




Оперативни Системи

Распоредување

проф. д-р Димитар Трајанов,
проф. д-р Невена Ацковска,
проф. д-р Боро Јакимовски,
проф. д-р Весна Димитрова,
проф. д-р Игор Мишковски,
проф. д-р Сашо Граматиков,
вонр. проф. д-р Милош Јовановиќ,
вонр. проф. д-р Ристе Стојанов,
доц. д-р Костадин Мишев



Цел на предавањето

- Распоредување на процеси
- Мерки за перформанси
- Разлики во начините за распоредување
 - Пакетни системи
 - Интерактивни системи
 - Системи кои работат во реално време

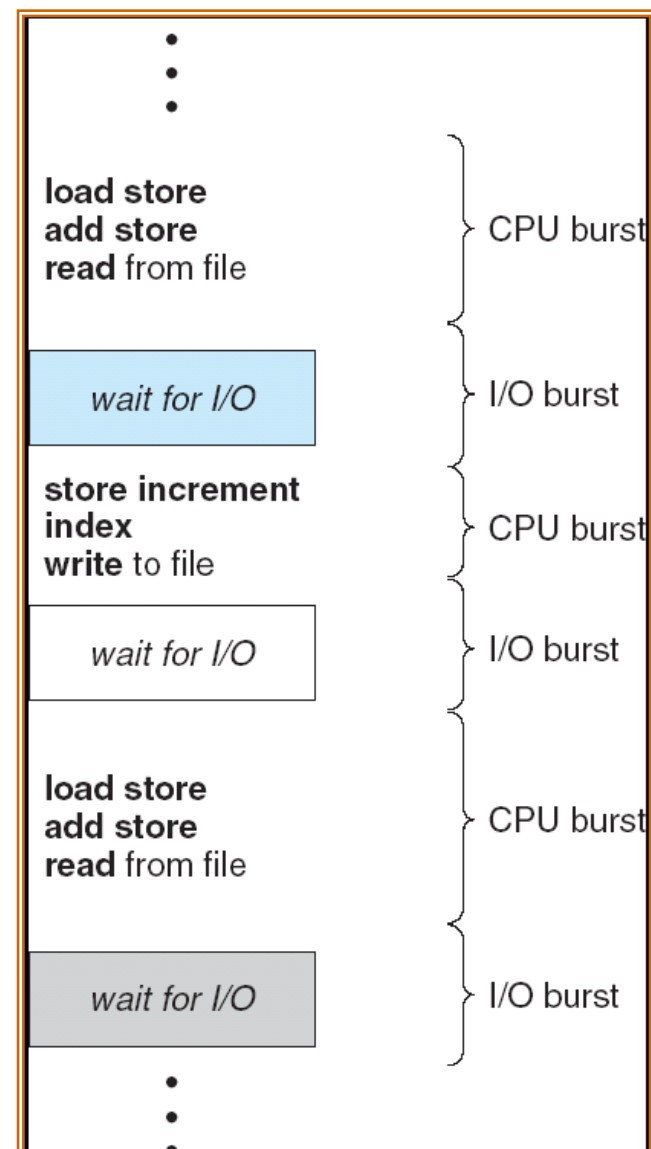


Распоредување (Scheduling)

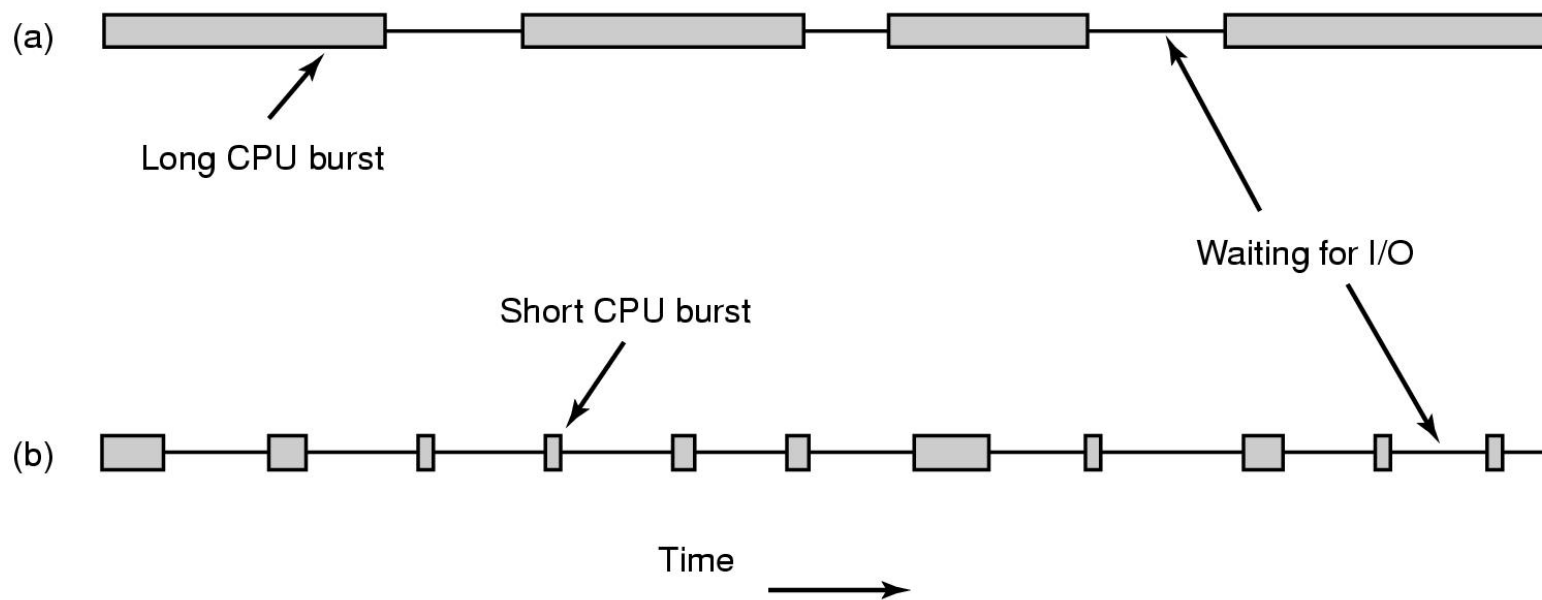
- Кога повеќе од еден процес се во ready состојба треба да се одлучи кој од нив ќе го добие CPU
- Делот од оперативниот систем кој ги прави овие одлуки се нарекува распоредувач (scheduler)
- Распоредувачот работи врз основа на некој алгоритам за распоредување

CPU v.s. I/O извршување

- Извршувањето на процесите се состои од циклуси на CPU извршување и I/O чекање



Поделба на процесите



a) CPU доминантен процес

б) I/O доминантен процес

Кога се распоредуваат процеси?

1. Кога еден процес креира процес-дете
2. Кога процес терминира
3. Кога процес од извршување доаѓа во состојба на чекање (I/O барање или повикување wait до завршување на процес дете)
4. Процес оди од состојба на извршување во состојба спремен (се случил прекин)
5. Процес се менува од состојба на чекање во спремна состојба (завршил I/O)



Типови ресурси

- Ресурсите можат да бидат поделени во една од двете групи
- Типот на ресурси одредува како ОС го управува

1. **Non-preemptible ресурси** (кои не се одземаат)

- Откако е доделен ресурсот, не може да се користи се' додека не биде ослободен
- Потребни се повеќе инстанци од ресурсот
- Пример: Блок на диск
- ОС менаџмент: **алокација**
 - Одлучува кој процес го добива кој ресурс

2. **Preemptible ресурси** (кои се одземаат)

- Може да е одземен ресурсот, потоа се враќа
- Може да има само еден од овој тип ресурс
- Пример: CPU
- ОС менаџмент: **распоредување**
 - Одлучува по кој редослед се опслужуваат барањата
 - Одлучува колку долго процесот го држи ресурсот



Како се извршува распоредувањето?

- Во дадени периодични временски моменти (прекини од системскиот часовник)
 - **Non-preemptive**: процесот се извршува додека тој самиот не блокира или заврши
 - **Preemptive**: процесот може да го користи CPU во рамките на некој предефиниран максимален временски период

Мерки за перформанси на распоредувачките алгоритми

- **Проток** (throughput) – број на процеси завршени во единица време
- **Време до завршување, време во систем** (turnaround time) – времето поминато од доставувањето до комплетирањето на процесот
- **CPU Искористеност** - процент од времето во кое процесорот е зафатен
- **Време на чекање** – време на чекање (за CPU) поминато во редицата на спремни процеси
- **Време на одзив** – во интерактивни системи времето на завршување не е добра мерка. Време од поднесување барање до прв добиен резултат



Категории на алгоритми за распоредување

- Во зависност од типот на системот постојат различни алгоритми
 - Пакетни (Batch)
 - Пресметка на камати, процесирање на случаи кај осигурителна компанија
 - Нема корисници
 - Non preemptive или preemptive со долги временски периоди
 - Интерактивни
 - Preemptive
 - Во реално време (Real-Time)
 - треба да бидат исполнети роковите
 - Понекогаш нема потреба од preemptive

Цели на алгоритмите за распоредување (1)

- За сите системи
 - Фер – секој процес да добие соодветно време на CPU
 - Спроведување на политиката за распоредување
 - Баланс – сите делови на системот да работат
- За пакетни системи
 - Максимизирај проток
 - Минимизирај време на завршување (од влегување во системот до терминирање)
 - CPU искористеност – CPU да работи цело време

Цели на алгоритмите за распоредување (2)

- Интерактивни системи
 - Време на одзив – одговори на барања брзо
 - Пропорционалност – потребно е да се исполнети корисничките барања
- Системи за работа во реално време
 - Почитувај рокови – без губење податоци
 - Предвидливост – избегни губење квалитет во мултимедијални системи



Interactive systems: Delay & User Reaction

Delay & User Reaction	
0 - 16ms	People are exceptionally good at tracking motion, and they dislike it when animations aren't smooth. Users perceive animations as smooth so long as 60 new frames are rendered every second. That's 16ms per frame, including the time it takes for the browser to paint the new frame to the screen, leaving an app about 10ms to produce a frame.
0 - 100ms	Respond to a user action within this time window and users feel like the result is immediate. Any longer, and the connection between action and reaction is broken.
100 - 300ms	Users experience a slight perceptible delay.
300 - 1000 ms	Within this window, things feel part of a natural and continuous progression of tasks. For most users on the web, loading pages or changing views represents a task.
1000+ms	Beyond 1 second, the user loses focus on the task they are performing.
10,000+ms	The user is frustrated and is likely to abandon the task; they may or may not come back later.



Распоредување во Batch-системите

- Прв – Дошол Прв – Услужен
(First-Come First-Served)
- Најкратката задача прва
(Shortest Job First)
- Најкраткото преостанато време следно
(Shortest Remaining Time Next)

Прв – Дошол Прв – Услужен (First-Come First-Served)

- Наједноставен алгоритам
- Лесно се програмира. Се користи поврзана листа
- Проблеми:
 - Ако има CPU-bound процес со 1 I/O пристап во секунда, и многу I/O-bound процеси кои треба да завршат по 1000 I/O операции, а им треба сосема малку CPU.
 - I/O-bound процесите ќе завршат за 1000 сек затоа што треба да чека 1000 прекини на секоја секунда кога би се прекинал CPU-bound процесот поради негово читање од диск.
 - Ако насилно се прекинува CPU-bound процесот секои 10ms тогаш I/O-bound процесите ќе завршат за 10 сек.

Прв – Дошол Прв – Услужен (First-Come First-Served)

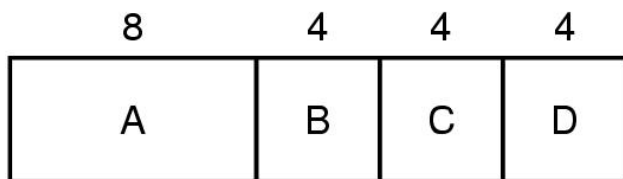
- Секој процес кој што ќе премине во состојба на спремен процес се придружува на крајот на редицата на спремни процеси за извршување.
- Прв започнува да се извршува оној процес кој што прв влегол во оваа редица на спремни процеси, втор започнува процесот кој влегол во редицата по него, итн.
- Средното време на чекање со овој алгоритам е често доста долго.

Најкратката задача прва (Shortest Job First)

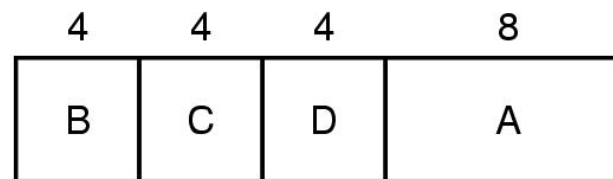
- Со овој алгоритам CPU се доделува на оној процес кој што има најмало очекувано време на извршување.
- Овој алгоритам им дава предност на извршување на малите процеси.
- Доколку процесите се со иста должина, тогаш се користи FCFS за да се разреши дилемата.
- Дава оптимални резултати кога процесите доаѓаат речиси истовремено

Најкратката задача прва (Shortest Job First)

- Потреба од познавање на времето на извршување на зададените работи
- Пример за извршување



(a)



(b)

- Средно време во системот
 - а) (A-8, B -12, C-16, D-20) **14 мин.**
 - б) **11 мин.**
- Алгоритамот дава најдобри времиња на завршување ако на почетокот се познати сите задачи кои треба да се завршат

Најкраткото преостанато време следно (Shortest Remaining Time Next)

- Preemptive верзија на SJF
- Треба да се познава времето на извршување
- Се пресметува преостанатото време за извршување

Најкраткото преостанато време следно (SRTN)

- Се избира процесот чие што преостанато време за извршување е помало.
- Кога ќе пристигне нов процес во редицата на спремни процеси, неговото вкупно време се споредува со преостанатото време на извршување на процесот кој веќе е во состојба на извршување.
- Ако на новиот процес му е потребно помало време за да заврши отколку на тековниот процес, тогаш тековниот процес се прекинува и новиот процес преминува во состојба на активен.

Распоредување во интерактивните системи

- Round-Robin
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling



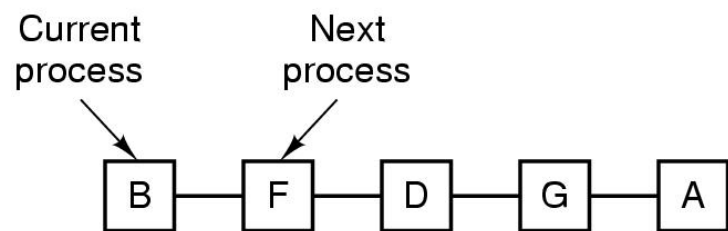
Кружно распоредување - Round Robin (RR)

- За секој процес се одделува мала количина CPU време (*временски квантум*)
 - Откако ќе истече времето процесот се вади од редица на спремни (preempted) и се додава на крајот на редицата на спремни
- Ако има n процеси во редица на спремни и квантумот е q , тогаш секој процес добива $1/n$ од CPU времето во порции од најмногу q временски единици наеднаш. Ниту еден процес не чека повеќе од $(n-1)q$ временски единици.
- Перформанси
 - q големо \Rightarrow FIFO
 - q мало $\Rightarrow q$ мора да е доволно големо во однос на времето за промена на контекст
- Големина на квантум кај различни оперативни системи
 - Windows Server – 120ms
 - Windows desktop верзии - 20-60ms
 - UNIX системи – 100ms

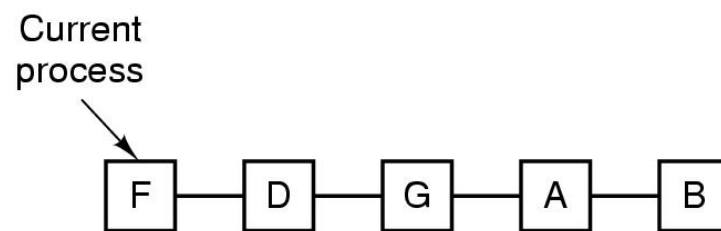


Round-Robin

- а) Листа на процеси спремни за извршување
- б) Листа на процеси спремни за извршување откако процесот В го искористил своето процесорско време (quantum)

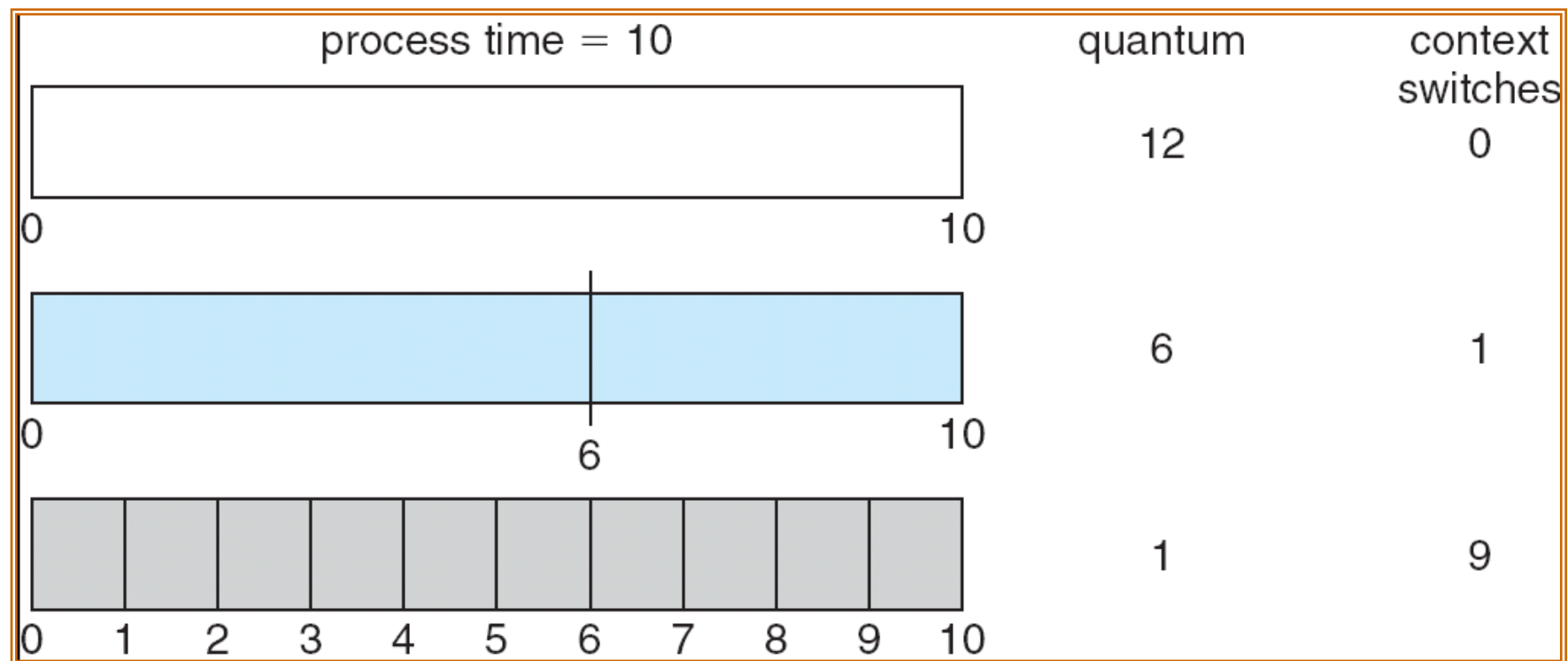


(a)



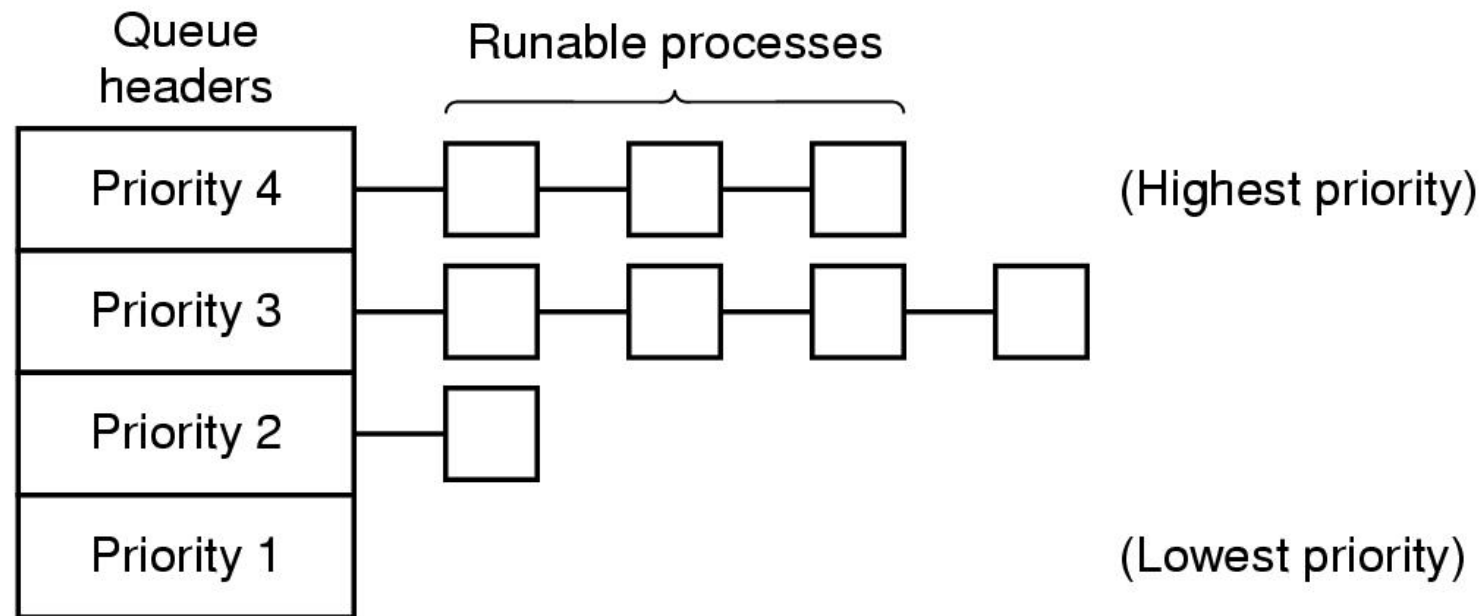
(b)

Временски квантум и време потребно за промена на контекст



Распореѓување со приоритети (Priority Scheduling)

- Дефинирање на различни приоритети на процесите според нивната важност

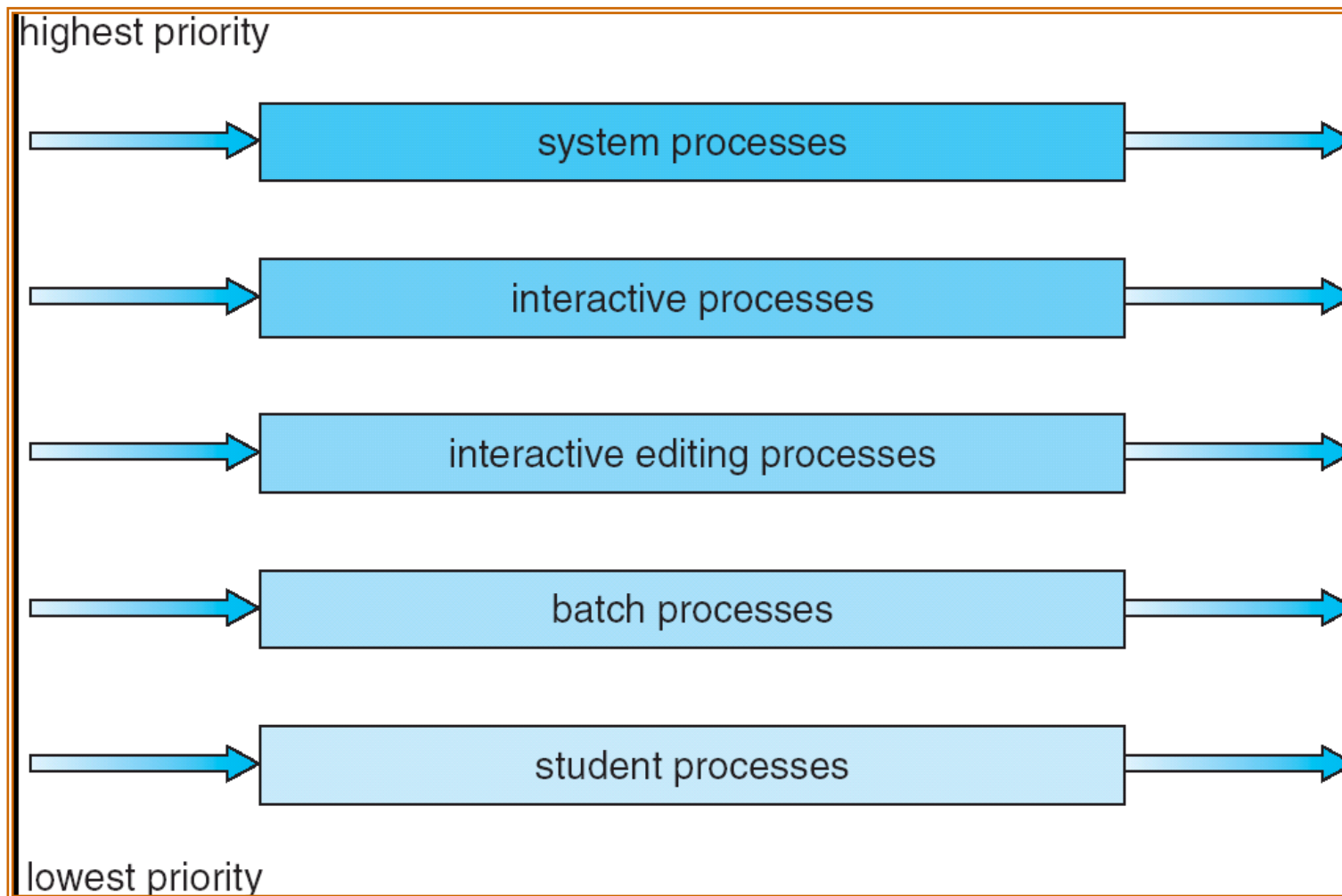


Распоредување по приоритети

- На секој процес му се придружува цел број кој го означува неговиот приоритет
- CPU се доделува на процесот со најголем приоритет (најмал број \equiv највисок приоритет)
 - SJF е распоредување со приоритети каде приоритетот е одлучен според предвиденото следно време на извршување
- Проблем \equiv Изгладнување – процеси со помал приоритет може никогаш да не се извршат
- Решение \equiv како поминува времето му се зголемува приоритетот на процесот



Повеќекратни Редови



Приоритет кај повеќекратни Редови

- Намалување на бројот на промени на процеси
- Се дефинираат приоритетни класи и секоја задача добива различно време за извршување

Приоритет	Периоди на извршување
највисок	1
највисок-1	2
највисок-2	4
највисок-3	8

Распоредување кај повеќекратни редови

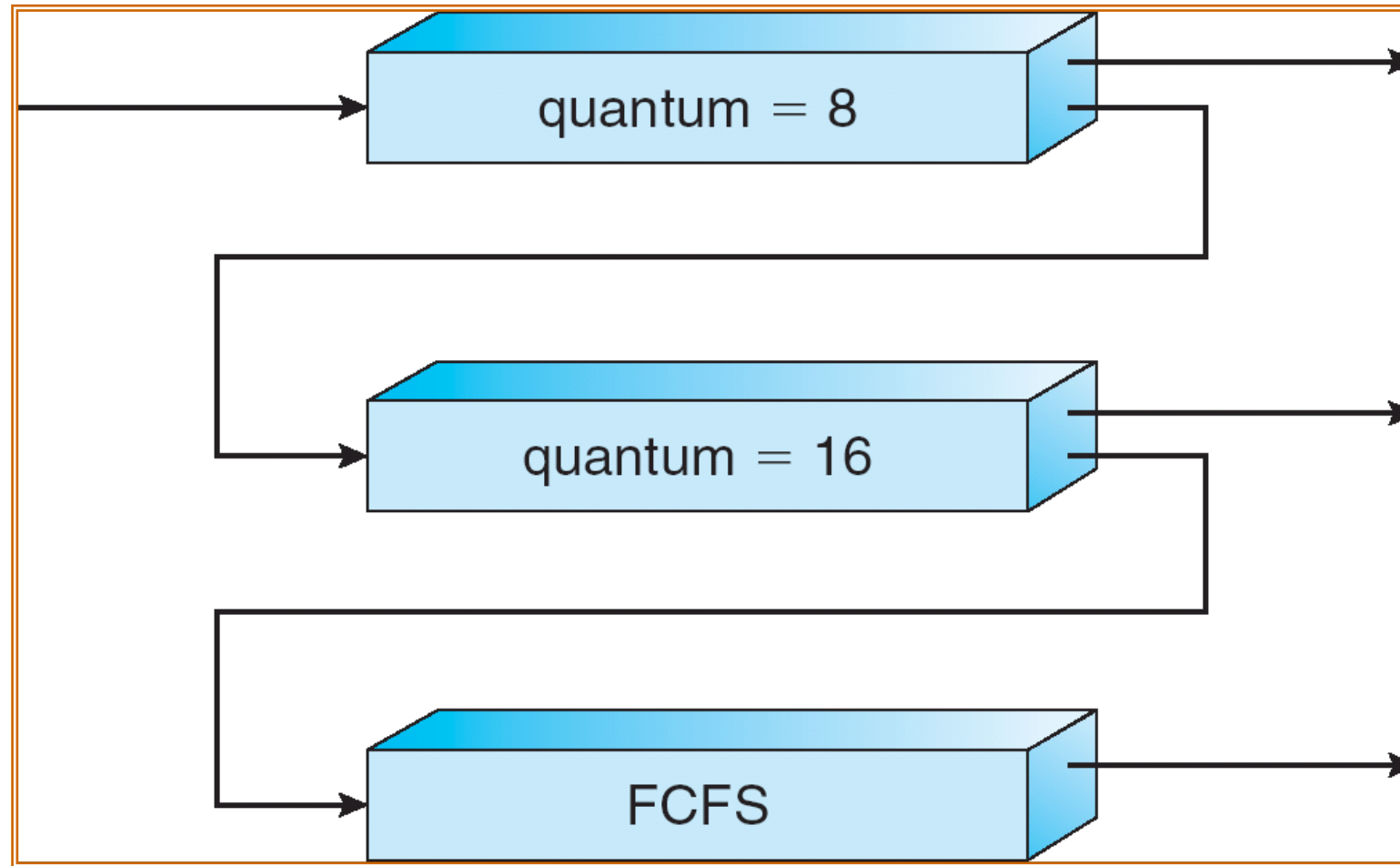
- Процесот може да има променлив приоритет
- Распоредувачот на повеќекратни редови со повратна информација работи според следните параметри:
 - број на редови
 - распоредувачки алгоритам за секој ред
 - метод кој кажува кога еден процес добива повисок приоритет
 - метод кој кажува кога му се намалува приоритетот на процес
 - метод кој одлучува во кој ред се сместува процесот кога му треба опслужување



Повеќекратни редови со повратна врска (Multilevel Feedback Queues)

- 3 редови:
 - Q_0 – RR со квантум 8 ms
 - Q_1 – RR со квантум 16 ms
 - Q_2 – FCFS
- Распоредување
 - Нов процес влегува во редот Q_0 . Кога ќе добие CPU, задачата добива 8 ms. Ако не заврши за 8 ms, се преместува во Q_1 .
 - Во Q_1 задачата се распоредува и добива дополнителни 16 ms. Ако не се заврши, се преместува во Q_2 .

Повеќекратни редови со повратна врска (Multilevel Feedback Queues)



Најкраткиот процес следен (Shortest Process Next)

- Верзија на SJF за интерактивни системи
- Проблем: кој од процесите кои тековно работат е најкус?!?
- Секоја команда се разгледува како посебна работа и се мери нејзиното време на извршување
- Врз основа на проценка прво се извршува најкратката команда



Едноставна проценка на време на извршување (Aging)

- Следното време на опслужување со CPU на еден процес може да се предвиди според времето на опслужување на истиот процес во минатото

- Нека

T_n е должина на n-то опслужување на тој процес (најсвежа информација, скорешна историја - recent history),

F_{n+1} е нашата предвидена вредност за следната должина на процесот (F_n е историја од минатото –past history)

Дефинираме рекурентна врска:

$$F_{n+1} = \alpha \cdot T_n + (1 - \alpha) \cdot F_n,$$

(α го контролира влијанието на скората историја од минатото)

Едноставна проценка на време на извршување (Aging) (2)

- Проценка на времетраење на следно извршување на процес :
- Нека имаме две претходни вредности за времето на извршување на еден процес, T_0 и T_1

$$F_1 = T_0, F_2 = \alpha \cdot T_1 + (1 - \alpha) \cdot F_1,$$
$$F_K = \alpha \cdot T_{K-1} + (1 - \alpha) \cdot F_{K-1}, K=3, 4, \dots$$

за $\alpha = 1/2$ (скората и историјата од минатото нека имаат еднаква тежина)

$$T_0,$$

$$T_0/2 + T_1/2,$$

$$T_0/4 + T_1/4 + T_2/2,$$

$$T_0/8 + T_1/8 + T_2/4 + T_3/2, \dots$$

(историјата се помалку влијае на иднината)

- - SJF е единствен оптимален (според времето на чекање) кога сите процеси доаѓаат во ист момент, во спротивно не е



Гарантирано распоредување (Guaranteed Scheduling)

- Потреба од гарантирање дека секој корисник ќе добие одредено процесорско време
- Се мери потрошеното време од секој корисник
- Се одредуваат приоритети за процесите врз основа на соодносот на реално потрошеното време и предефинираното време



Распоредување со Лотарија (Lottery Scheduling)

- За едноставна реализација на распоредување во кое ќе се доделат некои предефинирани проценти на процесорско време се користи распоредување со Лотарија
- Секој процес/корисник добива одреден број на тикети кои му овозможуваат користење на даден ресурс
- Кога ќе дојде време за распоредување на нов процес да го користи ресурсот се генерира случаен број во опсегот од 1 до бројот на тикети. Процесот кој го има бараниот број на тикет го добива ресурсот
- Процесите имаат по повеќе тикети. Бројот на тикети е право пропорционален со веројатноста дека тие ќе бидат избрани, односно ќе го добијат ресурсот

Фер-Деливо распоредување (Fair-Share Scheduling)

- Се користи за решавање на проблемот кога даден корисник, за да добие повеќе процесорско време, може да стартува повеќе процеси
- Идејата е да се извршува по еден процес од секој корисник
- Ако еден корисник има 4 процеси А,Б,В,Г а друг има еден процес Д тогаш процесите ќе се извршуваат по следниов редослед
 - АДБДВДГДАД...
- Ако првиот корисник треба да добие двапати повеќе време тогаш извршувањето би било
 - АБДВГДАБДВГД...



Распоредување во системи кои работат во реално време

- Главна цел е работата да се заврши во дадениот рок
- Мора да се пресмета дали перформансите на компјутерскиот систем задоволуваат
 - Ако има m периодични настани
 - Настанот i се случува со периода P_i и за да се опслужи бара C_i секунди
- Тогаш оптоварувањето може да е поднесе ако

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

Задача 1

- Нека се дадени 5 процеси: A, B, C, D и E и соодветно нивните времиња на извршување: 25, 22, 33, 35 и 28. Процесите пристигнуваат (скоро) истовремено.
- Да се нацрта соодветниот Гантограм (временски дијаграм) за секој од распределувачките алгоритми дадени во табелата и да се пополнат соодветните полиња во неа.



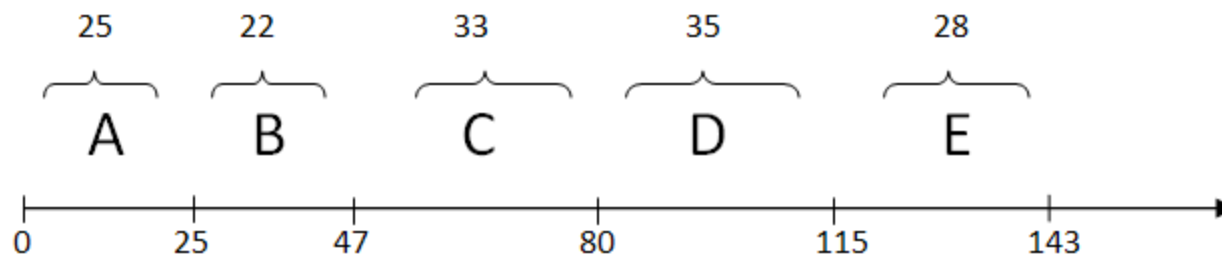
Задача 1

процес	време на извршување	време на пристигнување	Алгоритми за распределување					
			FCFS		SJF		Round Robin со квантум 10 ms	
			време на чекање	вкупно време во систем	време на чекање	вкупно време во систем	време на чекање	вкупно време во систем
A	25	0						
B	22	0						
C	33	0						
D	35	0						
E	28	0						



Решение:

- Гантограм за FCFS е:



- Вкупното време на чекање на процесите е:

Процес	Време на чекање
A	0
B	25
C	47
D	80
E	115

Просечно време на чекање: 53.4 ms

Решение:

- Вкупното време во системот на секој од процесите е времето кое поминало од нивното пристигнување, па сè до нивното комплетно завршување.

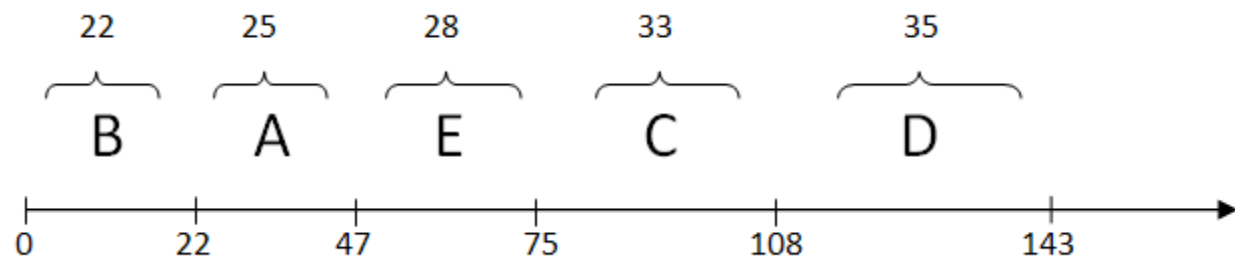
Процес	Време во системот
A	25
B	47
C	80
D	115
E	143

Просечно време на задржување во системот: 82 ms



Решение:

- Гантограм за SJF е:



- Вкупното време на чекање на процесите е:

Процес	Време на чекање
A	22
B	0
C	75
D	108
E	47

Просечно време на чекање: 50.4 ms

Решение:

- Вкупното време во системот на секој од процесите е времето кое поминало од нивното пристигнување, па сè до нивното комплетно завршување.

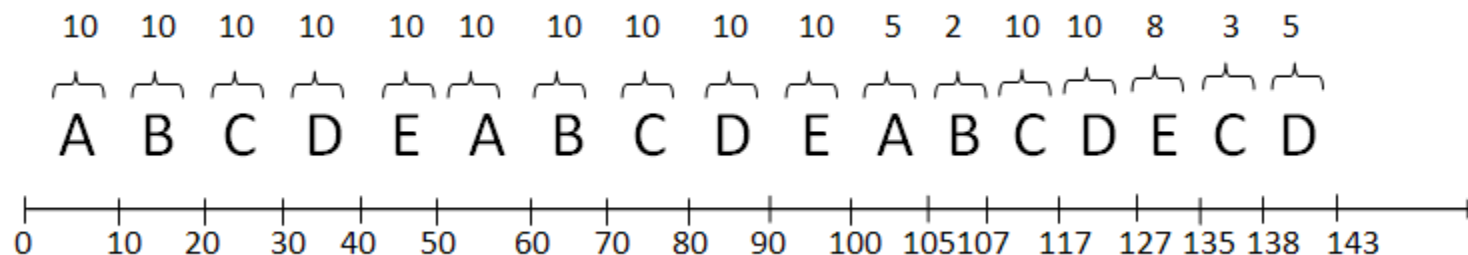
Процес	Време во системот
A	47
B	22
C	108
D	143
E	75

Просечно време на задржување во системот: 79 ms



Решение:

- Гантограм за RR со квантум 10 ms е:



- Вкупното време на чекање на процесите е:

Процес	Време на чекање
A	$0 + (50 - 10) + (100 - 60) = 80$
B	$10 + (60 - 20) + (105 - 70) = 85$
C	$20 + (70 - 30) + (107 - 80) + (135 - 117) = 105$
D	$30 + (80 - 40) + (117 - 90) + (138 - 127) = 108$
E	$40 + (90 - 50) + (127 - 100) = 107$

Просечно време на чекање: 97 ms

Решение:

- Вкупното време во системот на секој од процесите е времето кое поминало од нивното пристигнување, па сè до нивното комплетно завршување.

Процес	Време во системот
A	105
B	107
C	138
D	143
E	135

Просечно време на задржување во системот: 125.6 ms



Решение:

- После пресметките табелата би изгледала вака:

			Алгоритми за распределување					
процес	време на извршување	време на пристигнување	FCFS		SJF		Round Robin со квантум 10 ms	
			време на чекање	вкупно време во систем	време на чекање	вкупно време во систем	време на чекање	вкупно време во систем
A	25	0	0	25	22	47	80	105
B	22	0	25	47	0	22	85	107
C	33	0	47	80	75	108	105	138
D	35	0	80	115	108	143	108	143
E	28	0	115	143	47	75	107	135
Просек			53.4	82	50.4	79	97	125.6

Задача 2

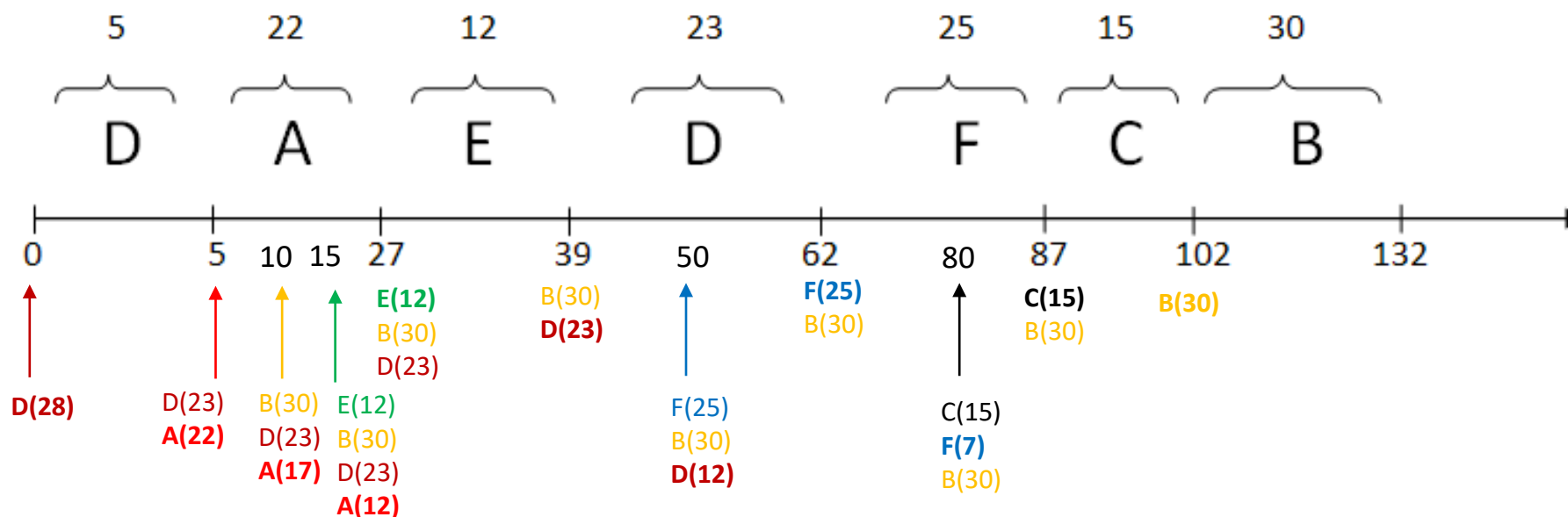
- Со помош на алгоритмот SRTN да се распределат 6 процеси A, B, C, D, E и F кои што имаат соодветно 22, 30, 15, 28, 12 и 25 времиња на извршување. Меѓутоа, процесите не се придружуваат истовремено на редицата на спремни процеси. Нивните времиња на придружување се: 5, 10, 80, 0, 15 и 50 ms соодветно.
- Да се нацрта соодветниот Гантограм и да се пресмета времето на одзив на секој од процесите во системот.

Решение:

P: A, B, C, D, E и F

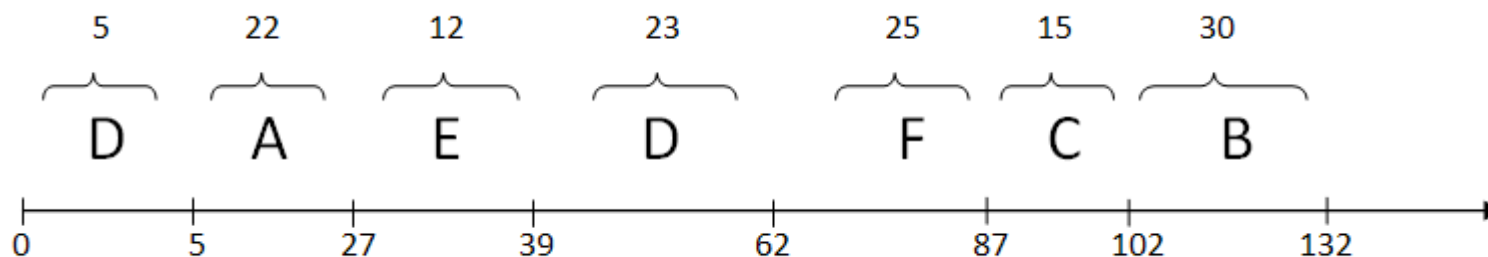
d: 22, 30, 15, 28, 12 и 25

T: 5, 10, 80, 0, 15 и 50



Решение:

- Гантограм за SRTN е:



- Времето на одзив се пресметува како резултат од времето на поднесување барање, до прв добиен резултат.
 - Притоа, ќе претпоставиме дека процесот веднаш штом ќе стане активен го дава својот прв резултат.

Решение:

- Времето на одзив за процесот А би било:
 - време на поднесување барање – 5ms
 - прв добиен резултат – 5ms
 - време на одзив: $5 - 5 = 0\text{ms}$
- Времето на одзив за процесот В би било:
 - време на поднесување барање – 10ms
 - прв добиен резултат – 102ms
 - време на одзив: $102 - 10 = 92\text{ms}$
- Времето на одзив за процесот С би било:
 - време на поднесување барање – 80ms
 - прв добиен резултат – 87ms
 - време на одзив: $87 - 80 = 7\text{ms}$



Решение:

- Времето на одзив за процесот D би било:
 - време на поднесување барање – 0ms
 - прв добиен резултат – 0ms
 - време на одзив: $0 - 0 = 0ms$
- Времето на одзив за процесот E би било:
 - време на поднесување барање – 15ms
 - прв добиен резултат – 27ms
 - време на одзив: $27 - 15 = 12ms$
- Времето на одзив за процесот F би било:
 - време на поднесување барање – 50ms
 - прв добиен резултат – 62ms
 - време на одзив: $62 - 50 = 12ms$



Распределување кај системи во реално време

- Систем во реално време е систем каде што времето ја игра најзначајна улога.
- Примери за таков вид системи се:
 - Системот во компакт диск плеерите, кој што ги зема битовите онака како што пристигнуваат од диск единицата и ги претвора во музика во многу краток временски интервал.
 - Ако пресметката би зела подолго време, музиката би звучела чудно.
 - Системите во авионите, системите кои ги контролираат роботите во една автоматизирана фабрика, мониторирањето на пациентите во одделите за интензивна нега, итн.



Распределување кај системи во реално време

- Настаните кои се случуваат во овие системи може да бидат периодични (се случуваат во регуларни интервали) или неперидични (се случуваат непредвидливо).
- Дали еден систем во реално време може да одговори на повеќепериодичен поток од настани (процеси), зависи од две работи:
 - колку време му треба на секој од процесите за да се изврши;
 - која е неговата периода на појавување;

Распределување кај системи во реално време

- Ако имаме m периодични процеси и секој процес се појавува со периода P_i и побарува C_i секунди од времето на CPU, тогаш системот ќе може да управува со тој поток само ако е задоволено следново:

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

- Системите во реално време кои што го задоволуваат овој критериум се наречени **распределиви системи**.

Задача 3:

- Нека е даден еден систем во реално време со три периодични настани. Процесите кои што ги отсликуваат овие настани се со периоди од 100, 200 и 500 msec, соодветно. Ако овие процеси побаруваат 50, 30 и 100 msec од времето на CPU, дали системот ќе може да управува со овие процеси, односно дали системот ќе биде распределив?
- Ако се додаде 4-ти процес со периода од 1 sec, колкаво треба да биде неговото максимално време на извршување за да може системот да биде распределив?

Решение:

- Системот за да биде распределив треба да го задоволува следново неравенство:

$$\sum_{i=1}^3 \frac{C_i}{P_i} \leq 1$$

- каде што $C_1 = 50$, $C_2 = 30$, $C_3 = 100$ и $P_1 = 100$, $P_2 = 200$, $P_3 = 500$, па имаме:

$$\sum_{i=1}^3 \frac{C_i}{P_i} = \frac{50}{100} + \frac{30}{200} + \frac{100}{500} = 0.5 + 0.15 + 0.2 = 0.85 \leq 1$$

- Значи овој систем може да управува со потокот од дадените повеќе периодични настани.



Решение:

- Ако се додаде уште еден процес со периода од 1sec = 1000 msec тогаш за системот да биде распределив треба да биде исполнето следново:

$$\frac{50}{100} + \frac{30}{200} + \frac{100}{500} + \frac{x}{1000} \leq 1$$

- Од каде што следува дека $x \leq 0.15$ или процесот да има максимално време на извршување од 150 msec.



Задача 4:

- За предвидување на времетраење на процеси се користи алгоритам на стареење со $\alpha=1/2$. Ако времетраењата на претходните 4 извршувања биле 40, 20, 40 и 15 ms, кое е предвидувањето за следното време на извршување?

Решение:

- Алгоритамот на стареење го пресметува следното предвидување како тежинска сума од претходните две извршувања, т.е. ако T_0 и T_1 се две последователни времиња на извршување, тогаш T_2 се пресметува како:

$$T_2 = \alpha \cdot T_1 + (1 - \alpha) \cdot T_0$$



Решение:

- Проценката за времетраењето на следното извршување на процес можеме да ја направиме на следниов начин:
- Нека ги имаме двете претходни вредности за времето на извршување на процесот, T_0 и T_1

$$F_1 = T_0,$$

$$F_2 = \alpha \cdot T_1 + (1 - \alpha) \cdot F_1,$$

$$F_k = \alpha \cdot T_{k-1} + (1 - \alpha) \cdot F_{k-1}, k=3, 4, \dots$$



Решение:

- Ако $\alpha=1/2$ се добиваат следните последователни предвидувања:

$$T_0$$

$$T_0/2 + T_1/2,$$

$$T_0/4 + T_1/4 + T_2/2,$$

$$T_0/8 + T_1/8 + T_2/4 + T_3/2,$$

- За вредностите во задачата ќе имаме предвидување:

$$T_0/8 + T_1/8 + T_2/4 + T_3/2 =$$

$$40/8 + 20/8 + 40/4 + 15/2 = 25ms$$



Решение:

- $T_0 = 40$
- $T_1 = 20$
- $T_2 = 40$

$$F_1 = T_0 = 40$$

$$F_2 = \alpha \cdot T_1 + (1 - \alpha) \cdot F_1 = 0.5 \cdot 20 + 0.5 \cdot 40 = 30$$

$$F_3 = \alpha \cdot T_2 + (1 - \alpha) \cdot F_2 = 0.5 \cdot 40 + 0.5 \cdot 30 = 35$$



Решение:

- $T_0 = 40$
- $T_1 = 20$
- $T_2 = 40$
- $\alpha = 0.2$

$$F_1 = T_0 = 40$$

$$F_2 = \alpha \cdot T_1 + (1 - \alpha) \cdot F_1 = 0.2 \cdot 20 + 0.8 \cdot 40 = 36$$

$$F_3 = \alpha \cdot T_2 + (1 - \alpha) \cdot F_2 = 0.2 \cdot 40 + 0.8 \cdot 36 = 36.6$$



Времињата на пристигнување и извршување на процесите A, B, C, D, E се дадени во следната табела:

Име на процес	Време на пристигнување	Време на извршување
A	1	13ms
B	8	13ms
C	8	13ms
D	1	16ms
E	12	14ms

Ако се користи **Multilevel Feedback Queues** каде Q0 користи Round Robin со квантум=2, Q1 користи Round Robin со квантум=4 и Q2 користи First Come First Served (FCFS), тогаш времето на одзив, вкупното време во системот и времето за чекање за процесите (изразено во ms) изнесува:

Процес	Време на одзив	Време во системот	Време на чекање
A	<input type="text"/>	<input type="text"/>	<input type="text"/>

Windows 11

- 1. Priority-Based Scheduling:** Windows assigns a priority level to each thread, with higher-priority threads getting more CPU time than lower-priority ones. The system dynamically adjusts these priorities to ensure responsiveness, especially for user-interactive applications.
- 2. Time Slicing:** The scheduler allocates time slices (also known as quanta) to each thread. When a thread's time slice expires, the scheduler can switch to another thread. Higher-priority threads may receive longer time slices than lower-priority threads.
- 3. Dynamic Prioritization:** The scheduler may boost the priority of threads dynamically to improve the responsiveness of applications. For example, when a user interacts with an application, its threads may receive a priority boost.



Windows 11

- 1.Symmetric Multiprocessing (SMP) and Hyperthreading:** Windows 11 can distribute threads across multiple CPU cores or logical processors, balancing the load to improve performance.
- 2.Processor Affinity and Scheduling:** The scheduler takes processor affinity into account, which is the assignment of specific threads to specific cores or processors. This can be managed by Windows automatically or set by applications.
- 3.I/O and Interrupt Prioritization:** The system gives priority to I/O and interrupts to ensure that hardware devices are responsive and that I/O-bound processes do not suffer from unnecessary delays.



Windows 11

- 1.Foreground vs. Background Differentiation:** Windows tends to prioritize the applications in the foreground to ensure they are responsive to the user. Background applications may have lower priority and receive less CPU time.
- 2.Scheduler Hints:** Windows 11 introduced features that allow the OS to be more energy-efficient, such as 'Eco mode', which de-prioritizes specific applications to conserve power. This is part of broader scheduling decisions that include considerations for power efficiency.
- 3.Multilevel Feedback Queue:** Windows uses a multilevel feedback queue that allows processes to move between different priority levels based on their behavior and needs. This helps balance the system's responsiveness with the need to run background tasks efficiently.
- 4.Priority Levels:** Windows provides 32 priority levels: 16 "real-time" levels and 16 "variable" levels. User-mode processes typically run at variable levels, while the system can reserve real-time levels for the most time-critical tasks.



Linux

- Linux uses the Completely Fair Scheduler (CFS) as its default scheduling algorithm since version 2.6.23. CFS is designed to provide fair CPU allocation to running processes:
- **Fairness:** The key goal of CFS is to model an "ideal, precise multitasking CPU" on real hardware. Each task gets to run for a proportionate share of the CPU time.
- **Red-Black Tree:** CFS uses a red-black tree data structure to maintain a "timeline" of processes, aiming to run each process for a small slice of time, after which the process is placed back in the tree based on its virtual runtime.
- **Task Weighting:** Each task is assigned a weight, and CFS uses these weights to ensure that tasks get CPU time in proportion to their weights, achieving fairness.
- **Nanosecond Granularity:** CFS uses nanosecond granularity in accounting for the process's time, allowing it to be very precise in its calculations.
- **Group Scheduling:** CFS supports group scheduling, where each group is allocated a share of CPU time, and processes within the group compete for that share.





Прашања?



"Ss. Cyril and Methodius" University - Skopje
FACULTY OF COMPUTER
SCIENCE AND ENGINEERING

