Jezgro i upravljanje procesima

Nemanja Maček

Sadržaj

- Jezgro
- Pojam procesa
- Predstavljanje procesa
- Dijagram stanja procesa
- Raspoređivanje procesa
- Operacije nad procesima
- Rutine za interprocesnu komunikaciju
- Niti - laki procesi

Kernel

- **Kernel** (jezgro) je osnovni je deo svakog OS.
- U hijerarhijskom (slojevitom) modelu, kernel je najbliži hardveru.
  - Kernel je veza, odnosno interfejs između hardvera i ostalih slojeva OS.
  - U slojevitom modelu NT arhitekture ispod kernela se nalazi sloj apstrakcije hardvera (engl. *Hardware Abstraction Layer*) koji omogućava OS da vidi različit hardver na isti način.
- Osnovna funkcija kernela je upravljanje procesima, odnosno:
  - stvaranje okoline u kojoj mogu postojati procesi,
  - dodeljivanje procesora procesima i
  - obezbeđivanje mehanizama za interprocesnu komunikaciju.
- Procesor (sa jednim jezgrom) je **nedeljivi resurs**!
  - Na jednom procesoru sa jednim jezgrom u jednom trenutku može se izvršavati samo jedan proces.
  - Kernel određuje kada i na koje vreme će proces dobiti procesor.
  - Ova pojava je poznata pod imenom multipleksiranje i predstavlja osnovu kvaziparalelnosti.

Kernel

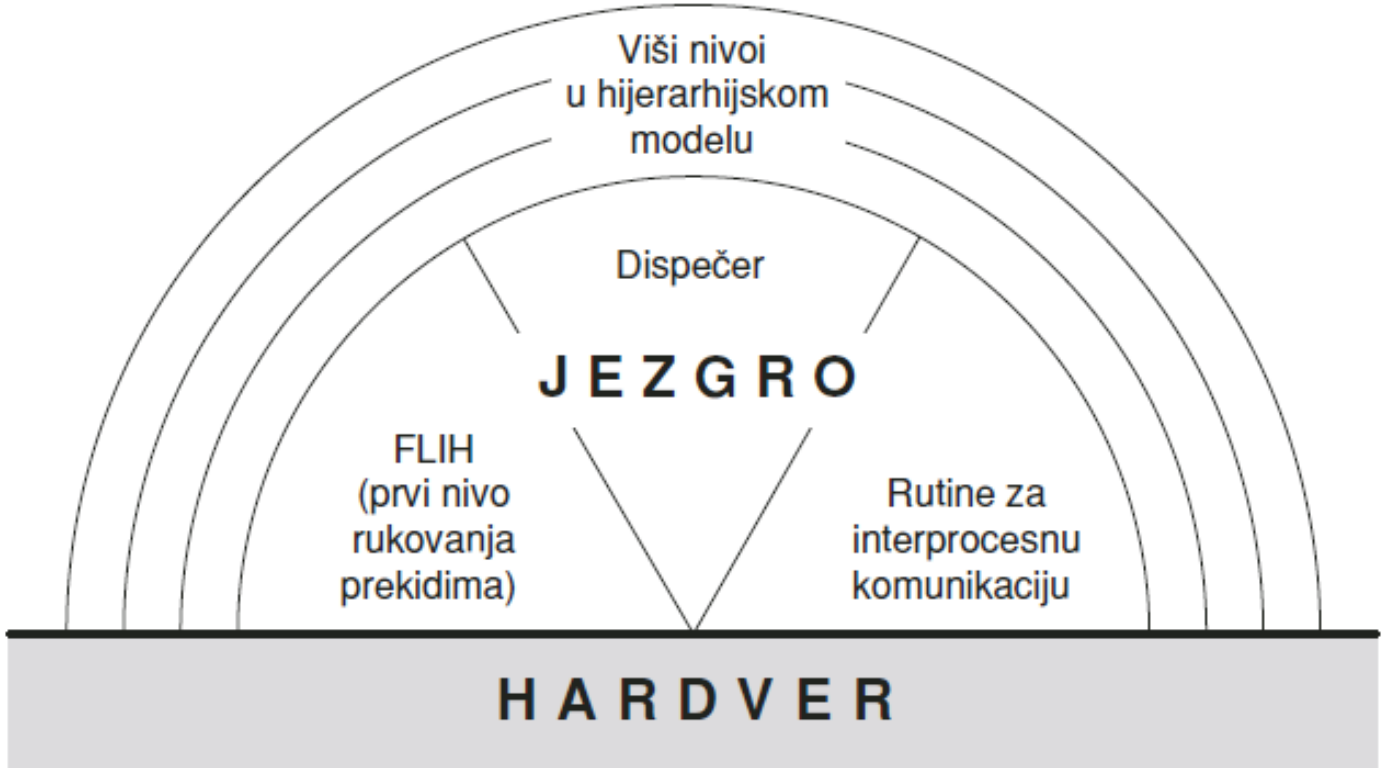
- Da bi kernel ostvario svoju osnovnu funkciju, neophodno je da na nivou hardvera postoje određene komponente koje omogućavaju nadogradnju hardvera kernelom.
  - **Mehanizam prekida**.
    - Mehanizam prekida obezbeđuje izvršenje prekidne (engl. *interrupt*) rutine, odnosno prebacivanje kontrole izvršavanja sa korisničkog na kontrolni program.
    - Najmanje što mehanizam prekida treba da uradi jeste da sačuva vrednost programskog brojača prekinutog korisničkog programa i pokrene kontrolni program.
    - Kontrolni program dalje određuje izvor prekida i odgovara na određeni način.
  - **Zaštitni mehanizam adresiranja memorije**.
    - Sprečava pogrešno adresiranje, odnosno mogućnost da jedan proces upiše svoje prateće podatke u deo memorije koji je dodeljen drugom procesu.
    - Ovaj mehanizam automatski čuva integritet procesa i podataka koji se nalaze u operativnoj memoriji.

Kernel

- **Privilegovani instrukcijski set**.
  - Privilegovani set su instrukcije koje su dostupne OSu, ali ne i korisničkim programima.
  - Ove instrukcije omogućavaju OSu da dodeli procesor drugom procesu, pristupi zaštićenim registrima u memoriji, izvrši ulazno-izlaznu operaciju i slično.
  - Prilikom izvršavanja privilegovanih instrukcija OS se nalazi u kontrolnom režimu (engl. *supervisory mode*).
  - Korisnički program ne može da izvrši privilegovanu instrukciju direktno.
  - Korisnički program pomoću sistemskog poziva zahteva od OSa da izvrši privilegovanu instrukciju, nakon čega OS prelazi u kontrolni režim i izvršava tu instrukciju.
- **Real-time časovnik**.
  - Pomoću real-time časovnika vrši se kontrola i evidencija potrošnje resursa računara za sve pojedinačne procese.
  - Ovaj mehanizam se može koristiti za raspoređivanje i zakazivanje izvršenja raznih poslova.

Kernel

- **Delovi kernela**.
- Jezgro *paper-model* OSa (teoretski model „na papiru“) može se podeliti na tri osnovne celine: prvi nivo obrade prekida, dispečer sistema i rutine za ostvarivanje interprocesne komunikacije.

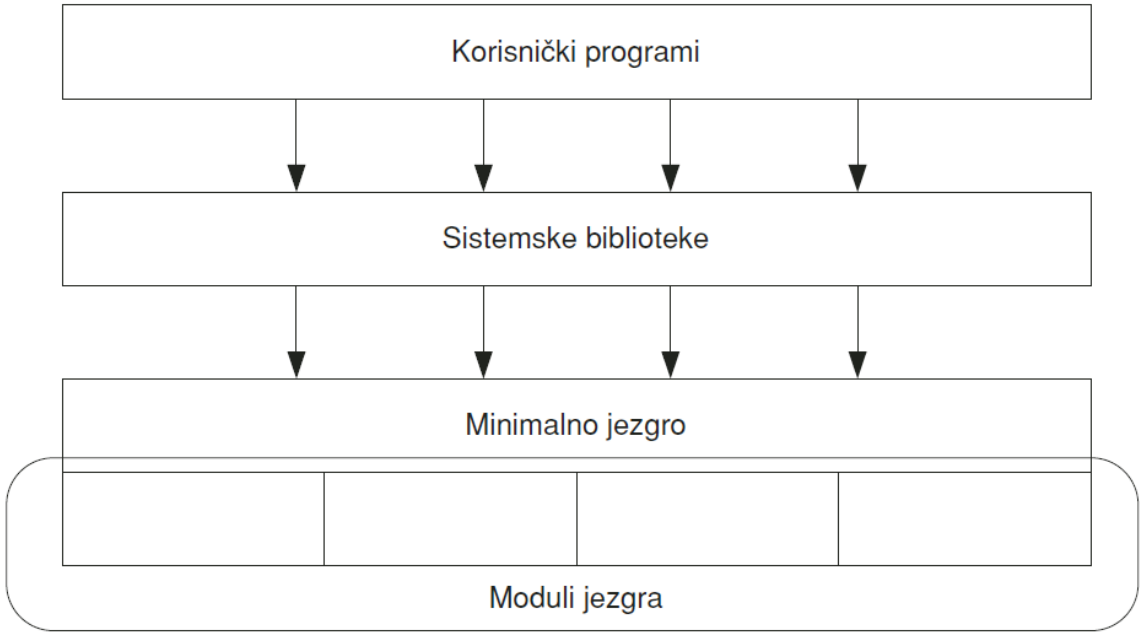


Kernel

- **Prvi nivo obrade prekida** (engl. *First Level Interrupt Handler*, FLIH).
  - Rutine za određivanje izvora prekida i iniciranje servisa (opsluživanje nekih vrsta prekida).
  - FLIH odgovara na spoljašnje prekide i sistemske pozive.
  - Posle izvršavanja koda koji čini ovaj deo OSa, prekid se smatra opsluženim, a dispečer dalje odlučuje kom će procesu dalje predati procesor na korišćenje.
- **Dispečer sistema** (planer poslova niskog nivoa, engl. *low-level scheduler*).
  - Deo kernela koji dodeljuje procesor procesima.
  - Procesor se uvek dodeljuje na osnovu nekog algoritma (na primer, *Shortest Job First*).
- **Rutine za ostvarivanje interprocesne komunikacije**.
  - Deo jezgra OSa koji obezbedjuje različite načine komunikacije među procesima, kao što su:
    - slanje poruka (engl. *send message*, *post message*),
    - semaforske tehnike,
    - imenovane cevi (engl. *named pipes*, mehanizam karakterističan za UNIX sisteme),
    - korišćenje deljive memorije.

Kernel

- **Koncept modularnog jezgra**.
- Moduli su delovi kernelskog koda koji mogu da se napune u memoriju ili izbace iz memorije nezavisno od ostatka kernela.
- Moduli implementiraju drajvere za hardver, novi sistem datoteka, mrežne protokole, itd.
- Omogućavaju mikro-kernel arhitekturu (realizaciju minimalne stabilne konfiguracije jezgra).



Pojam procesa

- **Proces** je program ili deo programa u stanju izvršavanja, zajedno sa svim resursima koji su potrebni za rad. Drugim rečima:
  - Program je **pasivan objekat**, odnosno datoteka na disku.
  - Kada se ta datoteka učita u memoriju, ona postaje proces, odnosno **aktivan objekat** koji ima svoje resurse, poput registara i memorije.
- To znači sledeće:
  - Tri korisnika koji obavljaju neku aktivnost biće predstavljena sa tri razna procesa.
  - Jedan program, koji je sam sebe razdelio na dva dela radi istovremenog (simultanog) izvođenja, biti predstavljen sa dva razna procesa.
- Sam operativni sistem je takođe sastavljen od niza procesa.
- Svaki proces ima tri fundamentalne memorijska dela, odnosno sekcije:
  - **programsku sekciju** koja se ne menja i koja sadrži programski kod,
  - **stek sekciju** (engl. *stack section*) koja sadrži privremene podatke (parametre za procedure, povratne adrese, itd ...),
  - **sekciju podataka** (engl. *data section*).

Pojam procesa

- **Kontrolni blok procesa** (engl. *Process Control Block*, PCB) je memorijska struktura koju generiše OS.
- Zahvaljujući PCB izvršenje programa se može prekidati i nastavljati više puta.
- PCB sadrži osnovne informacije o procesu koje OS koristi za upravljanje tim procesom:
  - ime ili jedinstveni identifikator procesa (PID),
  - kontekst (okolina) procesa,
  - prioritet procesa,
  - informacije vezane za memoriju procesa,
  - lista otvorenih datoteka,
  - status zauzetih ulazno-izlaznih resursa,
  - trenutno stanje procesa.
- **Kontekst procesa** čine podaci koji se čuvaju prilikom oduzimanja procesora, a koje generiše sam hardver: programski brojač, vrednosti registara i pokazivači na deo memorije koji proces koristi.
- Deo PCB u kome se čuva kontekst se još naziva i **hardverski kontrolni blok** procesa.
- PCB se u realnim OS nikad ne nalazi u istom memorijskom području kao i proces.

Dijagram stanja procesa

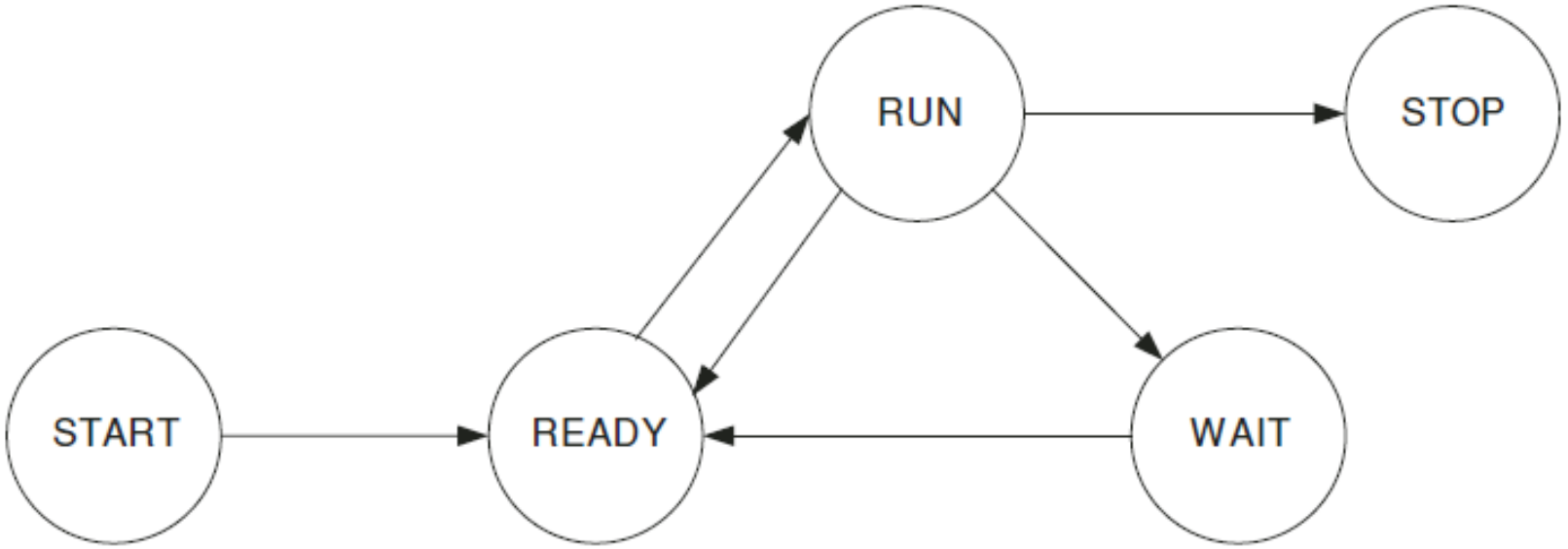
- **Stanje procesa** (engl. *process state*) opisuje ono što se u datom trenutku događa sa procesom.
- Broj stanja u kojima se proces može naći zavisi od konačnog automata koji je korišćen za opis.
- U ovom slučaju, radi jednostavnosti izlaganja, korišćen je konačni automat sa pet stanja:
  - Stanje START, tj. **trenutak formiranja procesa**.
  - **Stanje izvršavanja** (RUN, RUNNING).
    - Procesor izvršava istrukcije ovog procesa.
    - Broj procesa u RUN stanju u jednom trenutku određen je brojem procesora (jezgara).
  - **Stanje čekanja na procesor** (READY, RUNNABLE).
    - Proces je dobio sve potrebne resurse osim pocesora, spreman je za rad i čeka da mu se dodeli procesor.
  - **Stanje čekanja na resurs** (WAIT, UNRUNNABLE).
    - Proces čeka na neki događaj (na primer, da drugi proces završi sa štampanjem) jer su mu za dalje izvršavanje potrebni neki resursi koji trenutno nisu na raspolaganju.
  - **Kraj izvršenja procesa** (STOP).
    - Proces u stanju STOP oslobađa sve resurse koje je zauzeo.

Dijagram stanja procesa

- Proces se u svakom trenutku mora naći u jednom od ovih stanja a tranzicije iz jednog u drugo stanje se vrše zavisno od događaja vezanih za proces, resurse koje proces koristi itd.
- Provođenje procesa iz jednog stanja u drugo (engl. *state transition*) obavlja OS.
- Prilikom upravljanja procesima, dispečer prati i ispituje stanja u **kontrolnim blokovima procesa**.
- Na osnovu toga, procesi se mogu dodati u **red za čekanje na procesor**.
  - Dodavanje u red se svodi na zamenu podataka u kontrolnim blokovima i pokazivača pomoću kojih se formiraju redovi.
- Svaki proces počinje u stanju START, koje predstavlja neku vrstu pripreme.
  - Nakon toga se dovodi u stanje READY, odnosno stanje čekanja na dodelu procesa u procesorskom redu.
  - Ovo opisuje prvu tranziciju u dijagramu stanja.
- Posle nje, proces prolazi kroz dodatne tranzicije u konačnom automatu.

Dijagram stanja procesa

- Konačni automat sa pet stanja.

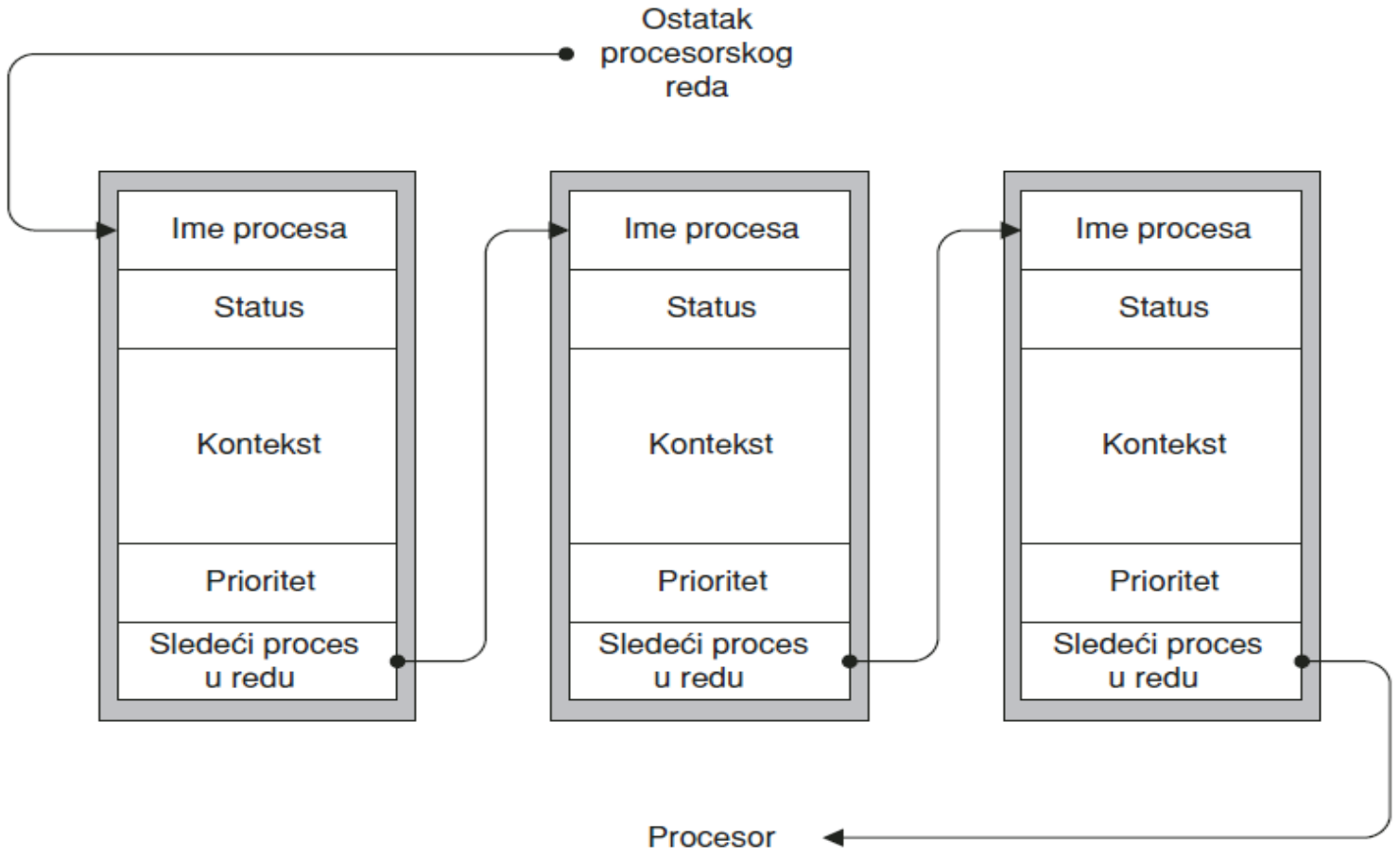


Dijagram stanja procesa

- **READY-RUN**. Dodela procesora procesu koji je došao na početak reda za dodelu procesora.
  - Proces prelazi iz stanja čekanja na procesor u stanje izvršavanja kada procesor prekine izvršenje tekućeg procesa, tako da može da otpočne izvršavanje novog procesa.
- **RUN-READY**. Oduzimanje procesora procesu nakon isticanja kvantuma u kojem mu je dodeljen.
  - Ova tranzicija je moguća samo u višeprocesnim operativnim sistemima.
  - Ukoliko je OS *pre-emptive* procesor se može oduzeti i ukoliko naiđe proces višeg prioriteta.
- **RUN-WAIT**. Oduzimanje procesora ako je neki resurs potreban za izvršenje procesa zauzet.
  - Ova tranzicija je moguća i u jednoprocesnim i u višeprocesnim operativnim sistemima.
  - Ova tranzicija se može desiti u slučaju da proces čeka rezultat operacije koje obavlja drugi proces što je tipično za odnos proizvođač-potrošač.
- **WAIT-READY**. Proces se vraća na kraj procesorskog reda nakon oslobađanja resursa koji je neophodan za rad procesa.
  - Tranzicija WAIT-RUN nije moguća u višeprocesnim operativnim sistemima!
- **RUN-STOP**. Proces završava sa radom (prirodno ili nasilno).

Dijagram stanja procesa

- Svi procesi u stanju READY mogu se povezati u procesorski red koristeći strukturu liste.
- Dispečer koristi procesorski red prilikom upravljanja procesima.

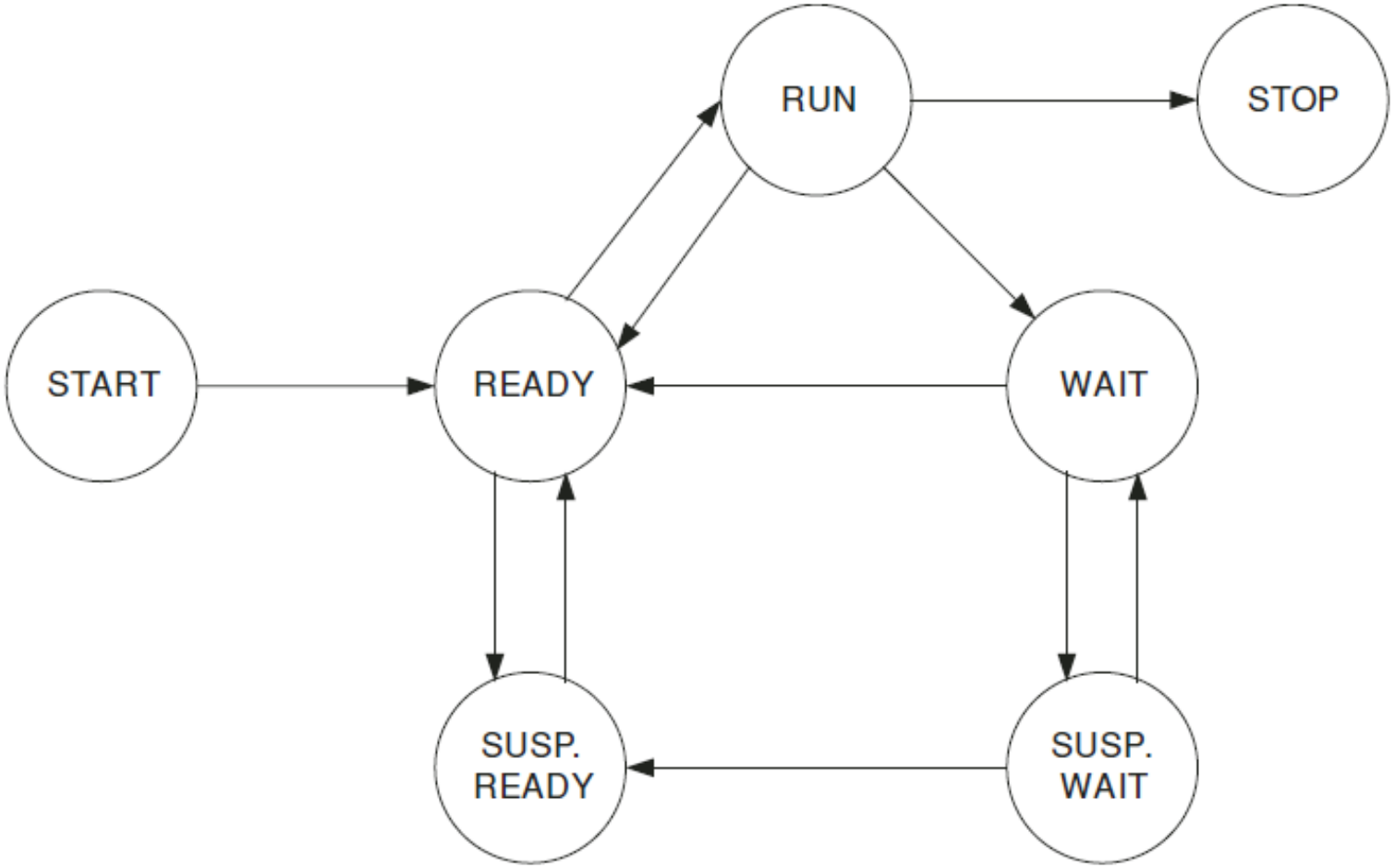


Dijagram stanja procesa

- **Prošireni dijagram stanja procesa**.
- Na nekim OS uvodi se mogućnost privremenog prekida izvršenja procesa.
- Proces čije je izvršenje privremeno prekinuto je **suspendovan** i ne takmiči se za resurse.
- Ovaj model automata karakterističan je za UNIX operativne sisteme.
  - Korisnik koji inicira proces ima pravo da ga supenduje.
  - Korisnik može da odmrzne proces, tj. da nastavi izvršenje suspendovanog procesa.
  - Sam OS može po potrebi da suspenduje određeni broj procesa i tako spreči pojavljivanje zastoja i spreči efekat zasićenja usled prevelike količine keširanih podataka.
  - Suspendovanje procesa se može javiti kao posledica upotrebe virtuelne memorije, odnosno *swap* prostora.
- Formiraju se dva dodatna stanja:
  - **SUSPENDED-READY**. Proces je suspendovan u stanju čekanja na procesor.
  - **SUSPENDED-WAIT**. Proces je suspendovan u stanju čekanja na resurs.

Dijagram stanja procesa

- Konačni automat sa sedam stanja.



Dijagram stanja procesa

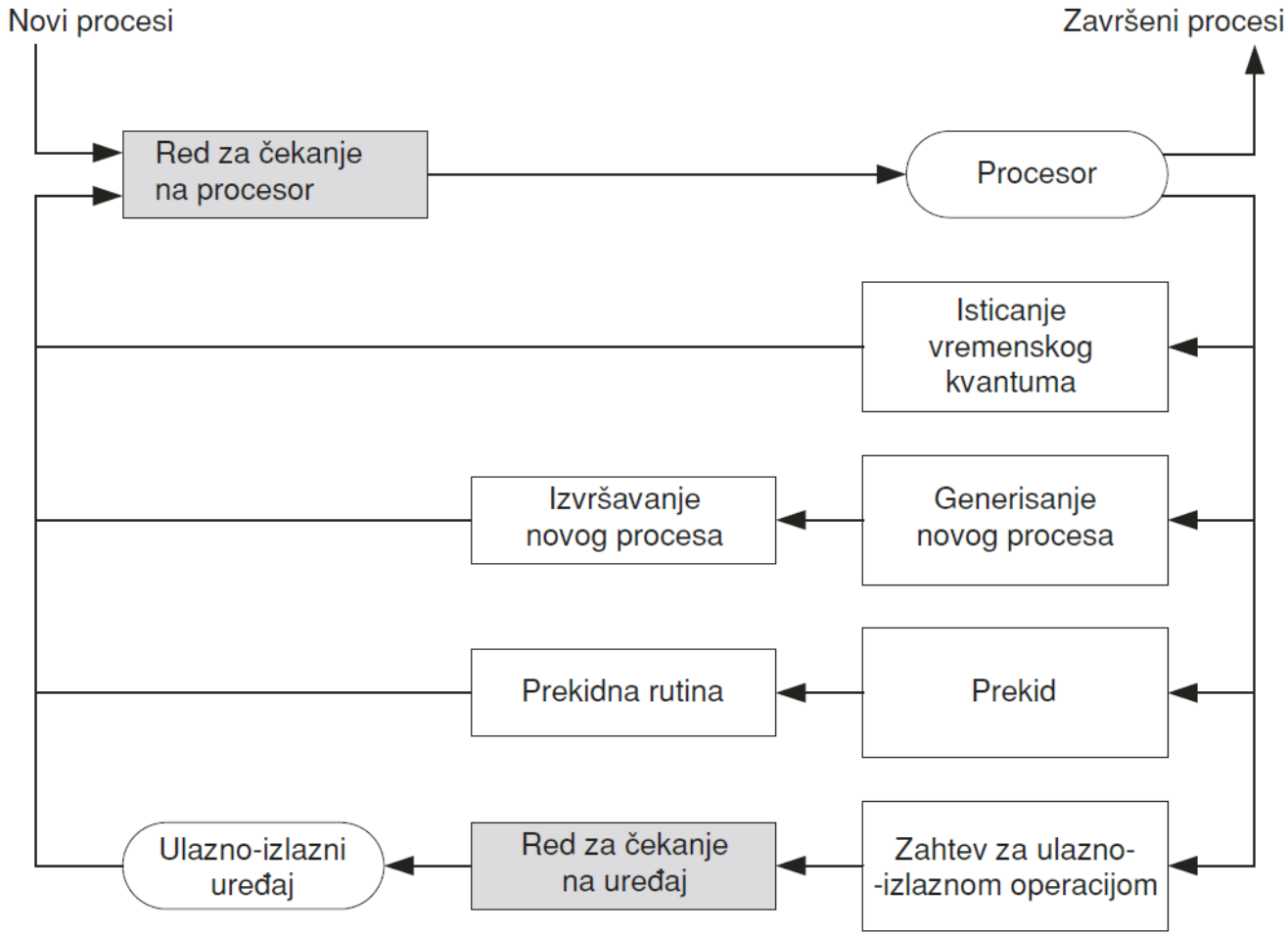
- Nove tranzicije u proširenom dijagramu stanja:
- **READY-SUSPENDED READY**. Dovođenje procesa iz stanja READY u suspendovano stanje.
  - Proces prolazi kroz avu tranziciju ako:
    - na sistemu postoji previše procesa u stanju READY,
    - treba izbeći zastoj (engl. *deadlock*)
    - korisnik eksplicitno suspenduje proces.
- **WAIT-SUSPENDED WAIT**. Dovođenje procesa u suspendovano stanje iz stanja WAIT.
- **SUSPENDED WAIT-SUSPENDED READY**. Resurs neophodan za dalje izvršenje procesa je slobodan, ali proces je i dalje suspendovan.
- **SUSPENDED READY-READY**. Proces je odmrznut i dovodi se na kraj procesorskog reda.
- **SUSPENDED WAIT-WAIT**. Proces je odmrznut, ali resurs neophodan za njegovo izvršenje nije oslobođen.

Raspoređivanje procesa

- **Redovi čekanja na procesor**.
  - Povezane liste formirane od PCB procesa sa definisanim redosledom izvršavanja procesa.
  - Svi procesi koju su spremni za rad i nalaze se u operativnoj memoriji čuvaju se u tom redu.
  - OS takođe uvodi i poseban red čekanja za svaki ulazno-izlazni uređaj (engl. *I/O queue*).
  - Svaki red čekanja na uređaj sadrži povezanu listu PCB procesa koji taj uređaj zahtevaju.
- Novi proces se inicijalno postavlja u red čeknja za spremne procese u kome čeka dodelu procesora, nakon čega napušta red i počinje da se izvršava.
- Proces koji se nalazi u stanju izvršavanja može:
  - ostati bez procesora kada mu istekne vremenski kvantum,
  - kreirati novi proces i čekati u blokiranom stanju da se novi proces izvrši,
  - ostati bez procesora kada se dogodi prekid,
  - postaviti I/O zahtev nakon čega se prebacuje u red čekanja na ulazno-izlazni uređaj.
- Proces se vraća u red čekanja na procesor dok se ne završi.
  - Nakon završetka proces oslobađa sve zauzete resurse.

Raspoređivanje procesa

- **Redovi čekanja na procesor**.



Raspoređivanje procesa

- Rutine OS za raspoređivanje procesa:
- **Planer poslova**.
  - U hijerarhijskom modelu se nalazi iznad kernela i obavlja sledeće funkcije:
    - deli poslove na procese,
    - na osnovu određenih algoritama dodeljuje prioritete procesima,
    - dovodi procese u red čekanja na procesor.
  - Poziva se kada se pojave novi procesi ili kada jedan ili više procesa završi aktivnosti.
- **Dispečer**.
  - Dispečer dodeljuje procesor procesima koji se nalaze u procesorskom redu:
    - tekućem procesu nakon isteka vremenskog kvantuma,
    - drugom procesu ukoliko je tekući proces prešao iz stanja RUN u stanje WAIT ili READY.
  - Poziva se veoma često i od njega se očekuje veoma brz odziv kako bi mogao adekvatno da odgovori na sukcesivne pozive.
  - Vreme koje je potrebno dispečeru da zaustavi jedan proces i dodeli procesor drugom je poznato kao kašnjenje dispečera (engl. *dispatch latency*).

Raspoređivanje procesa

- Izvršavanje jednog procesa se sastoji iz:
  - jednog ili više ciklusa izvršavanja na procesoru (engl. *CPU burst*),
  - jednog ili više ciklusa čekanja na ulazno-izlazne operacije (engl. *I/O burst*).
- Na osnovu resursa koje dominantno koriste postoje:
  - Procesi koje dominantno koriste procesor (engl. *CPU bound*)
    - generišu vrlo malo U/I zahteva,
    - najveći deo svog vremena troše na upotrebu procesora.
  - Procesi koji dominantno koriste ulazno-izlazne operacije (engl. *I/O bound*):
    - najveći deo vremena izvršavanja otpada na ulazno-izlazne cikluse,
    - na upotrebu procesora i memorije se troši relativno malo vremena,
    - relativno brzo će se blokirati čekajuči na U/I operacije (ulazak u WAIT stanje).
- Planer poslova mora da napravi dobru mešavinu I/O i CPU internzivnih procesa.
  - Ako većina procesa koju planer odabere intenzivno koristi U/I uređaje, procesor će biti slabo iskorišćen.

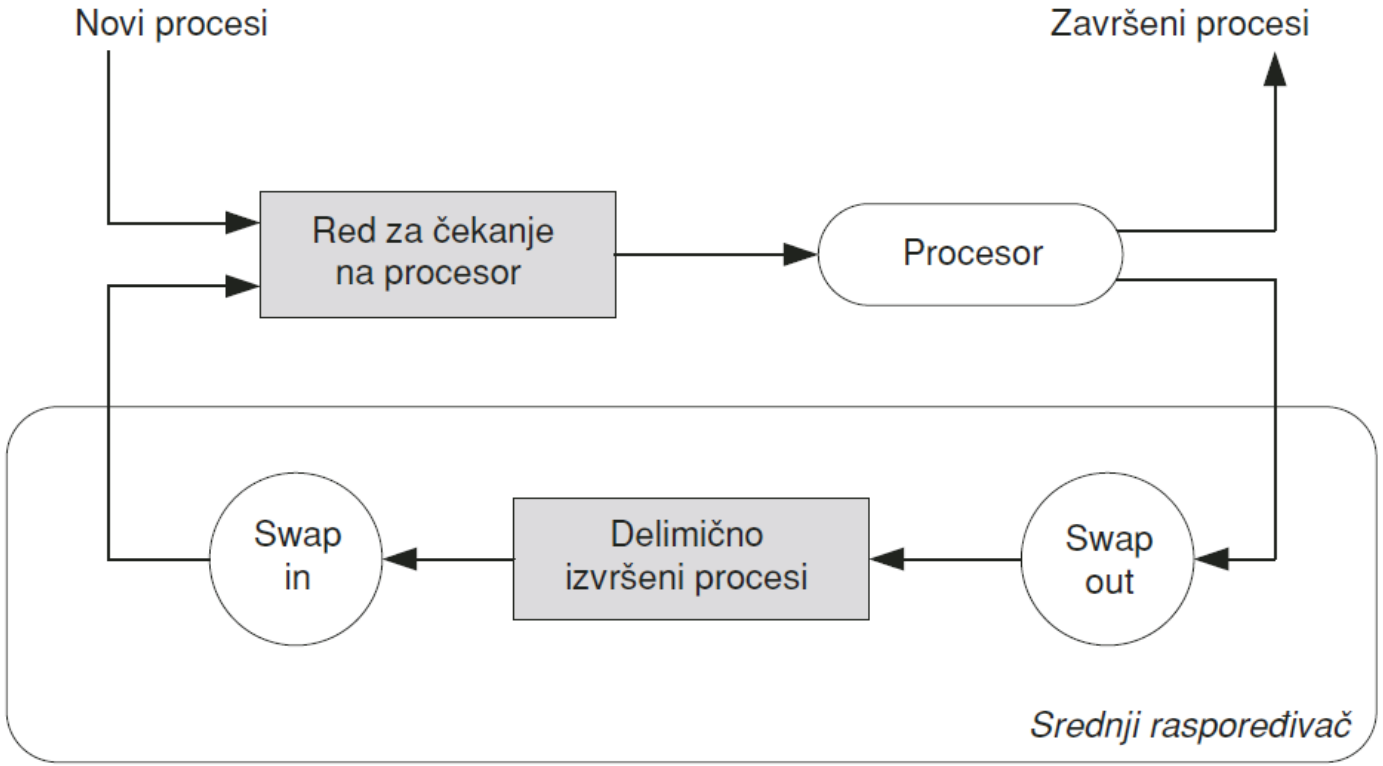
Raspoređivanje procesa

- **Zamena konteksta procesa** (engl. *context switch*).
- Kontekst procesa su podaci koji se čuvaju prilikom oduzimanja procesora, a koji omogućuju nastavak izvršenja procesa.
- Prilikom dodele procesora drugom procesu dispečer:
  - pamti stanje procesa koji se prekida, kao bi se proces mogao kasnije nastaviti i
  - puni memoriju stanjem novog procesa kome se dodeljuje procesor.
- Navedene operacije su poznatije pod nazivom zamena konteksta procesa.
  - Prilikom zamene konteksta poslednje stanje tekućeg procesa se čuva u njegovom PCB.
  - Poslednje stanje procesa koji će se dalje izvršavati rekonstruiše se iz odgovarajućeg PCB.
  - Nakon toga se procesu predaje kontrola za nastavak izvršenja.
- Zamena konteksta je premašenje sistema ali je neophodna za multiprogramiranje.
  - Premašenje zavisi od performansi procesora i memorije, broja registara koji se moraju sačuvati, optimalnih asemblerskih instrukcija, korišćenja raznih memorijskih tehnika itd.

Raspoređivanje procesa

- **Srednji nivo raspoređivanja**.
- Srednji raspoređivač (engl. *intermediate scheduler*) je posledica upotrebe *swap* tehnike, odnosno virtulene memorije kod interaktivnih sistema.
- Suština je u sledećem:
- Svaki novokreirani proces odlazi u red čekanja a dispečer odlučuje kome će da dodeli procesor.
  - Neki procesi mogu biti suspendovani, odnosno privremeno prekinuti i upisani na disk (engl. *swap space*), čime se oslobađa memorija za druge procese.
  - Premeštanjem sa diska u memoriju, suspendovani procesi se vraćaju u red čekanja na procesor.
- Srednji raspoređivač obavlja:
  - suspenzije procesa (engl. *swap-out*),
  - povratka procesa u stanje spremnosti (engl. *swap-in*),
  - izbor procesa za obe funkcije.

- **Srednji nivo raspoređivanja**.

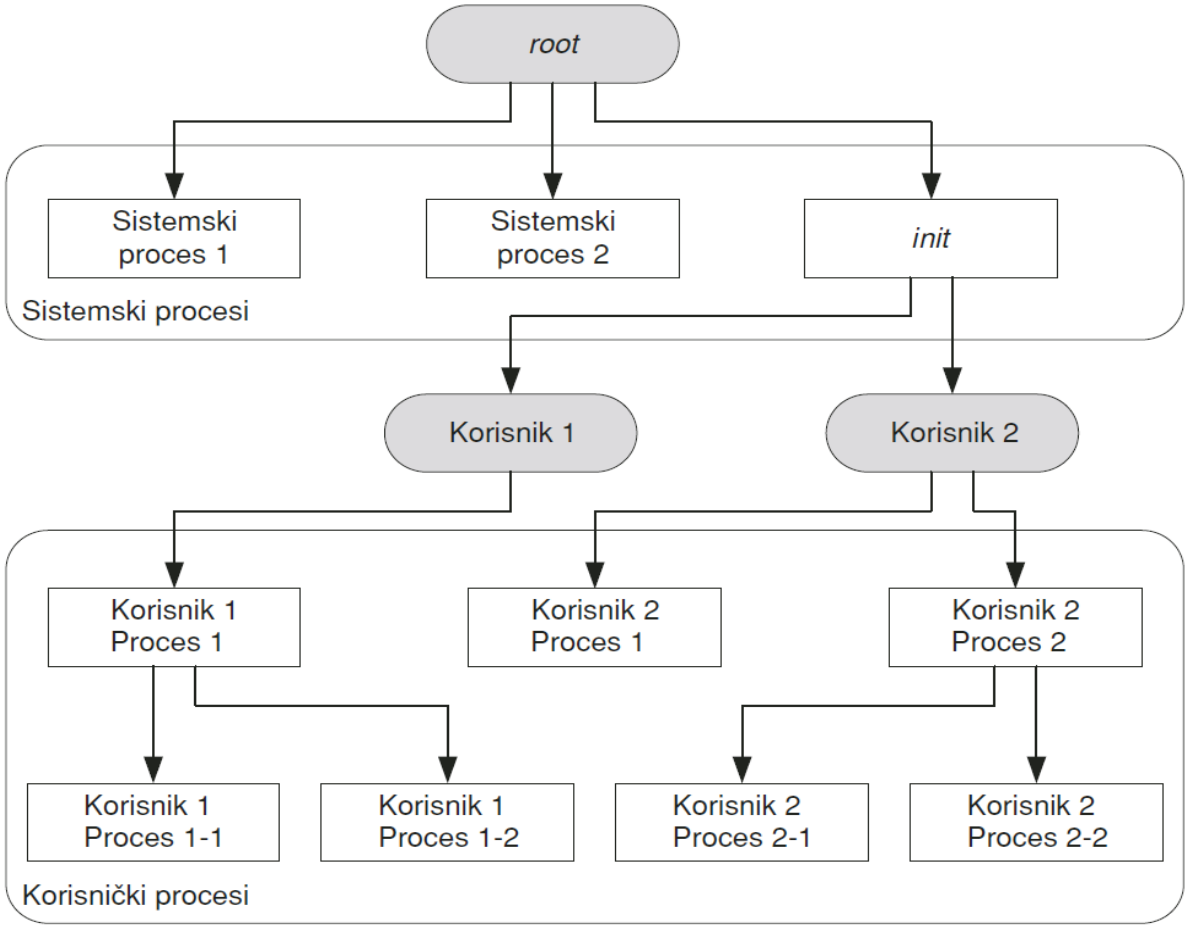


Operacije nad procesima

- Ranije opisani koncept jezgra uz korišćenje kontrolnih blokova procesa dozvoljava OS da nad procesima izvrši sledeće operacije:
  - kreiranje novog procesa – kreiranje kontrolnog bloka i ažuriranje procesne liste,
  - kreiranje veza proces-proces roditelj,
  - uništenje procesa brisanjem kontrolnog bloka iz procesne liste,
  - promenu stanja procesa – obavljanje tranzicija u dijagramu stanja (npr. dodela procesora),
  - promenu prioriteta procesa.
- **Kreiranje procesa**.
  - Proces može odgovarajućim sistemskim pozivom da kreira nove procese.
  - Proces koji kreira nove procese je roditeljski proces (engl. *parent process*)
  - Proces koji je roditelj kreirao naziva dete proces (engl. *child process*).
  - Proces roditelj može da nastavi da se izvršava nezavisno i konkurentno sa detetom ili da se blokira i čeka dok proces dete ne završi svoje aktivnosti.
  - Svaki proces koji je dete nekog drugog procesa može dalje kreirati nove procese.
    - Na taj način se formira hijerarhijska procesna struktura (stablo aktivnih procesa).

Operacije nad procesima

- Hijerarhijska procesna struktura (stablo aktivnih procesa).



Operacije nad procesima

- **Kreiranje procesa** – primer (Linux OS).
  - Svaki proces dobija univerzalni identifikator **PID** (engl. *Process Identifier*).
  - Novi procesi se kreiraju pomoću sistemskog poziva `fork()`.
    - Oba procesa (roditelj i dete) nastavaljaju izvršenje nakon sistemskog poziva `fork()`.
    - Sistemski poziv `fork()` duplicira adresni prostor roditelja i dodeljuje ga detetu.
    - Proces dete dobija PID 0 kao povratnu vrednost funkcije `fork()`, i to se smešta u promenljivu `pid` u našem slučaju, dok je njegov PID drugi i zna ga proces roditelj.
    - Rezultat:
      - Adresni prostor oba procesa je napunjen istim programom (kodom).
      - Identifikatori procesa su različiti.
  - Dete proces zatim izvršava sistemski poziv `exec()`.
    - Ovaj poziv puni program u adresni prostor dodeljen detetu i izvršava ga.
  - Proces roditelj izvršava sistemski poziv `wait()`.
    - Time se blokira dok proces dete proces ne završi svoje aktivnosti i ne vrati status roditeljskom procesu.

Operacije nad procesima

- **Kreiranje procesa** – primer (Linux OS).

```
# include <stdio.h>
main (int argc, char *argv[]) {
  int pid;
  pid = fork ();
  // proces dete
  if (pid == 0) {
    execlp ("/bin/sh", "ls", NULL);
  }
  // proces roditelj
  else {
    wait (NULL);
    /* proces dete završio svoje aktivnosti */
    exit (0);
  }
}
```

- Primer odgovara slučaju interpretera `/bin/sh` koji izvršava komandu `ls`.
- Roditelj kreira novi proces pomoću `fork` sistemskog poziva.
- Proces dete čiji je PID 0 izvršava sistemski poziv `exec` koji će u dodeljni adresni prostor napuniti i izvšiti program `ls`.
- Proces roditelj čiji je PID različit od 0, čeka da se `ls` komanda završi.
- Tek kada proces dete završi aktivnosti, roditelj će izaći iz `wait` sistemkog poziva.

Rutine za interprocesnu komunikaciju

- Rutine za interprocesnu komunikaciju se moraju implementirati u kernelu kako bi bile dostupne svim procesima i kako bi imale direktan pristup dispečeru.
- Interprocesna komunikacija se može ostvariti na više načina kao što su:
  - semaforske tehnike,
  - slanje poruka,
  - korišćenje deljive memorije (na primer, bafera),
  - korišćenje imenovanih cevi.

Rutine za interprocesnu komunikaciju

- Proces je **nezavistan** ukoliko nema direktnog uticaja na izvršenje drugih procesa i ukoliko na njegovo izvršenje direktno ne utiču drugi procesi.
  - Praktično, nezavisni procesi ne dele nikakve podatke sa drugim procesima.
- **Kooperativni procesi** su oni procesi koji jedni na druge međusobno utiču.
  - Praktično, to su procesi koji dele podatke ili bilo kakve resurse.
- **Konkurentno izvršavanje kooperatvnih procesa** zahteva mehanizam koji će dozvoliti procesima da međusobno komuniciraju između sebe i da sinhronizuju svoje akcije.
- Problem sinhronizacije se može objasniti na čuvenom problemu **proizvođač-potrošač**.
  - Proizvođač je proces koji proizvodi informacije a potrošač je proces koji ih troši (koristi).
  - Ta dva procesa mogu da rade konkurentno ukoliko postoji deljivi bafer koji će puniti proizvođač, a prazniti potrošač.
  - Pravila sinhronizacije:
    - Potrošač ne može uzeti ništa iz bafera ako proizvočač to prethodno nije stavio u bafer.
    - Ukoliko je bafer pun, proizvođač ne može ubaciti informaciju dok potrošač najpre ne uzme nešto iz bafera.

Rutine za interprocesnu komunikaciju

- **Deljivi bafer** se može realizovan kao cirkularna struktura koja se sastoji od *N* elemenata [0, *N*-1].
  - Pokazivači `in` i `out` određuju prvo slobodno i prvo puno mesto u baferu.
  - Bafer je prazan kada je `in=out`.
  - Bafer je pun kada `(in+1) mod N = out`.

```
# define N 10
typedef struct {
   /* Promenljive koje čine jedan element bafera, odnosno jednu informaciju */
} item;
item buffer[N]; // Bafer veličine N
int in = 0; // Prvo slobodno mesto
int out = 0; // Prvo zauzeto mesto
```

Operacije nad procesima

- Proizvođač:

```
item next_produced;
while (1) {
  /* Proizvođač proizvodi informaciju */
  while (((in+1)%N) == out);
  /* Sinhronizacija (bafer je pun).
     Proizvođač čeka da potrošač nešto
     uzme iz bafera */
   buffer[in] = next_produced;
   in = (in+1) % N;
}
```

- Potrošač:

```
item next_consumed;
while (1) {
  while (in == out);
  /* Sinhronizacija (bafer je prazan).
     Potrošač čeka se da proizvođač
     nešto stavi u bafer /*
   next_consumed = buffer[out];
   out = (out+1) % N;
   /* Potrošač konzumira informaciju */
}
```

- Ovo rešenje je korektno samo u slučaju da se u baferu mogu naći najviše *N*-1 informacija.
- Za slučaj bafera u kome se mogu naći svih *N* informacija mora se obezbediti dodatna sinhronizacija (biće objašnjeno u izlaganju o sinhronizaciji procesa).

Rutine za interprocesnu komunikaciju

- **Slanje poruka**.
- Slanjem poruka omogućava se sinhronizacija aktivnosti bez upotrebe deljive memorije.
- Pogodno je za distribuirana okruženja gde procesi egzistiraju na različitim računarima.
- Da bi procesi mogli da komuniciraju slanjem poruka, neophodno je u kernel implementirati sledeće dve operacije (primitive):
  - **slanje poruke** (engl. *send message*, *post message*) i
  - **prijem poruke** (engl. *receive message*).
- Postoje različite šeme za realizaciju ovih primitiva kao što su:
  - **Sinhrono ili blokirajuće** (engl. *blocking*) slanje i primanje poruka.
    - Proces koji šalje poruku se blokira dok drugi proces ili sanduče ne prime poruku.
    - Proces koji prima poruku se blokira dok ne dobije poruku.
  - **Asinhrono ili neblokirajuće** (engl. *nonblocking*) slanje i primanje poruka.
- Moguće su različite kombinacije ovih metoda.
  - Primer: šema u kojoj su i slanje i primanje blokirajući se zove *rendezvous*.

Rutine za interprocesnu komunikaciju

- **Direktna komunikacija**.
- Procesi moraju da imaju mogućnost imenovanja radi jednoznačne identifikacije.
- Procedure za slanje i primanje moraju da sadrže **ime procesa** sa kim žele konuniciraju.
- Adresiranje: **simetrično** ili **asimetrično**.
- Asimetrično: pošiljaoc navodi ime primaoca a primalac ne mora da navede ime pošiljaoca.

```
send (P, message)
receive (Q, message) // simetrično adresiranje
receive (id, message) // asimetrično adresiranje
```

- Pravila direktne komunikacije:
  - Link se uspostavlja automatski između para procesa koji žele da komuniciraju.
  - Jedan link se uspostavlja samo za dva procesa.
- Nedostatak: pri promeni imena procesa sve reference sa starim imenom moraju se pronaći i modifikovati u novo ime (a te situacije nisu uvek poželjne).

Rutine za interprocesnu komunikaciju

- **Indirektna komunikacija**.
- Poruke se šalju ili primaju preko poštanskih sandučića (engl. *mailbox*) ili portova.
- Poštansko sanduče je objekat sa jedinstvenim ID u kome OS ostavlja poruke drugih procesa.
  - Dva procesa mogu komunicirati samo ako dele sanduče.
  - Proces može komunicirati sa procesom preko više različitih sadučića.
- Primer: slanje poruka u poštansko sanduče A i primanje poruka iz poštanskog sandučeta A:

```
send (A, message)
receive (A, message)
```

- Pravila indirektne komunikacije:
  - Link se uspostavlja između para procesa koji dele sanduče.
  - Linku se mogu pridružiti više od dva procesa.
  - Više različitih linkova može postojati između svakog para procesa, gde svaki link odgovara jednom sandučetu.

Rutine za interprocesnu komunikaciju

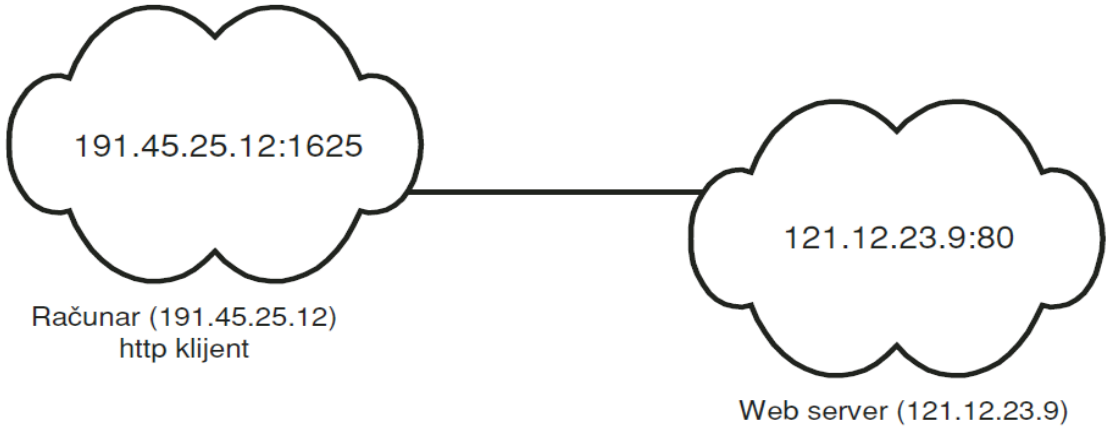
- **Indirektna komunikacija**.
- Gde nastaje problem?
  - Tri procesa (P1, P2 i P3) koji dele isto sanduče A.
  - Proces P1 šalje poruku u sanduče A.
  - Procesi P2 i P3 proveravaju da li je neka poruka smeštena u sanduče.
  - Pitanje: koji će proces preuzeti upućenu poruku?
- Ovaj problem se može rešiti na više načina:
  - Link preko jednog poštanskog sandučeta mogu da uspostave isključivo dva procesa.
  - Samo jedan proces može obaviti prijem u jednom trenutku vremena.
  - Adresiranje poruke (navođenje imena procesa primaoca).
- Sandučići mogu biti vlasništvo procesa ili OSa.
- Šta OS mora da obezbedi za indirektnu komunikaciju?
  - Mehanizme za kreiranje novog sandučeta za svaki par procesa koji želi da komunicira, slanje i primanje poruka kroz sanduče i brisanje sandučeta.

Rutine za interprocesnu komunikaciju

- **Semafori**.
- Semafor je celobrojna nenegativna promenljiva, čija vrednost štiti neki resurs (određuje da li je slobodan ili zauzet) i omogućava komunikaciju između procesa.
- Semafor ima svoju početnu vrednost *s* a nad njim se mogu izvršiti dve operacije: `signal` i `wait`.
  - Operacija `wait(s)` umanjuje vrednost semafora samo ako je *s*>0.
  - Ako je *s*=0 proces koji izvršava ovu operaciju ulazi u stanje čekanja na resurs.
  - Kada vrednost *s* postane pozitivna (drugi proces je oslobodio resurs) operacija `wait` će:
    - umanjiti vrednost semafora (*s*=*s*-1),
    - uvesti proces u kritičnu sekciju (deo koda u kome proces koristi resurs).
  - Kada proces želi da prekine korišćenje resursa on:
    - izvršava operaciju `signal(s)` čime uvećava *s* (*s*=*s*+1),
    - napušta kritičnu sekciju.
- Operacije `signal` i `wait` su nedeljive – ne mogu podeliti na više ciklusa.
  - Dva procesa ne mogu istovremeno izvršavati ove operacije nad istim semaforom.

Rutine za interprocesnu komunikaciju

- **Mehanizam utičnica** (engl. *socket*).
- Par procesa koji želi da komunicira preko mreže formira par utičnica (po jednu za svaki proces).
- Utičnica se definiše pomoću dva parametra **IP:port**.
- Koristi se klijent-server arhitektura: server osluškuje da li je stigao zathev klijenata na tom portu.
- Primer: računar želi da uspostavi Web sesiju sa serverom.
  - Klijent proces na strani računara kreira svoju utičnicu na portu 1625 i preko tog priključka komunicira sa tačno definisanom Web utičnicom (port 80) na strani Web servera.

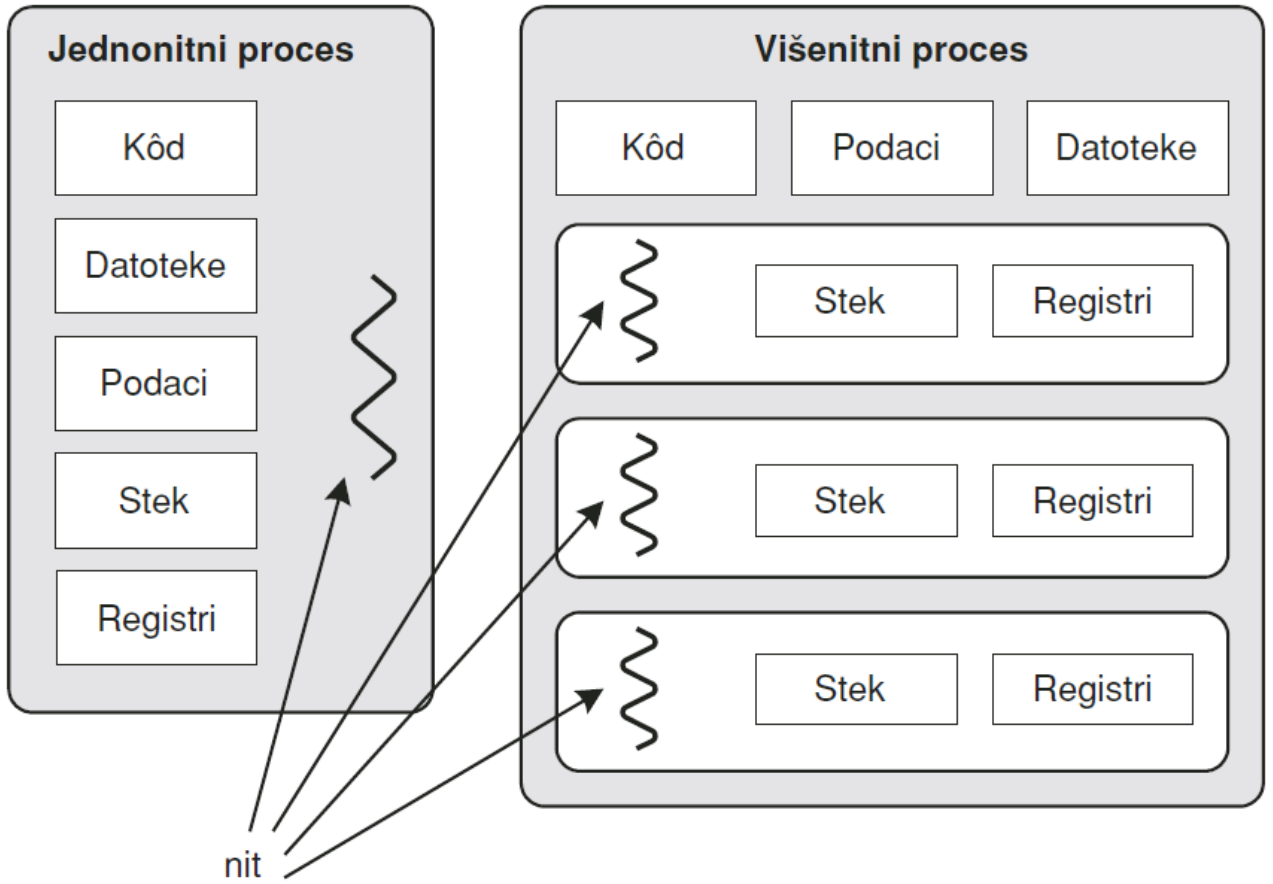


Laki i teški procesi

- Niti (engl. *threads*), odnosno laki procesi (engl. *lightweight process*, LWP) su osnovne celine za izvršavanje koda pod savremenim OS.
- Niti predstavljaju programsku celinu koja treba da obavi jedan posao.
- Niti (jedna ili više) pripadaju jednom tradicionalnom odnosno teškom procesu.
- Niti kao laki procesi i delovi jednog istog procesa imaju:
  - svoje unikatne resurse:
    - poseban identifikator niti (engl. *thread ID*),
    - posebnu vrednost programskog brojača, drugih registara procesora i stek,
    - podatke specifične za nit (ukoliko su potrebni);
  - zajedničke resurse sa ostalim nitima istog procesa:
    - kod sekcija, sekcija podataka, otvorene datoteke, signali.
- Proces sa svojim nitima može obavljati više poslova simultano
- Prebacivanje konteksta sa jedne niti na drugu daleko brže i efikasnije od prebacivanje konteksta između teških procesa.
  - Čuvaju se i rekontruišu samo registri procesora i stek.

Laki i teški procesi

- Jednonitni proces (engl. *singlethreaded*) i višenitni proces (engl. *multithreaded*).

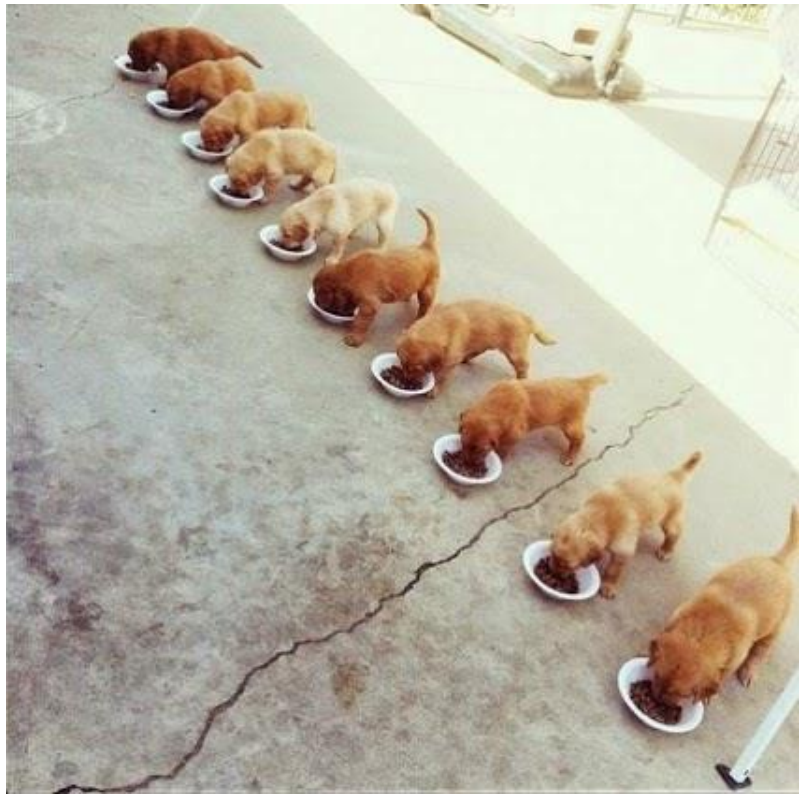


Laki i teški procesi

- **Rad sa nitima**.
- U višenitnoj arhitekturi menja se semantika sistemskih poziva za kreiranje i rad sa procesima.
- Mora postojati poziv za kreiranje teškog procesa kao i poziv za kreiranje jedne jedine niti.
- Prekidanje niti je aktivnost koja prekida izvršavnje niti pre njenog prirodnog završetka.
- Nit koja će biti prekinuta obično se naziva **ciljna nit** (engl. *target thread*).
- Prekidanje ciljne niti može se dogoditi po dva scenarija:
  - **Asinhroni prekid**. Aktivnost niti se odmah prekida bez obzira na stanje u kome se nalazi.
  - **Odloženi prekid** (teži za implementaciju). Ciljna nit proverava periodično da li treba da prekine aktivnost i ako treba to će učiniti u povoljnom trenutku kad obavi neku celinu.
- Signal upućen nekom višenitnom procesu zavisno od situacije može se proslediti:
  - samo onoj niti kojoj je namenjen,
  - svim nitima procesa,
  - samo nekim nitima procesa,
  - samo onoj niti procesa čija je namena da prima i obrađuje sve signale za proces.

Laki i teški procesi

- **Niti**: teorija (slika levo), vrlo čest slučaj u praksi (slika desno).



Literatura

1. B. Đorđević, D. Pleskonjić, N. Maček (2005): Operativni sistemi: teorija, praksa i rešeni zadaci. Mikro knjiga, Beograd.
2. R. Popović, I. Branović, M. Šarac (2011): Operativni sistemi. Univerzitet Singidunum, Beograd.

Hvala na pažnji

Pitanja su dobrodošla.