Operativni sistemi

- Jednoprocesorsko raspoređivanje -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet

Literatura

- *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
  - http://williamstallings.com/OperatingSystems/
  - http://williamstallings.com/OperatingSystems/OS9e-Student/

- Poglavlje 9: Jednoprocesorsko raspoređivanje
- Poglavlje 10:
  - 10.3 Raspoređivanje u sistemu Linux
  - 10.4 Raspoređivanje u sistemu Unix SVR4
  - 10.5 Raspoređivanje u sistemu Windows

Uvod u raspoređivanje (1)

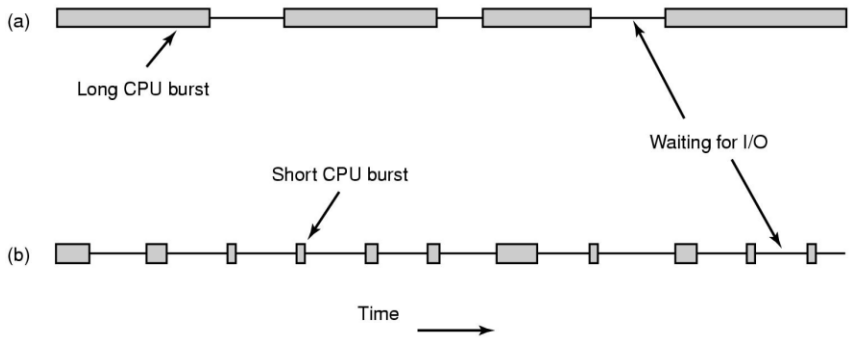
- **Raspoređivanje** (***Planiranje***) je osnovna funkcija OS-a
- Gotovo svi resursi sistema se pre upotrebe planiraju
- ***Planiranje CPU-a***, kao najvažnijeg resursa, ima centralno mesto u projektovanju OS-a
- CPU planiranje je osnova multiprogramskih OS-a
- Prebacivanjem CPU-a među procesima, OS obezbeđuje efikasnije korišćenje CPU-a
- Cilj multiprogramiranja je da se svakog trenutka izvršava neki proces čime se obezbeđuje efikasno korišćenje CPU-a
- Kod jednoprocesorskih sistema u jednom trenutku se može izvršavati samo jedan proces, ostali čekaju spremni u memoriji da se CPU oslobodi i da ih OS izabere za izvršenje

Uvod u raspoređivanje (2)

- Ideja multiprogramiranja je sledeća:
  - Jedan proces se izvršava do trenutka kada treba da čeka na neki događaj (obično da se izvede neka UI operacija)
  - Tog trenutka je CPU nezaposlen (*idle*)
  - Kod multiprogramiranja se to vreme koristi
  - OS oduzima CPU od procesa koji ga ne koristi i dodeljuje ga nekom od spremnih procesa
  - Kojem procesu?
  - Zavisno od algoritma planiranja

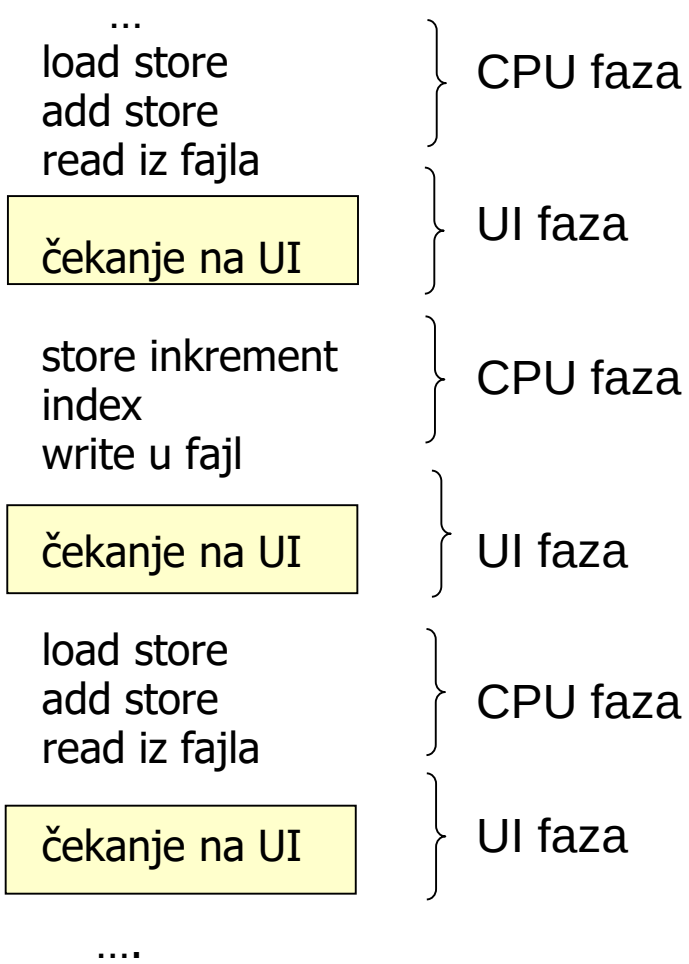
Ciklus CPU – UI (1)

- Procesi zahtevaju naizmenično korišćenje CPU-a i UI uređaja
- Izvršenje procesa je *ciklus* CPU izvršenje i UI čekanje. Procesi naizmenično menjaju ova dva stanja
- Svaki ciklus se sastoji od (obično veoma kratke) CPU faze (*bursta)* iza koga sledi (obično duža) UI faza (*burst)*
- Proces počinje i terminira se CPU fazom



- Postoje dve vrste procesa
  (a) Procesi orijentisani na CPU
  (b) Procesi orijentisani na UI
- Procesi (a) obično imaju duže CPU faze od procesa (b)

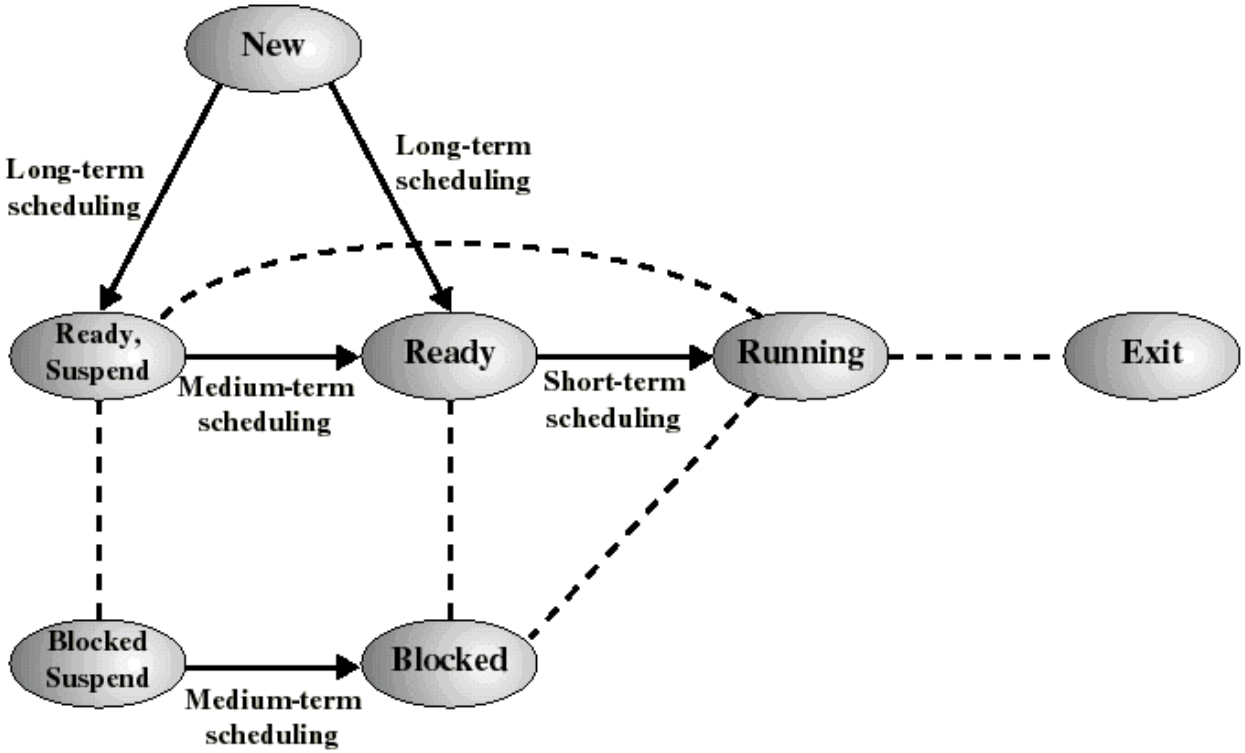
Ciklus CPU - UI (2)



Raspoređivanje procesora

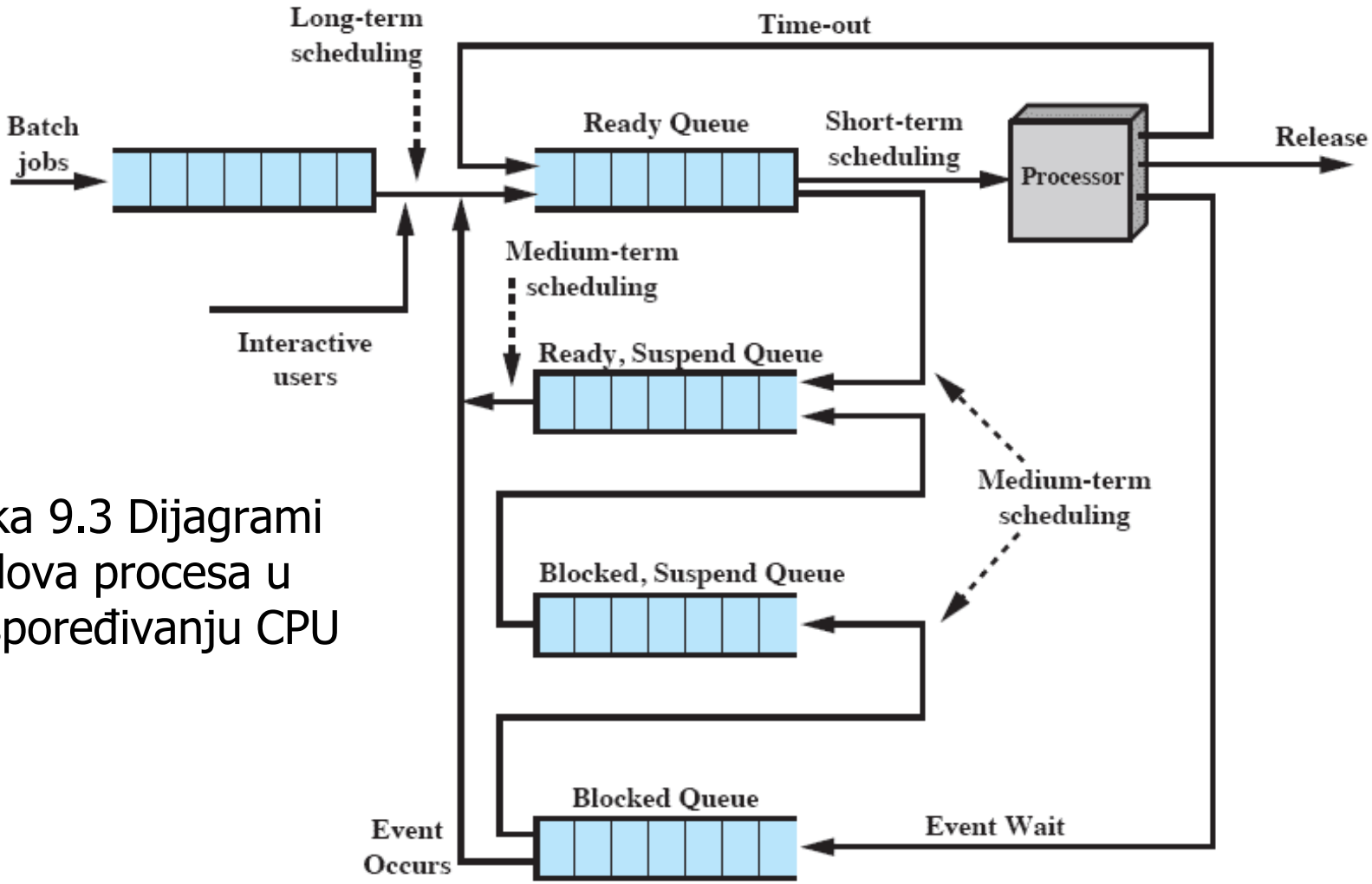
- Deo OS-a koji odlučuje kom će procesu iz liste spremnih procesa biti dodeljen CPU naziva se **kratkoročni raspoređivač** *(short-term scheduler)*, **dispečer** (dispatcher), ili **CPU planer** (scheduler)
- Algoritmi koje CPU planer koristi nazivaju se ***algoritmi raspoređivanja (planiranja)***
- Red spremnih procesa nije uvek implementiran kao FIFO red, i zavisno od algoritma planiranja može biti:
  - FIFO red, red po prioritetu, stablo, ili neuređena lančana lista
- Elementi reda spremnih procesa su upravljački blokovi procesa (PCB-ovi)
- Mi ćemo nadalje proučavati raspoređivanje kod ***jednoprocesorskih*** sistema

Tipovi raspoređivanja procesora



- **Dugoročno (Long-term):** koji proces pustiti u sistem - izvršava se prilikom kreiranja procesa
- **Srednjoročno (Medium-term):** koji proces preneti sa diska (swap područje) u glavnu memoriju
- **Kratkoročno (Short-term):** koji od spremnih procesa izabrati za izvršenje

Redovi raspoređivanja procesora



- Slika 9.3 Dijagrami redova procesa u raspoređivanju CPU

Dugoročno raspoređivanje

- Određuje koje kreirane procese pustiti u sistem na izvršenje
- Nakon izbora od strane dugoročnog planera, proces se dodaje u red kratkoročnog (u nekim OS – srednjoročnog) planera
- Kontroliše stepen multiprogramiranja
- Ako je više procesa aktivirano
  - Svaki proces dobija manji deo CPU vremena
  - Manja je verovatnoća da će svi procesi biti blokirani
    - Bolja iskorišćenost CPU
- Dugoročni planer pokušava da miksuje procese orijentisane na CPU i procese orijentisane na UI

Srednjoročno raspoređivanje

- Odlučuje koje procese preneti na disk ili koje uneti u memoriju sa diska za potrebe upravljanja multiprogramiranjem - upravlja funkcijom swap-ovanja (*swapping*)
- Sarađuje sa sistemom za upravljanje memorijom

Kratkoročno raspoređivanje

- Odlučuje koji će se proces sledeći izvršavati
- Kratkoročni raspoređivač se naziva još i dispečer (*dispatcher*)
- Dispečer se poziva kad god nastane događaj koji vodi blokiranju trenutno aktivnog procesa, ili koji obezbeđuje mogućnost da se prekine (*preempt*) trenutno aktivni proces u korist drugog procesa:
  - Prekid generatora takta
  - U/I prekidi
  - Sistemski pozivi operativnog sistema
  - Signali (na primer, od semafora)

Kada se vrši raspoređivanje?

- CPU raspoređivač donosi odluku u sledećim situacijama:
  - Kada se kreira novi proces
  - Kada se terminira neki proces
  - Kada se proces blokira na U/I ili semaforu (proces prelazi iz stanja aktivan u stanje blokiran)
  - Kada se javi U/I prekid (U/I je završen i proces prelazi iz stanja blokiran u stanje spreman)
  - Kada se javi prekid od generatora takta (kada proces prelazi iz stanja izvršenja u stanje spreman)

Kratkoročno raspoređivanje

- Kada OS izabere proces koji će se izvršavati neophodno je izvršiti promenu konteksta i aktiviranja izabranog procesa od tačke na kojoj je prekinut.
  - Prebacivanje konteksta podrazumeva čuvanje sadržaja registra CPU prekinutog procesa u okviru njegovog PCB i učitavanje prethodno zapamćenih vrednosti izabranog procesa u registre CPU
  - Prebacivanje u korisnički režim rada (*user mode*)
  - Prelaz na određenu lokaciju unutar izabranog programa koji je izabran od strane planera da bi se program restartovao od naredbe na kojoj je prethodno prekinut (istekom vremenskog kvanta), ili se sam blokirao
- **Dispečer** treba da je što je moguće brži jer se poziva pri svakoj promeni procesa
- Vreme koje dispečer potroši da bi stopirao jedan proces i startovao izabrani proces se naziva ***kašnjenje dispečerizacije (dispatch latency)***

Kriterijumi kratkoročnog raspoređivanja

- Orijentisani ka korisniku
  - ***Vreme odziva (Response time):*** Za interaktivne procese to je vreme koje protekne od slanja zahteva do početka prijema odgovora
  - ***Vreme zadržavanja/prolaska (Turnaround time):*** Vreme koje protekne od aktiviranja do kompletiranja procesa
    - Uključuje vreme izvršenja plus vreme koje proces provede čekajući na resurse uključujući i CPU
- Orijentisani ka sistemu
  - ***Propusna moć (Throughput):*** broj kompletiranih procesa u jedinici vremena. Za duge poslove to može biti 1 proces na sat, za kratke transakcije deo procesa u sekundi
  - ***Iskorišćenost procesora (CPU utilization):*** kreće se u opsegu 0 do 100%. Obično je 40% (lako opterećeni sistemi) do 90% (teško opterećeni sistemi)
  - ***Vreme čekanja (Waiting time):*** vreme koje proces provede čekajući u redu spremnih

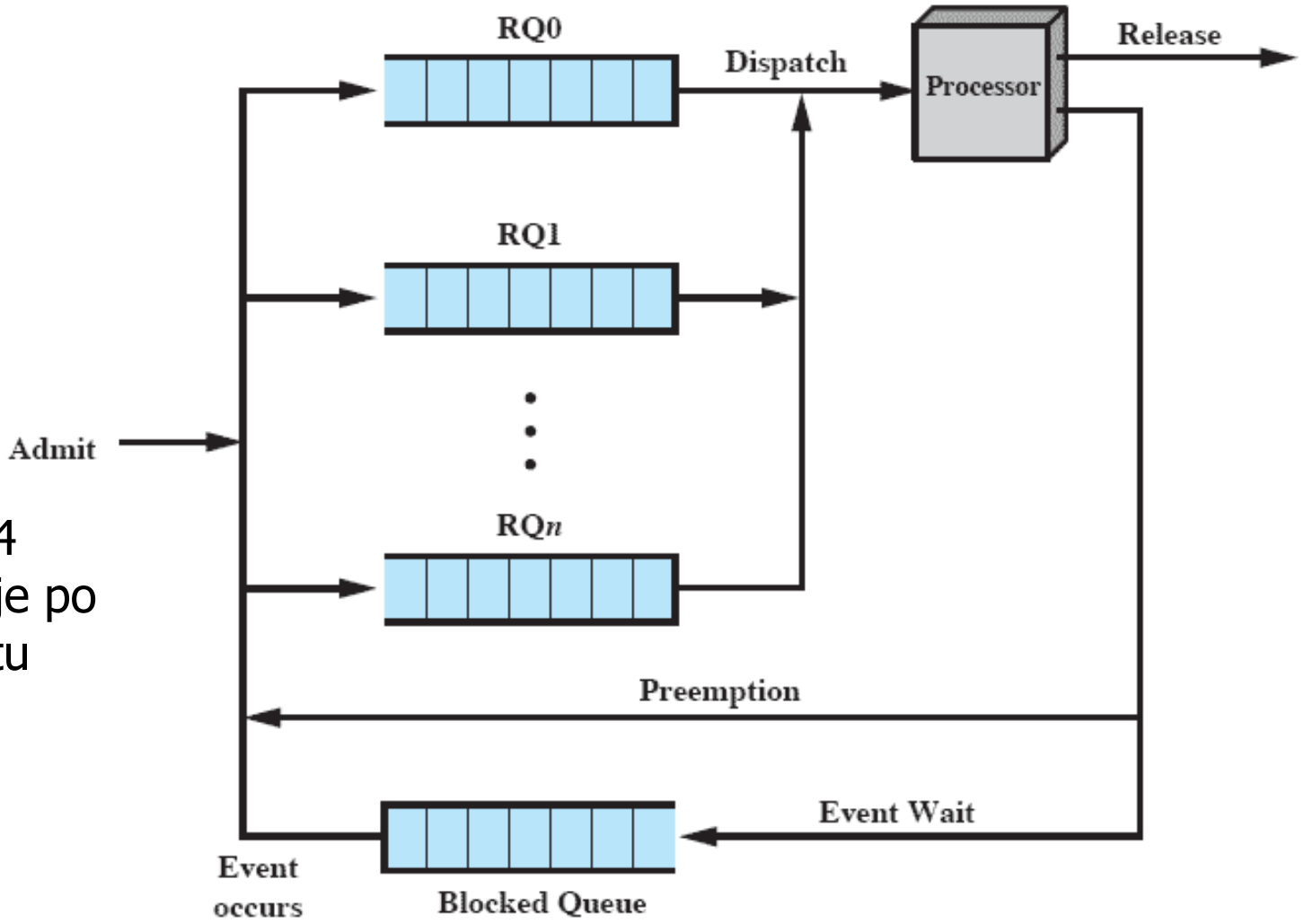
Ciljevi CPU raspoređivanja

- Maksimizirati iskorišćenost CPU i propusnu moć
- Minimizirati vreme zadržavanja, vreme čekanja i vreme odziva
- Pošto su ovi kriterijumi međusono zavisni nije moguća njihova istovremena optimizacija
  - Na primer, obezbeđivanje brzog vremena odziva može zahtevati algoritam planiranja koji će često izvršavati promenu konteksta procesa, čime se povećava režijski rad OS (*overhead*) i smanjuje propusna moć
- Za interaktivne sisteme se predlaže minimiziranje variranja vremena odziva
  - Poželjno je da sistem ima predvidljivo vreme odziva

Raspoređivanje po prioritetu

- U mnogim sistemima svakom procesu je dodeljen prioritet i planer uvek bira proces višeg prioriteta u odnosu na proces nižeg prioriteta
- U tom slučaju održava se više redova spremnih procesa u opadajućem redosledu njihovih prioriteta
- Kada treba da se obavi selekcija, OS bira jedan proces iz prvog nepraznog reda (najvišeg prioriteta) po određenoj politici planiranja
- Procesi nižeg prioriteta mogu "gladovati" za CPU

Redovi po prioritetu



- Slika 9.4 Uređenje po prioritetu

Algoritmi (politike) raspoređivanja

- ***Funkcija selekcije:*** određuje koji će proces iz reda spremnih procesa biti sledeći izabran za izvršenje
  - Može biti zasnovana na prioritetu procesa, zahtevima za resursima ili karakteristikama izvršenja procesa
- Ova funkcija se izražava korišćenjem tri veličine:
  - ***w*** = vreme provedeno u sistemu, u čekanju i izvršavanju
  - ***e*** = vreme provedeno u izvršavanju
  - ***s*** = totalno vreme zahtevano od strane procesa za izvršenje (vreme usluge) koje uključuje i *e*

Mod (režim) odluke

- ***Mod odluke*** definiše vremenski trenutak kada se izvršava funkcija selekcije
  - ***Bez prekidanja (nonpreemptive)***
    - Kada je proces u stanju izvršenja, izvršava se sve dok se ne terminira ili dok se ne blokira zbog UI
  - ***Sa prekidanjem (preemptive)***
    - OS može prekinuti proces koji se izvršava i premestiti ga u red spremnih procesa
    - Omogućava bolje usluge jer nijedan proces ne može dugo monopolizovati procesor

Algoritmi raspoređivanja

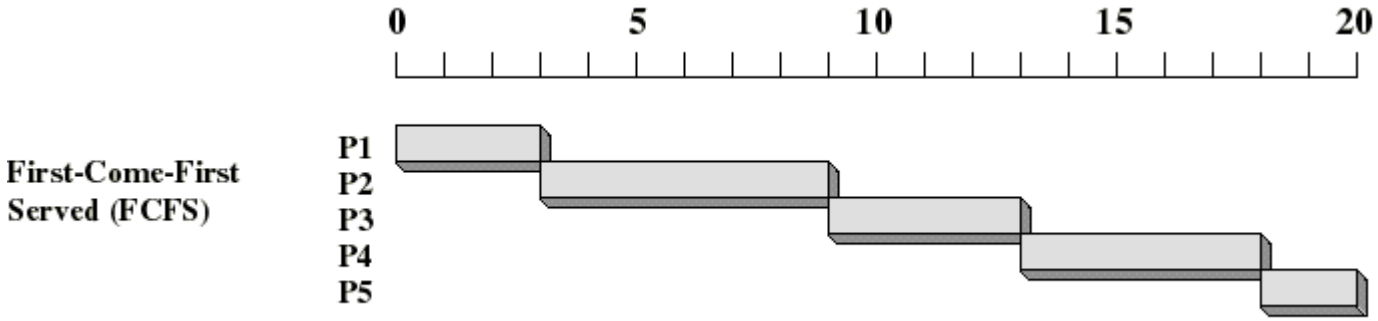
- *First Come First Served* (FCFS) – prvi došao prvi uslužen, bez prekidanja
- Kružno planiranje (Round-Robin)
- Najkraći proces sledeći (*Shortest Process Next* – SPN), bez prekidanja
- Najkraće preostalo vreme (*Shortest Remaining Time* - SRT), sa prekidanjem
- Sledeći sa najvećim odnosom odziva (*Highest Response Ratio Next* – HRRN)
- Planiranje u redovima u više nivoa, sa i bez povratne sprege (*Feedback*)

Primer koji će biti korišćen

Proces	Vreme pristizanja *(Arrival Time)*	Vreme obrade *(Service Time)*
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

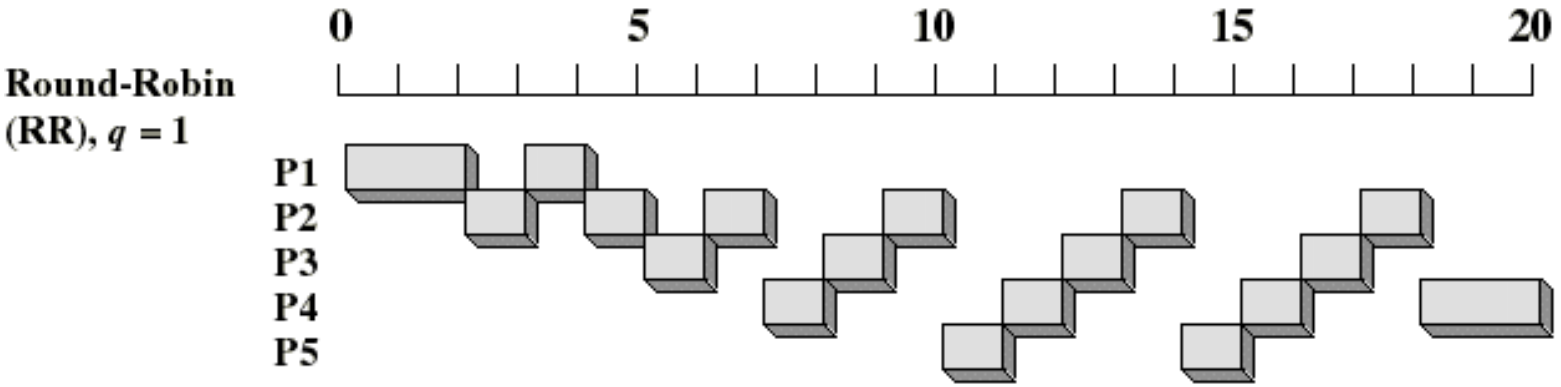
- Vreme obrade = ukupno procesorsko vreme potrebno u jednoj CPU fazi
- Procesi sa dugim vremenom obrade su procesi orijentisani na CPU i pominju se kao "dugi procesi"

Prvi došao prvi uslužen
First Come First Served (FCFS)



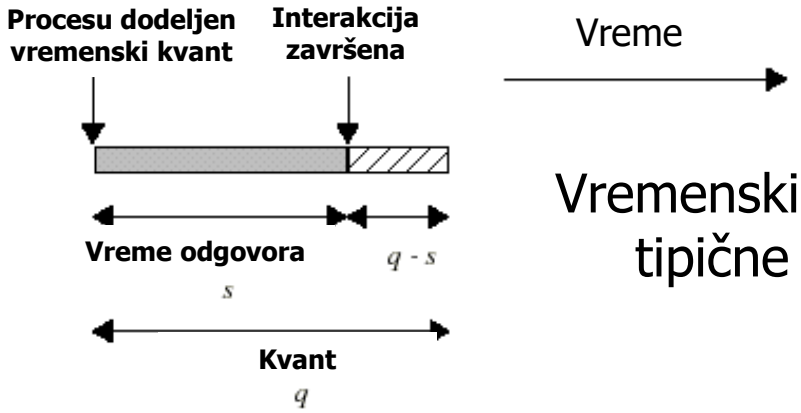
- **Funkcija selekcije:** proces koji najduže čeka u redu spremnih procesa (otuda, FCFS) **- max[*w*]**
- **Mod odluke:** bez prekidanja (*nonpreemptive*)
  - Proces se izvršava dok se ne terminira, ili dok sam sebe ne blokira
- **Nedostaci:**
  - Proces koji nema U/I monopolisaće procesor
  - Favorizuje procese orijentisane na CPU
  - Procesi orijentisani na U/I čekaće čak i kada su završili U/I (loše iskorišćenje U/I uređaja)
  - Da bismo držali zauzetim U/I uređaje treba dati nešto veći prioritet procesima koji su orijentisani na U/I

Round-Robin (RR)

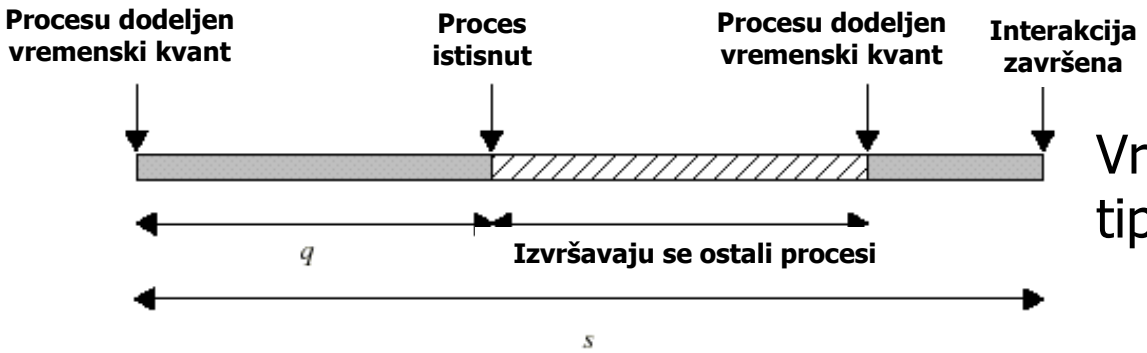


- **Funkcija selekcije:** ista kao kod FCFS
- **Mod odluke:** sa prekidanjem (*preemptive*)
  - Procesu se dopušta da se izvršava dok mu ne istekne dodeljeni vremenski period (kvant tipično 10 do 100 ms)
  - Tada se javlja prekidni signal od časovnika i proces se vraća u red spremnih procesa

Vremenski kvant za Round Robin



Vremenski kvant veći od tipične interakcije



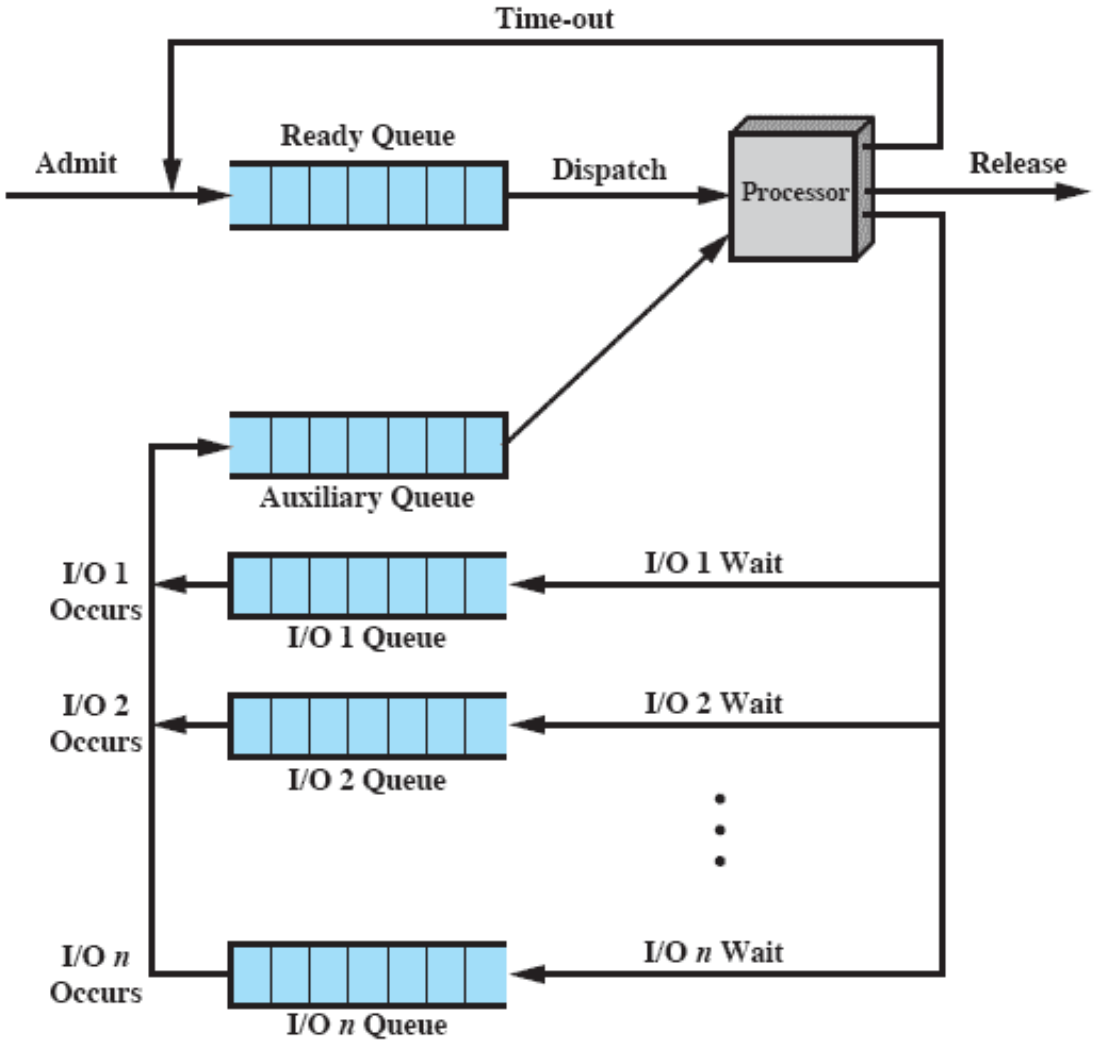
Vremenski kvant manji od tipične interakcije

- Vremenski kvant mora biti veći od vremena potrebnog za rukovanje prekidom od časovnika i promenu konkteksta (*context switch*)
- Treba da je veći od tipične interakcije, ali ne mnogo veći da bi se izbeglo kažnjavanje procesa orijentisanih na UI

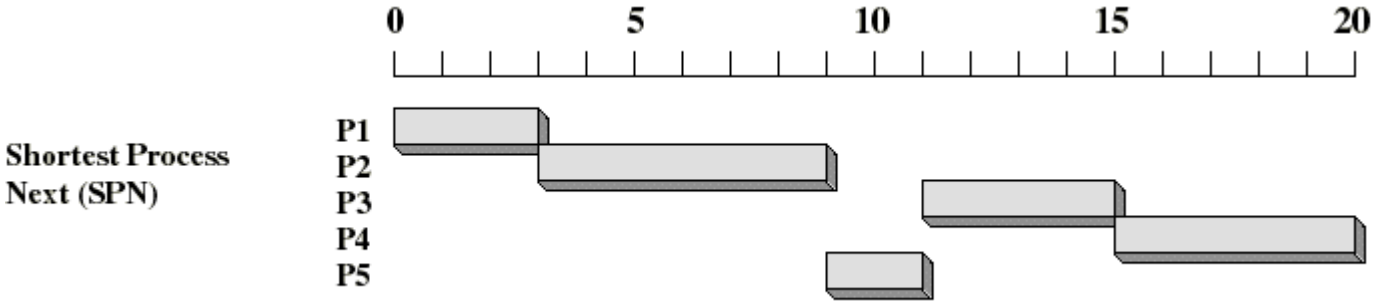
Nedostaci Round-Robin algoritma

- **Nedostaci:**
  - Još uvek se favorizuju procesi orijentisani na CPU
  - Procesi orijentisani na UI koriste CPU kraće nego što je vremenski kvant i zatim se blokiraju čekajući na UI
  - Procesi orijentisani na CPU izvršavaju se ceo vremenski kvant i vraćaju nazad u red spremnih procesa (tako da dolaze ispred blokiranih procesa)
- **Rešenje: virtuelni Round-Robin (VRR)**
  - Kada se UI završi, blokirani proces se stavlja u pomoćni red spremnih procesa koji ima prednost u odnosu na glavni red spremnih procesa
  - Proces izabran iz pomoćnog reda se izvršava onoliko dugo koliko mu je ostalo pre blokiranja zbog UI

Virtuelni Round-Robin (VRR)



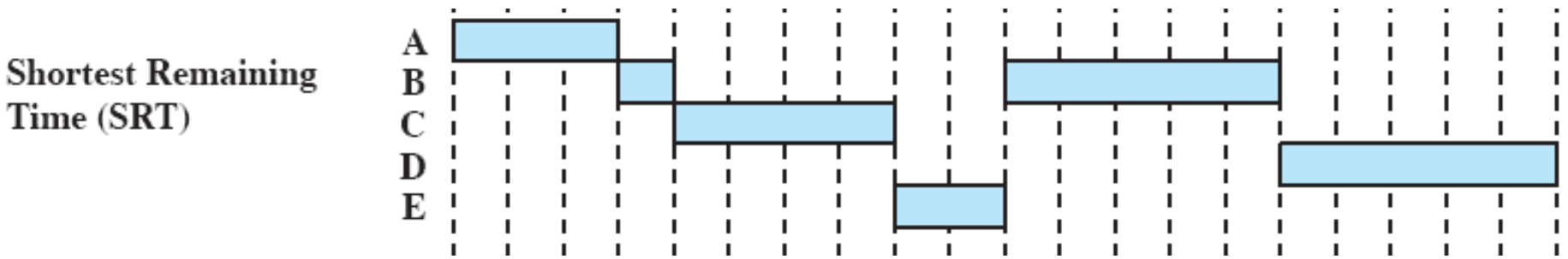
Najkraći posao sledeći
Shortest Process Next (SPN)



- **Funkcija selekcije:** proces sa najkraćim očekivanim vremenom obrade (*CPU burst time*) – **min[*s*]**
- **Mod odluke:** bez prekidanja
- Prvo se biraju procesi orijentisani na U/I
- Procena potrebnog vremena obrade $S_{n+1}=\alpha T_n+(1-\alpha)S_n \quad (0<\alpha<1)$
- **Nedostaci:**
  - Moguće izgladnjivanje dugih procesa sve dok pristižu kraći poslovi
  - Nepostojanje prekidanja nije pogodno za sisteme sa vremenskom podelom
  - Procesi orijentisani na CPU implicitno imaju niži prioritet, dok bi procesi orijentisani na U/I mogli monopolizovati CPU ako stalno pristižu
  - SPN implicitno ugrađuje prioritet: prednost imaju najkraći poslovi

Najkraće preostalo vreme (*Shortest Remaining Time* – SRT)

- Verzija algoritma (politike) *Prvo najkraći posao sa prekidanjem*
- Treba proceniti vreme izvršenja i izabrati najkraće, pri čemu se trenutno aktivni proces može prekinuti (istisnuti) ako se pojavi proces sa manjim procenjenim vremenom izvršenja

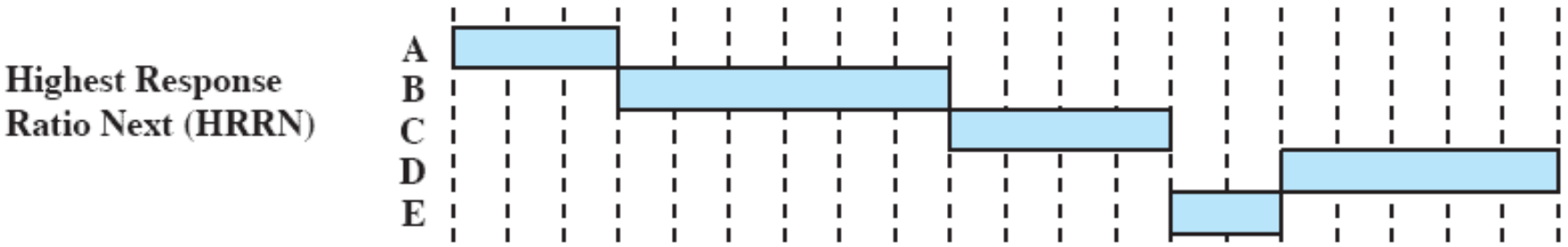


Sledeći sa najvišim odnosom odziva (HRRN)

- Bira sledeći proces sa najvišim odnosom (*Ratio*)

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

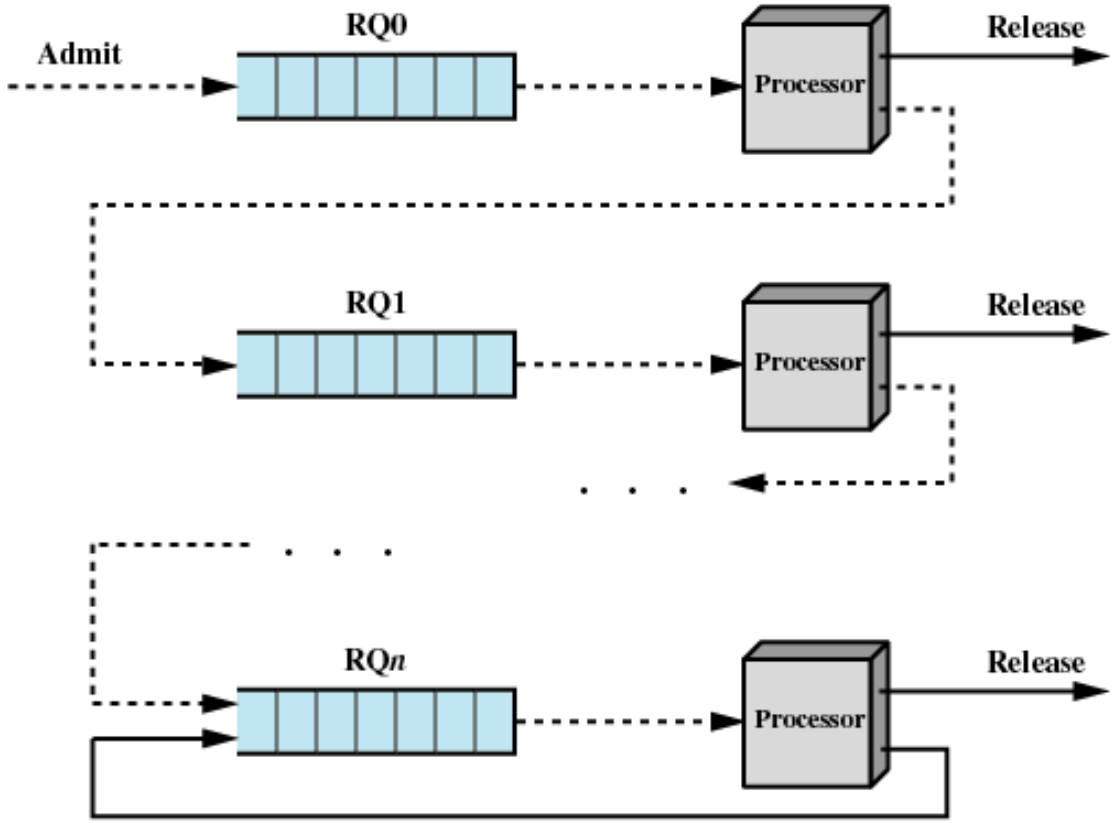
$$Ratio = \frac{w+s}{s}$$



Planiranje u više redova sa povratnom spregom (Feedback)

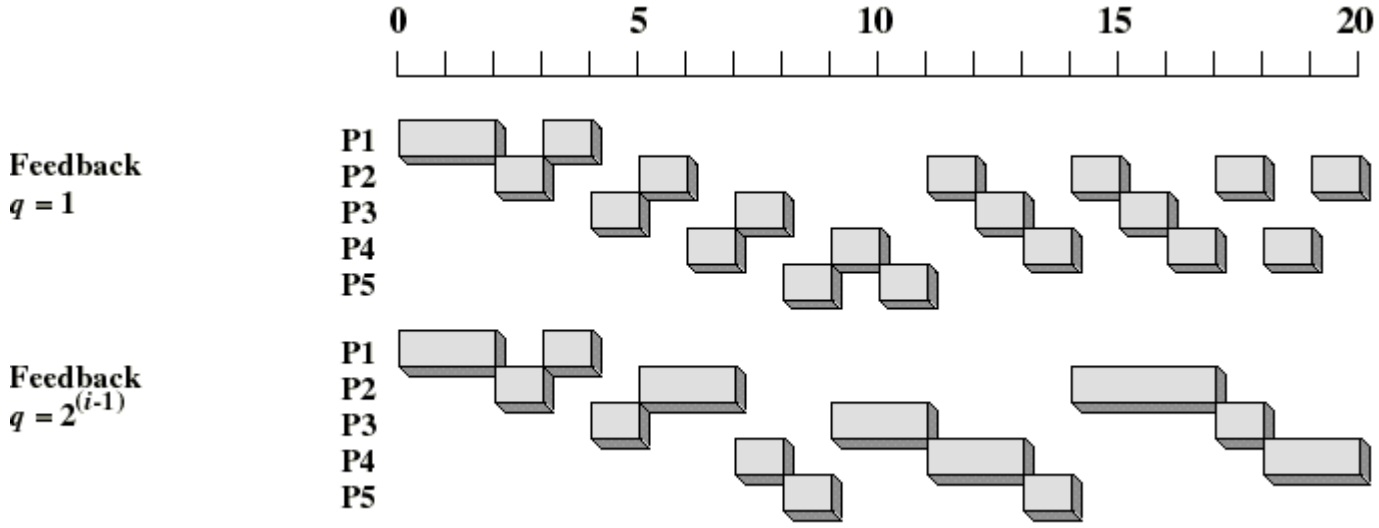
- To je raspoređivanje sa prekidanjem i dinamičkim prioritetima
- Postoji više redova spremnih procesa RQ_0, RQ_1,..., RQ_n (RQ - Ready Queue) različitog prioriteta:
  $P(RQ_0)>P(RQ_1)>...>P(RQ_n)$
- Novi proces se smešta u RQ_0
- Planer bira prvi proces iz RQ_0 i dodeljuje mu CPU na 1 vremenski kvant
- Kada istekne taj vremenski kvant, proces se premešta u RQ_1, a zatim RQ_2... dok ne dostigne RQ_n
- Procesi orijentisani na U/I ostaće u redu najvišeg prioriteta, dok će se procesi orijentisani na CPU pomerati u niže redove
- Planer za izvršenje bira proces iz RQ_i samo ako su prazni redovi RQ_{i-1} do RQ_0
- Obično je red RQ_n ciklični (round robin), pa proces tu kruži do terminiranja

Planiranje u više redova sa povratnom spregom (Feedback)



U svakom redu, osim u redu sa najnižim prioritetom, koristi se FCFS. U poslednjem redu koristi se Round Robin

Vremenski kvant za Feedback planiranje



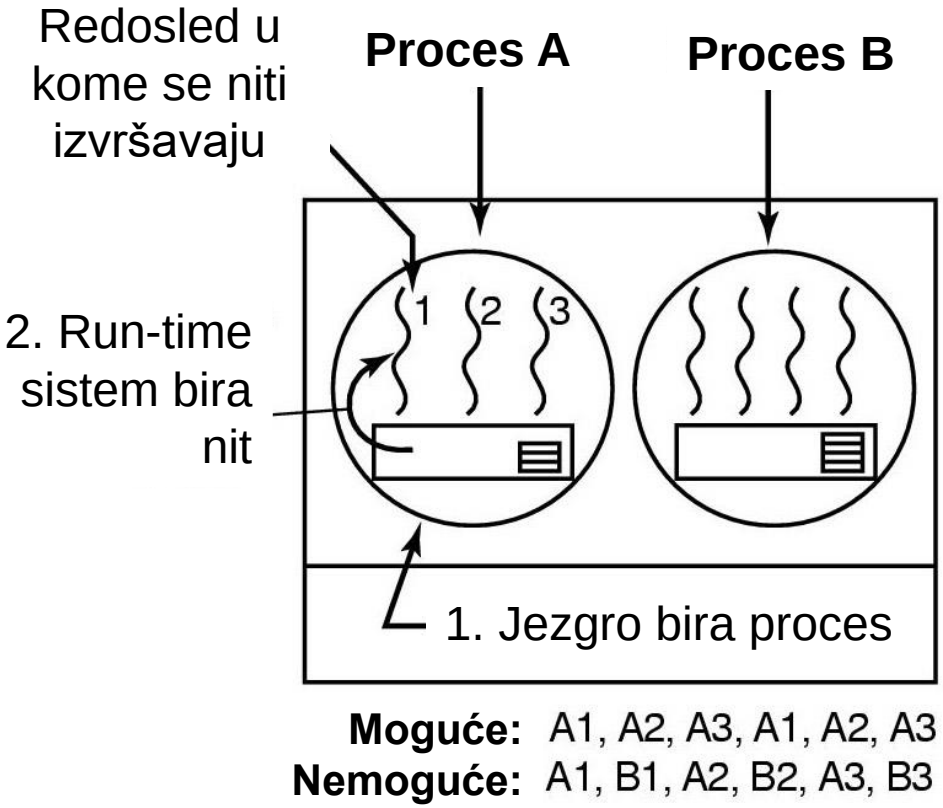
- Sa fiksnom dužinom kvanta, vreme zadržavanja dužih procesa može biti alarmantno
- Da bi se kompenzovao ovaj nedostatak sa nivoom reda uvećava se vremenski kvant
  - Primer: vremenski kvant reda $RQ_i = 2^{i-1}$
- Duži procesi još uvek mogu patiti od "izgladnjivanja"
  - Moguće rešenje je posle izvesnog vremena prevesti proces u neki red višeg prioriteta

Karakteristike algoritama raspoređivanja

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Raspoređivanje niti na nivou korisnika

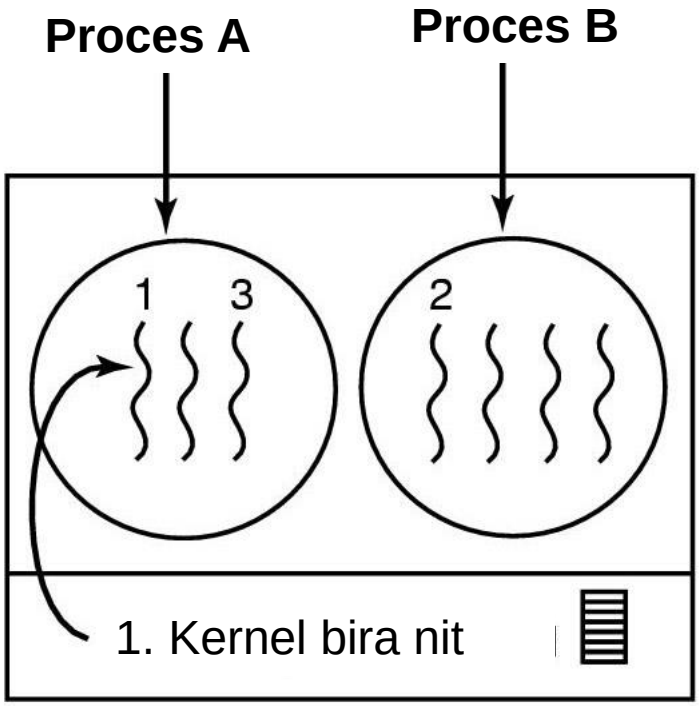


Tanenbaum, 2014

Moguće planiranje niti na korisničkom nivou

- 50-msec kvant procesa
- Niti se izvršavaju po 5 msec, jedna za drugom dok ne iskoriste ceo dodeljen kvant (CPU burst)

Raspoređivanje niti na nivou kernela



1. Kernel bira nit unutar procesa koja će se izvršavati

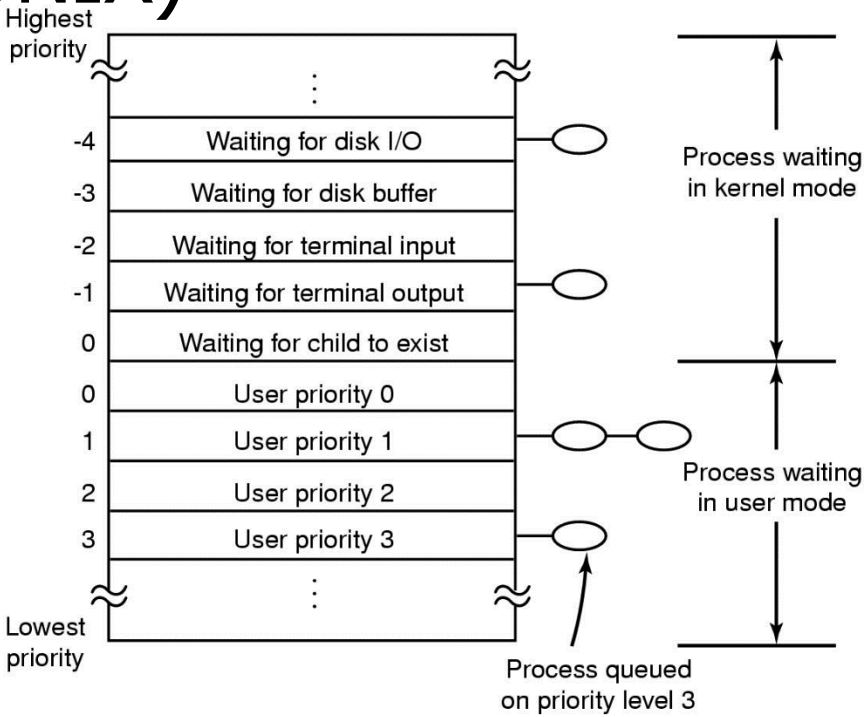
Tanenbaum, 2014

Moguće: A1, A2, A3, A1, A2, A3
Takođe moguće: A1, B1, A2, B2, A3, B3

- Razlike u planiranju niti na nivou kernela i na nivou korisnika:
  - Performanse - Zamena korisničkih niti u okviru procesa je daleko brža od zamene niti na nivou kernela koje zahteva kompletnu promenu konteksta
  - Algoritam planiranja – Za planiranje niti na nivou korisnika može se implementirati algoritam planiranja specifičan za konkretnu aplikaciju

Raspoređivanje u UNIX (System V R3, 4.3 BSD UNIX)

- Koristi se planiranje u dva nivoa:
  - Planer višeg nivoa premešta procese iz memorije na disk, i obrnuto
  - Planer nižeg nivoa (dispečer) bira proces kome će dodeliti CPU iz reda spremnih procesa
- Svaka verzija Unix-a ima nešto drugačiji algoritam nižeg nivoa, ali u suštini to je uvek planiranje po prioritetu sa više redova
- Procesi koji se izvršavaju u korisničkom režimu imaju pozitivan prioritet, a oni koji se izvršavaju u kernel modu negativan
- U redovima se nalaze samo spremni procesi koji su u memoriji
- Prioritet procesa se preračunava svake sekunde



- Planer pretražuje redove počev od reda najvišeg prioriteta i bira prvi proces iz prvog nepraznog reda
- Dodeljuje mu CPU na 1 kvant, nakon čega ga prekida, osim ako se proces nije u međuvremenu blokirao, i smešta na kraj reda odakle je izabran (RR planiranje)

Raspoređivanje u Unix (2)

- U redove prioriteta se smeštaju samo procesi spremni za izvršavanje.
- Planer pretražuje redove počev od reda najvišeg prioriteta.
- Aktivira se prvi proces iz prvog nepraznog reda i to za jedan vremenski kvant (100 ms-1s) ili dok se ne blokira, npr. na U/I zahtev.
- Kada završi sa dodeljenim kvantom proces se smešta na kraj istog reda
  - U okviru svakog reda se koristi Round Robin za izbor procesa
- Prioritet svakog procesa (*Priority*) se izračunava svake sekunde:

$$Priority(i) = CPU(i)/2 + nice + Base$$

  - *CPU(i)* – mera korišćenja procesora od strane procesa u tokom poslednjeg vremenskog intervala *i*. Što je *CPU_usage* veći to je prioritet procesa manji.

$$CPU(i) = CPU(i-1)/2$$

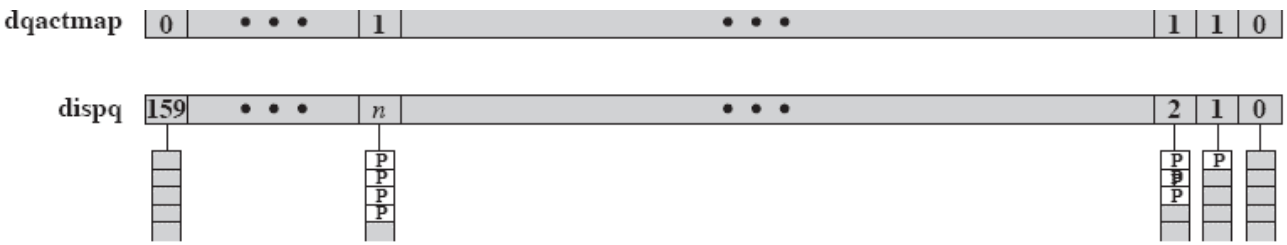
  - *nice* – svakom procesu se dodeljuje ova vrednost (podrazumevano je 0). Sistemski poziv: *nice(value)* može se postaviti vrednost nice u opsegu 0-20 a administrator može dodeliti vrednost -20 – -1.
  - *Base* – Osnovni prioritet procesa
- U skladu sa izračunatim prioritetom proces se smešta u odgovarajući red ili prekida trenutno aktivni proces.

Raspoređivanje u Unix (3)

- Uloga *base* prioriteta je podela svih procesa u fiksne kategorije određenog nivoa prioriteta
- *CPU* i *nice* vrednosti su ograničene tako da proces ne može da menja svoju kategoriju
- Ove kategorije obezbeđuju optimizovani pristup sekundarnim memorijama (diskovima) i brz odgovor na sistemske pozive.
- U redosledu opadajućem po prioritetu, kategorije su:
  - Swapper
  - Upravljanje blok U/I uređajima
  - Manipulacija datotekama
  - Upravljanje znakovnim U/I uređajima
  - Korisnički procesi

Raspoređivanje u UNIX System V R4

- Najviši prioritet je dat *real-time* procesima, zatim *kernel-mod* procesima, a najniži korisničkim procesima (*time-shared* procesima)
- 160 nivoa prioriteta podeljenih u 3 kategorije
- Red za raspoređivanje (*dispq*) se pridružuje svakom niovu prioriteta, zajedno sa bitmap vektorom - *dqactmap*
- Unutar time-shared procesa prioritet se menja; ukoliko proces iskoristi ceo kvant, prioritet se smanjuje.
- Kvant time-shared proc. 10ms (0) -100ms (59)



Priority Class	Global Value	Scheduling Sequence
Real-time	159 • • • • 100	first ↓
Kernel	99 • • 60	↓
Time-shared	59 • • • • 0	↓ last

Raspoređivanje u Linux (verzija 2.4 i ranije)

- Linux ima planiranje niti (*light weight* procesi) - implementirane u jezgru
- Linux razlikuje tri klase prioriteta
  - ***Real-time FIFO (najviši prioritet):*** ne mogu se istiskivati, osim od novo-učitane RT FIFO niti (SCHED_FIFO)
  - ***Real-time Round-Robin:*** iste su kao RT FIFO, jedino što se mogu prekidati (SCHED_RR)
  - ***Time sharing*** (SCHED_OTHER)
- Nijedna od ovih klasa nije real-time u pravom smislu; nema specificiranja *deadline-a*, nema garancije za ispunjenje *deadline-a*.
- Razlog za uvođenje ovog termina je usklađivanje sa standardom P1003.4 (Real-time ekstenzije za UNIX)
- Podrazumevani prioritet niti je 20, ali se može promeniti sistemskim pozivom *nice(value) , value=[-20,19]* pri čemu prioritet dobija vrednost jednaku *20-value*, otuda prioritet niti u opsegu *1≤prioritet≤40*
- Svaka nit ima dodeljeni vremenski kvant za izvršavanje (vremenski takt na svakih 1 ms – *jiffy,* a kvant je određena multiplikacija takta)
- Funkcije kernela ***schedule(), do_timer(), switch_to()***

Raspoređivanje u Linux (2) (verzija 2.4 i ranije)

- Vremenski kvant je promenljiva *counter* u *task_struct* (10-200ms) i u izrazu je predstavljena kao *quantum*
- Planer najpre izračunava goodness:

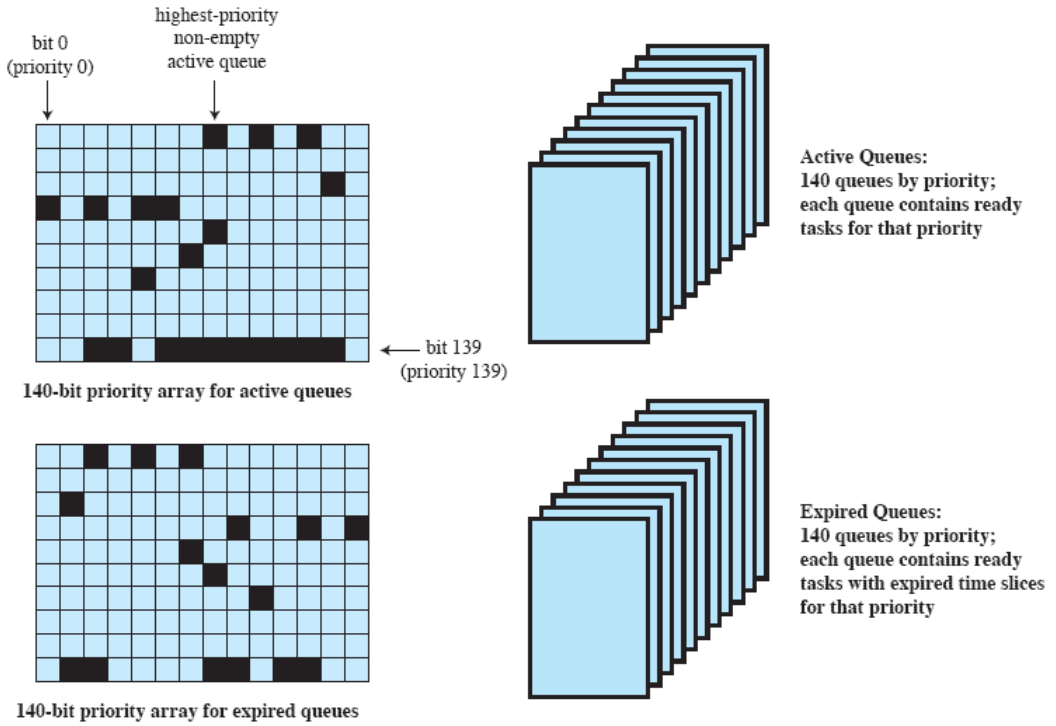
```
if ((class = = real_time) goodness = 1000 + priority;
if ((class = = time_sharing && quantum > 0) goodness = quantum + priority;
if ((class = = time_sharing && quantum = = 0) goodness = 0;
```

- Planer bira nit koja ima najveći **goodness**. Sa svakim vremenskim taktom vremenski kvant aktivne niti se smanjuje za 1
- Nit se prekida ako bude ispunjen neki od sledećih uslova:
  - Kvant niti je postao 0
  - Nit se blokira na semaforu, I/O operaciji ili nečem drugom,
  - Prethodno blokirana nit sa većim goodness-om je postala spremna.
- Ako je kvant svih spremnih niti 0 (blokirane niti mogu imati kvant koji nije 0) planer resetuje kvantove svih niti:

$$quantum = (quantum / 2) + priority$$

Raspoređivanje u Linux (2.6+)

- Linux 2.4 planer za SCHED_OTHER klasu niti nije bio dovoljno skalabilan za povećan broj procesora (jezgara) i niti
- Linux 2.6 koristi novi planer zasnovan na prioritetu, poznat kao O(1) planer
- 140 nivoa prioriteta
  - 0-99 za RT i 100-139 za TS niti
- Za svaki procesor Linux planer održava dve strukture podataka za:
  - Aktivne redove
  - Istekle redove (prethodno iskoristili vremenski kvant)
- Prednost se daje nitima orijentisanim na U/I u odnosu na niti orijentisane na CPU

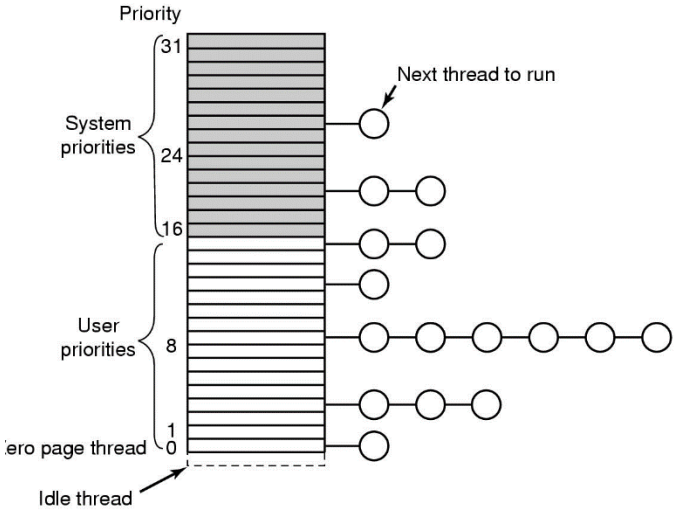


Raspoređivanje u Linux

- Za svaki procesor planer bira prvu nit iz nepraznog reda najvišeg prioriteta; niti u redu se organizuju u round-robin stilu
- Niti se mogu premeštati iz reda jednog procesora u red drugog
- Prioritet *ne-real time* niti je u opsegu 100-139, inicijalno 120
- Dinamički prioritet se izračunava na osnovu prethodnog izvršavanja/čekanja niti
- *Real-time* niti nemaju dinamički prioritet
- Vremenski kvant je promenljiv – 10-200ms

Raspoređivanje u Windows

- Nema centralno planiranje niti
- Planiranje u više nivoa sa povratnom spregom
- Planer podržava 32 nivoa planiranja
- Niti u redovima po RR algoritmu
- Postoje 4 kategorije prioriteta:
  - 16-31 sistemski: rezervisano za sistem i za niti kojima sistem administrator može dodeljivati prioritet
  - 1-15 korisnički
  - 0 zero: za potrebe menadžera memorije
  - -1 idle: *idle* nit se izvršava kad niko ne koristi CPU



- Planer pretražuje redove od 31 pa naniže i bira nit sa početka nepraznog reda i dodeljuje mu 1 kvant
- Pošto vreme istekne nit se vraća na kraj tog reda, a planer bira nit sa početka reda
- Ako nijedna nit nije spremna, izvršava se *idle* nit

Raspoređivanje u Windows (2)

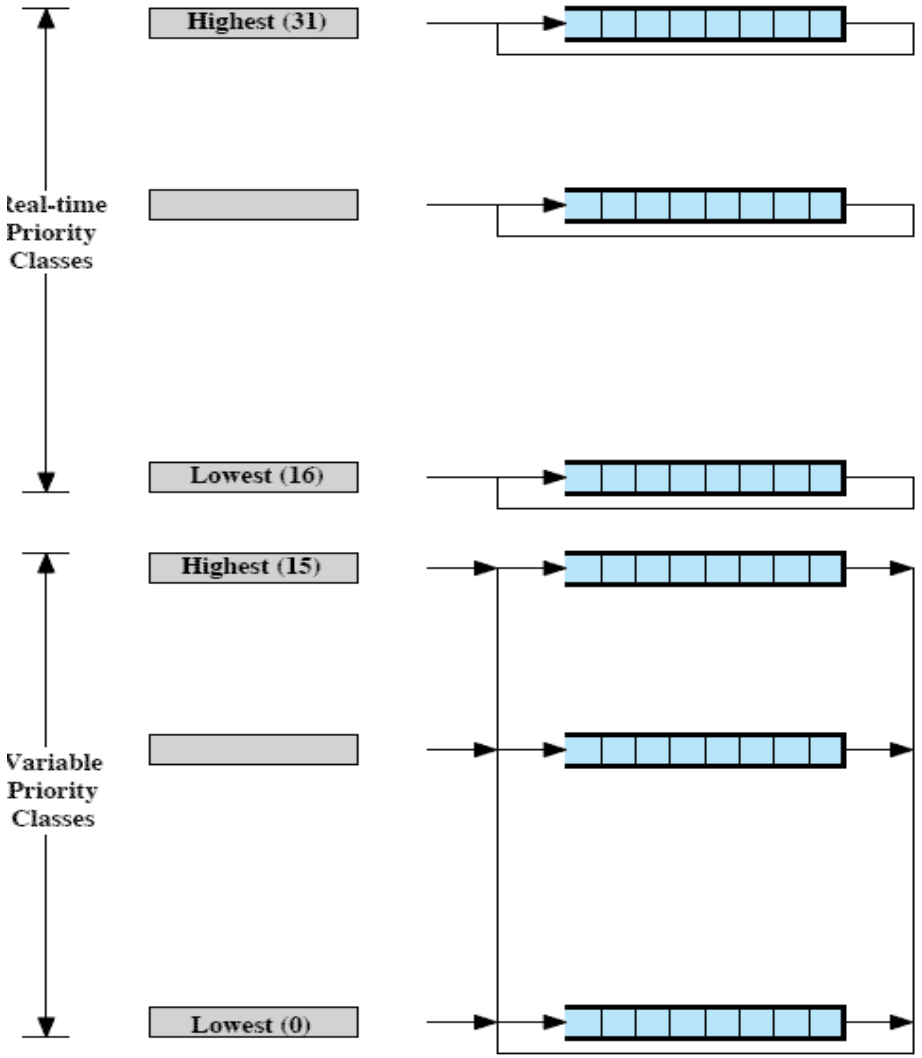
- Planiranje se vrši na nivou niti.
- Ne postoji centralni planer (scheduler) procesa (niti) već je kod za planiranje razbacan po jezgru sistema.
- Kada trenutno aktivna nit ne može više da se izvršava ulazi u kernel mode i aktivira planer.
- Aktiviranje planera se vrši ukoliko je ispunjen neki od sledećih uslova:
  - Nit se blokira na semaforu, mutex-u, događaju (event)
  - Nit signalizira neki objekat i aktivira nit većeg prioriteta (npr. operacija na semaforu)
  - Istekao je vremenski kvant.
- Planer se poziva automatski u sledećim situacijama:
  - Završetak I/O operacije – proverava se da li trenutno aktivna nit može biti zamenjena.
  - Istek perioda čekanja (timed wait)

Raspoređivanje u Windows (3)

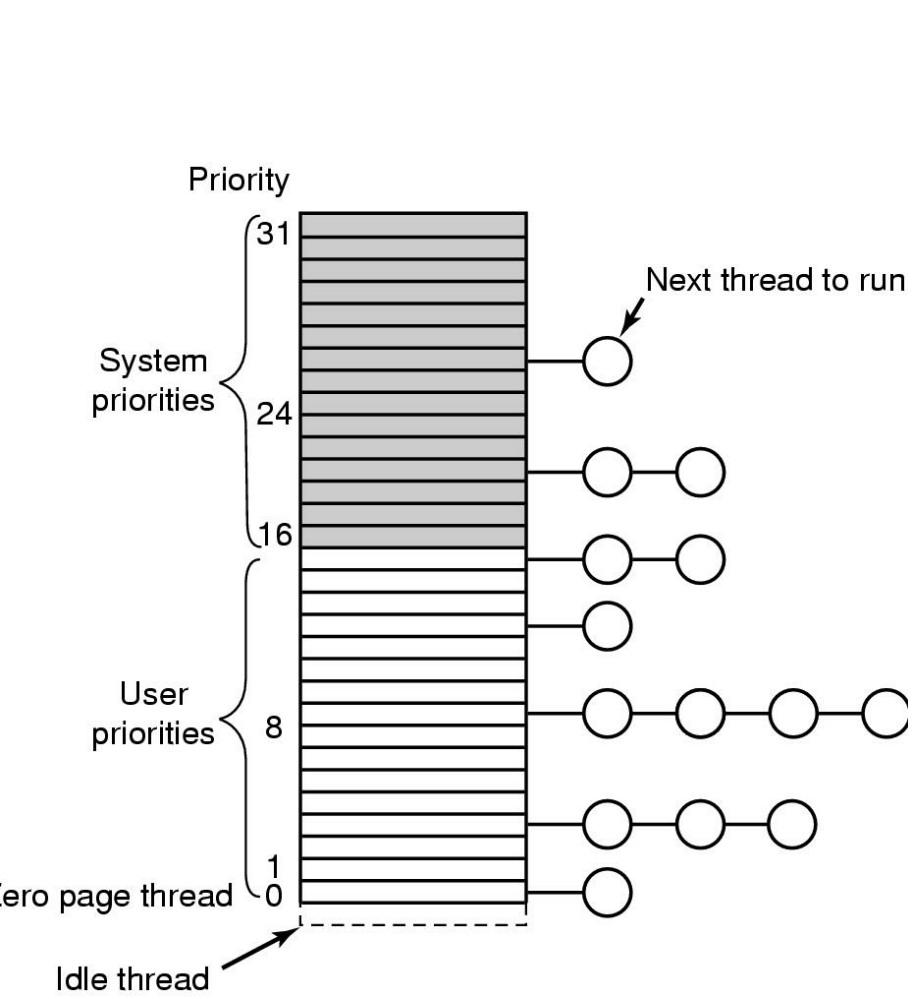
- Algoritam planiranja je fully preemtive multi-level queue.
- Za svaki red se koristi Round Robin.
- ***Fully preemptive*** – prekidanje niti se može desiti u bilo kom trenutku a ne samo na kraju vremenskog kvanta.
- Prioritet procesa i niti (42 moguće kombinacije):
- SetPriorityClass – postavlja prioritet svih niti procesa
  - Realtime
  - High
  - Above normal
  - Normal
  - Bellow normal
  - Idle
- SetThreadPriority – postavlja prioritet niti relativno u odnosu na druge niti istog procesa
  - Time critical
  - Highest
  - Above normal
  - Normal
  - Bellow normal
  - Lowest
  - Idle

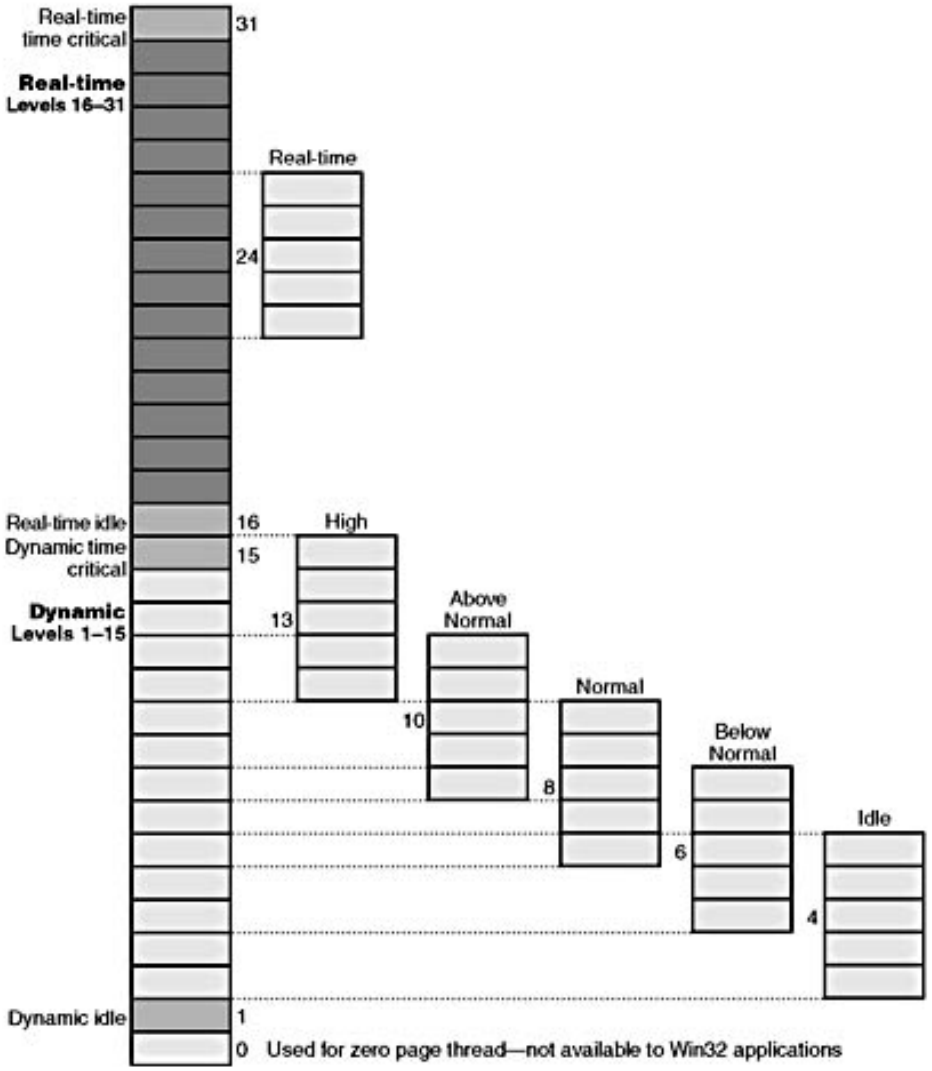
Prioriteti procesa i niti

		Win32 process class priorities					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1



Raspoređivanje u Windows (4)



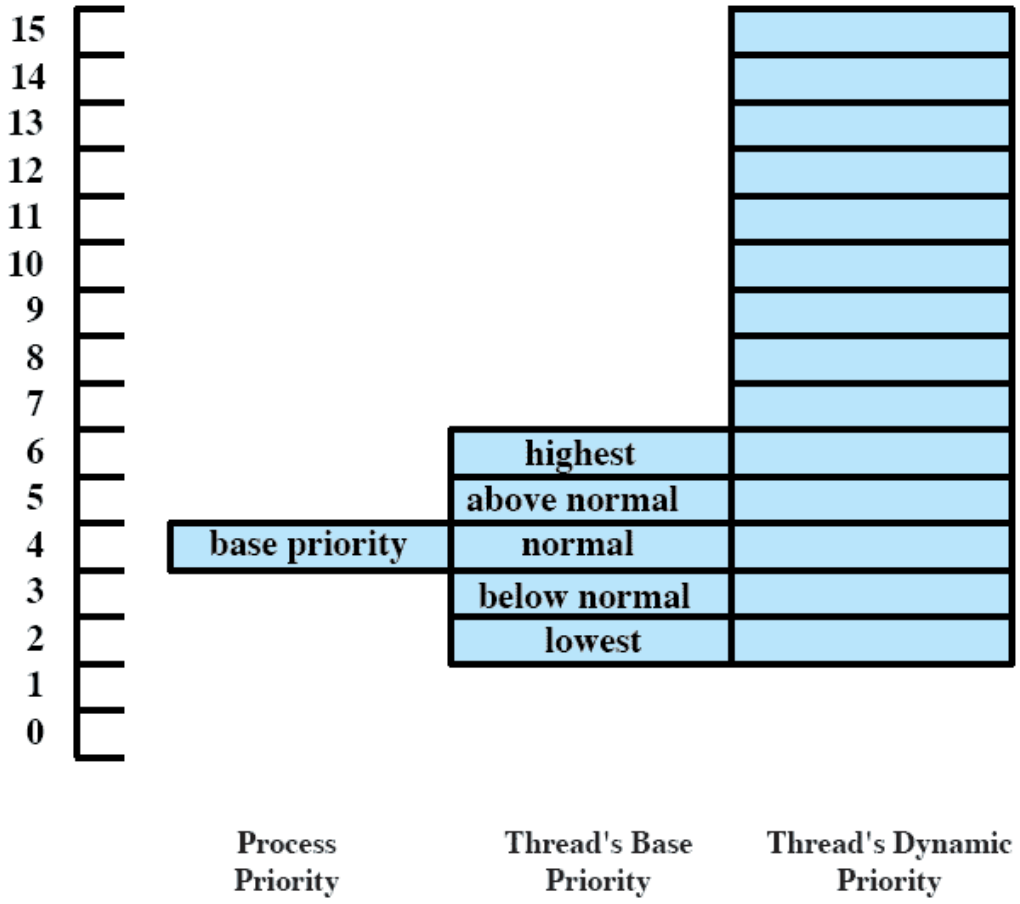


Raspoređivanje u Windows (5)

- *Zero page thread* je nit koja se izvršava u pozadini koristeći CPU vreme kada nema drugih niti koje su spremne za izvršavanje. Ima zadatak da ažurira stranice za upravljanje memorijom.
- Vremenom je osnovni algoritam modifikovan tako da nitima sa nivoa nižih od 15 prioritet može biti povećan po završetku I/O operacije. Time se postiže da se veoma brzo startuju I/O operacije čime se postiže da su I/O uređaji stalno aktivni.
- Ukoliko nit iskoristi čitav vremenski kvant prebacuje se u niz nižeg prioriteta.

Raspoređivanje u Windows (6)

- Primer prioriteta procesa i niti



Raspoređivanje u Windows (7)

- Vremenski kvant:
  - Windows 2000 Professional – 20 milisekundi
  - Windows 2000 Server – 120 milisekundi
- Vremenski kvant se ručno može povećati: 2x, 4x, 6x u odnosu na default vrednost.
- Ako je nit čekala na nekom objektu za sinhronizaciju, nakon oslobađanja prioritet joj se povećava za 2 ako se radi o *foreground* niti (nit koja kontroliše prozor u fokusu) ili za 1 u ostalim slučajevima.
- Izmena osnovnog planera: kada prozor postane aktivan sve njegove niti dobijaju veći vremenski kvant.

Domaći zadatak

- Poglavlje **9 Jednoprocesorsko raspoređivanje**
  - 9.6 Ključni pojmovi, kontrolna pitanja i problemi
- CPU Scheduling animations
  - https://apps.uttyler.edu/Rainwater/COSC3355/Animations
  - First Come, First Served Scheduling
  - Shortest Job First Scheduling
  - Priority Scheduling
  - Round Robin Scheduling