

# Operativni sistemi 1

## Raspoređivanje procesa i dodela procesora

*Nemanja Maček*

# Sadržaj

- Uvodne napomene
- Algoritmi za raspoređivanje procesa
  - First Come, First Served
  - Shortest Job First
  - Raspoređivanje na osnovu prioriteta
  - Round Robin
- Raspoređivanje u više procesorskih redova
- Raspoređivanje u višeprocesorskoj okolini

# Uvodne napomene

- 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 sistemima procesor ne radi ništa dok proces čeka.
- Na višeprocesnim sistemima, memorija se puni većim brojem procesa.
  - Kad aktivni proces mora da čeka, OS mu oduzima procesor i dodeljuje ga nekom drugom procesu.
- Dodela procesora po nekom algoritmu je jedna od fundamentalnih funkcija OSa!
- Sličan princip se primenjuje i za druge resurse, kao što su U/I uređaji.
  - Veći broj zahteva se smešta u red za čekanje, a zatim se raspoređuju po nekom kriterijumu.

# Uvodne napomene

- Procesor se dodeljuje drugom procesu pod sledećim okolnostima:
  - kada proces pređe u stanje čekanja na resurs (1),
    - npr. čeka na završetak I/O operacije koju je inicirao,
  - kada proces roditelj čeka proces dete da završi aktivnosti (2),
  - prilikom tranzicije RUN-STOP, odnosno kada tekući proces završi aktivnosti (3),
  - prilikom tranzicije RUN-READY (4).
- Za slučaj (1), (2) i (3) **novi proces** mora biti izabran iz reda čekanja na procesor i aktiviran.
- **Raspoređivanje bez predpražnjenja** (engl. *nonpreemptive scheduling*).
  - Rapoređivanje bez prekidanja izvršenja tekućeg procesa.
  - Procesor se može oduzeti samo od procesa koji je završio svoje aktivnosti ili čeka na resurs.

# Uvodne napomene

- **Raspoređivanje sa predpražnjenjem** (engl. *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.
    - 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 kernela OSa.
  - Pravilo: da ne bi nastao haos, proces koji je pomoću sistemskog poziva prešao **u kernelski režim se ne može prekidati** sve dok je u tom režimu rada.

# Uvodne napomene

- **Kriterijumi dodele procesora**.
  - **Iskorišćenje procesora**.
    - Po ovom kriterijumu procesor treba da bude zauzet, odnosno da radi nešto bez prestanka.
  - **Propusna moć sistema** (engl. *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 osobinama.
  - **Vreme potrebno za kompletiranje procesa** (engl. *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, itd.

# Uvodne napomene

- **Kriterijumi dodele procesora**.
  - **Vreme čekanja** (engl. *waiting time*).
    - Ukupno vreme koje proces provede u redu čekanja na procesor (engl. *ready queue*).
    - Proces do kraja svog izvršavanja može više puta da bude u ovom redu čekanja kao i da na ovo vreme direktno utiču algoritmi za raspoređivanje.
  - **Vreme odziva** (engl. *response time*).
    - Vreme potrebno da se nakon slanja zahteva pojave prvi rezultati izvršenja procesa.
    - Ovo vreme je veoma bitno kod **interaktivnih sistema**.
      - Primer: da li želite da čekate na terminal na aerodromu više od 1h?
- Zavisno od namene sistema algoritmi se optimizuju za određene kriterijume, najmanje, najveće ili srednje vrednosti.
- Primer:
  - Za interaktivne sisteme je značajnije minimizirati najveće vreme odziva a ne srednje vreme.

# Uvodne napomene

- **Scheduling**.



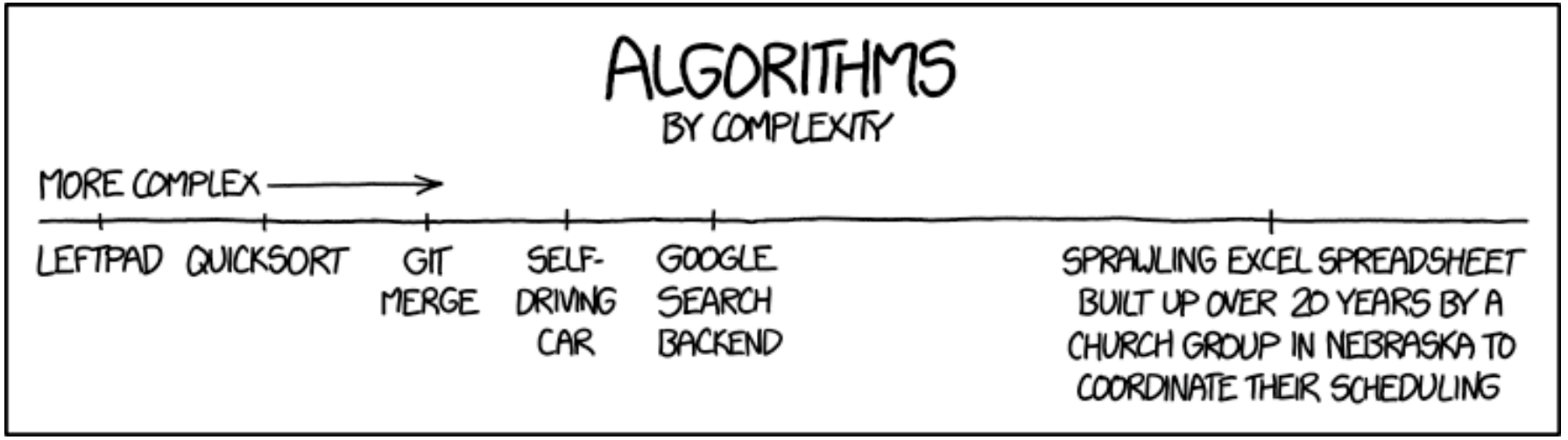

* preuzeto sa: http://www.cs.vu.nl/~frankh/dilbert/scheduling.gif

# Uvodne napomene

- **Ispitivanje algoritama**.
- Pitanje: kako odabrati algoritam za raspoređivanje procesa u nekom posebnom sistemu?
- Prvi problem: postavititi glavni kriterijum ili kombinaciju kriterijuma kao što su:
  - Maksimizovanje iskorišćenosti procesora uz ograničenje da max. vreme odziva bude 1 sec.
  - 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** (engl. *workload*) – može biti sintetičko ili realno.
  - Opterećenje se formira snimanjem svih procesa jednog realnog sistema u određenom periodu vremena u statističku datoteku (engl. *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 inforamcije o uspešnosti jednog algoritma na datom statističkom uzorku.

# Algoritmi za raspoređivanje procesa

- O složenosti algoritama.




* preuzeto sa stranice: https://www.explainxkcd.com/wiki/index.php/1667:_Algorithms

# First Come, First Served

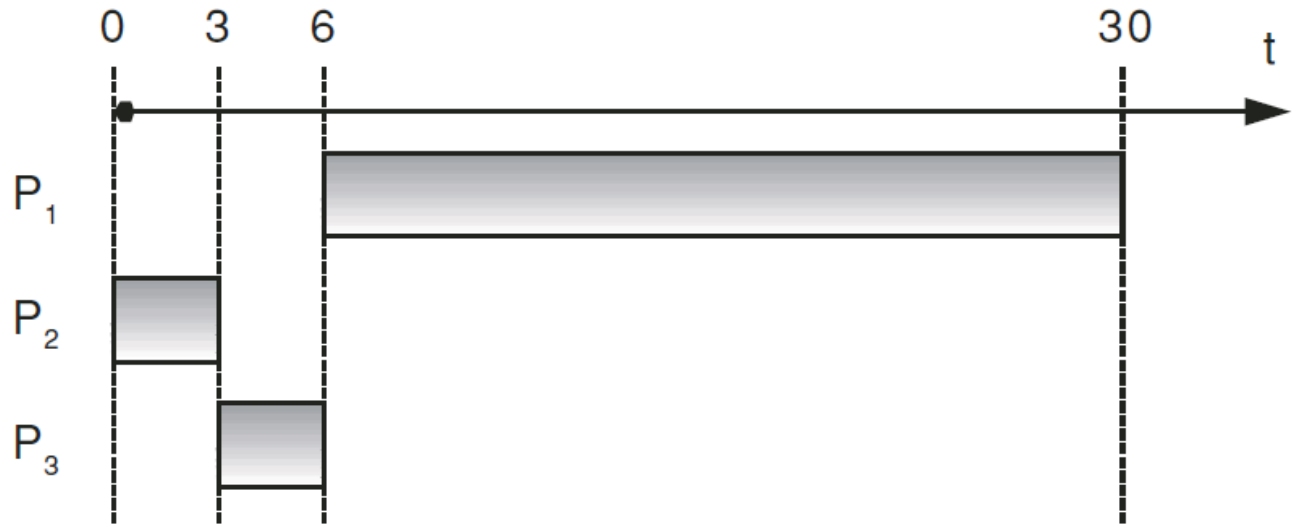
- **Prvi došao, prvi uslužen** (engl. *First Come*, *First Served*, FCFS) je najprostiji algoritam za
- raspoređivanje procesora.
- Procesi dobijaju procesor onim redom kojim su pristizali u red čekanja.
- Red čekanja funkcioniše po standardnom FIFO principu:
  - PCB procesa koji je ušao u red čekanja na procesor stavlja se na kraj liste.
  - Procesor 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.

# First Come, First Served

- Primer 1:
  - Procesi P1, P2 i P3 čija su vremena izvršavanja 24, 3 i 3msec nailaze u poretku P1, P2, P3.
  - Vremena čekanja (*waiting time*) za procese P1, P2 i P3 su 0, 24 i 27msec respektivno.
  - Srednje vreme čekanja $t_w$ = (0+24+27)/3 = 17msec.

# First Come, First Served

- Primer 2:
  - Promena redosleda nailaska procesa (P2, P3, P1).
  - Situacija se značajno menja.
  - Vremena čekanja za procese P1, P2 i P3 6, 0 i 3msec respektivno.
  - Srednje vreme čekanja je $t_w$ = (6+0+3)/3 = 3msec.



# 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 izvršava u potpunosti ili do trenutka blokiranja zbog čekanja na U/I operaciju.
  - Zbog toga je FCFS je krajnje nepodesan za interaktivne sisteme.

# Shortest Job First

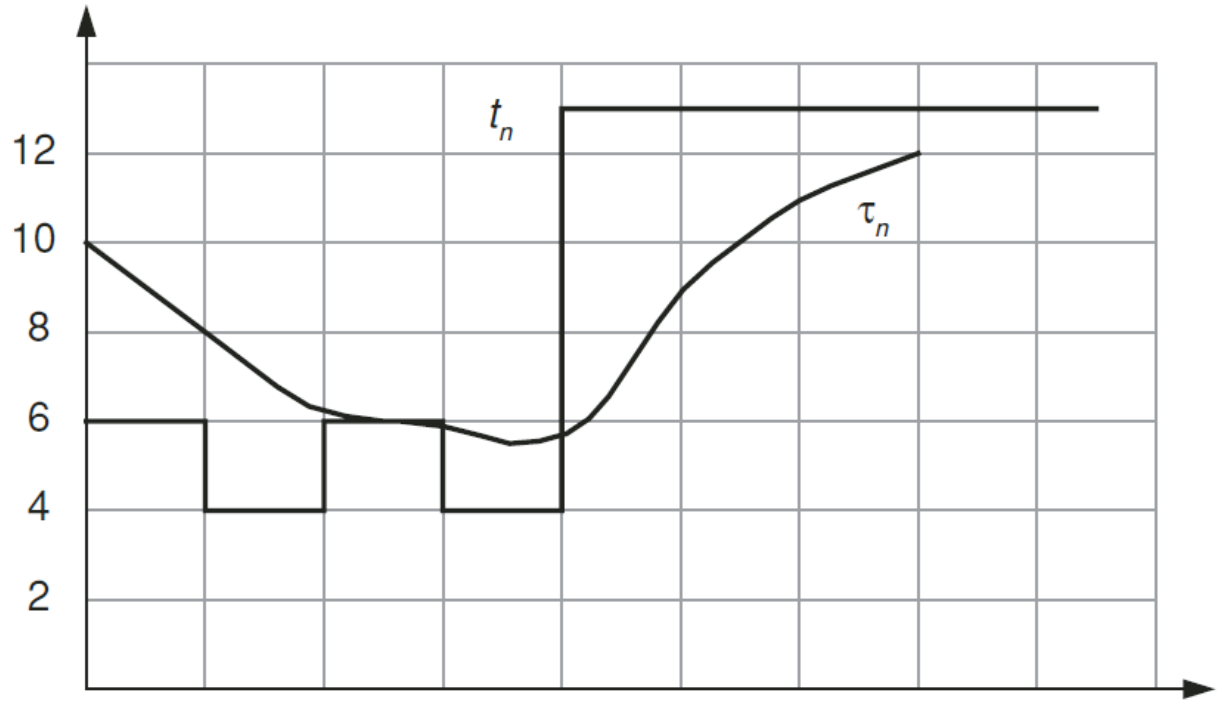
- Algoritam ***Shortest Job First*** (***SJF***) funkcioniše po principu: obavi prvo najkraći posao**.**
- Za sve procese u redu čekanja procenjuje se vreme potrebno za izvršavanje.
  - Procesor se dodeljuje onom  procesu kome treba najmanje vremena za izvršenje, odnosno procesu koji bi najkraće trajao.
  - Za 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 (engl. *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** (engl. *shortest next CPU burst*).

# 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:
  - $\tau_{n+1}$ je procenjena vrednost trajanja sledećeg CPU ciklusa
  - $t_n$ je realno trajanje prethodnog CPU ciklusa
  - $\tau_n$ je procenjena vrednost trajanja prethodnog CPU ciklusa
  - Koeficijent $\alpha$ je na intervalu $0 \leq \alpha \leq 1$.
    - Koeficijent $\alpha$ određuje relativnu težinu istorije procene u ondnosu na realno vreme poslednjeg poznatog ciklusa.
- Formula za procenu trajanja sledećeg ciklusa korišćenja procesora:
  - Kombinuje prethodnu vrednost $t_n$ i istoriju procene trajanja ciklusa koja je sublimisana u $\tau_n$
  - $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

# Shortest Job First

- Primer procene trajanja sledećeg ciklusa korišćenja procesora:
  - $a$=0,5, $\tau_0$=10 vremenskih jedinica, $t_0$=6, $t_1$=4, $t_2$=6, $t_3$=4, $t_4$=$t_5$=$t_6$=13 vremenskih jedinica.
  - Procenjena vremena su: $\tau_1$=8, $\tau_2$=6, $\tau_3$=6, $\tau_4$=5, $\tau_5$=9, $\tau_6$=11, $\tau_7$=12.



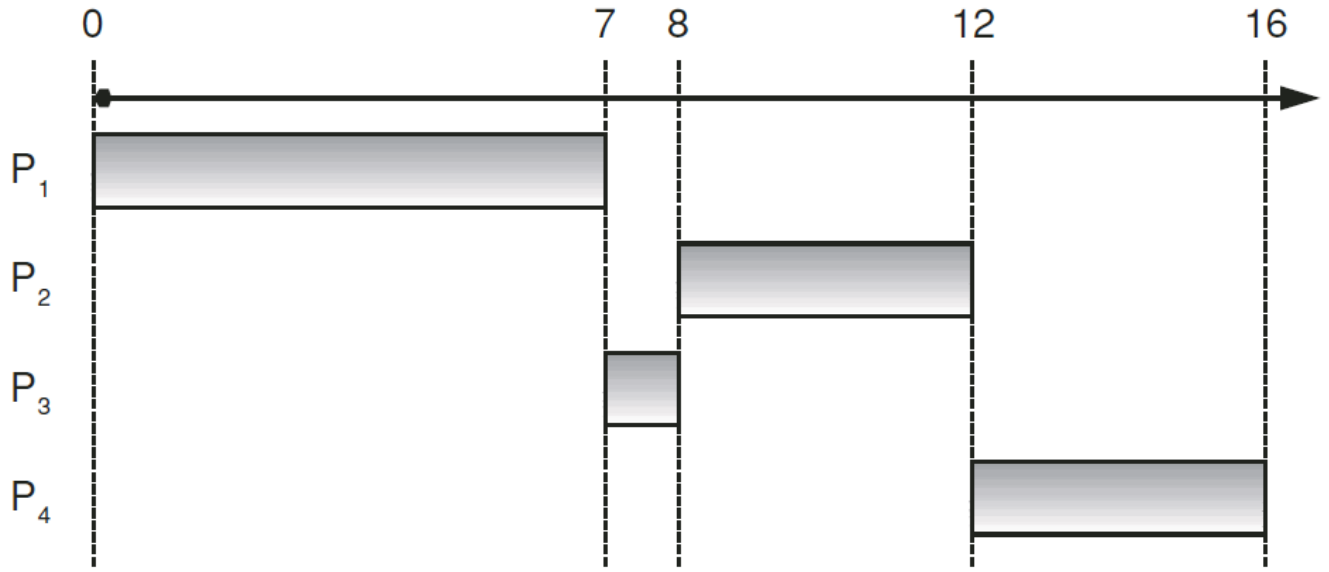
# Shortest Job First

- Postoje dve varijante SJF algoritma:
  - Bez pretpražnjenja (engl. *non-preemptive*)
    - Uvek završiti tekući proces bez obzira kakav se novi proces pojavio u redu čekanja.
  - Sa pretpražnjenjem (engl. *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 **raspređivanje sa najmanjim preostalim vremenom** (engl. *shortest-remaining time first*).
- Za SJF veoma je bitno znati dve informacije:
  - Vreme nailaska procesa u red čekanja (engl. *arrival time*)
  - Vreme potrebno za izvršenje procesa (engl. *CPU burst time*).
- Napomena: kod *FCFS* je bitan samo jedan parametar, a to je vreme nailaska procesa u red.

# Shortest Job First

- Primer: SJF bez pretpražnjenja.

| Proces | Vreme nailaska u sistem | Vreme izvršavanja |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 4 | 1 |
| $P_4$ | 5 | 4 |



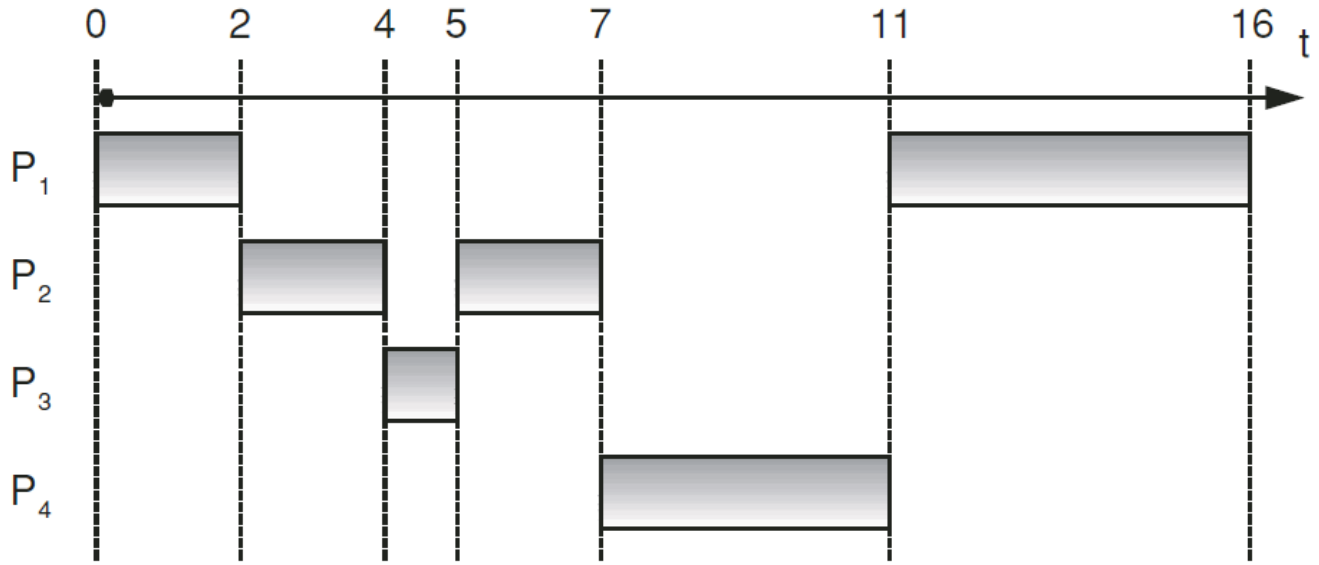
# Shortest Job First

- Primer: SJF bez pretpražnjenja.
  - U prvom trenutku u redu se nalazi samo proces P1 čije izvršenje traje 7 vremenskih jedinica.
  - U toku izvršavanja procesa P1 u red pristižu preostala tri procesa.
  - Kada proces P1 završi svoje aktivnosti *SJF* odabira proces P3 zato što je najkraći.
  - Nakon završetka aktivnosti procesa P3 ($t$=8) u redu ostaju dva procesa sa jednakim vremenima izvršavanja.
  - SJF na njih primenjuje *FCFS* algoritam koji odabira proces P2 a zatim P4.
  - Vremena čekanja za procese su:
    - $t_w$ (P1) = 0
    - $t_w$ (P2) = 8-2 = 6
    - $t_w$ (P3) = 7-4 = 3
    - $t_w$ (P4) = 12-5 = 7
  - Srednje vreme čekanja je $t_w$ = (0+6+3+7)/4 = 4.
  - U slučaju *FCFS* srednje vreme čekanja bi bilo $t_w$ = (0+5+7+7)/4 = 4,75.

# Shortest Job First

- Primer: SJF sa pretpražnjenjem (SRTF).

| Proces | Vreme nailaska u sistem | Vreme izvršavanja |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 4 | 1 |
| $P_4$ | 5 | 4 |



# Shortest Job First

- Primer: SJF sa pretpražnjenjem (SRTF).
  - U prvom trenutku u redu čekanja se nalazi samo proces P1 (vreme izvršavanja 7), koji počinje da se izvršava.
  - U trenutku t=2 pojavljuje se proces P2 (vreme izvršenja 4) što je manje 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 novi proces P3.
  - Opet se obavlja analiza preostalih vremena: P1 zahteva još 5, P2 još 2 ciklusa, a P3 kao novi proces samo 1 ciklus, što će izazvati novo pražnjenje nakon koga P3 dobija procesor …
- Vremena čekanja za procese su:
  - $t_w$ (P1) = 11-2 = 9
  - $t_w$ (P2) = 5-4 = 1
  - $t_w$ (P3) = 0
  - $t_w$ (P4) = 7-5 = 2
- Srednje vreme čekanja je $t_w$ = (9+1+0+2)/4 = 3.

# Raspoređivanje na osnovu prioriteta

- 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 raspoređivaje.
  - Prioritet je obrnuto proporcionalan trajanju sledećeg CPU ciklusa procesa.
  - To znači da **duži procesi imaju manji prioritet** i obrnuto.
- Prioritetno-orijentisani algoritmi mogu biti realizovani:
  - Sa pretpražnjenjem (engl. *priority preemptive scheduling*).
    - Tekući proces se prekida izvršenje ukoliko se pojavi proces većeg prioriteta (provera se odvija uvek kada se pojavi novi proces).
  - Bez pretpražnjenja (engl. *priority non-preemptive scheduling*).
    - Proces koji je dobio kontrolu ne prekida izvršenje bez obzira što se pojavo proces većeg prioriteta.

# Raspoređivanje na osnovu prioriteta

- Glavni problem: **blokiranje niskoprioritetnih procesa**.
  - Procesi niskog prioriteta mogu veoma dugo (neograničeno) da čekaju na procesor.
  - Primer rešenja problema zakucavanja niskoprioritetnih procesa: povećanje prioriteta sa vremenom provedenim u redu čekanja na procesor.
  - 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** (engl. *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 *I/O bound* procesi bi dugo čekali na procesor i zauzimali mesto u operativnoj memoriji a U/I podsistem bi ostao prilično besposlen.

# Round Robin

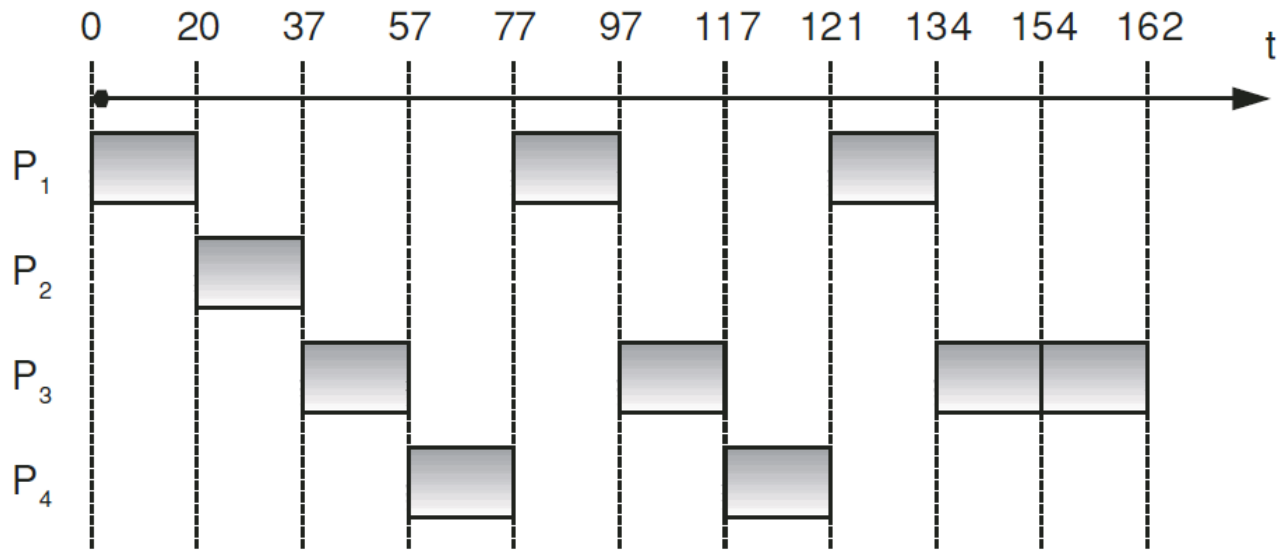
- Round Robin (*RR*) se može posmatrati kao **FCFS algoritam sa pretpražnjenjem**.
- Najpre se definiše **vremenski kvantum** (engl. *time slice*) i kružni red čekanja na procesor.
- Svaki prispeli proces ubacuje na kraj liste.
- Od sistema se očekuje da generiše prekidni signal po isteku vremenskog 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.
  - 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.

# Round Robin

- 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.
- Nedostaci:
  - Srednje vreme odziva *RR* algoritma je po pravilu jako dugo.
  - Prebacivanje konteksta između dva procesa unosi premašenje u vidu nekorisnog vremenskog kašnjenja.
  - 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, RR algoritam konvergira ka FCFS algoritmu.

# Round Robin

- Primer:
  - Kvantum: 20 vremenskih jedinica.
  - Četiri procesa (P1, P2, P3 i P4) su došla u red u skoro istom trenutku.
  - Vremena izvršavanja ovih procesa su 53, 17, 68 i 24 vremenskih jedinica, respektivno.

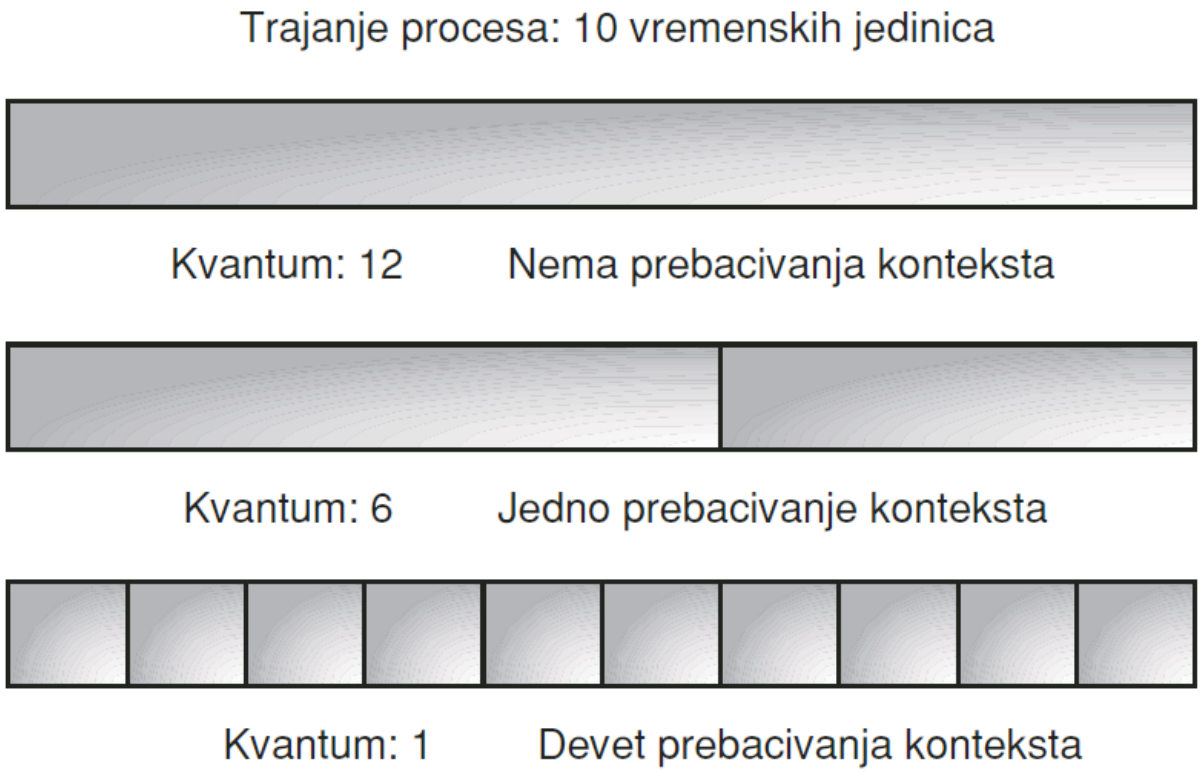


# Round Robin

- 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 (pretpostavlja se da dispečer obavlja prebacivanje konteksta trenutno).
- Realno:
  - 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.

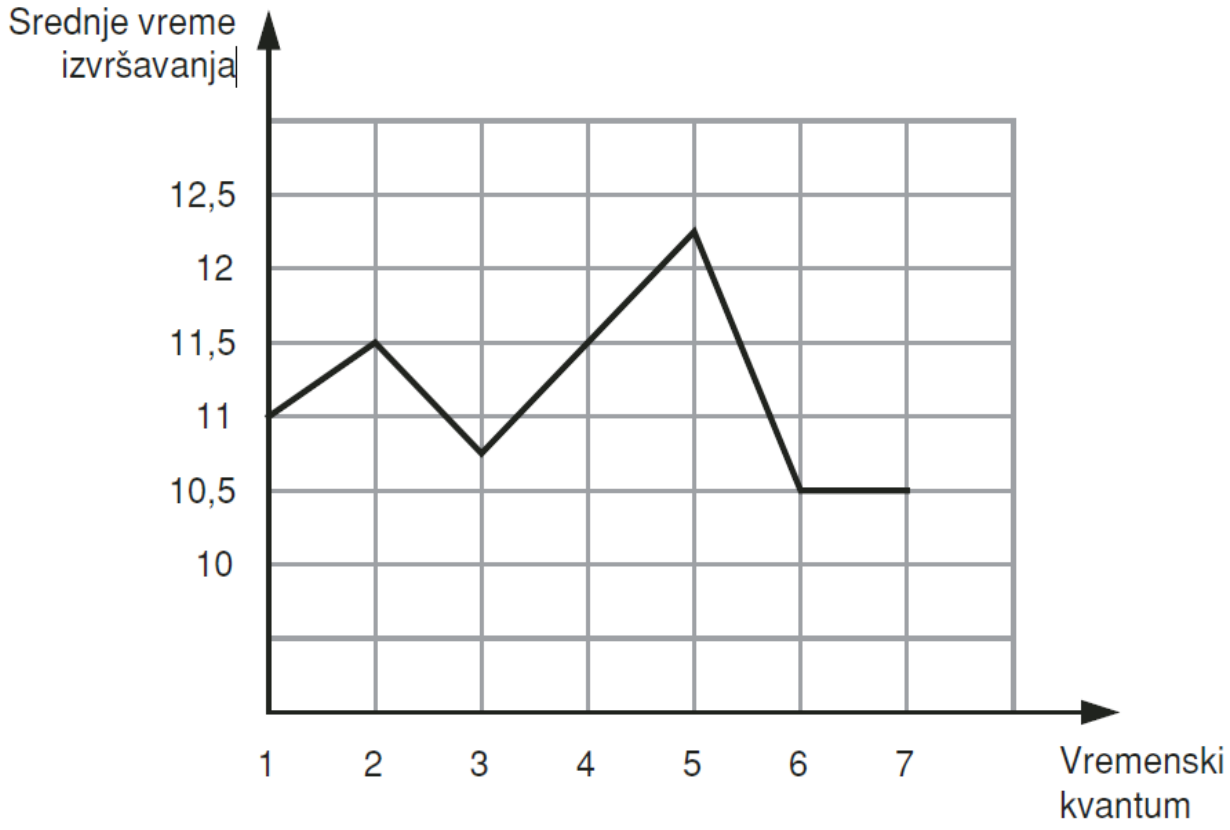
# Round Robin

- Analiza štetnog uticaja veoma malih vremenskih kvantuma.



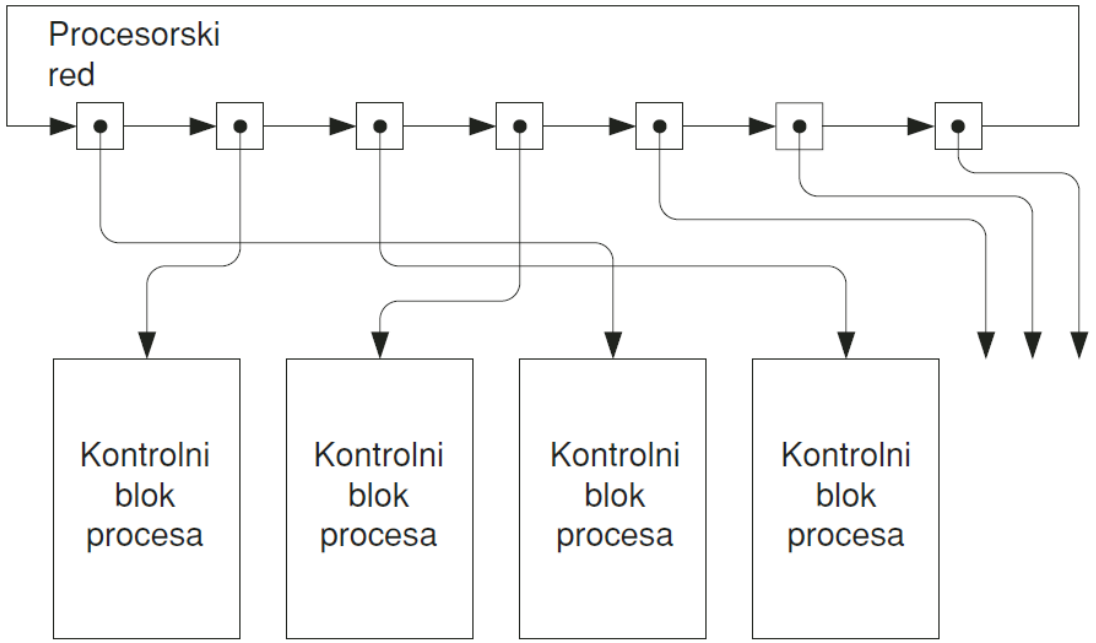
# Round Robin

- Srednje vreme izvršavanja procesa za male vrednosti kvantuma ne popravlja se obavezno sa povećanjem vremenskog kvantuma!
- Primer:
  - Vremena potrebna za izvršenje procesa 6, 3, 1 i 7 vremenskih jedinica.
  - Svi procesi pristižu u sistem u trenutku $t$=0.
  - Vremenski kvantumi veličine 2, 4 i 5 povećavaju vreme izvršenja umesto, da ga smanjuju.



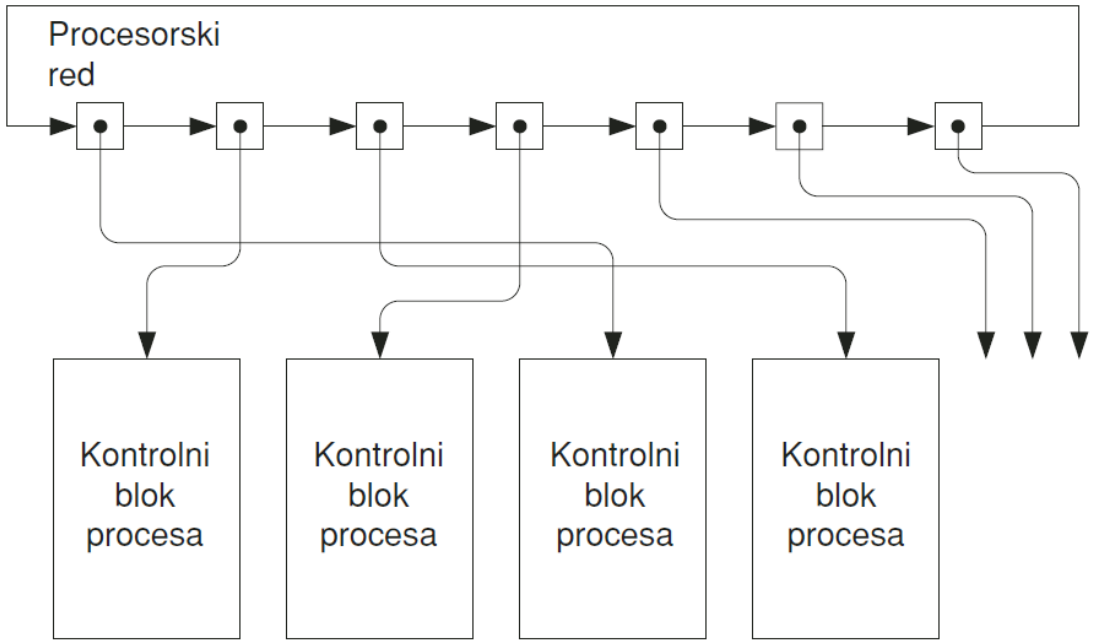
# Round Robin

- Pretpostavite da je RR implementiran tako da su članovi reda za dodelu procesora pokazivači na kontrolne blokove procesa.
- Pitanje: šta se postiže stavljanjem dva pokazivača na isti proces u red čekanja na procesor?



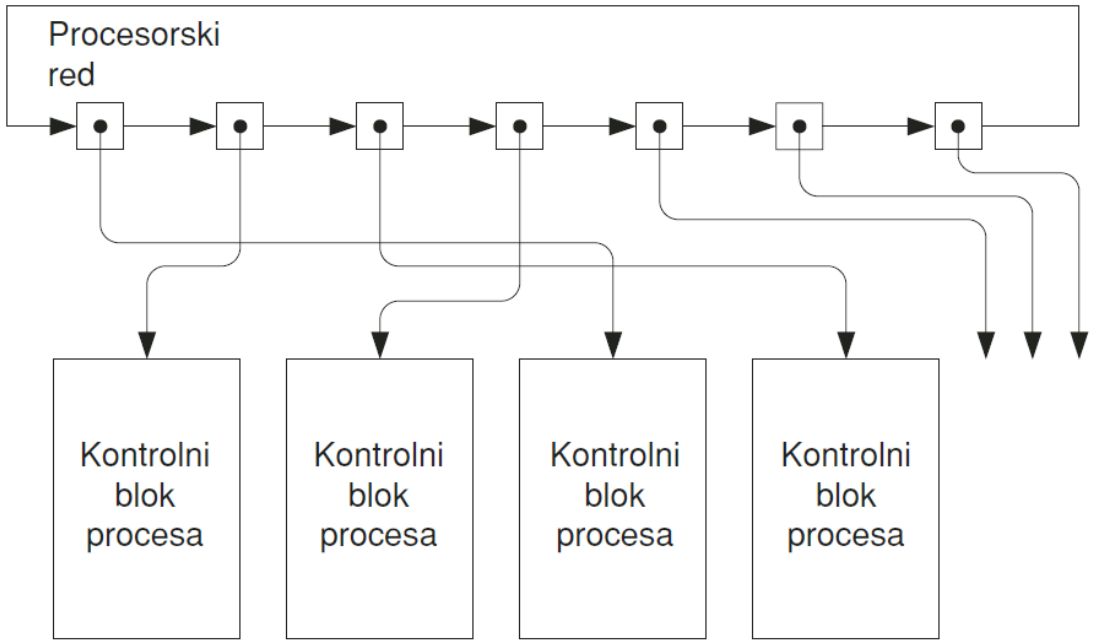
# Round Robin

- Odgovor: postiže se efekat sličan povećanju prioriteta procesa, jer proces češće dobija procesor na korićenje.



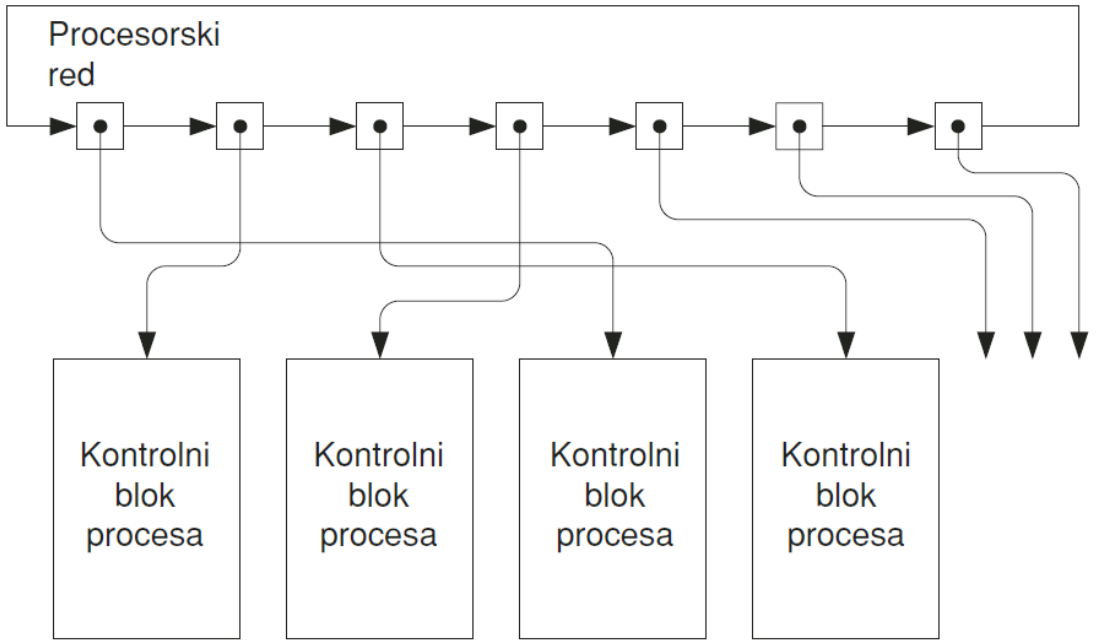
# Round Robin

- Pitanje: kako se može izmeniti osnovni RR tako da se isti efekat postigne bez uvođenja dvostrukih pokazivača?



# Round Robin

- Odgovor: uvođenjem dva različita vremenska kvantuma, pri čemu se veći kvantum dodeljuje procesima sa višim prioritetom, a manji procesima sa nižim prioritetom.



# Raspoređivanje u više procesorskih redova

- Ova klasa algoritama odnosi se na sisteme i sitacije u kojima se procesi mogu klasifikovati u različite grupe.
- Primer: procesi koji rade **u prvom planu** (engl. *foreground*) i procesi koji rade **u pozadini** (engl. *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.
- Svaki red čekanja ima poseban nezavistan 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.

# Raspoređivanje u više procesorskih redova

- 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).

# Raspoređivanje u više procesorskih redova

- **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.
- Ideja: procesi se na osnovu svojih karakteristika (vezanih za korišćenje procesora i U/I uređaja) razdvoje u različite redove.
  - Proces koji zahteva puno procesorskog vremena ubacuje se u niskoprioritetni red.
  - Interaktivni i U/I dominantni peocesi ubacuju se u visokoprioritetne redove.
  - Uvodi se vremenska komponenta koja spečava zakucavanje:
    - Ako proces čeka isuviše dugo, prebacuje se u red višeg prioiteta.
- Parametri koji definišu redove čekanja sa povratnom spregom su:
  - broj redova,
  - algoritam za raspoređivanje za svaki red posebno,
  - metode za određivanje reda u koji će proces ući po kreiranju i kada proces može da pređe u red višeg, odnosno nižeg prioriteta.
- Ovo rešenje primenjuju praktično svi savremeni OS.

# Raspoređivanje u višeprocesorskoj okolini

- Raspoređivanje je daleko složenije jer zahteva ozbiljnu sinhronizaciju procesora.
- Pretpostavićemo da su svaki procesor može da izvršava bilo koji proces iz reda čekanja.
- U tom slučaju može se:
  - Obezbediti **poseban red čekanja za svaki procesor**.
    - Problem: jedan procesor može biti potpuno besposlen ukoliko je odgovarajući red prazan a drugi procesor gotovo potpuno opterećen.
  - Konstruisati **zajednički red čekanja**.
    - OS može koristiti više tehnika za raspoređivanje:
      - **Simetrično** (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** (međuprocesorski *master-slave* odnos).
        - *Master* procesor izvršava kernelski kod i U/I operacije.
        - Ostali procesori izvršavaju korisnički kod.
        - Nije efikasno jer sve U/I operacije izvršava samo jedan procesor.

# 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.**