Operativni sistemi
- Procesi i niti -

Veljko Stanković

Uvod

- Proces predstavlja centralni koncept OS-a: abstrakciju programa koji se izvršava.
- Proces predstavlja osnovnu radnu jedinicu sistema.
- Svaki sistem se sastoji iz kolekcije procesa.
  - OS procesi izvršavaju sistemske programe
  - Korisnički procesi izvršavaju korisničke programe
- OS je zadužen za:
  - kreiranje i brisanje korsničkih i sistemskih procesa
  - dodelu resursa procesima
  - sinhronizaciju procesa
  - komunikaciju između procesa
  - razrešenje konflikata između procesa

Uvod

- Rani računari su omogućavali izvršavanje jednog programa koji je imao potpunu kontrolu nad resursima sistema.
- Evolucija ka sistemima kod kojih se više programa izvršava konkurnetno zahtevala je čvršću kontrolu nad njihovim izvršavanjem.

Operacije nad procesima
- Kreiranje procesa -

1010011
1110100
1100001
1010110

— Do kreiranja procesa mogu dovesti sledeća četiri događaja:
- Inicijalizacija sistema.
- Kreiranje novog procesa od strane procesa koji se već izvršava.
  - ☑ Proces roditelj (Bazni proces)
  - ☑ Proces naslednik
- Korisnički zahtev za kreiranjem procesa.
- Inicijalizacija izvršavanja paketa programa (batch).

— Podproces može nakon kreiranja:
- da dobije posebne resurse od OS-a za izvršavanje,
- da se izvršava primenom dela resursa baznog procesa,
  - ☑ Na ovaj način se sprečava da jedan proces kreira preveliki broj podprocesa i time preoptereti sistem.

Operacije nad procesima
- Kreiranje procesa -

1010011
1110100
1100001
1010110

— Prilikom kreiranja procesa:
- Bazni proces se može izvršavati konkurentno sa svojim podprocesima,
- Bazni proces čeka da svi njegovi podprocesi završe sa izvršavanjem na sistemu.

— Adresni prostor
- Podproces koristi kopiju adresnog prostora baznog procesa
  - ☑ UNIX sistem, naredba **fork**.
  - ☑ Lakša komunikacija baznog procesa i podprocesa.
- Podproces se posebno učitava.

Operacije nad procesima
- Prekid izvršenja procesa -

1010011
1110100
1100001
1010110

— Proces završava sa izvršavanjam iz jednog od sledećih razloga:
- Regularni kraj izvršavanja.
- Prekid izvršavanja usled greške.
- Prekid izvršavanja usled fatalne greške.
  - ☑ Prekid se vrši od strane sistema, izvan procesa.
- Namernim prekidanjem izvršavanja od strane drugog procesa ili korisnika.

— Bazni proces može prekinuti izvršavanje podprocesa iz sledećih razloga:
- Podproces je prekoračio dodeljene resurse za izvršavanje.
- Izvršenje podprocesa nije više neophodno.
- Prekida se izvršavanje baznog procesa i OS ne dozvoljava izvršavanje podprocesa koje je on kreirao.

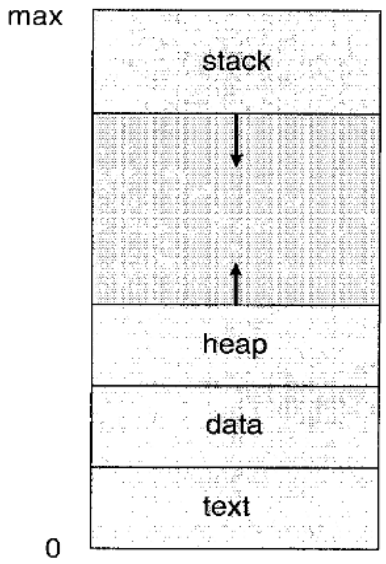
Proces

1010011
1110100
1100001
1010110

— Proces predstavlja program koji se izvršava.

— Proces predstavlja jedinicu dodele resursa.

- Proces je opisan:
  - ☑ Sadržajem programskog brojača
  - ☑ Sadržajem CPU registara
  - ☑ Sadržajem magacina
    - Gde su smeštene priveremeni podaci i promenljive
  - ☑ Globalnim promenljivama programa
- Program predstavlja pasivan entitet, sadržaj fajla na disku.
- Proces predstavlja aktivan entitet koji je opisan skupom resursa potrebnih za njegovo izvršenje.
- Iako dva procesa mogu biti rezultat izvršavanja istog programa, oni se posmtraju kao nezavisni entiteti.



Opis procesa

- Proces je na jedinstven način opisan svojom slikom (process image).
- Slika procesa se sastoji iz:
  - Koda programa koji proces izvršava.
  - Podataka nad kojima se izvršavaju instrukcije.
  - Process Control Block - PCB
- Polja PCB kontrolnog bloka
  - Identifikator
    - ☑ Broj koji na jedinstven način pridružen procesu.
  - Stanje procesa
    - ☑ Opisuje fazu izvršavanja procesa
  - Prioritet procesa
    - ☑ Prioritet u odnosu na druge procese pri dodeli CPU.
  - Programski brojač (PC)
    - ☑ Ukazuje na adresu sledeće instrukcije koja treba da se izvrši

Opis procesa

— Polja PCB kontrolnog bloka (nastavak)

- CPU registri
  - ☑ Sadržaj registara CPU se mora sačuvati zajedno sa sadržajem programskog brojača kako bi proces mogao da nastavi sa svojim izvršavanjem u slučaju prekida.
- Informacija o dodeli memorije
  - ☑ Informacije o tabeli straničenja, segmentnoj tabeli, vrednosti baze i granice.
- I/O status
  - ☑ Lista I/O uređaja dodeljenih procesu, lista otvorenih fajlova, itd.
- Podaci o vremenu izvršavanju na CPU i ukupnom vremenu izvršavanja, vremenskim ograničenjima, itd.

Kontrolni blok procesa (PCB)

- Tipične informacije u okviru PCB -

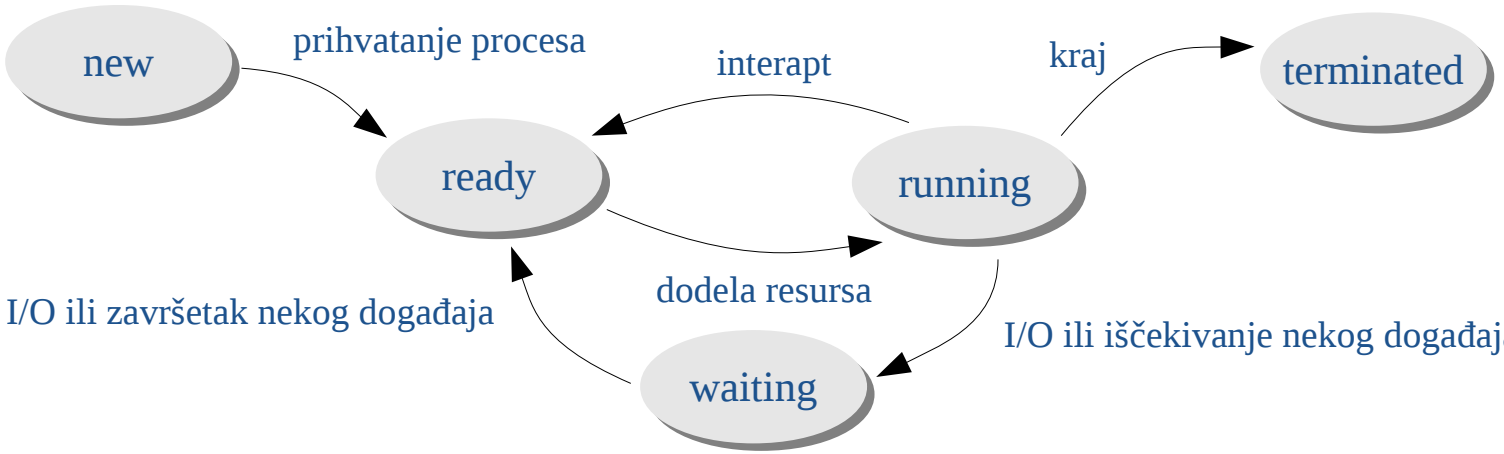
1010011
1110100
1100001
1010110

- Upravljanje procesima
  - Registri
  - Programski brojač (PC)
  - Programska statusna reč (PSW)
  - Pokazivač magacina
  - Prioritet
  - Stanje
  - Parametri za alokaciju CPU
  - ID procesa
  - Bazni proces
  - Grupa procesa
  - Signali
  - Vreme početka izvršavanja procesa
  - Vreme izvršavanja na CPU
  - Vreme izvršavanja podprocesa
- Upravljanje memorijom
  - Pokazivač tekstualnog segmenta
  - Pokazivač segmenta podataka
  - Pokazivač na segment magacina
- Fajl sistem
  - Root direktorijum
  - Radni direktorijum
  - Fajl deskriptori
  - Korisnički ID
  - ID grupe

Stanje procesa

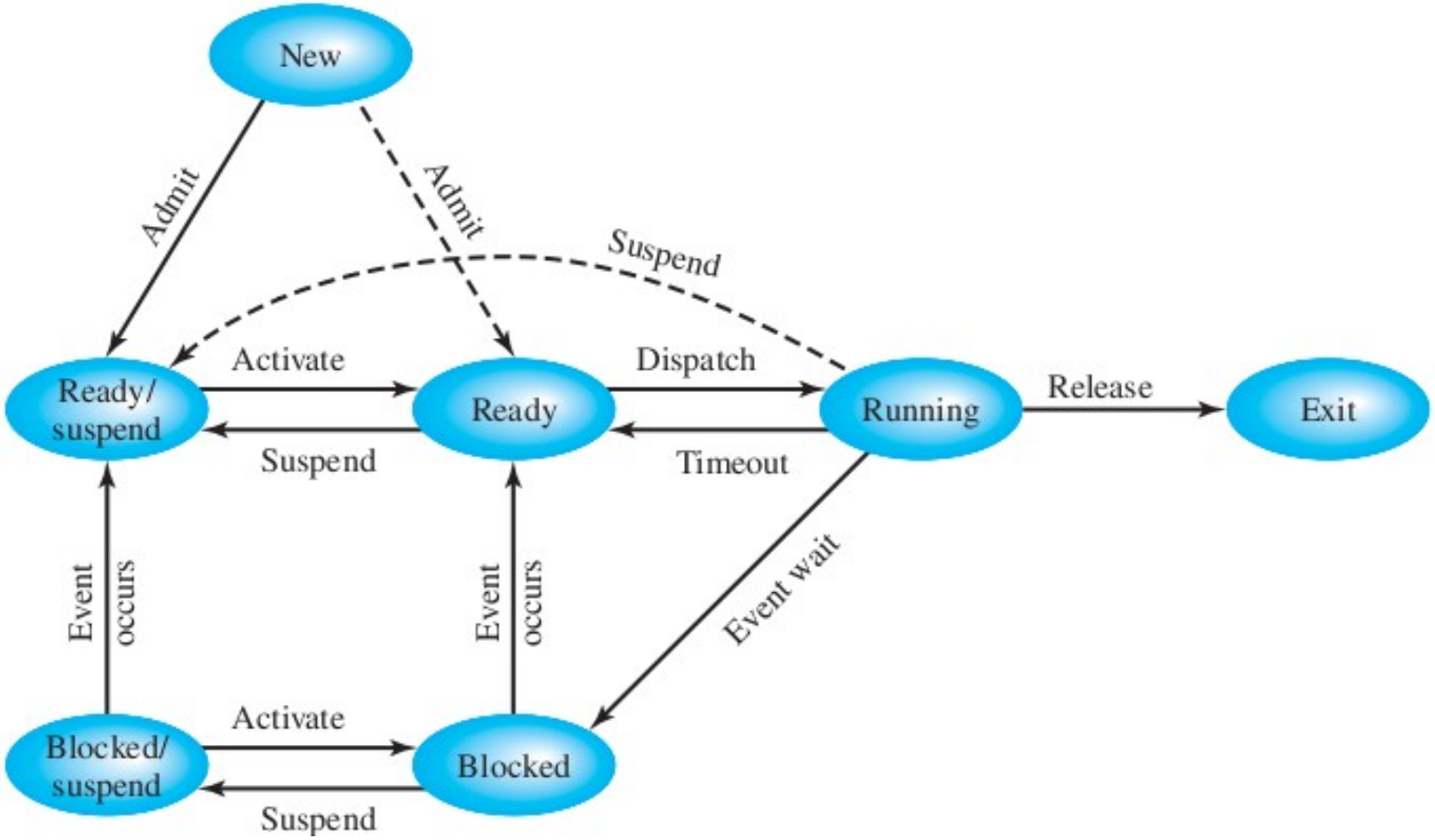
- Tokom svog izvršavanja proces prolazi kroz različita stanja
- Stanja procesa
  - **New** – Kreira se proces, proces započinje sa svojim izvršavanjem.
  - **Running** – Izvršavaju se instrukcije procesa
  - **Waiting** – Proces čeka neki događaj (I/O ili završetak nekog drugog procesa)
  - **Ready** – Proces je spreman za izvršavanje. Čeka da mu se dodele resursi sistema.
  - **Terminated** – Proces je završio sa svojim izvršavanjem.
- Samo jedan proces se može izvršavati na CPU.
- Više procesa mogu biti spremni na izvršenje ili da čekaju na izvršenje.

Stanje procesa



Stanje procesa

1010011
1110100
1100001
1010110

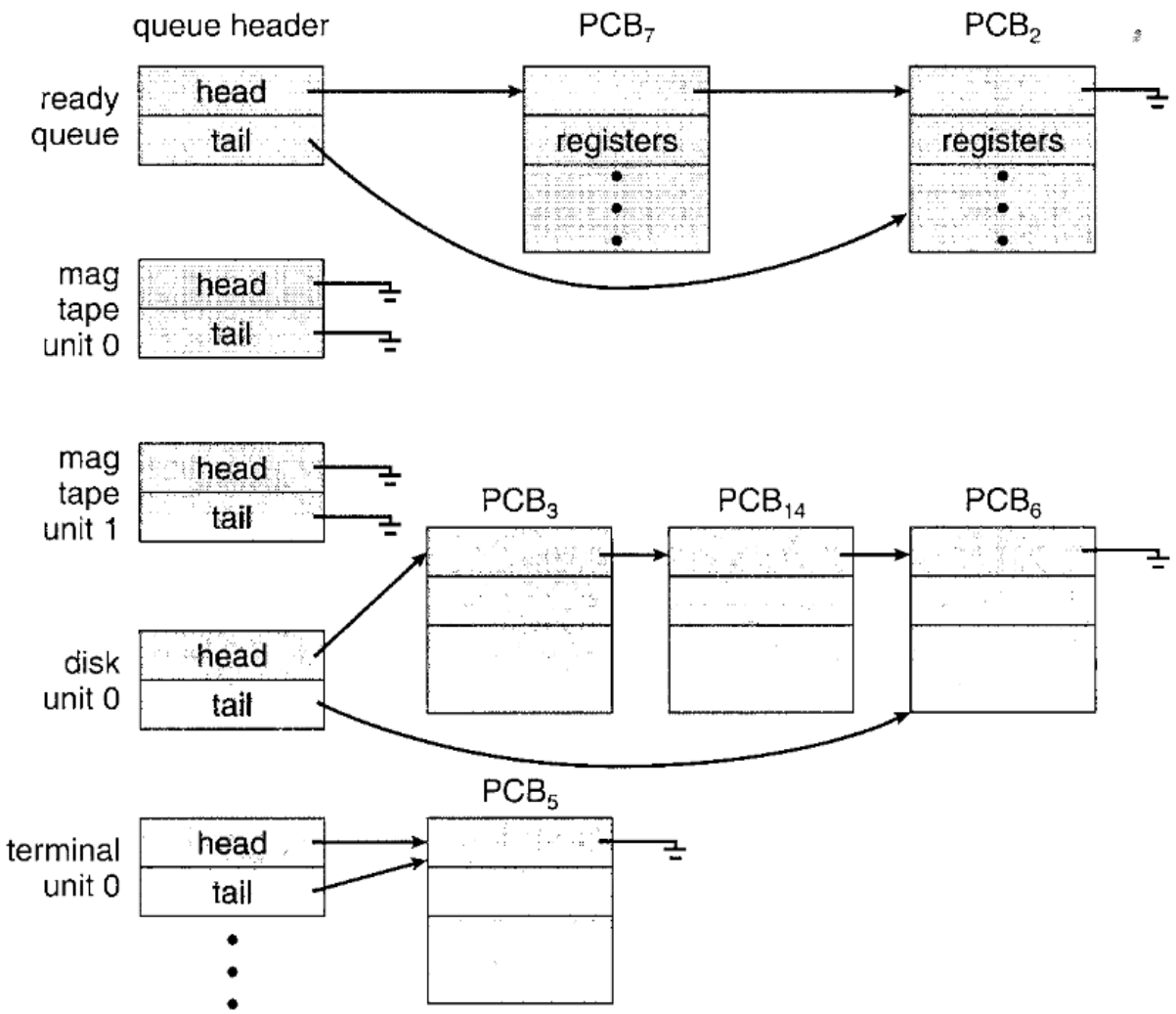


Dodela resursa procesima
(Process scheduling)

- Mutlitasking omogućava izvršavanje više procesa u isto vreme čime se povećava efikasnot iskorišćenja sistema.
- Cilj multitaskinga jeste naizmenična dodela CPU procesima čime se omogućava interakcija korisnika sa programima u toku njihovog izvršavanja.
- Svaki proces koji započne svoje izvršavanje na sistemu se smešta u red čekanja.
- Procesi koji se već nalaze u glavnoj memoriji sistema i čekaju na izvršenje se nalaze u **ready** redu čekanja (ready queue).
  - **ready** red čekanja je organizovan kao povezana lista.

Dodela resursa procesima
(Process scheduling)

1010011
1110100
1100001
1010110



Dodela resursa procesima (Process scheduling)

- Tokom izvršenja može se desiti da proces zahteva pristup I/O uređaju
  - Njegovo izvršenje se prekida i proces se smešta u red čekanja odgovarajućeg I/O uređaja (device queue)
    - ☑ Svaki I/O uređaj ima svoj red čekanja.
- Prilikom početka izvršavanja, svaki proces se smešta u ready red čekanja.
- Tokom izvršavanja može se desiti
  - Proces može da zahteva pristup I/O uređaju
    - ☑ smešta se u waiting red čekanja
  - Proces može da kreira novi proces i čeka na završetak njegovog izvršavanja
    - ☑ smešta se u waiting red čekanja
  - Izvršavanje procesa može biti prekinuto zbog interapta
    - ☑ proces tada biva ponovo smešten u ready red čekanja

Dodela resursa procesima
(Process scheduling)

- Tokom svog izvršavanja proces se može naći u različitim redovima čekanja.
- OS mora da rasporedi procese u ovim redovima čekanja u skladu sa nekim pravilom.

Dodela resursa procesima
(Process scheduling)

1010011
1110100
1100001
1010110

- Alokatori resursa (schedulers)
  - Dugoročni alokatori (long-term schedulers)
    - ☑ Izvršavaju se sa manjom frekvencijom i donose dalekosežnije odluke o izvršavanju procesa.
    - ☑ Donose odluku koje procese treba učitati sa sekundarne memorije u glavnu.
    - ☑ Definiše stepen multiprogramiranja, tj koliko se programa mogu izvršavati istovremeno.
    - ☑ Ako je broj stepen multiprogramiranja stabilan, tj srednji broj procesa koji se kreiraju je približno jednak srednjem broju procesa koji završavaju izvršavanje, LTS se aktivira samo kada neki proces završi sa izvršavanjem.
    - ☑ Često se ne koriste, već se svaki novi proces smešta odmah u ready red čekanja.
      - Stabilnost sistema zavisi od fizičkih ograničenja, na primer raspoložive memorije.
  - Kratkoročni alokatori (short-term schedulers)
    - ☑ Često se izvršavaju i moraju biti veoma brzi
    - ☑ Donose odluku koji proces iz ready reda čekanja će se izvršavati sledeći na CPU.

Dodela resursa procesima (Process scheduling)

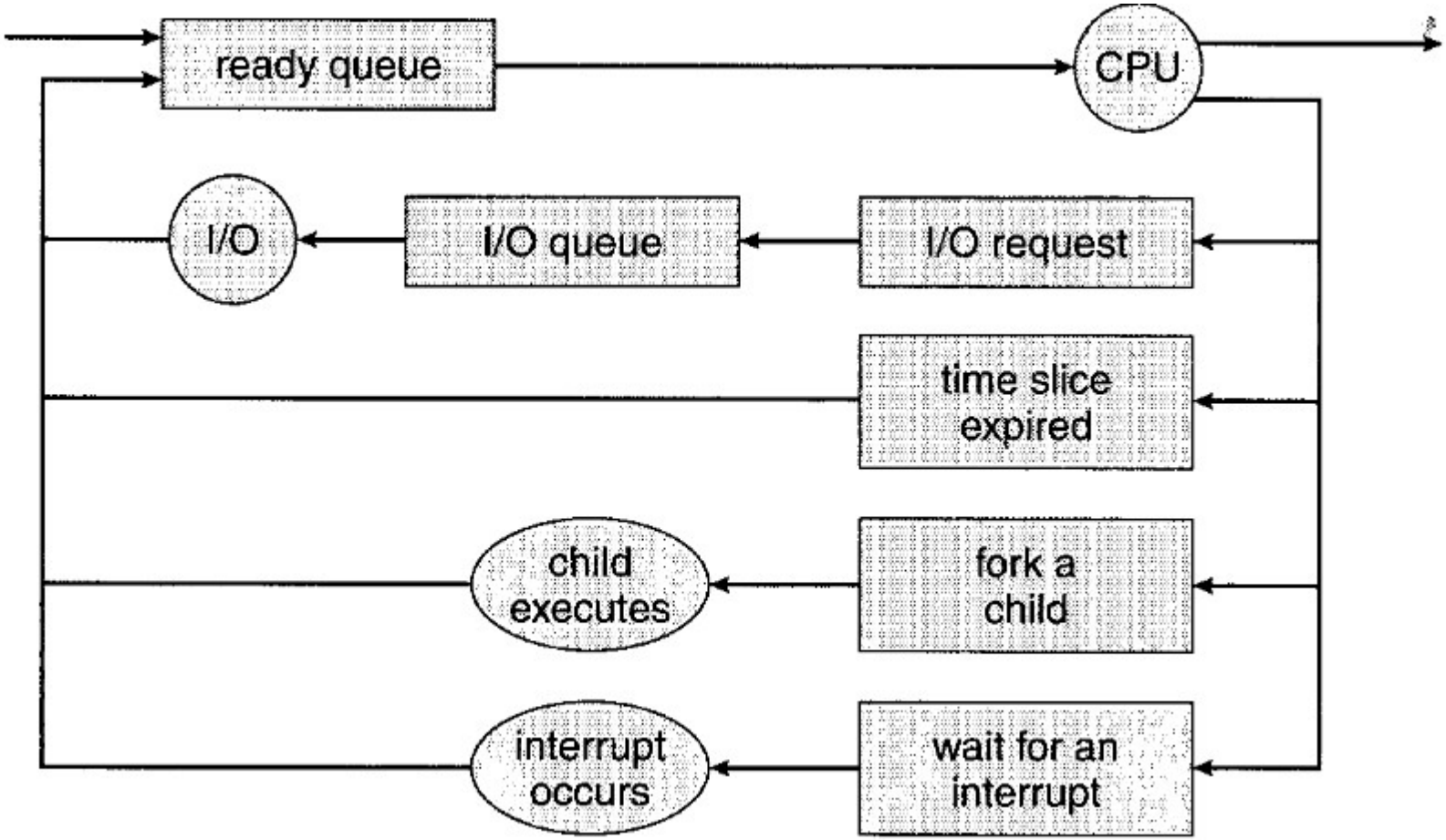
1010011
1110100
1100001
1010110

- Alokatori resursa (schedulers)
  - Srednjoročni alokatori (mid-term schedulers)
    - ☑ Cilj je da se u određenim vremenskim intervalima proveri iskorišćenost sistema i neki od procesa uklone iz ready reda čekanja.
    - ☑ Povećava se efikasnost iskorišćenja sistema.
    - ☑ Smanjuje se stepen multiprogramiranja.
    - ☑ Proces kasnije može nastaviti sa izvršavanjem.
      - Postupak smeštanja procesa privremeno na HDD se naziva svapovanje (swapping).

- Razlika između alokatora resursa je u učestanosti sa kojom se donose odluke o redosledu izvršavanja procesa.
  - U stabilnom stanju, srednji broj procesa koji se kreira je jednak srednjem broju procesa koji završavaju svoje izvršavanje.

Dodela resursa procesima
(Process scheduling)

1010011
1110100
1100001
1010110



Dodela resursa procesima (Process scheduling)

- Procesi se mogu podeliti na:
  - I/O ograničene procese
    - ☑ Proces koji provede više vremena u komunikaciji sa I/O uređajem nego izvršavajuci se na CPU.
  - CPU ograničene procese
    - ☑ Proces koji provede više vremena izvršavajući instrukcije na CPU nego u komunikaciji sa I/O uređajima.
- Važno je da dugoročni alokator resursa odabere dobru kombinaciju I/O i CPU ograničenih procesa kako bi svi resursi sistema bili stalno zauzeti.
- Dugoročni alokatori nisu uvek prisutni u OS.
  - Postoji samo kratkoročni alokator i stabilnost ovih sistema zavisi od fizičkih ograničenja sistema i ponašanja korisnika.
    - ☑ Ako performanse sistema opadnu do određenog nivoa neki korisnici će jdnostavno odustati i preći na izvršavanje nekih drugih zadataka.

Promena konteksta (Context switch)

1010011
1110100
1100001
1010110

- Promena konteksta
  - Prelazak sa izvršavanja jednog procesa na drugi zahteva skladištenje stanja starog procesa i unošenja stanja novog procesa.
  - Vreme potrebno za promenu konteksta CPU predstavlja čist gubitak jer tokom njega sistem ne obavlja nijedan koristan zadatak.
  - Brzina promene konsteksta zavisi od brzine memorije, broja registara i postojanja specijalnih instrukcija.
- Uticaj hardvera
  - Kod sistema kod kojih postoji veliki broj registara, promena konteksta se sastoji samo u promeni pointera na drugi skup registara.
    - ☑ Registri organizovani u banke registara.
    - ☑ Svaka banka se sastoji iz:
      - Parametarskih registara (parametri koji se preuzimaju od procesa roditelja).
      - Lokalni registri (koriste se za čuvanje lokalnih promenljivih).
      - Privremeni registri (koriste se za razmenu parametara koji se prenose procesu nasledniku).

Kooperacija između procesa

- Procesi koji se konkurentno izvršavaju na sistemu mogu biti:
  - Međusobno nezavisni
    - ☑ Ne može da utiče na druge ili ne mogu da utiču na njega ostali procesi tokom njegovog izvršavanja.
    - ☑ Ne deli nikakve podatke sa drugim procesima.
  - Kooperirajuci procesi
    - ☑ Proces može da utiče na druge procese ili mogu da utiču na njega drugi procesi.
    - ☑ Dele podatke.

Kooperacija između procesa

- Razlozi implementacije kooperirajućih procesa:
  - Razmena informacija
    - ☑ Više korisnika može biti zainteresovano za konkurentni pristup istim podacima.
  - Ubrzavanje izvršenja instrukcija.
    - ☑ Neki zadatak možemo podeliti na manje zadatke koji se mogu izvršavati u paraleli.
  - Modularnost
    - ☑ Ako želimo da razvijemo modularni sistem gde su funkcije sistema podeljene u odvojene procese.
  - Pogodnosti primene
    - ☑ Ako jedan korisnik želi da izvršava više od jednog zadatka.
- Moraju se razviti posebni mehanizmi za komunikaciju između procesa.

Niti
(Threads)

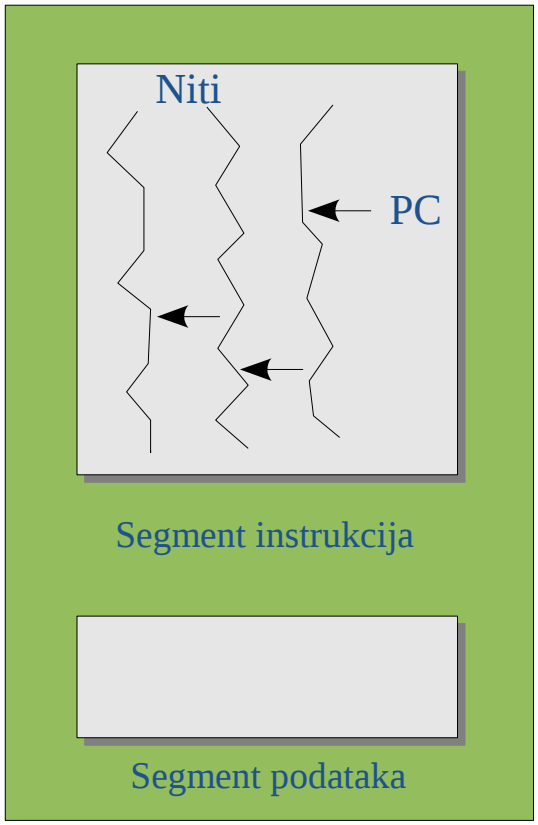
- Proces je definisan preko resursa koje koristi pri izvrsenju i tacke do koje je stigao tokom izvrsenja.
- Postoje situacije kada je potrebno da procesi dele resurse tokom izvrsenja
  - Dosta vremena se gubi pri promeni konteksta procesa.
  - Uvodi se pojam niti
- Nit (Thread) je odredjena
  - Stanjem programskog brojaca
  - Stanjem registara
  - Stanjem magacina
- Niti koje se konkurentno izvrsavaju nisu nezavisne na isti nacin kao i procesi
  - Za razliku od procesa, niti unutar jednog procesa dele isti adresni prostor i globalne promenljive unutar njega

Niti
(Threads)

- Vise procesa koji se konkurentno izvrsavaju
  - Svaki proces je nezavisna celina i odredjen je svojim PC, registrima, magacinom i adresnim prostorom
- Vise niti koje se konkurentno izvrsavaju zahtevaju manje resursa od vise procesa koji bi obavljali isti zadataka
  - Usteda na memoriji, broju fajlova i alokaciji CPU
- Niti su po mnogo cemu slicne procesima
  - Mogu se naci u istim stanjima kao i proces
  - Samo jedna nit se moze izvrsavati u jednom trenutku
  - Svaka nit ima svoj PC i magacin
  - Niti mogu kreirati podniti
- Za razliku od procesa, niti nisu u potpunosti nezavisni entiteti
  - Posto dele adresni prostor, mogu da menjaju sadrzaj memorije drugih niti

Niti
(Threads)

— Zastita pristupa podacima razlicitih niti nije potrebna jer pripadaju istom procesu i po prirodi nisu medjusobno stetni (a nije ni moguca)



Niti (Threads)
- Parametri niti -

- Parametri procesa zajednicki za sve niti
  - Adresni prostor
  - Globalne promenljive
  - Otvoreni fajlovi
  - Podprocesi
  - Alarmi koji ocekuju opsluzivanje
  - Signali i parametri signala
  - Statistika procesa
- Parametri pojedinacnih niti
  - Programski brojac
  - Registri
  - Magacin
  - Stanje

Niti
(Threads)

- Proces zapocinje sa izvrsavanjem jedne niti
- Niti mogu kreirati druge niti
  - thread_create + ime nove niti koja treba da se izvrsava
  - Nema potrebe navoditi druge parametre poput adresnog prostora jer se obe niti izvrsavaju u istom adresnom prostoru
- Niti mogu biti organizovane hijerarhijski
- Nit zavrsava izvrsenje pozivom funkcije thread_exit
- Jos jedan uobicajeni poziv jeste thread_yield
  - Nit dobrovoljno prekida sa svojim izvrsavanjem i prepusta CPU drugoj niti
  - Omogucava multitaksin niti slicno procesima posto nema prekida koji bi ih na to prislio kao kod procesa

Niti
(Threads)

- Prednosti primene niti
  - Omogucavaju paralelno izvrsavanje zadataka
    - ☑ Bolje efikasnost iskoriscenja resursa, ako jedna nit prestane sa izvrsavanjem, druga nastavlja
  - Mogu se kreirati i prekinuti brze nego procesi jer su manje i zahtevaju manje resursa
  - Kombinovanjem CPU i I/O ogranicenih niti ubrzava se izvrsenje zadataka
  - Kod multiprocesorskih sistema omogucavaju paralelnu obradu

Niti

- Različite niti u okviru procesa nisu potpuno nezavisne u poređenju sa procesima koji se izvršavaju na istom računaru.
- Pošto niti istig procesa dele adresni prostor, one mogu da utiču na podatke drugih niti.
- Zaštita između niti je nemoguća i nije potrebna jer niti po definiciji sarađuju u okviru jednog procesa na izvršavanju zajedničkog zadatka
- Stanja kroz koja nit prolazi prilikom izvršavanja su slična stanjima u kojima može da se nađe proces tokom svog izvršavanja.
  - Razlika je u tome da kada jedna nit čeka na neki događaj ili je suspendovana, da li je i ceo preoces takođe u istom stanju ili izvršavanje preuzima druga nit.
- Kao i u slučaju procesa, dobitak na performansama je najveći kada imamo kombinaciju CPU i IO ograničenih niti.

Niti

- ULT vs KLT
- Prednosti ULT
  - CPU ne ovdi računa o implementaciji niti.
  - Primena ULT ne zavisi od toga da li OS podržava ili ne rad sa nitima.
  - Svaki proces, analogno tabeli procesa, održava tabelu niti.
  - Multipleksiranje niti je mnogo brže pošto aokaciju CPU obavljaju lokalne procedure i nema potrebe za promenom konteksta ili skladištenjem svih podataka iz registara CPU.
    - ☑ Moguće je prilagoditi algoritam za dodelu CPU sakom procesu individualno.
  - Glavni cilj jeste da imamo blokirajuće niti koje nisu u mogućnosti da blokiraju izvršavanje drugih niti i tako blokiraju ceo proces već se proces kontinualno izvršava izvršavanjem različitih niti.

Niti

— Nedostaci ULT

- Svaka nit, potencijano može dovesti do blokade celog procesa.
  - ☑ Zamena stranica kod virtuelne memorije.
- Moramo da obezbedimo ravnopravno izvršavanje niti u okviru procesa kako jedna nit nebi uzurpirala ceo proces.

Niti

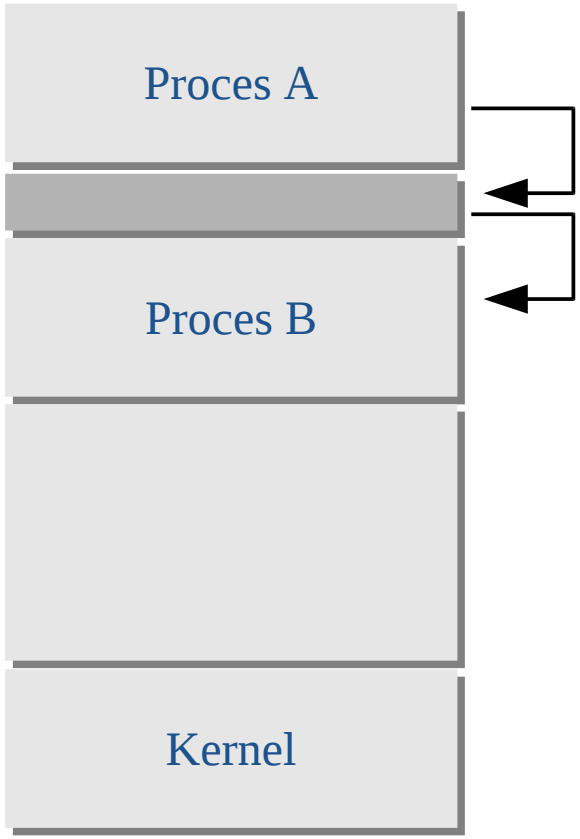
- KLT
- OS ima zadatak da prati sve podatke vezane za izvršavanje niti.
- Kada se neka nit blokira, OS odlučuje koja će nit isto ili nekog drugog procesa nastaviti da se izvršava.
- Proces kreiranja i prekida rada niti zahteva više resursa CPU pa procesi često vrše recikliranje niti gde nit sa istim opisom izvršava nek idrugi zadatak.
-

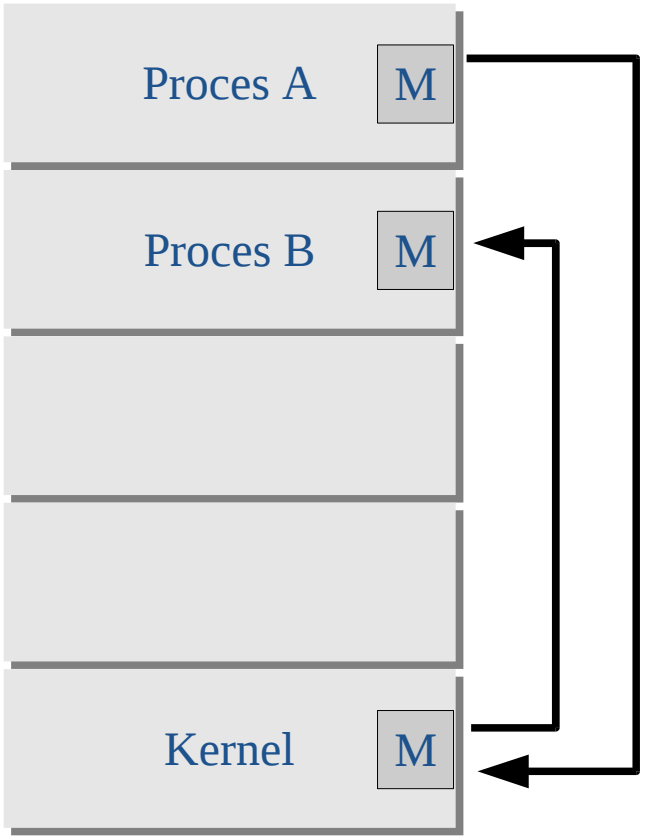
Komunikacija izmedju procesa

- Komunikacija izmedju procesa (IPC – Interprocess Communication)
- Dva osnovna modela za komunikaciju između procesa
  - Komunikacija preko zajednicke memorije (shared memory)
    - ☑ Definise se deo memorije koji je zajednicki za dva procesa koji kooperiraju.
    - ☑ Razmena informacija se vrsi tako sto procesi upisuju i citaju podatke iz zajednicke memorije.
    - ☑ Maksimalna brzina pri komunikaciji (brzina ogranicena brzinom memorije)
    - ☑ Sistemski pozivi se koriste samo za uspostavljanje zajednicke memorije.
  - Komunikacija razmenom poruka
    - ☑ Pogodna za razmenu manje kolicine podataka.
    - ☑ Lakse se implementira od modela sa zajednickom memorijom pri komunikaciji izmedju racunara.
    - ☑ Implementirana uz pomoc sistemskih poziva, pa stoga zahteva vise vremena za opsluzivanje od strane kernela
- Cesto se desava da su oba modela implementirana na OS-u.

Komunikacija izmedju procesa

1010011
1110100
1100001
1010110





Komunikacija izmedju procesa
- Komunikacija preko zajednicke memorije -

— Neophodno je definisati deo memorije zajednicki za dva ili vise procesa.

- Obicno se segment zajednicke memorije nalazi unutar adresnog prostora procesa koji kreira zajednicku memoriju.
- Ostali procesi moraju da dodaju prostor zajednicke memorije svom adresnom prostoru.
  - ☑ OS stiti adresni prostor procesa od drugih procesa.
  - ☑ Procesi moraju da eksplicitno dozvole pristup drugih procesa delu svog adresnog prostora koji se koristi kao zajednicka memorija.
  - ☑ OS nakon toga prestaje da kontolise pristup zajednickom delu memorije.
  - ☑ Procesi moraju sami da osiguraju korektan i organizovan pristup memoriji.
- Komunikacija preko bafera
  - ☑ Bafer neogranicene velicine
    - Izvor upisuje podatke brzinom kojom ih generise
    - Odrediste moze biti primorano da ceka na podatke ako ih cita vecom brzinom
  - ☑ Bafer ogranicene velicine

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

1010011
1110100
1100001
1010110

- Omogucava komunikaciju i sinhronizaciju izmedju procesa bez potrebe da dele adresni prostor
  - Narocito korisna pri komunikaciji izmedju procesa u distribuiranom okruzenju
- Implementira se primenom najmanje dve vrste poruka
  - send(message)
  - receive(message)
- Duzina poruka koje se razmenjuju mogu biti:
  - fiksne
    - ☑ Laksa implementacija
    - ☑ Proces komunikacije komplikovaniji
  - promenljive duzine
    - ☑ Slozenija implementacija na sistemskom nivou
    - ☑ Komunikacija jednostavnija

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

1010011
1110100
1100001
1010110

- Izmedju procesa mora postojati komunikacioni link da bi mogli da komuniciraju
  - Direktna ili inidirektna komunikacija
  - Sinhrona ili asinhrona komunikacija
  - Automatsko ili eksplicitno baferovanje

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

- Imenovanje procesa
- Prilikom komunikacije se javlja potreba za imenovanjem (adresiranjem) procesa
  - Direktna komunikacija
    - ☑ Scaki proces koji zeli da komunicira mora eksplicitno da naznaci kom/od kog procesu/a salje/prima poruku.
      - send(P, message) – Posaljui poruku message procesu P
      - receive(Q, message) – Primi poruku procesa Q
    - ☑ Veza uspostavljena na ovaj nacin ima sledece osobine:
      - Veza se uspostavlja direktno izmedju bilo koja dva procesa koji zele da komuniciraju. Procesi trebaju samo da znaju medjusobna imena (ID, adresu).
      - Veza se dodeljuje tacno jednom paru procesa.
      - Izmedju jednog para procesa moze postojati tacno jedna veza.
    - ☑ Asimetricna direktna komunikacija
      - Samo izvor poruke mora da navede adresu odredista, prijemni proces ne mora da imenuje posiljaoca
      - receive(id, message) – prijem poruke od bilo kog procesa. *id* – identifikacija posiljaoca

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

— Nedostatak direktne simetricne ili asimetricne komunikacije se odnosi na to da promenom ID procesa, moramo da nadjemo i promenimo sve moguce reference ka starom ID procesa.

— Kod indirektne komunikacije, poruke se salju i promaju preko **mailbox**-ova/**porta**.

- Mailbox/port se moze posmatrati kao objekat u koji se poruke mogu smestati i iz koga se poruke mogu preuzimati.
- Procesi mogu medjusobno da komuniciraju preko vise razlicitih mailbox-ova.
- Komunikacija se obavlja pozivima:
  - ☑ send(A, message) – Posalji poruku u mailbox A
  - ☑ receive(A, massage) – Preuzmi poruku iz mailboxa A

— Osobine veze kod inirektne komunikacije

- Veza se uspostavlja samo ako oba procesa imaju zajednicki mailbox
- Istu vezu mogu deliti vise procesa
- Izmedju dva procesa mogu postojati vise veza koje su definisane razlicitim mailbox-ovima.
  - ☑ Primer: Proces P1 salje poruku preko mailboxa *A*, procesi P2 i P3 preuzimaju poruku. Koji proces ce preuzeti poruku?
    - Zavisi od broja procesa koji mogu komunicirati preko jednog mailboxa.
    - Zavisi od broja procesa koji mogu istovremeno da izvrse komandu receive.
    - Da li sistem odredjuje koji proces ce da primi poruku?

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

- Pravimo razliku da li mailbox pripada korisniku ili OS-u.
  - Mailbox pripada procesu
    - ☑ Ako mailbox pripada procesu tada se on kreira u okviru adresnog prostora procesa.
    - ☑ Pravi se razlika izmedju vlasnika mailbox-a koji moze preko njega samo da prima poruke i posiljaoca koji preko njega moze samo da salje poruke.
    - ☑ Po zavrsetku izvrsenja procesa, mailbox se takodje brise. ⇨ Svaki proces koji naknadno pokusa da posalje poruku preko tog mailbox-a mora biti obavesten da ono vise ne postoji.
  - Mailbox pripada OS-u
    - ☑ Postoji kao nezavistan entitet. Ne pripada nijednom narocitom procesu.
    - ☑ OS mora da poseduje mehanizme koji bi omogucili procesu da:
      - Kreira novi mailbox
      - Salje i prima poruke preko mailbox-a
      - Obrise mailbox
    - ☑ Proces koji kreira mailbox je inicijalno jedini vlasnik mailbox-a i samo on moze da prima poruke preko tog mailboxa.
    - ☑ Vlasnistvo nad mailbox-om i privilegije preuzimanja poruka se mogu preneti na druge procese prigodnim sistemskim pozivima.

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

— Sinhronizacija procesa

- Slanje i prijem poruka moze biti sinhron (blocking) ili asinhron (nonblocking).
  - ☑ Sinhrono slanje poruka
    - Proces koji salje poruku je blokiran dok prijemni proces ne preuzme poruku.
  - ☑ Asinhrono slanje poruka
    - Proces koji salje poruku nastavlja sa izvrsavanjem nakon sto posalje poruku.
  - ☑ Sinhrono preuzimanje poruka
    - Prijemni proces prestaje sa izvrsavanjem dok ne preuzme poruku.
  - ☑ Asinhrono preuzimanje poruka
    - Prijemni proces preuzima ili validnu poruku ili Null.
- U OS moguce sve ove kombinacije.

Komunikacija izmedju procesa
- Komunikacija razmenom poruka -

- Bez obzira da li se radi o direktnoj ili indirektnoj komunikaciji, poruke se smestaju u posebne redove (queue, buffer, FIFO).
- Redovi cekanja se mogu implementirati kao:
  - Kapaciteta nula
    - ☑ Duzina reda je jednaka nuli. Poruka se odmah salje ka odredistu.
  - Ogranicenog kapaciteta
    - ☑ Red cekanja ima konacnu duzinu *n*, tj. maksimalno *n* poruka se mogu smestiti u njemu. Ako red nije pun, proces smesta poruku u red i nastavlja sa izvrsavanjem. Ako je red pun, proces prestaje sa izvrsavanjem dok se red ne isprazni.
  - Neogranicenog kapaciteta
    - ☑ Teoretski, bilo koji broj poruka se mogu naci u redu cekanja.

Vezbe

- Potrebno je da program sadrzi u zaglavlju sledece heder fajlove kako bi mogli da koristimo naredbe vezane za upravljanje procesima:
  - unistd.h
  - sys/types.h
- Naredba za kreiranje procesa

```
pid_t fork (void)
```

- Naredba za izvrsavanje programa
  - Omogucava da nova nit procesa pokrene izvrsavanja novog programa
  - Menja se sadrzaj adresnog prostora osim argumenata i parametara okruzenja

```
int execv (const char *filename, char *const argv[])
```

— Primer struktire proc

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  volatile int pid;            // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Vezbe

— Stanja procesa

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

— Kontekst procesa

```
// Saved registers for kernel context switches.
// Don't need to save all the segment registers (%cs, etc),
// because they are constant across kernel contexts.
// Don't need to save %eax, %ecx, %edx, because the
// x86 convention is that the caller has saved them.
// Contexts are stored at the bottom of the stack they
// describe; the stack pointer is the address of the context.
// The layout of the context matches the layout of the stack in swtch.S
// at the "Switch stacks" comment. Switch doesn't save eip explicitly,
// but it is on the stack and allocproc() manipulates it.
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

Vezbe

— Inicijalizacija procesa

```
// Set up first user process.
void
userinit(void)
{
  struct proc *p;
  extern char _binary_initcode_start[], _binary_initcode_size[];

  p = allocproc();
  initproc = p;
  if((p->pgdir = setupkvm()) == 0)
    panic("userinit: out of memory?");
  inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
  p->sz = PGSIZE;
  memset(p->tf, 0, sizeof(*p->tf));
  p->tf->cs = (SEG_UCODE << 3) | DPL_USER;
  p->tf->ds = (SEG_UDATA << 3) | DPL_USER;
  p->tf->es = p->tf->ds;
  p->tf->ss = p->tf->ds;
  p->tf->eflags = FL_IF;
  p->tf->esp = PGSIZE;
  p->tf->eip = 0;  // beginning of initcode.S

  safestrcpy(p->name, "initcode", sizeof(p->name));
  p->cwd = namei("/");

  p->state = RUNNABLE;
}
```

— Kreiranje procesa

```
int pid;

pid = fork();
if(pid > 0){
  printf("parent: child=%d\n", pid);
  pid = wait();
  printf("child %d is done\n", pid);
} else if(pid == 0){
  printf("child: exiting\n");
  exit();
} else {
  printf("fork error\n");
}
```

Vezbe

— Pokretanje novog programa

```
char *argv[3];

argv[0] = "echo";
argv[1] = "hello";
argv[2] = 0;
exec("/bin/echo", argv);
printf("exec error\n");
```

Domaci zadatak

1010011
1110100
1100001
1010110

Domaci zadatak

1010011
1110100
1100001
1010110