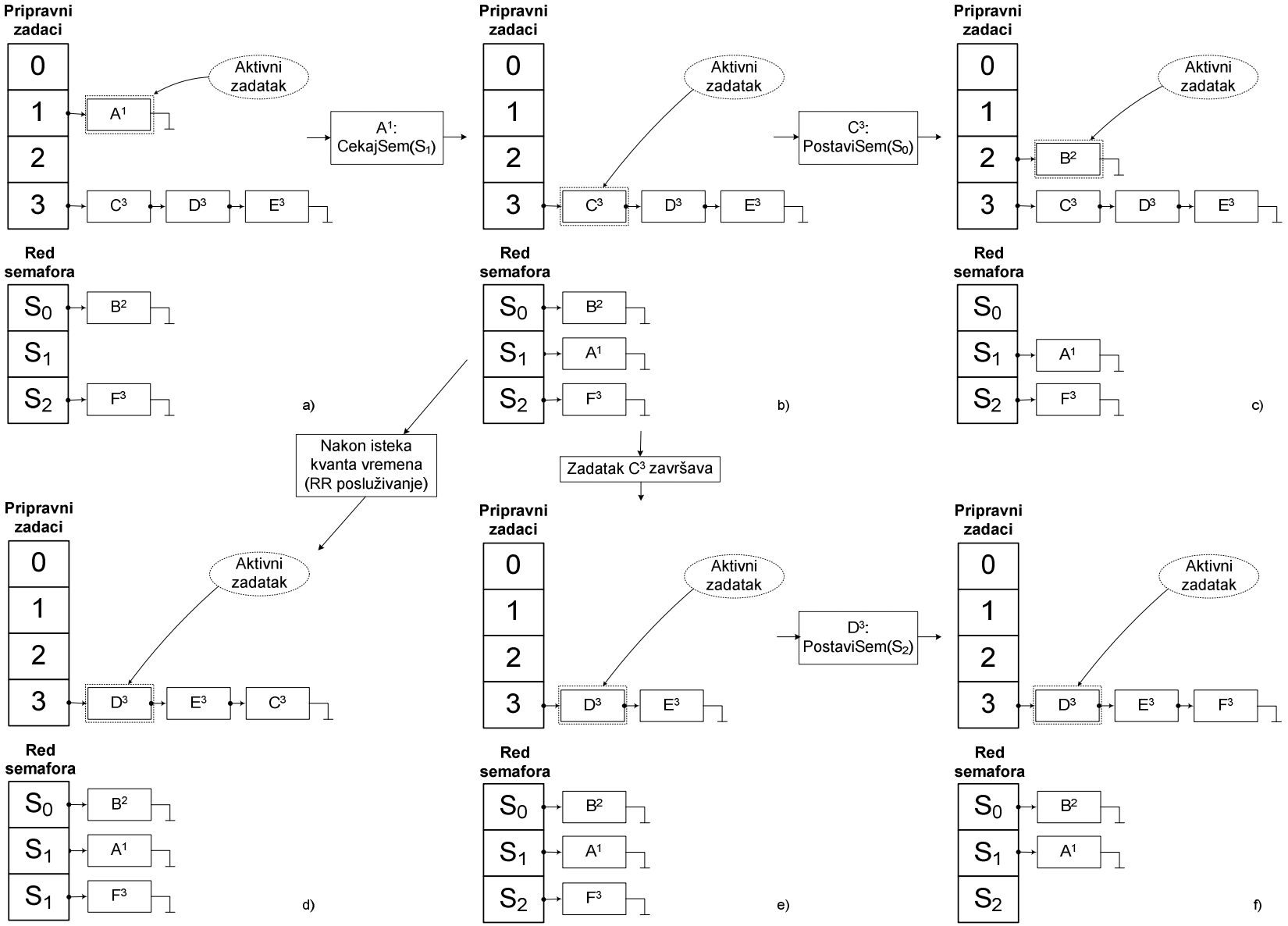
- Sustavi koji ne rade u stvarnom vremenu rade razliku između ***RT zadataka*** i ***običnih zadataka***
  - RT zadaci su većinom podržani, tj. RT raspoređivanje, ali se ono razlikuje od raspoređivanja ostalih zadataka
- Raspoređivanje **RT zadataka**
  - osnovni princip raspoređivanja je **prioritet**
  - zadatak većeg prioriteta **uvijek** istiskuje zadatak manjeg
  - pripravni su zadaci složeni u prioritetnim razinama
    - u istoj se razini (redu) nalaze zadaci jednakog prioriteta
  - ako se u istoj razini nalazi više dretvi (istog prioriteta) tada se u postupku raspoređivanja mora uzeti i drugi princip:
    - **po redu prispijeća** – uzima se prvi zadatak u razini i izvodi dok ne završi ili se ne blokira (na nekom jezgrinom pozivu), ili
    - **kružno posluživanje** – svaki zadatak dobiva samo dio procesorskog vremena (*kvant vremena*); po isteku tog vremena uzima se idući zadatak u redu (a ovaj ide na kraj)



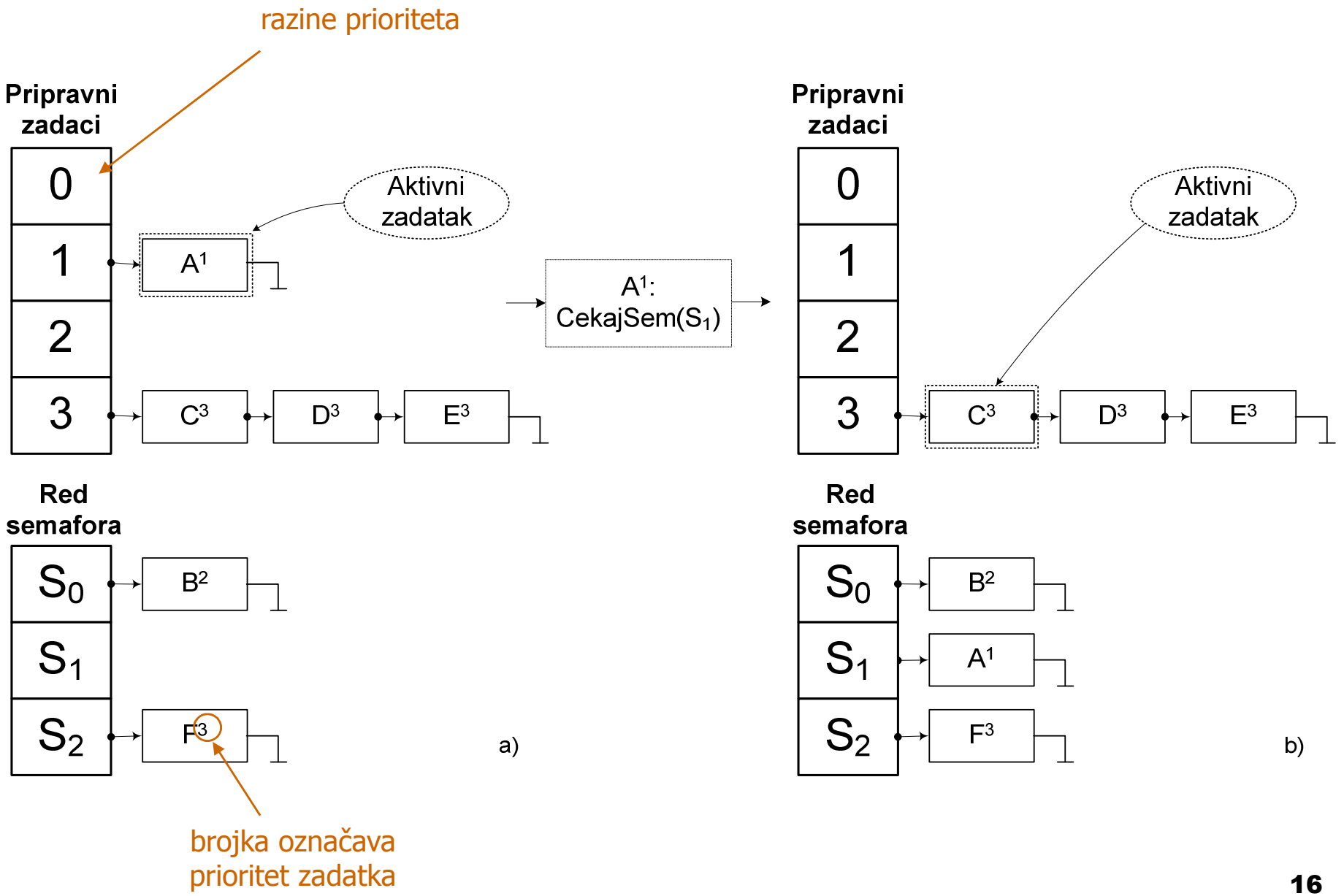
Organizacija

reda pripravnih dretvi

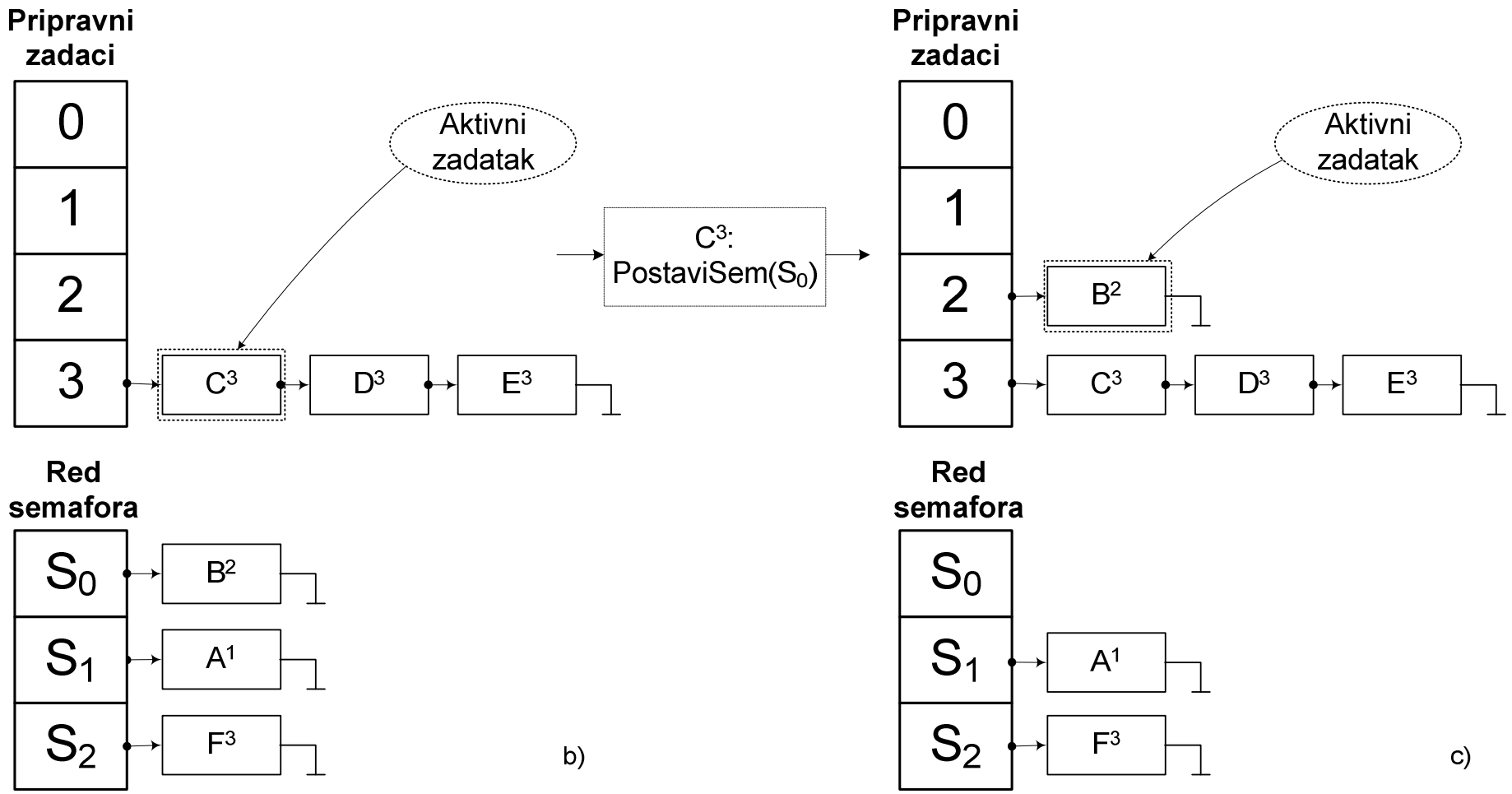
Primjer stanja sustava i odluke raspoređivanja



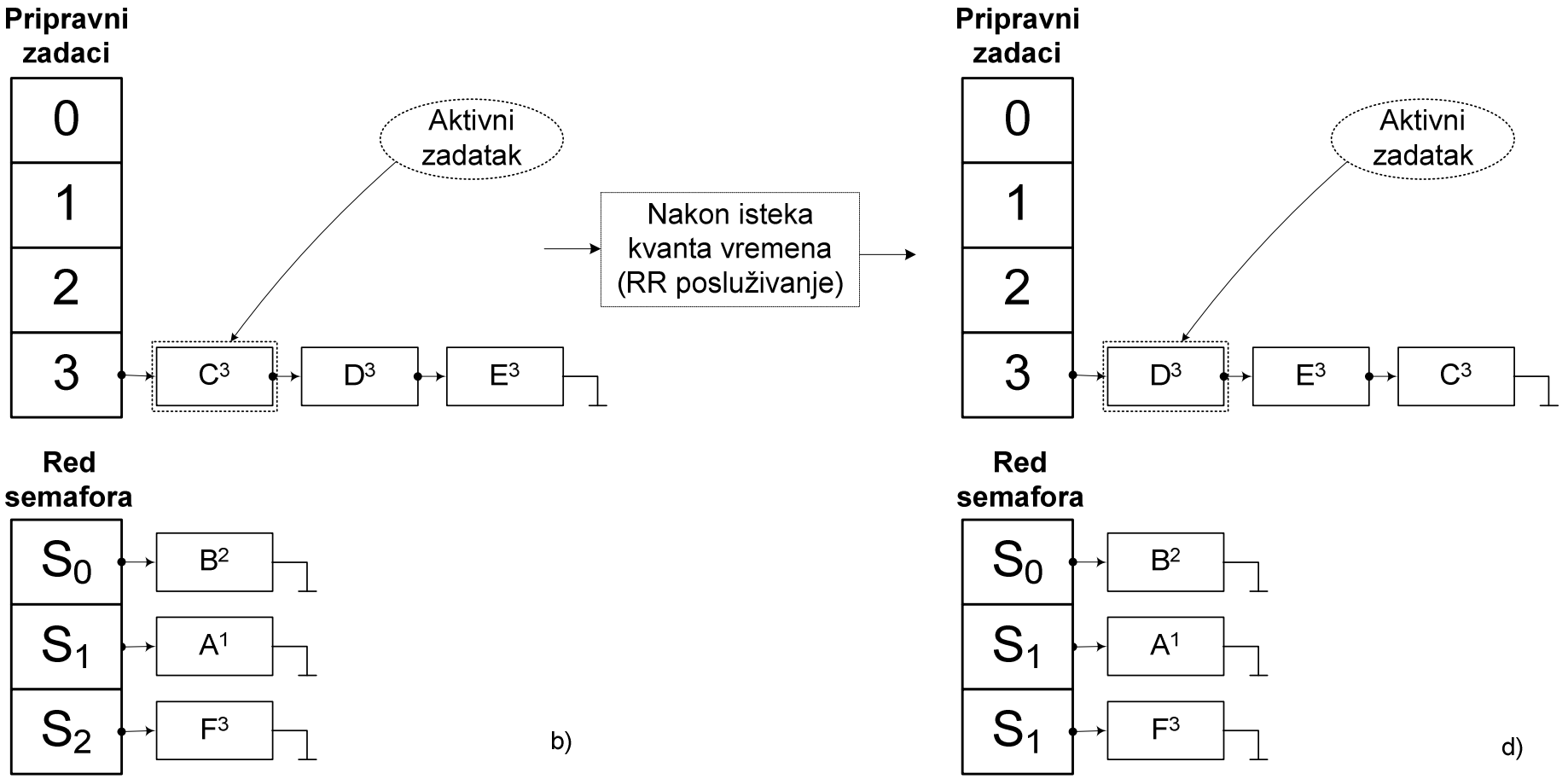
Primjer raspoređivanja: a) → b)



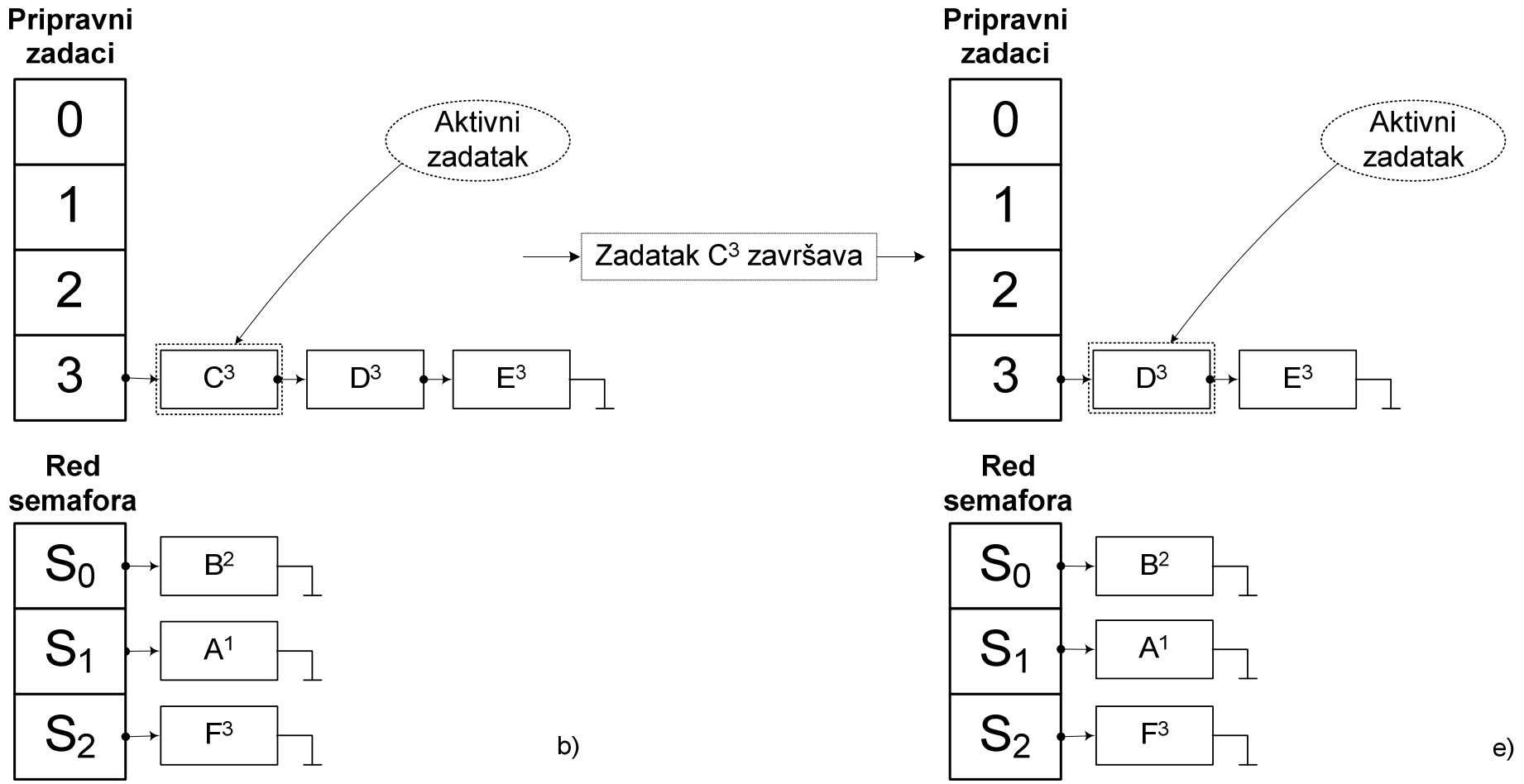
Primjer raspoređivanja: b) → c)



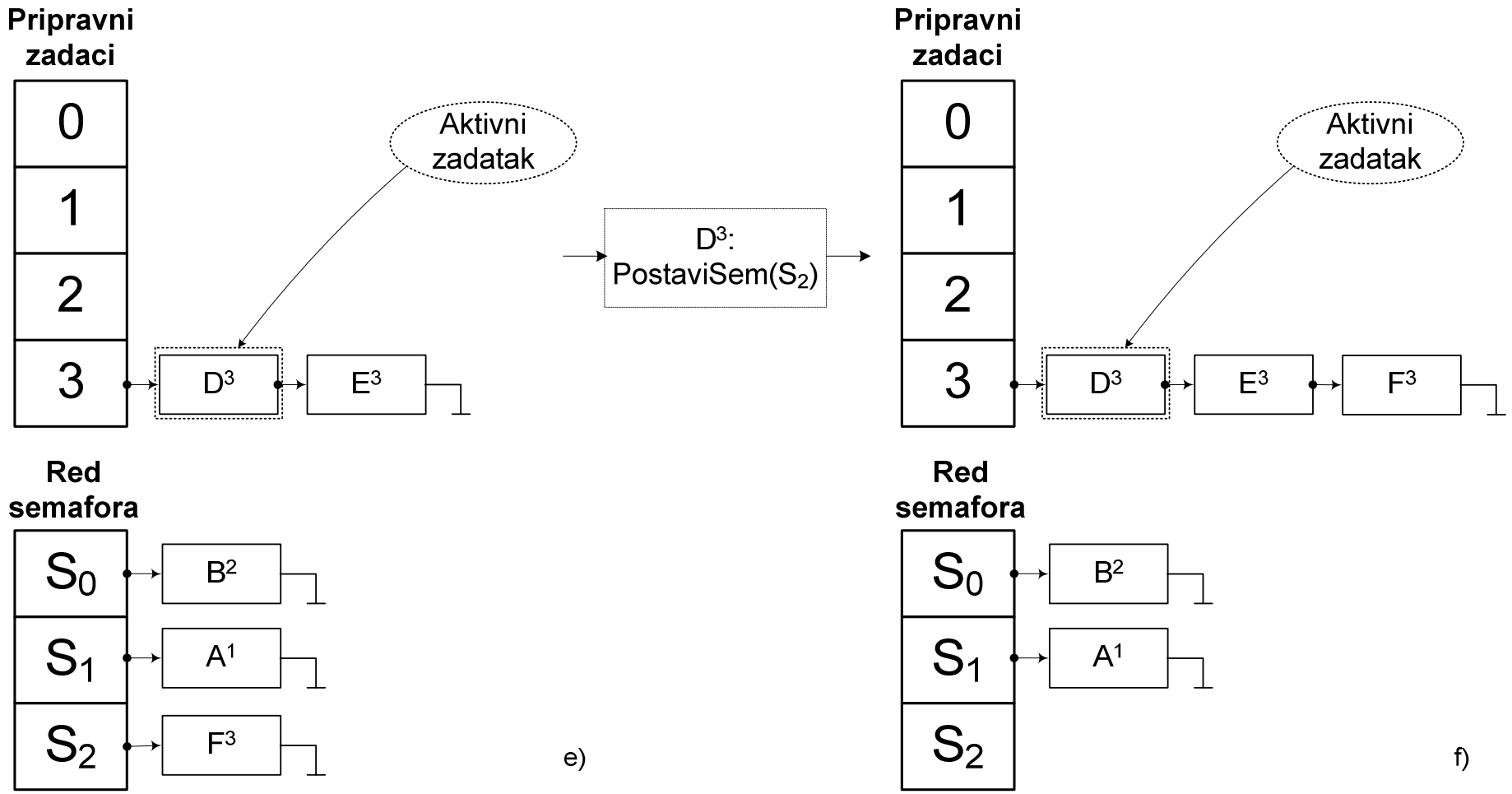
Primjer raspoređivanja: b) → d)



Primjer raspoređivanja: b) → e)



Primjer raspoređivanja: e) → f)



Raspoređivanje u ostalim, ne-RT, okolinama

- Raspoređivanje *običnih zadataka*

- Što je to *obični zadatak?*
  - zadatak sa “ublaženim vremenskim ograničenjima”, ili bez vremenskih ograničenja
    - “ublažena” – dozvoljena kašnjenja u rangu i iznad 100 ms
    - ništa se kritično neće dogoditi i ako se neka ograničenja ne zadovolje
    - međutim, znatna ili česta kašnjenja smanjuju kvalitetu sustava (korisnik je “manje” zadovoljan sustavom)

  - “radni” zadaci – intenzivno (i duže) koriste procesor
    - vrijeme do završetka je barem nekoliko sekundi
    - vrlo rijetko traže interakciju s korisnikom: obično samo naredbu pokretanja
    - ne koriste (izravno) UI uređaje, ne upravljaju ih ili nadziru

Primjeri mogućih principa raspoređivanja za obične zadatke

- **Pravednost** – pravedno raspodijeliti procesorsko vrijeme svim zadacima u sustavu (prema njihovim prioritetima)
- **Procesorska iskoristivost**
  - uz višu iskoristivost više će se posla obaviti
- **Broj završenih zadataka**
  - dovrši što više zadataka – favoriziraju se kratki zadaci
- **Minimizacija čekanja u redovima**
  - smanjuje se (prosječno) vrijeme koje zadaci čekaju prije nego li su posluženi (npr. smanji kvant vremena)
- **Kraće vrijeme odziva**
  - favoriziraj “interaktivne zadatke”– oni rijetko koriste procesor, pa im zato odmah posveti pažnju čim budu pripravni (obično upravljaju s korisničkim sučeljem ili UI jedinicama)
- **Optimizacija korištenja priručnih spremnika u sustavima s više procesorskih jedinki**
  - zadržavaj zadatak na istom procesoru – u sukcesivnim izvođenjima (nakon zamjene s drugim zadacima) ti zadaci mogu pronaći svoje podatke još uvijek u priručnom spremniku!

Višerazinsko raspoređivanje s povratnom vezom

- Izvorno "*Multilevel Feedback Queue*" (MFQ u nastavku)

- Teoretski način (princip) raspoređivanja običnih zadataka koji se načelno koristi u većini današnjih (običnih) operacijskih sustava

MFQ raspoređivanje (izvor Wikipedia)

- Ciljevi MFQ raspoređivanja za višenamjenske sustave:
  - preferirati kratke zadatke
  - preferirati zadatke koji koriste UI naprave
  - na osnovi rada zadatka vrlo brzo odrediti u koju kategoriju zadatak pripada te ga prema tome i raspoređivati

MFQ raspoređivanje (izvor Wikipedia, nastavak)

- U raspoređivanju se koriste višerazinski FIFO redovi:
  a) uvijek se raspoređuje/izvodi prvi zadatak u najvišem nepraznom redu
  b) kad u sustav dođe **novi zadatak**, on se stavlja u najvišu razinu (na kraj reda u toj razini)
  c) nakon nekog vremena taj zadatak dođe na prvo mjesto reda, a potom se raspoređuje na procesor
  d) ako se u tom izvođenju zadatak **obavi do kraja** – napušta sustav
  e) ako se u tom izvođenju **blokira** (prepušta kontrolu drugima), tada se pri povratku u pripravno stanje stavlja u **isti red** (na kraj)
  f) ako u izvođenju **potroši cijeli kvant** vremena (i još bi), tada se zadatak premješta **u prvu nižu razinu**
  g) postupak se ponavlja (koraci c) do f)) dok god zadatak ne završi ili ne padne u **najnižu razinu**

MFQ raspoređivanje (izvor Wikipedia, nastavak)

- U najnižoj razini zadaci se poslužuju **podjelom vremena** (*Round Robin*) dok ne završe i napuste sustav
- Opcionalno, ako se zadatak blokira na UI jedinici (*ili drugdje**) na “duže vrijeme”, pri povratku u pripravno stanje (kad se odblokira) zadatak se **promovira** stavljanjem u prvi viši red
- MFQ raspoređivanje daje zadatku jednu priliku da ostane u istoj razini prije nego li je pomaknut u nižu*
- *u stvarnim sustavima je stvar “labavija” – ako zadatak duže vrijeme “ne radi” (blokiran je), on biva pomican u više razine (nema “samo jednu priliku”)

Raspoređivanje običnih dretvi u Linux-u

- U nastavku se koristi pojam ***dretva*** umjesto ***zadatka***, jer je to uobičajen naziv u okviru operacijskih sustava (u teoriji “zadatak”, u praksi “dretva”)

- Linux koristi “Potpuno pravedan raspoređivač”
  - *Completely Fair Scheduler - CFS*
  - koristi se od jezgre oznake 2.6.23 (2007.)

- Osnovni CFS princip (teorija)
  - za sustav s *N* procesora i *M* dretvi (*N*<*M*), svaka dretva **treba dobiti *N/M* udio** u ukupnom procesorskom vremenu (uz pretpostavku istih prioriteta)
  - za svaku se dretvu izračunava/prati “**vremenski dug** prema dretvi” (*wait runtime*) koji se računa kao razlika između ***pripadajućeg*** i ***stvarno dodijeljenog*** vremena
  - **dretva s najvećim dugom se odabire za izvođenje**

Raspoređivanje običnih dretvi u Linux-u

- Primjer za sustav dretvi istog prioriteta (“*nice*”)
  - U nekom trenutku dretva τ_1 je raspoređena te dobiva kvant vremena *T*
  - Po isteku kvanta vremena raspoređivač ažurira “dug” za sve dretve prema (indeks identificira dretvu):
    - $wr_1 = wr_1 + T/N - T$ – dug prema τ_1 se smanjuje!
    - $wr_2 = wr_2 + T/N$ – dug prema τ_2 se povećava
    - $wr_3 = wr_3 + T/N$ – dug prema τ_3 se povećava
    - …
- U implementaciji se koriste “crveno-crna stabla” za organizaciju “reda” pripravnih dretvi
  - “dug” prema dretvi određuje njen položaj u stablu

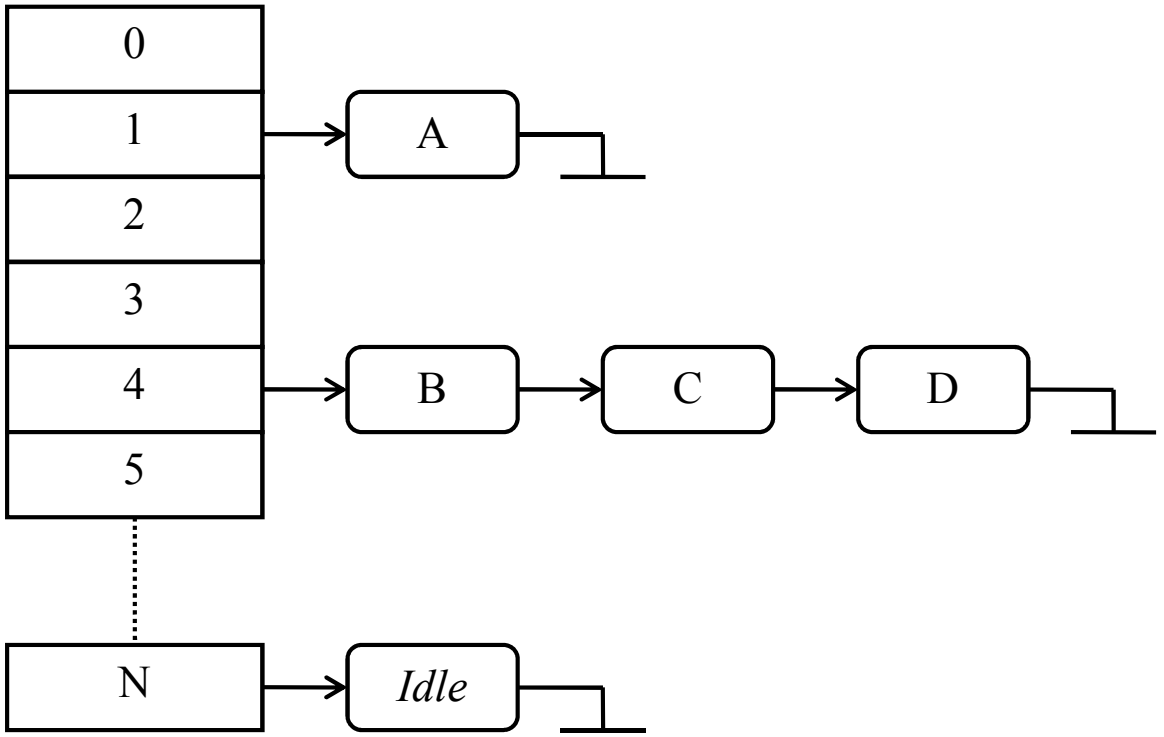
Raspoređivanje u Win32 sustavima

- Dretve se raspoređuju **prema njihovim prioritetima** – dretve najvećeg prioriteta dobivaju gotovo svo vrijeme, ostale tek toliko da se ne dogodi izgladnjivanje
- **Osnovni prioritet** dretve se izračunava iz *prioritetne klase procesa* te *prioritetne razine dretve*
  - za obične dretve prioritet je u rangu **1-15** (16-31 za RT)
- Prioritet se može i povećati
  - dretve u fokusu (*foreground*)
  - dretve koje rade s UI napravama (i rijetko koriste procesor)
- Prioritet se može i smanjiti za procesorski zahtjevne dretve, ali samo do njihova osnovna prioriteta
- Dretve najvećeg (istog) prioriteta se raspoređuju podjelom vremena

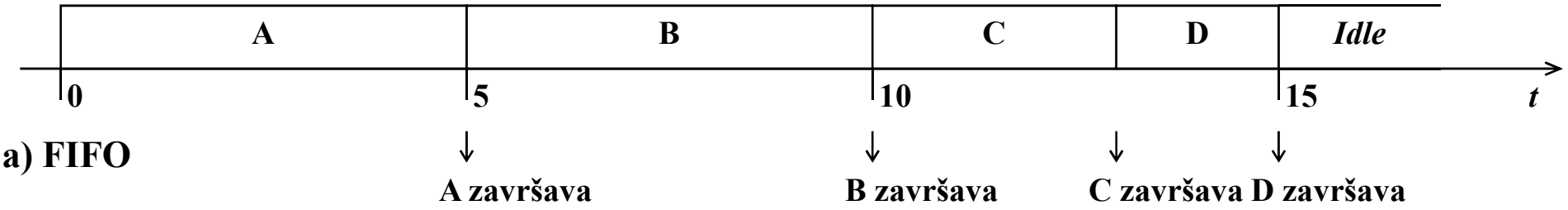
Primjer: usporedba različitih raspoređivanja

(Izvor: *Operacijski sustavi*, L. Budin i ostali, 2010.)

- Četiri zadatka A, B, C i D s vremenima izračunavanja $T_P(A) = 5$, $T_P(B) = 5$, $T_P(C) = 3$ i $T_P(D) = 2$ (vr. jedinica)
- prioriteti su prikazani na slici – manji broj = veći prioritet

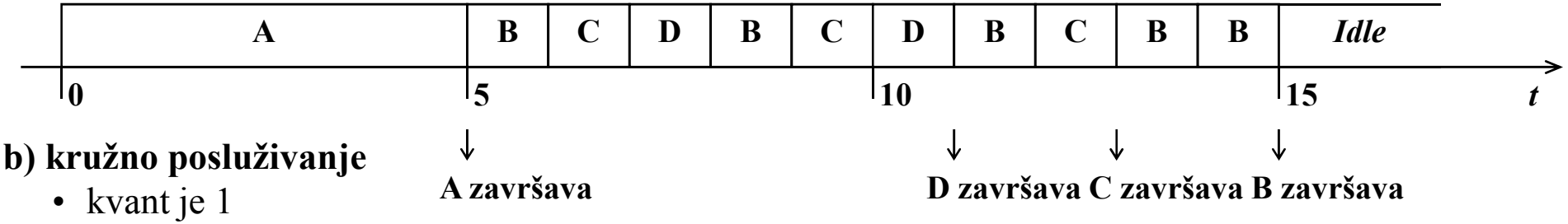


Usporedba različitih raspoređivanja na primjeru



a) FIFO

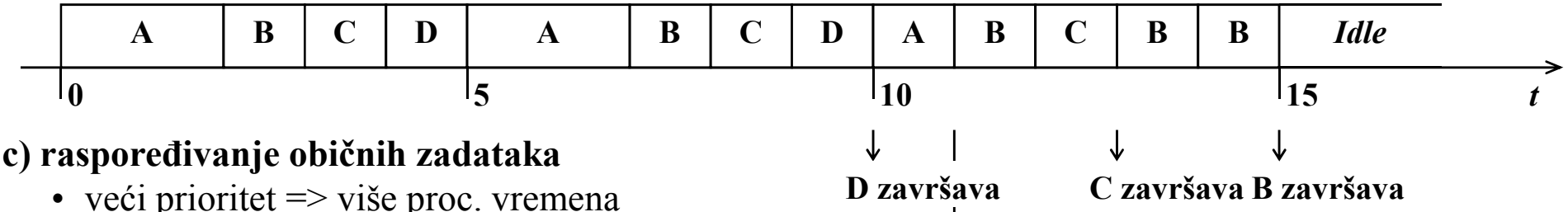
A završava B završava C završava D završava



b) kružno posluživanje

- kvant je 1
- B, C i D imaju isti prioritet – na njih se primjenjuje kružno posluživanje

A završava D završava C završava B završava



c) raspoređivanje običnih zadataka

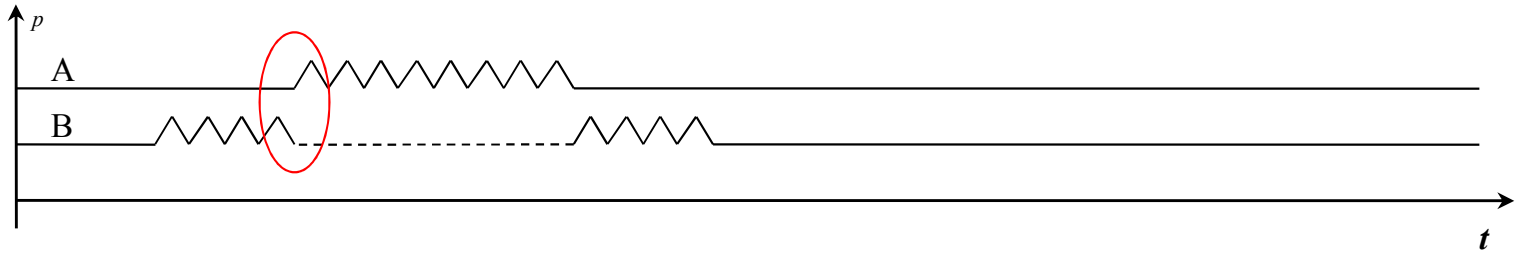
- veći prioritet => više proc. vremena
- u ovom primjeru "više vremena" je postignuto većim kvantom za zadatke većeg prioriteta (A dobiva kvant = 2, dok je on za ostale = 1)

D završava C završava B završava

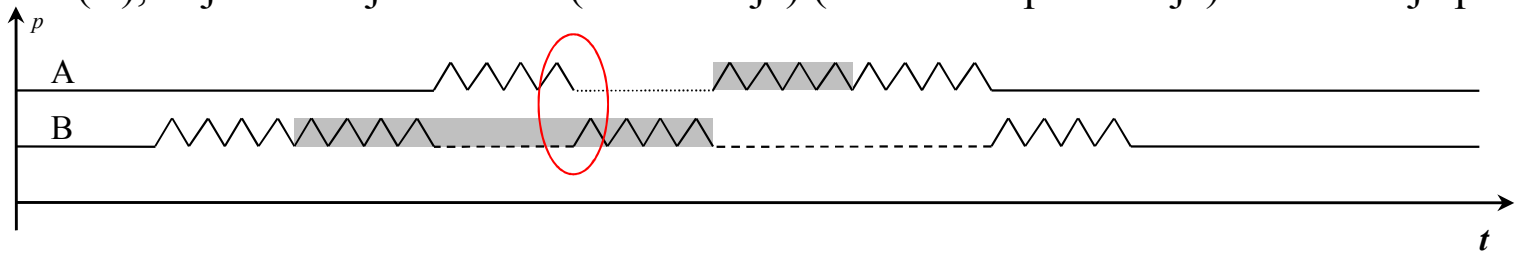
A završava

Problem inverzije prioriteta (RT raspoređivanje)

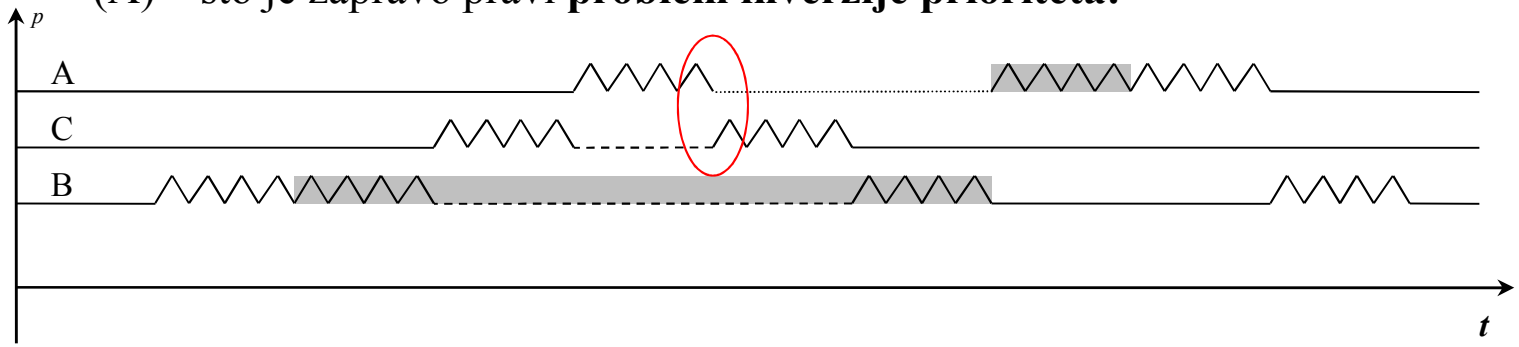
a) zadatak višeg prioriteta (A), istiskuje zadatak nižeg prioriteta (B) (očekivano)



b) zadatak višeg prioriteta (A) se blokira na sredstvu koje je zauzeo zadatak nižeg prioriteta (B), koji nastavlja s radom (B nastavlja) (očekivano ponašanje) – "inverzija prioriteta"



c) zadatak višeg prioriteta (A) se blokira (kao i u b)), ali zadatak srednjeg prioriteta (C) dodatno odgađa izvođenje zadatka koji ima sredstvo, a time i zadatak najvećeg prioriteta (A) – što je zapravo pravi **problem inverzije prioriteta!**

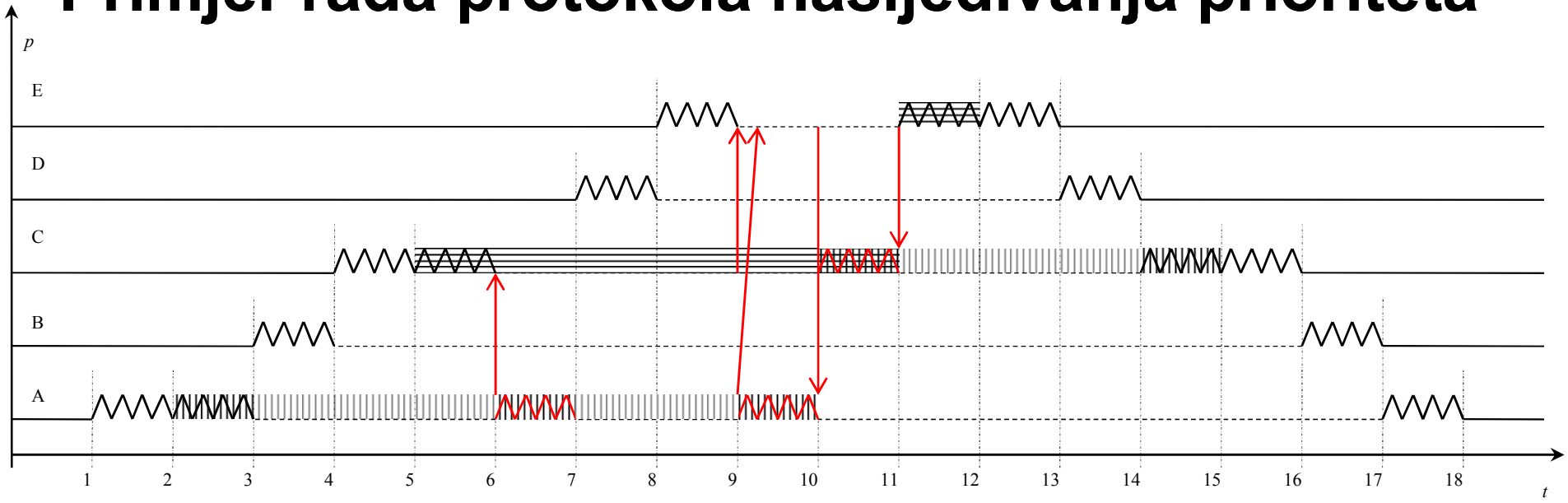


Rješenja za problem inverzije prioriteta

- **Protokol nasljeđivanja prioriteta**
  - kada se zadatak A višeg prioriteta blokira zbog sredstva koje je prethodno zauzeo zadatak B nižeg prioriteta, tada se zadatku B podigne prioritet na razinu zadatka A (B “naslijedi prioritet od A”)
  - kada zadatak B otpusti sredstvo, vraća mu se prijašnji prioritet (a A nastavlja, jer dobiva otpušteno sredstvo)
  - zadaci “srednjeg prioriteta” (manjeg od A ali većeg od B) neće u tom slučaju odgađati zadatak A!
- **Protokol stropnog prioriteta**
  - kada zadatak zauzme sredstvo tada mu se prioritet poveća na unaprijed definirani stropni prioritet za to sredstvo (tj. ako je taj prioritet veći od prioriteta zadatka)
  - otpuštanjem sredstva, zadatku se vraća prijašnji prioritet

- Protokoli se ugrađuju u sinkronizacijske funkcije
- Protokoli su neophodni za RT sustave

Primjer rada protokola nasljeđivanja prioriteta



Opis događaja po trenucima:

1 – A, kao jedina pripravna dretva započinje s radom
2 – A zauzima sredstvo S_1 (okomite crte)
3 – B započinje s radom i istiskuje A jer ima najveći prioritet (u tom trenutku)
4 – C započinje s radom i istiskuje B jer ima najveći prioritet
5 – C zauzima sredstvo S_2 (vodoravne crte)
6 – C treba S_1 koji je A zauzeo, A nasljeđuje prioritet od C i nastavlja s radom (s prioritetom od C)
7 – D započinje s radom i istiskuje A jer ima najveći prioritet
8 – E započinje s radom i istiskuje D jer ima najveći prioritet
9 – E treba S_2 koji je C zauzeo, C nasljeđuje prioritet od E, A nasljeđuje prioritet od C i nastavlja s radom (s prioritetom od E)
10 – A oslobađa S_1, vraća mu se početni prioritet, C zauzima S_1 i nastavlja s radom (s prioritetom od E)
11 – C oslobađa S_2, vraća mu se početni prioritet, E zauzima S_2 i nastavlja s radom
12 – E oslobađa S_2 i nastavlja s radom
13 – E završava, D nastavlja s radom (ima najveći prioritet)
14 – D završava, C nastavlja s radom (ima najveći prioritet)
15 – C oslobađa S_1 i nastavlja s radom
16 – C završava, B nastavlja s radom (ima najveći prioritet)
17 – B završava, A nastavlja s radom (jedina dretva)
18 – A završava

Primjer rada – datoteka: nas_prio.c

Raspoređivanje – sažetak

- Raspoređivanje RT zadataka:
  - **raspoređivanje prema prioritetu** – primarni princip
  - kada ima više pripravnih dretvi istog najvišeg prioriteta tada se koriste **FIFO** i/ili **kružno posluživanje (RR)**
  - obzirom da je prioritet uvijek privi princip, pri imenovanju načina raspoređivanja koristi se drugi princip!
  - FIFO i RR su implementirani u gotovo svim sustavima
    - ponašanje je isto! (razlike su u oznakama prioriteta i u formatu sučelja)
- Raspoređivanje običnih zadataka:
  - osnovni princip je ravnopravna (pravedna) podjela procesorskog vremena (uzimajući u obzir prioritet)
  - zadaci većeg prioriteta dobivaju više procesorskog vremena, ali veći prioritet nije razlog istiskivanja!
  - implementacija je različita na različitim sustavima (uzima se u obzir posebnost sustava i tome se prilagođava i raspoređivanje zadataka)

Osnovni koncepti operacijskih sustava

POSIX sučelje za raspoređivanje

Raspoređivanje: algoritam i prioritet

- Raspoređivanje je određeno:
  - **odabirom algoritma raspoređivanja** *(policy)*:
    - **`SCHED_FIFO`** – FIFO raspoređivanje RT dretvi
    - **`SCHED_RR`** – kružno raspoređivanje RT dretvi
    - **`SCHED_SPORADIC`** – sporadično raspoređivanje RT dretvi
    - **`SCHED_OTHER`** – raspoređivanje običnih dretvi (ne-RT)
  - **prioritetom dretve** (veći prioritet, prije dolazi na red)
    - prioritet je broj u rangu:
      - `int sched_get_priority_min(int policy);`
      - `int sched_get_priority_max(int policy);`
- Algoritam i prioritet mogu se odabrati pri stvaranju dretve, ali se mogu i naknadno mijenjati
- Uobičajeno je da stvaranje RT dretvi može raditi samo korisnik s visokim privilegijama (*root*/*administrator*)

Postavljanje parametara raspoređivanja za novu dretvu

- Parametar tipa **`pthread_attr_t`** koji se šalje u funkciju stvaranja nove dretve (**`pthread_create`**) sadrži (može sadržavati) podatke o željenim algoritmima raspoređivanja dretve, te njen osnovni prioritet
- Funkcije za postavljanje parametara raspoređivanja
  - **`pthread_attr_setinheritsched`**
    - da li da nova dretva naslijedi parametre od dretve koja ju stvara ili ne:
      - (`PTHREAD_INHERIT_SCHED`, `PTHREAD_EXPLICIT_SCHED`)?
  - **`pthread_attr_setschedpolicy`**
    - postavljanje algoritma (`SCHED_FIFO`, `SCHED_RR`, `SCHED_SPORADIC` ili `SCHED_OTHER`)
  - **`pthread_attr_setschedparam`**
    - definiranje osnovnog prioriteta

Mijenjanje parametara raspoređivanja postojećoj dretvi

- Mijenjanje algoritma i prioriteta:
  - **pthread_setschedparam(**
    pthread_t *thread*,
    int *policy*,
    const struct sched_param **param*)

- Mijenjanje samo prioriteta
  - **pthread_setschedprio(**
    pthread_t *thread*,
    int *prio*);

- Ekvivalentne funkcije na razini procesa:
  - **sched_setscheduler(pid, policy, param)**
  - **sched_setparam(pid, param)**

Utjecaj sinkronizacijskih funkcija na raspoređivanje

- Monitor (**`pthread_mutex_lock/unlock`**)
  - može se postaviti protokol nasljeđivanja prioriteta ili protokol stropnog prioriteta:

```
int pthread_mutexattr_setprotocol(
        pthread_mutexattr_t *attr,
        int protocol);
```

  - vrijednosti za parametar ***`protocol:`***

    `PTHREAD_PRIO_INHERIT` – nasljeđivanje prioriteta
    `PTHREAD_PRIO_PROTECT` – stropni prioritet
    `PTHREAD_PRIO_NONE`

  - Postavljanje vrijednosti stropnog prioriteta:

```
int pthread_mutex_setprioceiling(
        pthread_mutex_t *mutex,
        int prioceiling,
        int *old_ceiling);
```

POSIX sučelja – problemi

- POSIX sučelja su u konstantnom razvoju (“usavršavaju se”, prilagođavaju, …)
- Međutim, POSIX sučelja su definicije *sučelja* i samo su preporuke koje dozvoljavaju različite implementacije (“ispod”)
- U različitim sustavima neke su funkcionalnosti različito implementirane (ili i nisu implementirane (potpuno))
  - pogotovo u “običnim” sustavima (npr. Linux, Solaris)

Primjer korištenja sučelja

- Datoteka: nas_prio.c