Raspoređivanje procesa i dodela procesora

Na jednoprocesorskim sistemima, u jednom trenutku izvršava se samo jedan proces, dok svi drugi procesi moraju sačekati da se procesor oslobodi i da dođu na red. U višeprocesorskoj arhitekturi, veći broj procesa može se izvršavati istovremeno, tj paralelno (po jedan na svakom procesoru). Osnova tehnike multiprogramiranja je naizmenična dodela procesora većem broju aktivnih procesa. Na taj način se obezbeđuje maksimalno iskorišćenje procesora, jer u svakom trenutku postoji proces koji će se izvršavati (u krajnjem slučaju to može biti i proces koji ne radi ništa – *idle* proces). U višeprocesnim operativnim sistemima, planer poslova niskog nivoa (dispečer), na osnovu nekog algoritma bira proces iz reda čekanja na procesor i dodeljuje mu procesor na korišćenje.

Uvodne napomene

Ideja multiprogramiranja je relativno 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 neki događaj, npr, završetak ulazno-izlazne komande. Na prostim sistemima, procesor ne radi ništa dok proces čeka. Na kvalitetnijim sistemima , memorija se puni većim brojem procesa, tako da aktivnom procesu koji mora da čeka, operativni sistem oduzima procesor i dodeljuje ga nekom drugom procesu.

Dodela procesora po nekom algoritmu jedna je od osnovnih funkacija operativnih sistema. Sličan princip se primenjuje i na druge resurse, kao što su ulazno-izlazni uređaji: veći broj zahteva se smešta u red čekanja, a zatim se raspoređuje po nekom kriterijumu.

Raspoređivanje s pretpražnjenjem

Procesor se dodeljuje drugom procesu pod sledećim okolnostima:

- Kada proces pređe u stanje čekanja na resurs (čeka na završetak U/I operacije koju je inicirao)
- Kada proces roditelj čeka proces dete da završi aktivnosti
- Prilikom tranzicije RUN-STOP, tj kada tekući proces završi aktivnosti
- Prilikom tranzicije RUN-READY
- Prilikom tranzicije WAIT-READY (u slučaju da je proces koji je napustio stanje WAIT i prešao u stanje READY višeg prioriteta od procesa koji se trenutno izvršava)

U slučaju ***raspoređivanja bez pretpražnjenja***, procesor se može oduzeti samo od procesa koji je završio svoje aktivnosti ili čeka na resurs. Pri ***raspoređivanju s pretpražnjenjem (preemptive scheduling)***, procesor se može oduzeti procesu koji nije završio svoje aktivnosti i nije blokiran.

Jedan od fundamentalnih hardverskih komponenata neophodna za pretpražnjenje jeste ***tajmer***, koji periodično postavlja prekidni signal (prekidanje procesa se u literaturi pominje i kao pražnjenje). Prednosti ovakvog raspoređivanja su da se procesor može jako brzo dodeliti procesu višeg prioriteta koji zahteva izvršenje. Po pravilu, da ne bi nastao haos, proces koji je pomoću sistemskog poziva prešao u režim jezgra ne može se prekinuti sve dok je u tom režimu. Ovo je karakteristično za UNIX, ali i za većinu drugih operativnih sistema. Problemi koji se mogu javiti pri raspoređivanju s pretpražnjenjem nastaju zbog moguće nekonsistentnosti zajedničkih podataka (proces koji je otpočeo ažuriranje podataka je prekinut, a kontrola je dodeljena drugom procesu koji koristi iste podatke).

Kriterijumi dodele procesora

Većina algoritama za raspoređivanje procesa optimizovana je prema nekom od sledećih kriterijuma:

- ***Iskorišćenje procesora (CPU utilization)*** – po ovom kriterijumu procesor treba da bude stalno zauzet, tj da radi bez prestanka. Iskorišćenje procesora varira od 0% do 100%. Na realnim sistemima, iskorišćenje ide od 40% na umereno opterećenim sistemima, do 90% na veoma opterećenim sistemima

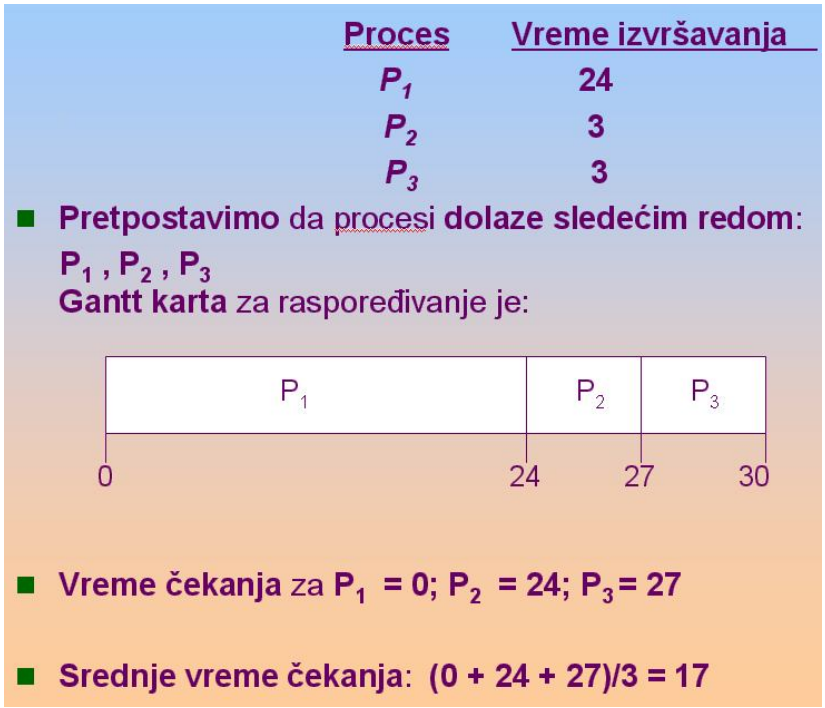
- ***Propusna moć (throughput)*** **–** definiše se kao broj procesa izvršen u jedinici vremena
- ***Vreme potrebno za kompletiranje procesa (turnaround time)***- ukupna količina vremena za koje se izvrši pojedinačni proces. Ovo vreme se računa od trenutka nastanka do završetka procesa, a uključuje i vreme potrebno da proces uđe u red čekanja, vreme provedeno u redu čekanja, vreme korišćenja procesora i izvršenja ulazno-izlaznih operacija
- ***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 redu čekanja, a na ovo vreme direktno utiču algoritmi za raspoređivanje
- ***Vreme odziva (response time)*** **–** vreme za koje se nakon slanja zahteva pojave prvi rezultati izvršenja procesa. Ovo vreme je veoma bitno u interaktivnim sistemima – proces koji otpočne rad, treba relativno brzo da prikaže početne rezultate na ekranu, zatim da ponovo izvrši neka izračunavanja ili ulazno-izlaznu operaciju, a onda opet prikaže rezultate i tako redom.

Algoritmi za dodelu procesora

First Come, First Served

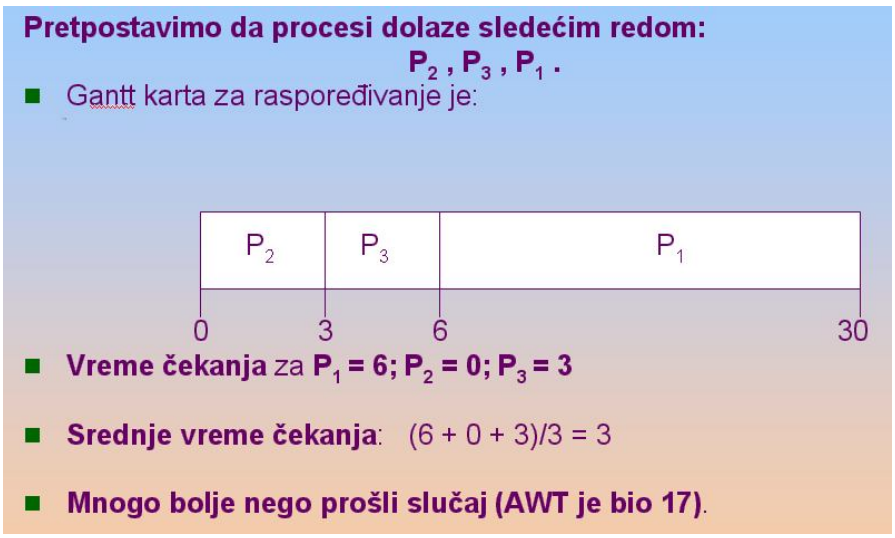
FCFS (prvi došao, prvi uslužen) najprostiji je algoritam za raspoređivanje procesora – procesi dobijaju procesor onim redom kojim su pristizali u red čekanja. Red čekanja funkcioniše po standardnom FIFO principu: kontrolni blok procesa koji je ušao u red čekanja na procesor stavlja se na kraj liste, a procesor se uvek dodeljuje onom procesu koji je na početku liste.

Srednje vreme čekanja (*waiting time)* za ovaj algoritam je vrlo dugačko. Uzmimo primer tri procesa P_1, P_2, P_3 čija su vremena izvršavanja 24, 3, 3 ms. Pretpostavimo da procesi nailaze u sledećem poretku: P_1, P_2, P_3. Vreme čekanja za proces P_1 iznosi 0, za P_2 iznosi 24, a za P_3 27 ms, a srednje vreme čekanja je (0+24+27)/3=17 ms.



Grantov dijagram aktivnosti 1(FCFS)

Ukoliko se promeni redosled nailaska procesa, situacija se drastično menja. Vreme čekanja za procese P_1, P_2, P_3 iznose 6, 0 i 3 ms, a srednje vreme čekanja je t_w=(6+0+3)/3=3ms.



Grantov dijagram aktivnosti br 2 (FCFS)

Na osnovu ovih primera zapaže se da srednje vreme čekanja zavisi kako od ***dužine trajanja procesa*** tako i od ***sekvence njihovog nailaska u sistem***. Kod algoritma FCFS moguća je pojava ***konvoj efekta*** – svi procesi čekaju da se završi jedan proces koji dugo traje. ***FCFS se obično izvodi bez pretpražnjenja***, tj bez prekidanja procesa, jer ne poštuje prioritete procesa već samo vreme dolaska u red čekanja. To znači da se proces koji je dobio procesorsko vreme izvršava u potpunosti ili do trenutka blokiranja zbog čekanja na ulazno-izlaznu operaciju. Zbog toga je FCFS krajnje nepodesan za interaktivne sisteme.

Shortest Job First

Suština algoritma ***Shortest Job First*** *(prvo najkraći posao)* jeste u sledećem: za sve procese u redu čekanja procenjuje se vreme potrebno za izvršenje, a ***procesor se dodeljuje onom procesu kome treba najmanje vremena za izvršenje***, tj procesu koji bi najmanje trajao. Na one procese iz reda koji imaju ***ista procenjena vremena izvršavanja , primenjuje se algoritam FCFS.***

Osnovni nedostatak algoritma SJF leži u tome što sistem ne može unapred znati koliko će odgovarajući procesi trajati tj ne može tačno proceniti trajanje celog procesa. Predikcija trajanja procesa zasniva se na činjenici da se proces sastoji od više ciklusa korišćenja procesora (*CPU burst)*, koji se prekidaju ulazno-izlaznim operacijama. Trajanje sledećeg ciklusa korišćenja procesora procenjuje se na osnovu trajanja prethodnih ciklusa korišćenja procesora. Zato se algoritam preciznije može nazvati i najkraći sledeći CPU ciklus (*shortest next CPU burst)*

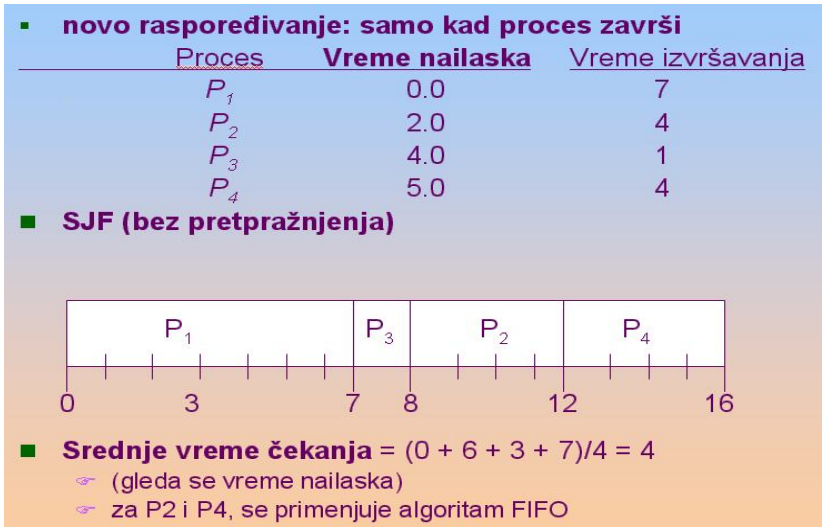
Postoje dve varijante algoritma SJF: ***s pretpražnjenjem (preemptive)*** ili ***bez pretpražnjenja (non-preemptive)***. Ove dve varijante se ponašaju različito kada se nov proces pojavi u redu čekanja dok se tekući proces izvršava. Varijanta SJF ***bez pretraživanja uvek će završiti tekući proces***, bez obzira na to kakav se novi proces pojavio u redu čekanja. U varijanti ***SJF s pretpražnjenjem, ukoliko je vreme potrebno za izvršenje novog procesa kraće od vremena potrebnog za završetak aktivnosti tekućeg procesa, procesor će biti dodeljen novom procesu.***

Za SJF je veoma bitno znati dve informacije: vreme nailaska procesa u red čekanja (*arrival time)* i vreme potrebno za izvršenje procesa (*CPU burst time)*. Za FCFS je bitan jedan parametar, a to je vreme nailaska procesa u red.

Prvi primer je algoritam bez pretpražnjenja, što znači da će se svaki proces izvršiti do kraja. U nekom trenutku, u redu čekanja se nalazi samo proces P1 čije izvršenje traje 7 vremenskih jedinica. U toku izvršavanja procesa P_1, u red stižu preostala tri procesa. Kada proces P_1 završi svoje aktivnosti , algoritam SJF odabira procec

P_3 zato što je najkraći. Nakon završetka aktivnosti procesa P_3 (t=8), u redu čekanja ostaju dva procesa s jednakim vremenima izvršavanja , tako da se na njih primenjuje algoritam FCFS koji za dodelu procesora najpre odabira proces P_2, a zatim P_4. Vremena čekanja za pojedine procese su:

- $t_w(P_1)=0$
- $t_w(P_2)=8-2=6$ * vrednost 2 predstavlja vreme nailaska u sistem, a u trenutku 8 je počeo da se izvršava
- $t_w(P_3)=7-4=3$
- $t_w(P_4)=12-5=7$
- $t_w=(0+6+3+7)/4=4$ – srednje vreme čekanja



SJF bez pretpražnjenja

Drugi primer je SJF s pretpražnjenjem. Pretpostavljamo da je situacija identična prethodnoj: na sistemu se u trenucima 0, 2, 4, 5 formiraju četiri procesa čija su vremena izvršenja 7, 4, 1 i 4 vremenske jedinice. ***Algoritam je s pretpražnjenjem, tako da se u slučaju pojave novog procesa proverava koliko je vremena ostalo tekućem procesu, a koliko je potrebno novom za izvršenje***. Po potrebi procesor se prazni i ustupa novom kraćem procesu.

U prvom trenutku, u redu čekanja se nalazi samo proces P_1 (vreme izvršavanja 7 vremenskih jedinica), koji počinje da se izvršava.

U trenutku t=2:

- pojavljuje se proces P_2, čije je vreme izvršenja 4 vremenske jedinice, što je kraće od vremena preostalog za izvršenje procesa P1 (5 vremenskih jedinica).
- Zbog toga se procesor u trenutku t=2 prazni i predaje procesu P2.

U trenutku t=4, pojavljuje se nov proces P3. Opet se analiziraju preostala vremena:

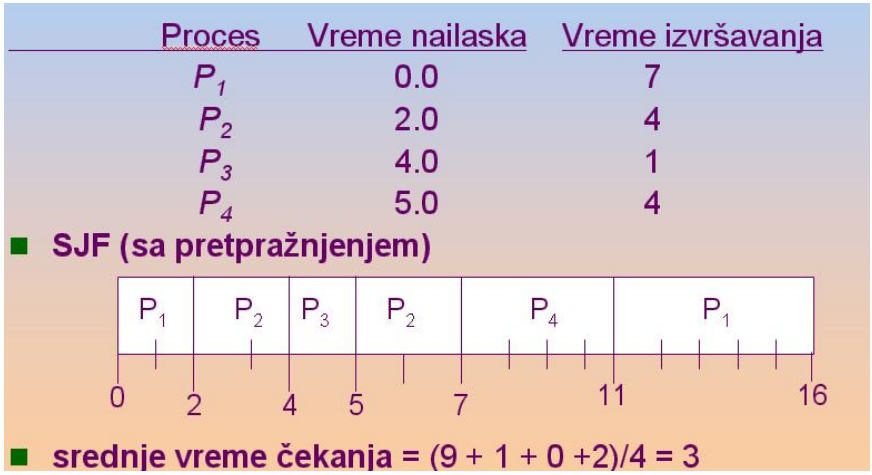
- P1 zahteva još 5 ciklusa
- P2 još 2 ciklusa
- P3 – kao nov proces samo 1 ciklus
- što će izazvati novo pražnjenje nakon koga P3 dobija procesor.

U trenutku t=5, P3 je završio aktivnosti, ali je istovremeno naišao proces P4. Dalje raspoređivanje se obavlja na sledeći način:

- u t=5, P1 ima zaostalo vreme od 5 ciklusa, P2 od 2 ciklusa, a nov proces P4 zahteva 4 ciklusa, tako da će procesor biti dodeljen procesu P2
- u trenutku t=7, proces P2 završava aktivnosti , pa nastupa novo raspoređivanje: P1 ima zaostalo vreme od 5 ciklusa, tako da se procesor dodeljuje procesu P4, koji zahteva 4 ciklusa. P4 koristi procesor do završetka svojih aktivnosti, posle čega se on – u trenutku t=11, dodeljuje procesu P1.

Vreme čekanja za pojedinačne procese iznose:

- $tw(P_1)=11-2=9$
- $tw(P_2)=5-4=1$
- $tw(P_3)=0$
- $tw(P_4)=7-5=2$
- $tw=(9+1+0+2)/4=3$ – srednje vreme čekanja



SJF sa pretpražnjenjem

Raspoređivanje na osnovu prioriteta procesa

U prioritetno orijentisanim algoritmima, svakom procesu se dodeljuje prioritet, posle čega algoritam bira proces s najvećim prioritetom. Prioritet je celobrojna vrednost, a najčešće se koristi konvencija po kojoj manji broj znači veći prioritet i obrnuto (vrednost 0 predstavlja maksimalni prioritet). Ako su procesi jednakog prioriteta, odluka se donisi po principu FCFS.

SJF je specijalan slučaj prioritetno orijentisanih algoritama za raspoređivanje: prioritet je obrnuto proporcionalan trajanju CPU ciklusa procesa. To znači da duži procesi imaju manji prioritet i obrnuto.

Prioritetno orijentisani algoritmi mogu biti realizovani ***s pretpražnjenjem (priority preemmptive scheduling) – tekući proces u tom slučaju prekida izvršenje ukoliko se pojavi proces većeg prioriteta (provera se odvija uvek kada se pojavi novi proces)***. Ukoliko je algoritam realizovan ***bez pretpražnjenja (priority non-preemptive sheduling), proces koji je dobio kontrolu ne prekida izvršenje iako se pojavio proces većeg prioriteta.*** Prioriteti mogu biti:

- interno definisani, na osnovu resursa koje proces zahteva i odnosa procesora i ulazno-izlanih aktivnosti prisutnih u procesu
- eksterno dodeljeni (programer može favorizovati određene procese po nekim eksternim kriterijumima)

Glavni problem prioritetno orijentisnih algoritama je blokiranje niskoprioritetnih procesa. Procesi niskog prioriteta mogu veoma dugo (neograničeno) da čekaju na procesor. Npr, ako na jako opterećenim sistemima novi procesi neprestano pristižu, pa se može dogoditi da procesi visokog prioriteta potpuno blokiraju niskoprioritetne procese. Problem zakucavanja niskoprioritetnih procesa razrešava se tako što im se prioritet povećava u skladu sa s vremenom provedenim u redu čekanja na procesor. Tako se rezultujući prioritet formira na osnovu:

- početnog prioriteta, koji proces dobija kada uđe u red čekanja na procesor
- vremena provedenog u redu čekanja (aging)

Kako s vremenom provedenim u redu čekanja rezultujući prioritet raste, niskoprioritetni procesi dobijaju veću šansu da dodelu procesora.

Ako prioritete dinamički određuje operativni sistem, dobro je uzeti u obzir sledeća razmatranja:

- procesima koji intenzivno koriste ulazno-izlazne uređaje treba dodeliti viši prioritet jer su ovakvi procesi često blokirani, a procesor ima treba u kraćim intervalima
- procesima koji intenzivno koriste procesor i radnu memoriju treba dodeliti niži prioritet; procesor se može bolje iskoristiti ukoliko se najpre dodeli procesima koji intenzivno koriste ukazno-izlazne uređaje; ovi procesi će procesor koristiti relativno kratko, a zatim se blokirati, posle čega se nesmetano mogu izvršavati procesi iz grupe procesa koji intenzivno koriste procesor; u obrnutom slučaju, U/I procesi dugo bi čekali na procesor i zauzeli bi mesto u radnoj memoriji,a ulazno-izlazni sistem bi ostao prilično besposlen

Round Robin (RR)

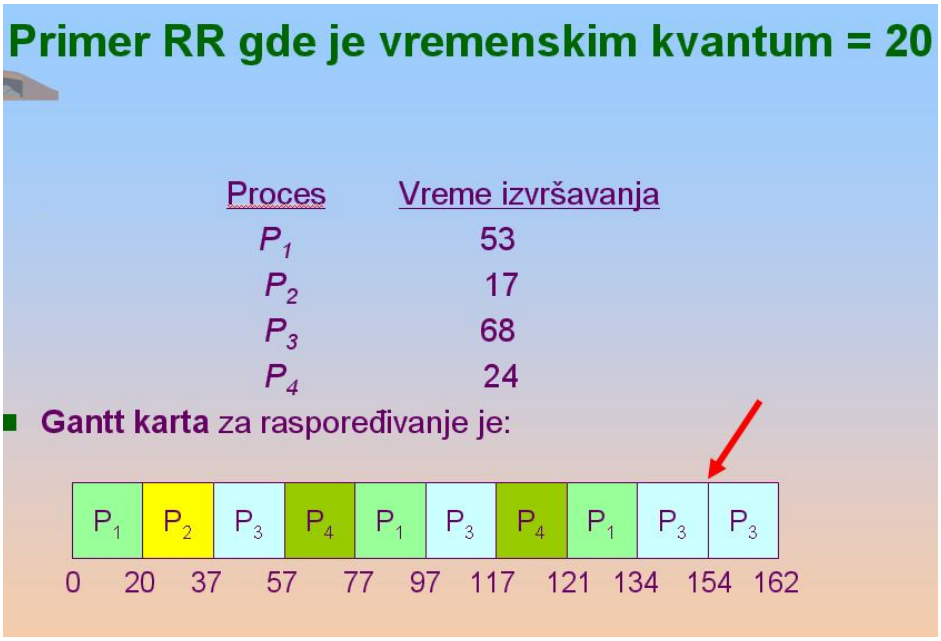
Algoritam Round Robin namenski je projektovan za interaktivne sisteme. Round Robin se može posmatrati kao algoritam FCFS sa pretpraženjenjem, tj ***sa pravilnim vremenskim prebacivanjima između procesa***. RR funkcioniše na sledeći način: ***najpre se definiše kratak vremenski interval*** (vremenski kvantum, ***time slice***), reda veličine od 10 do 100 ms i ***kružni red čekanja na procesor***. Svaki prispeli proces ubacuje se na kraj liste. Od sistema se očekuje da po isteku vremeskog kvantuma generiše prekidni signal, nakon čega se procesu oduzima kontrola nad procesorom i predaje sledećem procesu iz kružne liste. U okviru jednog vremenskog kvantuma moguće su sledeće situacije:

- proces je završio aktivnost pre isteka vremenskog kvantuma – proces tada oslobađa procesor, a algoritam RR uklanja proces iz kružne liste i predaje kontrolu sledećem procesu
- proces nije završio aktivnosti, ali mora da prekine izvršenje kada mu istekne vremenski kvantum; algoritam RR postavlja prekinuti proces na kraj reda čekanja, a kontrolu predaje sledećem procesu iz liste
- proces se blokirao zbog čekanja na ulazno-izlazne operacije; blokirani proces oslobađa procesor, a algoritam 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 prema procesima – ***svi procesi dobijaju procesor na korišćenje, ravnomerno i ravnopravno***. Ako imamo n procesa u redu čekanja, svakom procesu pripada 1/n procesorskog vremena. Ukoliko je vremenski kvantum dužine q, nijedan proces u stanju READY neće čekati na sledeću dodelu procesora više od (n-1)q vremena. Međutim, srednje vreme odziva algoritma RR po pravilu je jako dugo. Treba imati u vidu da ***prebacivanje konteksta između dva procesa unosi premašenje (overhead) – u ovom slučaju kašnjenje***. Ovo premašenje je jedan od glavnih činilaca prilikom izbora veličine kvantuma. Što je kvantum manji, procesor će biti ravnomernije raspodeljen, ali će i prebacivanje konteksta između procesa biti češće, tako da će premašenje sistema biti veće. Ukoliko je kvantum veći, algoritam RR konvergira ka algoritmu FCFS.

U nastavku je dat primer raspoređivanja procesora algoritmom Round Robin s kvantumom od 20 vremenskih jedinica. Pretpostavimo da su četiri procesa P_1, P_2, P_3, P_4 došla u red skoro u istom trenutku i da je formirana kružna lista P_1->P_2->P_3->P_4. Neka su vremena izvršenja ovih prcesa 53, 17, 68 i 24 vremenske jedinice.

- Procesor najpre dobija proces P_1
- Po isticanju kvantuma, procesor mu se oduzima i predaje sledećem procesu iz kružne liste, a to je P_2, dok P1 odlazi na kraj kružne liste
- Proceso P_2 će se završiti pre isteka kavntuma, tako da će se u trenutku t=37 procesor dodeliti sledećem procesu iz kružne liste, procesu P_3, a proces P_2 se kao završeni proces briše iz liste
- Procesu P_3 su potrebna četiri kvantuma, a procesu P_4 dva
- U t=57, P_3 završava svoj kvantum i ustupa mesto procesu P_4, a potom P_4 ustupa mesto P_1, pa ovaj P_3 i tako redom
- U t=121 proces P_4 završava svoje aktivnosti i ispada iz liste
- U t=134 završava proces P_1
- Poslednja dva kvantuma pripadaju procesu P_3



Gantov dijagram aktivnosti za RR

Generalno gledano, RR ima najbolje vreme odziva (response time), a veće srednje vreme završetka procesa (turnaround time) u odnosu na SJF, čak i u slučaju zanemarivanja vremena potrebnog za prebacivanje konteksta procesa.

Performanse algoritma RR zavise od veličine vremenskog kvantuma. U slučaju veoma malih vremenskih kvantuma, algoritam RR se naziva deljenje procesora (CPU sharing), jer tada svaki proces radi brzinom jednakom 1/n brzine realnog procesora. Ova brzina predstavlja idealan slučaj – pretpostavlja se da dispečer trenutno prabacuje kontekst.

Realno, dispečer je realizovan softverski i izvršava se kao zaseban proces, pa unosi kašnjenje pri svakom prebacivanju konteksta. Realna brzina izvršavanja svakog procesa manja je od 1/n brzine realnog procesora i opada sa porastom učestalosti prebacivanja konteksta.

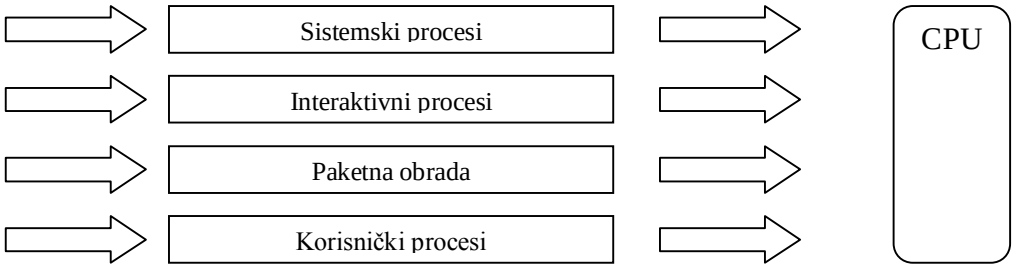
Raspoređivanje u više redova čekanja

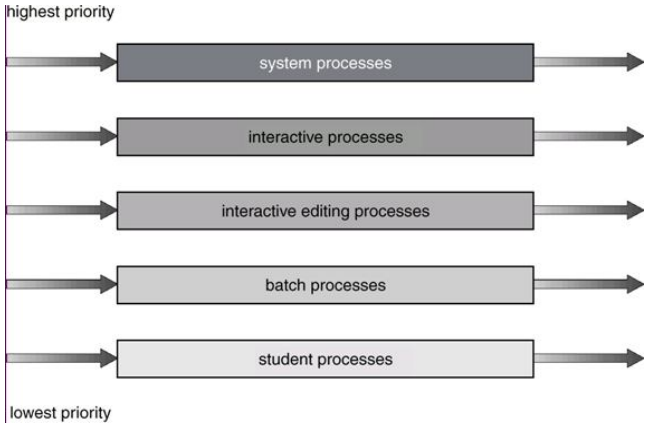
Ova klasa algoritama za raspoređivanje odnosi se na sisteme i situacije u kojima ***se procesi mogu klasifikovati u različite grupe***. Npr, procesi se mogu podeliti na interaktivne procese, tj procese koji rade u prvom planu (foreground) i procese koji rade u pozadini (background). Ove dve klase procesa imaju različite zahteve u pogledu vremena odziva i zbog toga se koriste različiti algoritmi za raspoređivanje procesora. Po pravilu, 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 svojih osobina, kao što su veličina memorije, prioritet i tip. ***Svaki red čekanja ima poseban nezavisan algoritam za raspoređivanje koji odgovara njegovim potrebama.*** Npr, može se napraviti red čekanja ua interaktivne procese koji rade u pozadini, koji se opslužuje po algoritmu FCFS.

U slučaju više redova čekanja, za izbor između procesa smeštenih u različitim redovima, primenjuje se prioritetna šema s pretpražnjenjem. Svaki red čekanja ima apsolutni prioritet u odnosu na niže redove. Procesi iz niže klase moraju da sačekaju ili da daju prednost procesima iz viših klasa, a ako su dobili procesor, oslobodiće ga ukoliko se pojavi proces iz više klase.

Pored apsolutnog prioriteta, može se uvesti i RR tehnika između različitih redova, što znači da svaki red dobija procesor na neko vreme, a u tom vremenu biraju se procesi iz tog reda, po algoritmu koji je definisan za taj red. Npr, u slučaju da imamo dva reda (interaktivni red i red za procese koji rade u pozadini), sistem može dodeliti 80% procesorskog vremena interaktivnom redu i 20% vremena pozadinskom redu.





Raspoređivanje u više procesorskih redova

Raspoređivanje u višeprocesorskim sistemima

Raspoređivanje procesa u višeprocesorskim sistemima daleko je kompleksnije 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 ili konstruisati zajednički red čekanja.

U prvom slučaju, jedan procesor može biti potpuno besposlen ukoliko je odgovarajući procesorski red prazan, a drugi procesor moe biti maksimalno opterećen. Zato se umesto zasebnih redova formira zajednički red čekanja, koji obezbeđuje ravnomerno opterećenje svih procesora.

U slučaju zajedničkog reda čekanja, operativni sistem može koristiti više tehnika za raspoređivanje:

1. svaki procesor ispituje zajednički red i odabira procese koje će izvršavati. Opasnost postoji prilikom pristupa zajedničkim podacima. Takođe, dva procesora ne smeju da preuzmu isti proces. Ovaj način multiprocesiranja poznat je kao ***simetrično multiprocesiranje***
2. jedan procesor, koji je proglašen glavnim, određuje koje će procese izvršavati drugi procesori (međuprocesorski odnos nadređen – podređen, master/slave); ovo multiprocesiranje naziva se ***asimetričnim***: nadređeni procesor izvršava kôd jezgra i ulazno-izlazne operacije, dok ostali procesori izvršavaju korisnički kôd, koji im definiše nadređeni procesor; navedeni sistemi nisu efikasni jer sve ulazno-izlazne operacije izvršava samo jedan procesor