OPERATIVNI SISTEMI

III - Raspoređivanje procesa

III - Raspoređivanje procesa

S A D R Ž A J

3.1 - Uvod

- ideja multiprogramiranja je jasna i jednostavna
- svaki proces ostaje u stanju izvršenja dok mu
  - ne istekne vremenski kvantum ili
  - dok ne dođe u situaciju da mora da čeka na neki događaj (npr. završetak U/I komande)
- na prostim OS procesor ne radi ništa dok proces čeka
- na višeprocesnim OS radna memorija se puni većim brojem procesa
  - kada aktivni proces iz nekog razloga mora da čeka, OS mu oduzima procesor i dodeljuje ga nekom drugom procesu
- dodela procesora po nekom algoritmu je jedna od fundamentalnih funkcija OS-a
- sličan princip se primenjuje i za druge resurse, kao što su U/I uređaji

3.1 - Uvod

- procesor se dodeljuje drugom procesu pod sledećim okolnostima
  1. kada proces pređe u stanje čekanja na resurs, npr. čeka na završetak I/O operacije koju je inicirao
  2. kada proces roditelj čeka proces dete da završi aktivnosti
  3. prilikom tranzicije RUN-STOP, odnosno kada tekući proces završi sve svoje aktivnosti
  4. prilikom tranzicije RUN-READY
  5. prilikom tranzicije WAIT-READY, u slučaju kada je proces koji je napustio stanje WAIT i prešao u stanje READY višeg prirориteta od procesa koji se trenutno izvršava

Raspoređivanje bez predpražnjenja (*nonpreemptive scheduling*)

- raspoređivanje bez prekidanja izvršenja tekućeg procesa
- procesor se može oduzeti samo od procesa koji je završio svoje aktivnosti ili čeka na resurs (slučajevi od 1 do 3)

3.1 - Uvod

Raspoređivanje sa predpražnjenjem (*preemptive scheduling*)

- procesor se može oduzeti procesu koji nije završio svoje aktivnosti i nije blokiran
- prednost
  - procesor se može jako brzo dodeliti procesu višeg prioriteta koji zahteva izvršenje
- problem
  - moguća nekonzistentnost zajedničkih podataka
  - primer: proces koji je otpočeo ažuriranje podataka je prekinut, a kontrola dodeljena drugom procesu koji koristi iste podatke
- jedna od hardverskih komponenti neophodna za pretpražnjenje je tajmer koji peroidično postavlja prekidni signal
- pretpražnjenje ima uticaj na konstrukciju jezgra OS-a
  - da ne bi nastao haos, proces koji je pomoću sistemskog poziva prešao u režim jezgra se ne može prekidati sve dok je u tom režimu rada

3.1 - Uvod

Kriterijumi dodele procesora

- iskorišćenje procesora
  - po ovom kriterijumu procesor treba da bude zauzet, odnosno da radi nešto bez prestanka
- propusna moć sistema (*throughput*)
  - definiše se kao broj procesa izvršen u jedinici vremena
  - za dugačke procese ovaj odnos može biti 1 proces na sat, za kratke transakcione procese može biti na primer 10 procesa u sekundi
  - propusna moć zavisi od vrste procesa i brojnih hardverskih i sistemskih osobina
- vreme potrebno za kompletiranje procesa (*turnaround time*)
  - ukupno vreme potrebno da se izvrši pojedinačni proces
  - računa se od trenutka kreiranja do završetka procesa, a uključuje i vreme potrebno da proces uđe u red čekanja, vreme provedeno u redu čekanja, vreme korišćenja procesora i izvršenja U/I operacija

3.1 - Uvod

Kriterijumi dodele procesora

- vreme čekanja (*waiting time*)
  - ukupno vreme koje proces provede u redu čekanja na procesor (ready queue)
  - proces do kraja svog izvršavanja može više puta da bude u ovom redu čekanja
  - na ovo vreme direktno utiču algoritmi za raspoređivanje
- vreme odziva (*response time*)
  - vreme potrebno da se nakon slanja zahteva pojave prvi rezultati izvršenja procesa
  - ovo vreme je veoma bitno kod interaktivnih sistema
  - zavisno od namene sistema algoritmi se optimizuju za određene kriterijume
    - nekad se forsiraju najmanje ili najveće vrednosti, a nekada srednje
    - na primer, za interaktivne sisteme je značajnije minimizirati najveće vreme odziva a ne srednje vreme

3.1 - Uvod

Ispitivanje algoritama

- kako odabrati algoritam za raspoređivanje procesa u nekom posebnom OS-u?
- prvo je potrebno postaviti glavni kriterijum ili kombinaciju kriterijuma kao što su
  - maksimizovanje iskorišćenosti procesora uz ograničenje da max. vreme odziva bude1sekunda
  - maksimizovanje propusne moći tako da srednje vreme izvršavanja bude linearno proporcionalno ukupnom vremenu izvršavanja
- kada se defniše kriterijum obavlja se analiza različitih algoritama po pravilu putem simulacije
- za simulaciju je potrebno radno opterećenje (*workload*) koje može biti sintetičko ili realno

3.1 - Uvod

Ispitivanje algoritama

- opterećenje se formira snimanjem svih procesa jednog realnog sistema u određenom periodu vremena u statističku datoteku (*trace file*)
- simulira se red čekanja koji može biti izveden u jednom ili u više nivoa
- svaki red ima svoj kapacitet, vremensku raspodelu nailaska procesa, i algoritam koji se primenjuje za raspoređivanje procesa
- kada simulator obavi svoje, dobijaju se informacije o uspešnosti jednog algoritma na datom statističkom uzorku

3.2 - Algoritmi za dodelu procesora

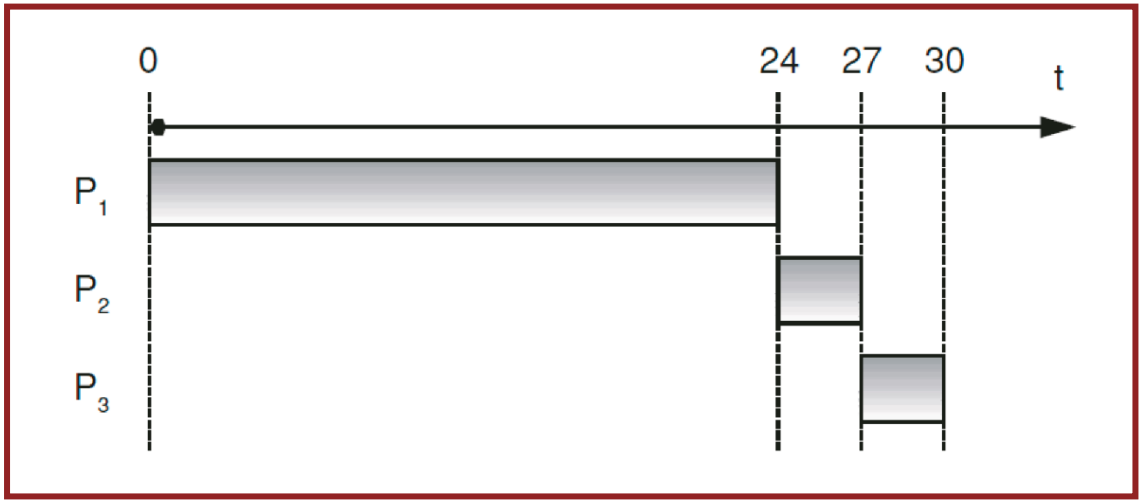
Prvi došao, prvi uslužen (FCFS, *First Come, First Served*)

- najprostiji algoritam za raspoređivanje procesora
- procesi dobijaju procesor onim redom kojim su pristizali u red čekanja
- red čekanja funkcioniše po standardnom FIFO (*First In First Out)* principu
  - PCB procesa koji je ušao u red čekanja na procesor stavlja se na kraj liste
  - procesor se uvek dodeljuje onom procesu koji je na početku liste
- algoritam je lak i za razumevanje i za implementaciju
- međutim, srednje vreme čekanja (engl. *waiting time)* za ovaj algoritam je krajnje dugačko

3.2 - Algoritmi za dodelu procesora

Prvi došao, prvi uslužen (FCFS, *First Come, First Served*)

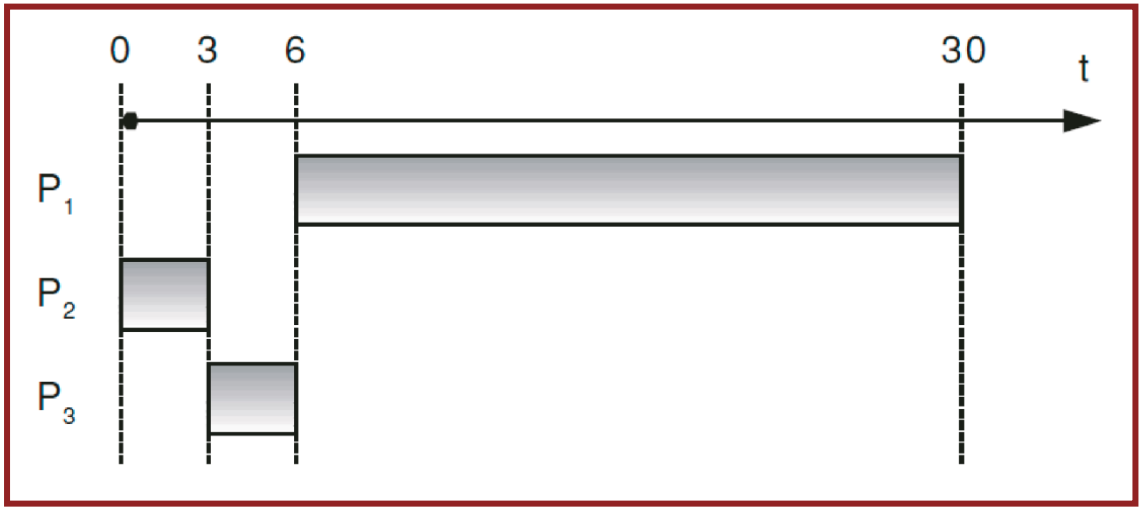
- primer
  - procesi P1, P2 i P3 čija su vremena izvršavanja 24, 3 i 3 ms (milisekunde) nailaze u poretku P1, P2, P3
  - vremena čekanja *(waiting time)* za procese P1, P2 i P3 su 0, 24 i 27 ms respektivno
  - srednje vreme čekanja t_w = (0+24+27)/3 = 17ms



3.2 - Algoritmi za dodelu procesora

Prvi došao, prvi uslužen (FCFS, *First Come, First Served*)

- ako se međutim izmeni redosled izvođenja procesa tako da on bude P2, P3, P1, situacija se značajno menja
- vremena čekanja za procese P1, P2 i P3 su 6, 0 i 3 ms respektivno
- srednje vreme čekanja je $t_w = (6+0+3)/3 = 3$ ms



3.2 - Algoritmi za dodelu procesora

Prvi došao, prvi uslužen (FCFS, *First Come, First Served*)

- na osnovu primera zapaža se da srednje vreme čekanja zavisi
  - od dužine trajanja procesa
  - od sekvence njihovog nailaska u sistem
- kod FCFS algoritma moguća je pojava konvoj efekta
  - svi procesi čekaju da se završi jedan proces koji dugo traje
  - analogija sa saobraćajem
    - veliko sporo vozilo (šleper) na uzbrdici prati veći broj manjih bržih vozila
    - brža vozila ne mogu da preteknu šleper zbog pune linije
- FCFS ne poštuje priorite procesa već samo vreme dolaska u red čekanja
  - zato se izvodi bez pretpražnjenja, odnosno prekidanja procesa
  - to znači da se proces koji je dobio procesorsko vreme izvršava u potpunosti ili do trenutka blokiranja zbog čekanja na U/I operaciju
  - zbog toga je FCFS krajnje nepodesan za interaktivne sisteme

3.2 - Algoritmi za dodelu procesora

Prvo najkraći posao (SJF, *Shortest Job First*)

- za sve procese u redu čekanja procenjuje se vreme potrebno za izvršavanje.
  - procesor se dodeljuje onom procesu kome treba najmanje vremena za izvršenje, odnosno procesu koji bi najkraće trajao
  - za one procese koji imaju ista procenjena vremena izvršavanja primenjuje se FCFS algoritam
- osnovni nedostatak
  - sistem ne može tačno da proceni trajanje procesa u celini
  - predviđanje trajanja procesa se zasniva na činjenici da se proces sastoji iz više ciklusa korišćenja procesora (*CPU burst)* koji se prekidaju U/I operacijama
  - procena trajanja se obavlja tako što se proceni trajanje sledećeg ciklusa korišćenja procesora na osnovu prethodnih ciklusa korišćenja procesora
  - algoritam se preciznije može nazvati najkraći sledeći CPU ciklus (*shortest next CPU burst)*

3.2 - Algoritmi za dodelu procesora

Prvo najkraći posao (SJF, *Shortest Job First*)

- Procena trajanja sledećeg ciklusa korišćenja procesora
  - jedan od načina da se obavi procena trajanja sledećeg ciklusa korišćenja procesora je određivanje srednje vrednosti na bazi svih prethodnih CPU ciklusa. Neka su
    - τ_{n+1} - procenjena vrednost trajanja sledećeg CPU ciklusa
    - $\mathbf{t_n}$ - realno trajanje prethodnog CPU ciklusa
    - τ_n - procenjena vrednost trajanja prethodnog CPU ciklusa
    - koeficijent α je u intervalu $0< \alpha <1$
    - koeficijent α određuje relativnu težinu istorije procesa, tj. poslednjeg poznatog ciklusa
  - formula za procenu trajanja sledećeg ciklusa korišćenja procesora kombinuje prethodnu vrednost $\mathbf{t_n}$ i istoriju procene trajanja ciklusa koja je sublimisana u $\boldsymbol{\tau_n}$

$$\tau_{n+1} = \alpha\, t_n + (1- \alpha)\, \tau_n$$

3.2 - Algoritmi za dodelu procesora

Prvo najkraći posao (SJF, *Shortest Job First*)

- postoje dve varijante SJF algoritma
  - bez pretpražnjenja (*non-preemptive)*
    - uvek završiti tekući proces bez obzira kakav se novi proces pojavio u redu čekanja
  - sa pretpražnjenjem (*preemptive)*
    - ukoliko je vreme potrebno za izvršenje novog procesa kraće od vremena potrebnog za završetak aktivnosti tekućeg procesa, procesor će biti dodeljen novom procesu
    - naziva se i raspoređivanje sa najmanjim preostalim vremenom (*shortest- remaining time first)*
- za SJF veoma je bitno znati dve informacije
  - vreme nailaska procesa u red čekanja (*arrival time)*
  - vreme potrebno za izvršenje procesa (*CPU burst time)*
- za razliku of SJF kod *FCFS* je bitan samo jedan parametar, a to je vreme nailaska procesa u red

3.2 - Algoritmi za dodelu procesora

Raspoređivanje na osnovu prioriteta procesa

- svakom procesu se dodeljuje prioritet a algoritam bira proces sa najvećim prioritetom
- prioritet je celobrojna vrednost a najčešće se koristi konvencija po kojoj
  - manji broj znači veći prioritet
  - veći broj znači manji prioritet (na primer, vrednost 0 je najviši prioritet)
  - u slučaju procesa sa jednakim prioritetom, odluka se donosi po FCFS principu
- SJF je specijalan slučaj prioritetno-orijentisanih algoritama za raspoređivanje
  - prioritet je obrnuto proporcionalan trajanju sledećeg CPU ciklusa procesa
  - to znači da duži procesi imaju manji prioritet i obrnuto

3.2 - Algoritmi za dodelu procesora

Raspoređivanje na osnovu prioriteta procesa

- prioritetno-orijentisani algoritmi mogu biti realizovani
  - sa pretpražnjenjem (*priority preemptive scheduling)*
    - tekući proces prekida izvršenje ukoliko se pojavi proces većeg prioriteta (provera se odvija uvek kada se pojavi novi proces)
  - bez pretpražnjenja (*priority non-preemptive scheduling)*
    - proces koji je dobio kontrolu ne prekida izvršenje bez obzira što se pojavio proces većeg prioriteta
- glavni problem ovog algoritma je blokiranje niskoprioritetnih procesa
  - procesi niskog prioriteta mogu veoma dugo (neograničeno) da čekaju na procesor
  - primer rešenja problema zakucavanja niskoprioritetnih procesa je povećanje prioriteta sa vremenom provedenim u redu čekanja na procesor

3.2 - Algoritmi za dodelu procesora

Raspoređivanje na osnovu prioriteta procesa

- glavni problem ovog algoritma je blokiranje niskoprioritetnih procesa
  - preciznije rečeno rezultujući prioritet se formira na osnovu
    - početnog prioriteta, koji proces dobija kada uđe u red čekanja na procesor
    - vremena provedenog u redu čekanja (*aging)*
- ako prioritete dinamički određuje OS dobro je uzeti u obzir
  - procesima koji intenzivno koriste U/I uređaje treba dodeliti viši prioritet jer su ovakvi procesi često blokirani a procesor im treba u kraćim intervalima
  - procesima koji intenzivno koriste procesor i operativnu memoriju treba dodeliti niži prioritet
    - u obrnutom slučaju U/I procesi bi dugo čekali na procesor i zauzimali mesto u operativnoj memoriji, a U/I podsistem bi ostao prilično besposlen

3.2 - Algoritmi za dodelu procesora

Round Robin (RR)

- Round Robin (RR) se može posmatrati kao FCFS algoritam sa pretpražnjenjem
- najpre se definiše vremenski kvantum (*time slice)* i kružni red čekanja na procesor
- svaki prispeli proces ubacuje se na kraj liste
- od sistema se očekuje da generiše prekidni signal po isteku vremenskog kvantuma
- RR funkcioniše na sledeći način
  - proces je završio aktivnost pre isteka vremenskog kvantuma
    - proces oslobađa procesor a RR uklanja proces iz liste i predaje kontrolu sledećem procesu
  - proces nije završio aktivnosti ali mora da prekine izvršenje kada mu istekne vremenski kvantum
    - RR postavlja prekinuti proces na kraj reda a kontrolu predaje sledećem procesu iz liste

3.2 - Algoritmi za dodelu procesora

Round Robin (RR)

- RR funkcioniše na sledeći način
  - proces se blokirao zbog čekanja na U/I operacije
    - blokirani proces oslobađa procesor a RR predaje kontrolu sledećem procesu iz liste
    - proces prelazi u red čekanja na kraj RR liste tek nakon povratka u stanje READY
- RR je fer je prema procesima (svi dobijaju procesor na korišćenje ravnomerno i ravnopravno)
  - ako imamo *n* procesa u redu čekanja, svakom procesu pripada 1/n procesorskog vremena
  - ako je kvantum dužine q, nijedan proces u stanju READY neće čekati više od (n-1)q vremena na sledeću dodelu procesora

3.2 - Algoritmi za dodelu procesora

Round Robin (RR)

- performanse RR algoritma zavise od veličine vremenskog kvantuma
- u slučaju većih vremenskih kvantuma RR konvergira ka FCFS algoritmu
- u slučaju veoma malih vremenskih kvantuma svaki proces radi brzinom jednakom 1/n brzine realnog procesora
  - ovo je idealan slučaj koji pretpostavlja da dispečer obavlja prebacivanje konteksta trenutno
- realna situacija je nešto drugačija
  - dispečer je realizovan softverski i izvršava se kao zaseban proces
  - kao takav unosi kašnjenje pri svakom prebacivanju konteksta
  - realna brzina izvršavanja svakog procesa je manja od 1/n brzine realnog CPU i opada sa porastom učestanosti prebacivanja konteksta

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

- ova klasa algoritama odnosi se na sisteme i situacije u kojima se procesi mogu klasifikovati u različite grupe
  - primer: procesi koji rade u prvom planu (*foreground)* i procesi koji rade u pozadini (*background)*
    - ove dve klase procesa imaju različite zahteve za vreme odziva i zbog toga se koriste različiti algoritmi za raspoređivanje procesora
    - interaktivni procesi imaju osetno veći prioritet u odnosu na procese koji rade u pozadini.
- algoritmi za raspoređivanje u više redova dele red čekanja u više razdvojenih redova
- procesi se ubacuju u odgovarajući red čekanja na osnovu osobina procesa, kao što su veličina memorije, prioritet i tip procesa

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

- svaki red čekanja ima poseban nezavisan algoritam za raspoređivanje koji odgovara njegovim potrebama
  - primer
    - poseban red čekanja za interaktive procese se opslužuje po RR algoritmu
    - poseban red čekanja za procese koji rade u pozadini, koji se opslužuje po FCFS algoritmu

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

- u slučaju više redova čekanja, kada je u pitanju selekcija procesa između različitih redova, primenjuje se prioritetna šema sa pretpražnjenjem
- pored apsolutnog prioriteta, može se uvesti i RR između različitih redova
  - na primer, OS može dodeliti 80% procesorskog vremena interaktivnom redu i 20% vremena pozadinskom redu

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

Povratna sprega između redova

- u prethodnoj šemi procesi pripadaju svojim redovima od nastanka do završetka aktivnosti
- ponekad se zahteva fleksibilnost koja će omogućiti prelazak procesa iz jednog u drugi red
- procesi se na osnovu svojih karakteristika (vezanih za korišćenje procesora i U/I uređaja) razdvajaju u različite redove
  - proces koji zahteva puno procesorskog vremena ubacuje se u niskoprioritetni red
  - interaktivni i U/I dominantni procesi ubacuju se u visokoprioritetne redove
  - uvodi se vremenska komponenta koja sprečava zakucavanje
    - ako proces čeka isuviše dugo, prebacuje se u red višeg prioriteta

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

Raspoređivanje u višeprocesorskim sistemima

- ovo raspoređivanje je daleko složenije jer zahteva ozbiljnu sinhronizaciju procesora
- pretpostavićemo da su procesori funkcionalno identični, tj. da svaki procesor može da izvršava bilo koji proces iz reda čekanja
- u tom slučaju može se
  - obezbediti poseban red čekanja za svaki procesor
    - ovde se može pojaviti problem da jedan procesor može biti potpuno besposlen ukoliko je odgovarajući red prazan a drugi procesor gotovo potpuno opterećen
  - konstruisati zajednički red čekanja

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

Raspoređivanje u višeprocesorskim sistemima

- u slučaju zajedničkog reda čekanja OS može koristiti više tehnika za raspoređivanje
  - simetrično raspoređivanje
    - svaki procesor ispituje zajednički red i bira procese koje će izvršavati
    - opasnost postoji prilikom pristupa zajedničkim podacima
    - takođe, dva procesora ne smeju da preuzmu isti proces
  - asimetrično raspoređivanje
    - jedan procesor, koji je proglašen glavnim, određuje koje će procese izvršavati drugi procesi (međuprocesorski *master-slave* odnos)
    - master procesor izvršava kernelski kôd i U/I operacije
    - ostali procesori izvršavaju korisnički kôd koji im je dodelio master procesor
    - nije efikasno jer sve U/I operacije izvršava samo jedan procesor

3.3 - Raspoređivanje u više redova čekanja, u višeprocesorskim sistemima i u realnom vremenu

Raspoređivanje u realnom vremenu

- OS koji rade u realnom vremenu dele se na dve klase
- čvrsti OS (*hard*)
  - zahteva se izvršavanje važnih poslova u garantovanom vremenskom roku, tj. intervalu
  - u ovim OS proces se stvara sa informacijom za koje vreme mora biti obavljen ili za koje vreme mora da izvrši U/I operaciju
  - raspoređivač mora da obezbedi procesu da se izvrši u predviđenom roku ii da ga odbaci
- meki OS (*soft*)
  - manje restriktivni
  - realni procesi dobijaju veći prioritet izvršavanja i on se ne menja u vremenu
  - kod ovih sistema je važno da dsipečersko kašnjenje mora da bude veoma malo