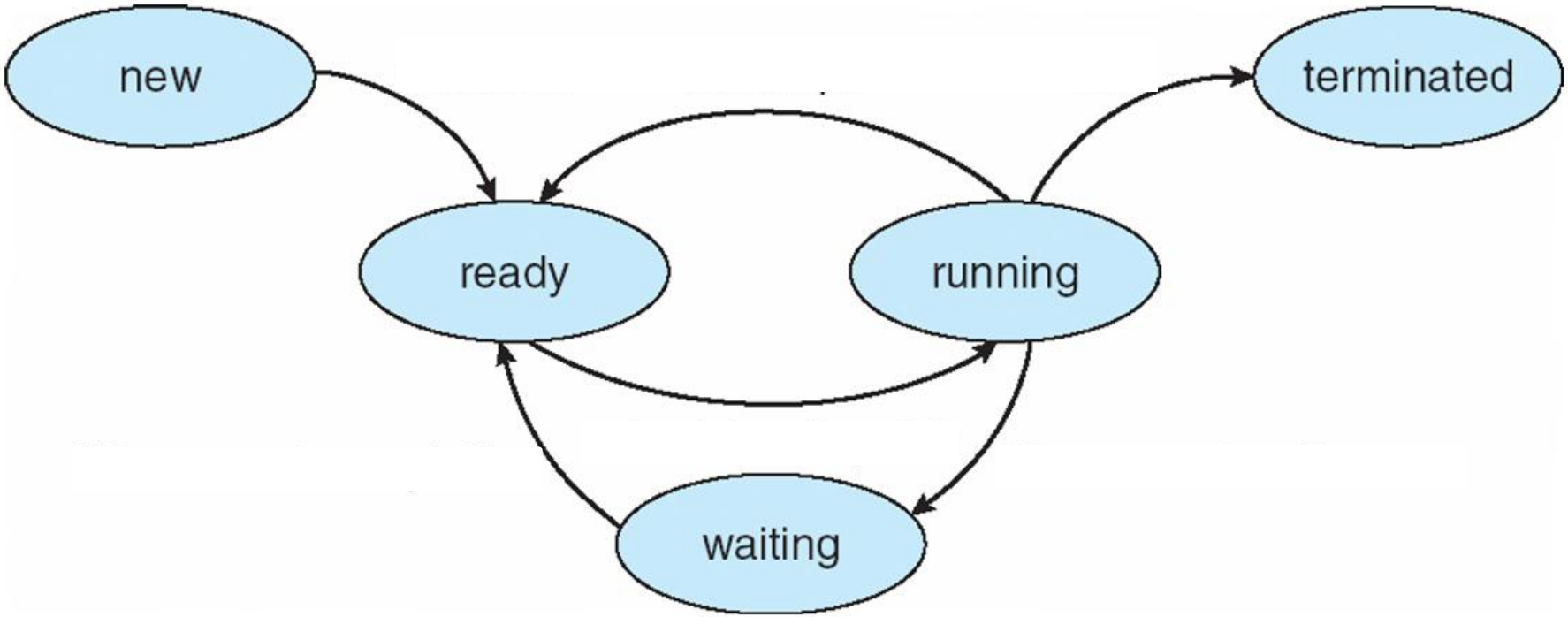
Планиране на процесора

доц. д-р инж. Христо Вълчанов

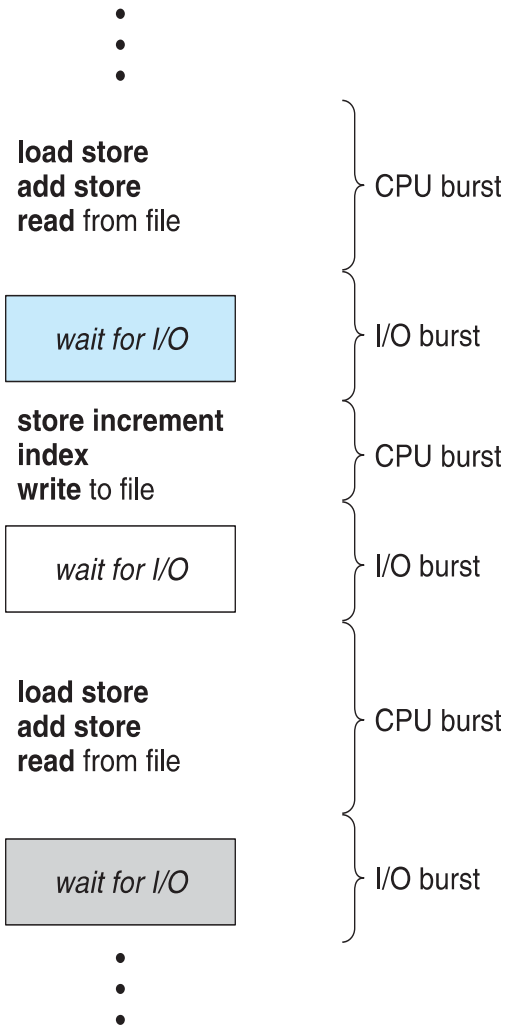
http://cs.tu-varna.bg

Състояния на процесите

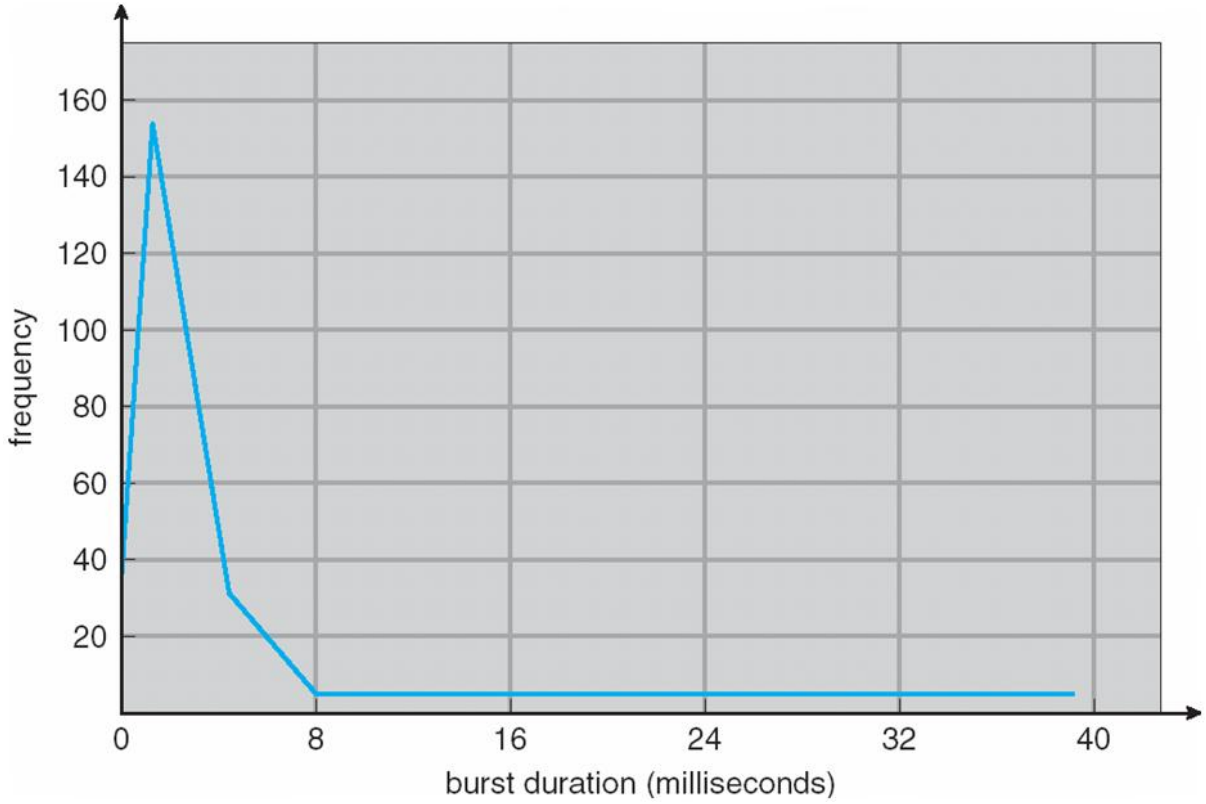


Основни концепции

- Максимално използване на CPU се получава чрез мултипрограмно изпълнение
- CPU–I/O Burst Cycle – изпълнението на процес включва цикли работа на CPU и I/O изчаквания
- **CPU burst** следван от **I/O burst**
- CPU burst разпределянето е от голямо значение



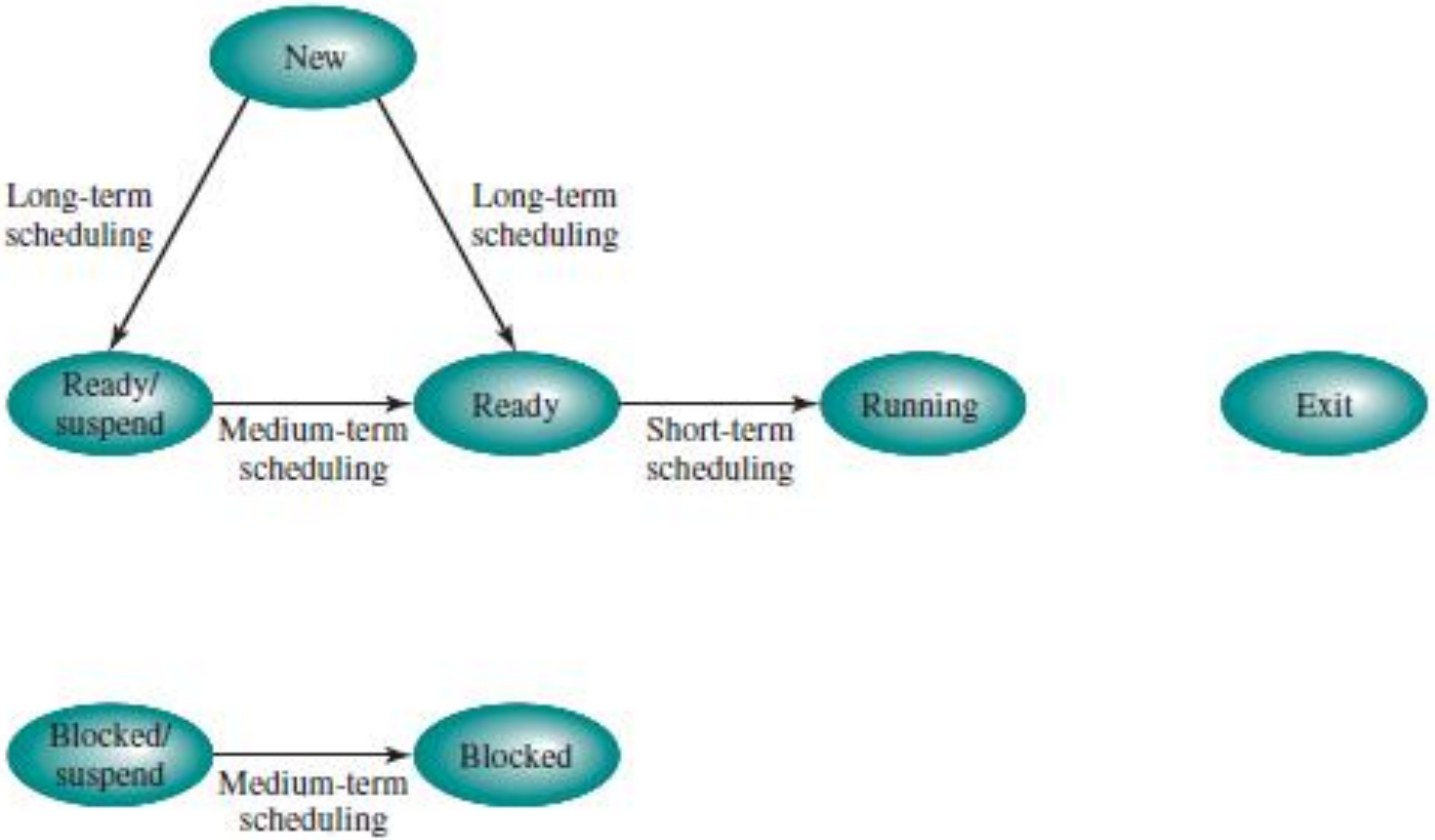
Хистограма на CPU-burst времената



Видове планирания

- **Short-term scheduling –** кой наличен процес ще се изпълни върху процесора;
- **Medium-term scheduling** – кои процеси да се добавят към готовите в паметта;
- **Long-term scheduling** – при създаване на нов процес;
- **I/O scheduling** – за кой процес I/O заявки да се обслужат от налично I/O устройство.

Видове планирания (2)



CPU планировчик (Scheduler)

- **Short-term scheduler** избира един от процесите *от Ready Queue* и заема за него CPU
  - Опашката може да бъде подредена по различни начини
- Кога се изисква планиране на CPU – ако процес:
  1. Се превключва от Running в Waiting състояние (заявка за I/O)
  2. Се превключва от Running в Ready (прекъсване)
  3. Се превключва от Waiting в Ready (завършване I/O)
  4. Завършва

Preemptive и non-preemptive планиране

- **Non-preemptive** – след като CPU е зает от процес, процесът притежава CPU, докато го освободи или при завършване или при преминаване в waiting състояние
- **Preemptive** -- CPU може да бъде даден на друг процес по всяко време. Проблемни ситуации:
  - Достъп до споделени данни
  - Изпълнение в kernel mode
  - Настъпване на прекъсвания по време на критични действия на ОС

Диспечер

- Диспечер е софтуерен модул, предоставящ контрола върху CPU на избраните за изпълнение процеси. Това включва:
  - Превключване на контекста
  - Превключване в потребителски режим
  - Преход към подходящ адрес в потребителката програма за възстановяване на нейното изпълнение
- **Dispatch latency** – времето, необходимо на диспечера да спре един процес и да стартира друг

Критерии за планиране

- **Използване на CPU** – съхраняване CPU натоварен колкото се може повече време
- **Throughput** – брой процеси, завършили своето изпълнение за единица време
- **Turnaround time** –време за изпълнение на отделен процес
- **Waiting time** – време за чакане на процес в ready queue
- **Response time** – време от момента на изпратената заявка до генерирането на първият отговор в интерактивните системи

Алгоритъм на планиране

- Оптимизационни критерии
  - Max CPU заемане
  - Max throughput
  - Min turnaround time
  - Min waiting time
  - Min response time
- Ще се разглежда за всеки процес случай на единичен CPU burst
- **Gantt chart** - бар-диаграма, описваща частично планиране с начално и крайно време за процес

First- Come, First-Served (FCFS) планиране

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Нека процесите са в последователност: P_1 , P_2 , P_3 във време 0.

- Waiting time за P_1 = 0; P_2 = 24; P_3 = 27
- Средно waiting time: (0 + 24 + 27)/3 = 17

FCFS планиране

Ако последователността на процесите:

$$P_2 , P_3 , P_1$$

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Средно waiting time: (6 + 0 + 3)/3 = 3
- По-добър резултат

- **Convoy effect** – късите процеси са зад по-дългите

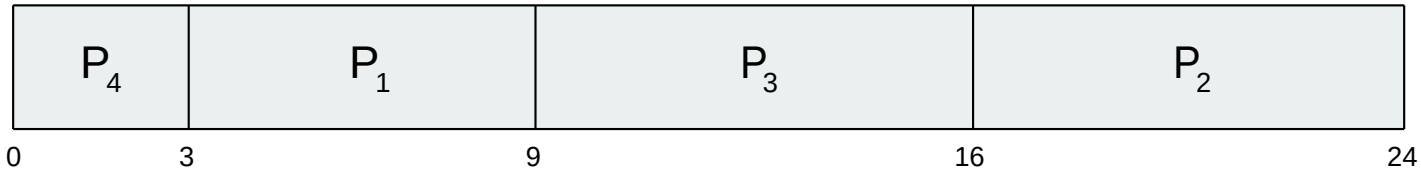
Shortest-Job-First (SJF) планиране

- С всеки процес се асоциира продължителността на неговия следващ CPU burst
  - Тези продължителности се използват за планиране на по-късите процеси
- SJF е оптимален – дава minimum средно време за изчакване за дадено множество от процеси
  - Трудността е определянето на продължителността на следващата CPU заявка

Пример за SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF диаграма

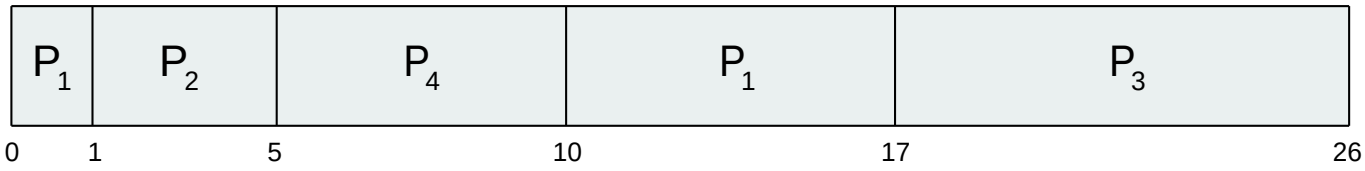


- Средно waiting time = (3 + 16 + 9 + 0) / 4 = 7

Shortest-remaining-time-first

- Preemptive версия на SJF се нарича **shortest-remaining-time-first**

Process	*Arrival* Time in ReadyQ	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



- Средно waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5

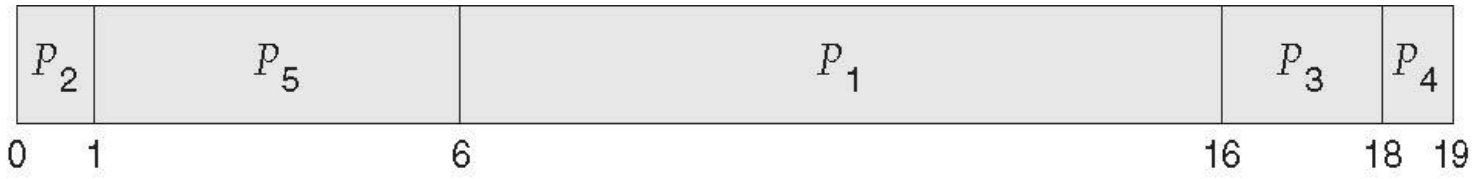
Приоритетно планиране

- Приоритет – цяло число, асоциирано с всеки процес
- CPU се заема от процеса с най-висок приоритет (по-малко число = по-висок приоритет)
  - Preemptive
  - Non-preemptive
- SJF е специален случай на приоритетно планиране. Приоритетът е инверсията на предвиждания следващ CPU burst
- Проблем ≡ **Starvation** – ниско-приоритетните процеси могат никога да не се изпълнят
- Решение ≡ **Aging** – приоритетът да нараства за процеси, чакащи дълго време

Пример

Процесите се появяват във време 0

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



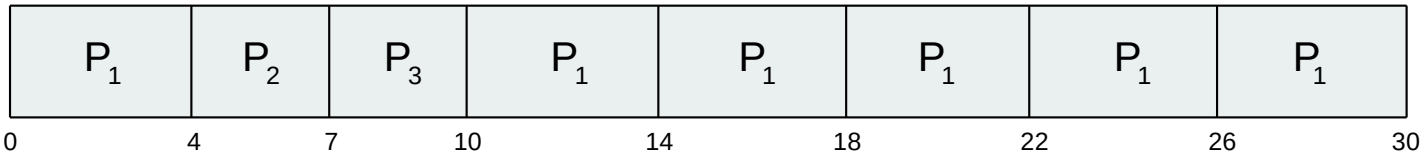
- Средно waiting time = 8.2 msec

Round Robin (RR)

- Всеки процес използва CPU кратко време (**time quantum** q). След изтичането на това време, процесът се изтласква и се добавя в края на ready queue.
- За планиране се използват прекъсвания от таймер
- Производителност
  - q е голямо $\Rightarrow$ FIFO
  - q е малко $\Rightarrow$ превключването води до големи разходи

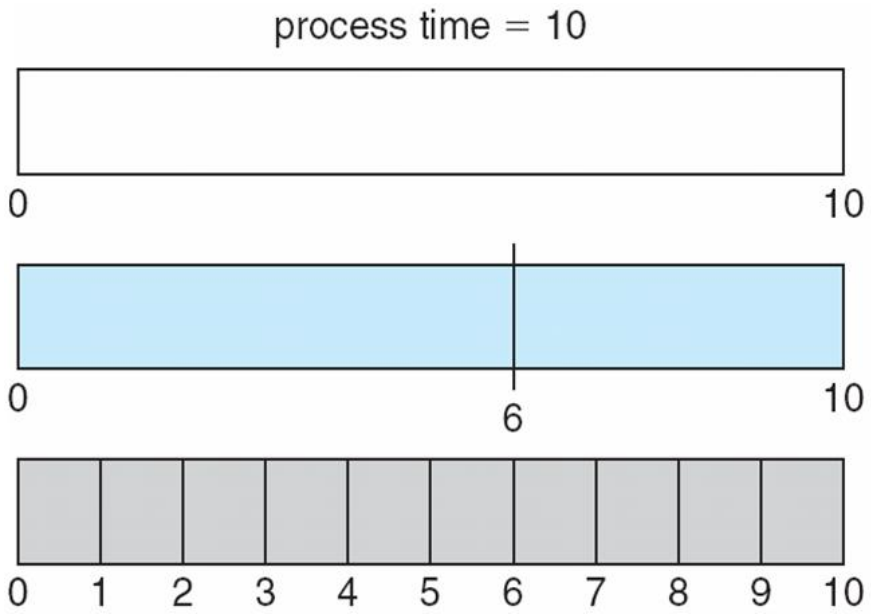
Пример RR с квант = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3



- Типично, RR има по-голям среден ***turnaround*** от SJF, но по-добър ***response***
- q трябва да е голямо в сравнение с времето за превключване на контекста
- q обикновено е от 10ms до100ms, превключването на контекста < 10 usec

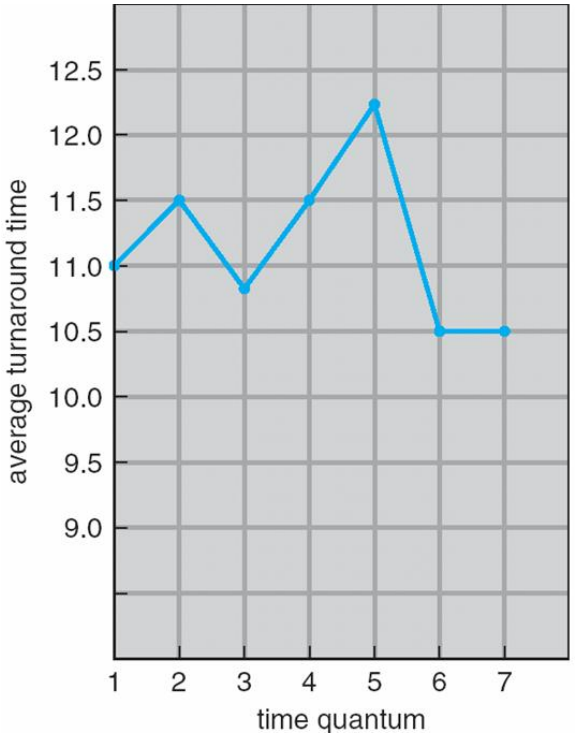
Кванти и време за превключване на контекста



quantum	context switches
12	0
6	1
1	9

Turnaround време и кванти

- Turnaround времето не е задължително да се подобрява ако продължителността на кванта нараства



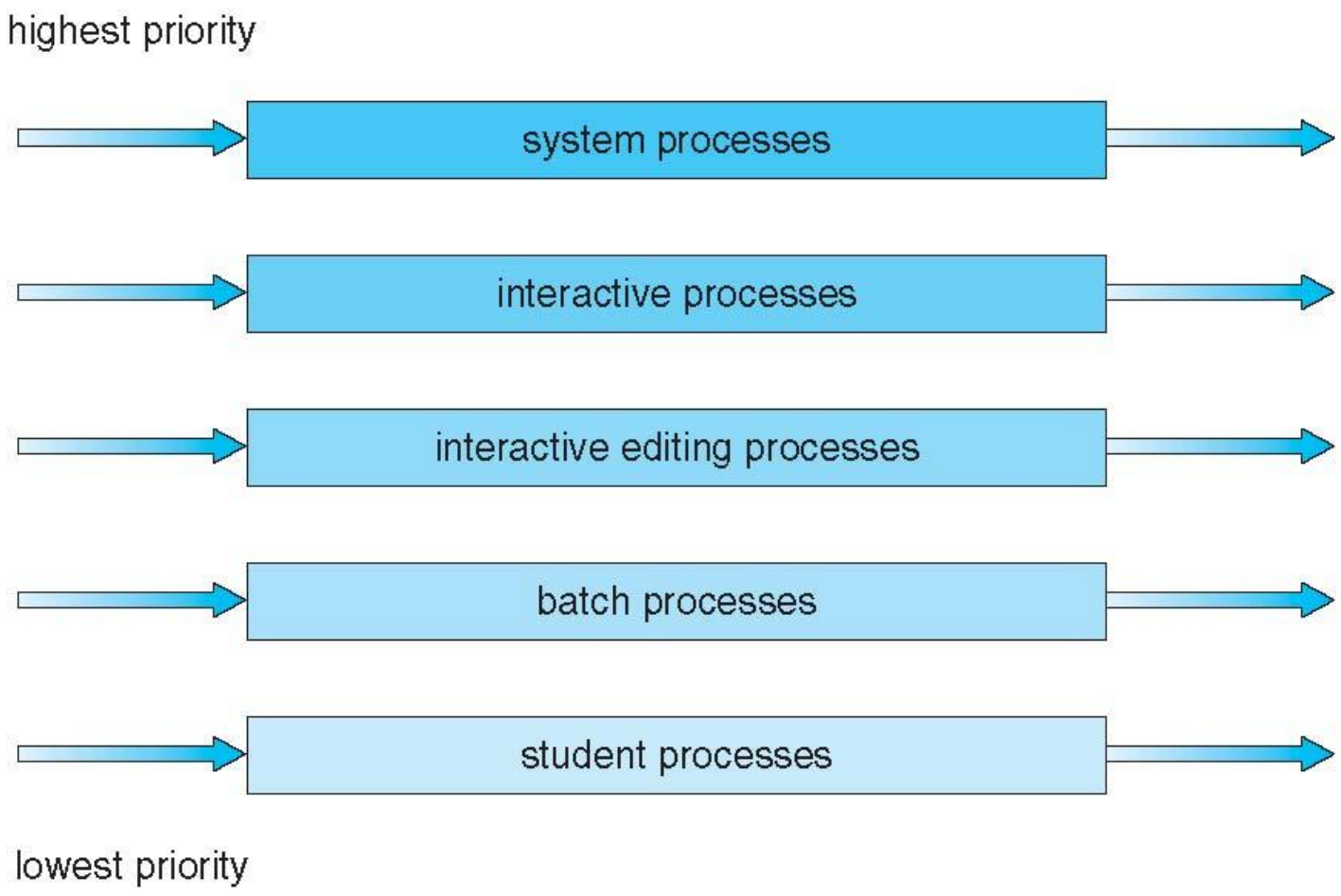
process	time
P_1	6
P_2	3
P_3	1
P_4	7

- По принцип, средното turnaround време може да се подобри ако повечето процеси завършат своя CPU burst само в един квант

Multilevel Queue

- Ready queue се разделя на няколко опашки според вида процеси, например:
  - **foreground** (interactive)
  - **background** (batch)
- Всяка опашка има собствен алгоритъм на планиране:
  - foreground – RR
  - background – FCFS
- Трябва да се планира между опашките:
  - Fixed priority scheduling – обслужват се първо всички процеси от foreground опашката, после от background опашката. Възможна е starvation.
  - Time slice – всяка опашка получава определена продължителност от CPU времето, която да планира между процесите си, например 80% за foreground при RR и 20% за background при FCFS

Multilevel Queue планиране

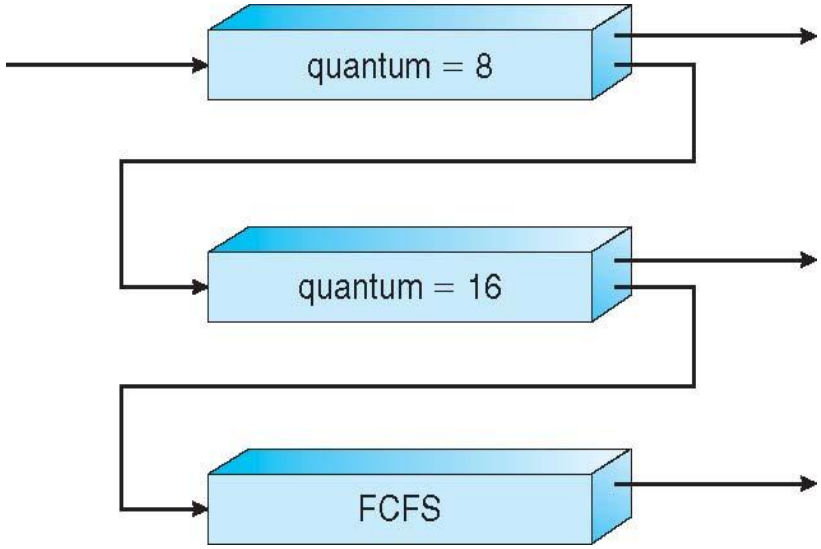


Multilevel Feedback Queue

- Множество опашки
- Процес може да бъде преместван между различни опашки
- Планирането се базира на :
  - Брой опашки
  - Използвания метод за определяне в коя опашка процес ще постъпи при необходимост от обслужване
  - Всяка опашка има отделен алгоритъм на планиране
  - Използвания метод за определяне кога да се постави процес в по-приоритетната опашка
  - Използвания метод за определяне кога да се намали приоритета на процес
- Aging може да се реализира чрез увеличаване приоритета на процес

Пример на Multilevel Feedback Queue

- Три опашки:
  - Q_0 – RR с квант 8 ms
  - Q_1 – RR с квант 16 ms
  - Q_2 – FCFS
- Планиране
  - Нова задача влиза в опашка Q_0 , обслужвана по FCFS
    - Когато заеме CPU, процесът получава 8 ms
    - Ако не завърши в рамките на 8ms, процесът се премества в Q_1
  - В Q_1 процесът отново се обслужва по FCFS и получава допълнителни 16ms
    - Ако все още не завърши, се изтласква и се премества в Q_2



Планиране в Linux

Приоритети

- Статичен
  - За real-time процеси
  - Диапазон 0 - 99
- Динамичен
  - За конвенционални процеси
  - Диапазон 100-139
  - Настройва се в зависимост от изпълнението и класа приоритет

Приоритет на процес

- Базиран на “nice” ниво на процеса.
  - По подразбиране 0, диапазон -20 to +19 (по-малкото е по-добро)
- I/O bound процеси получават по-висок приоритет.
- CPU bound процеси получават малък приоритет.

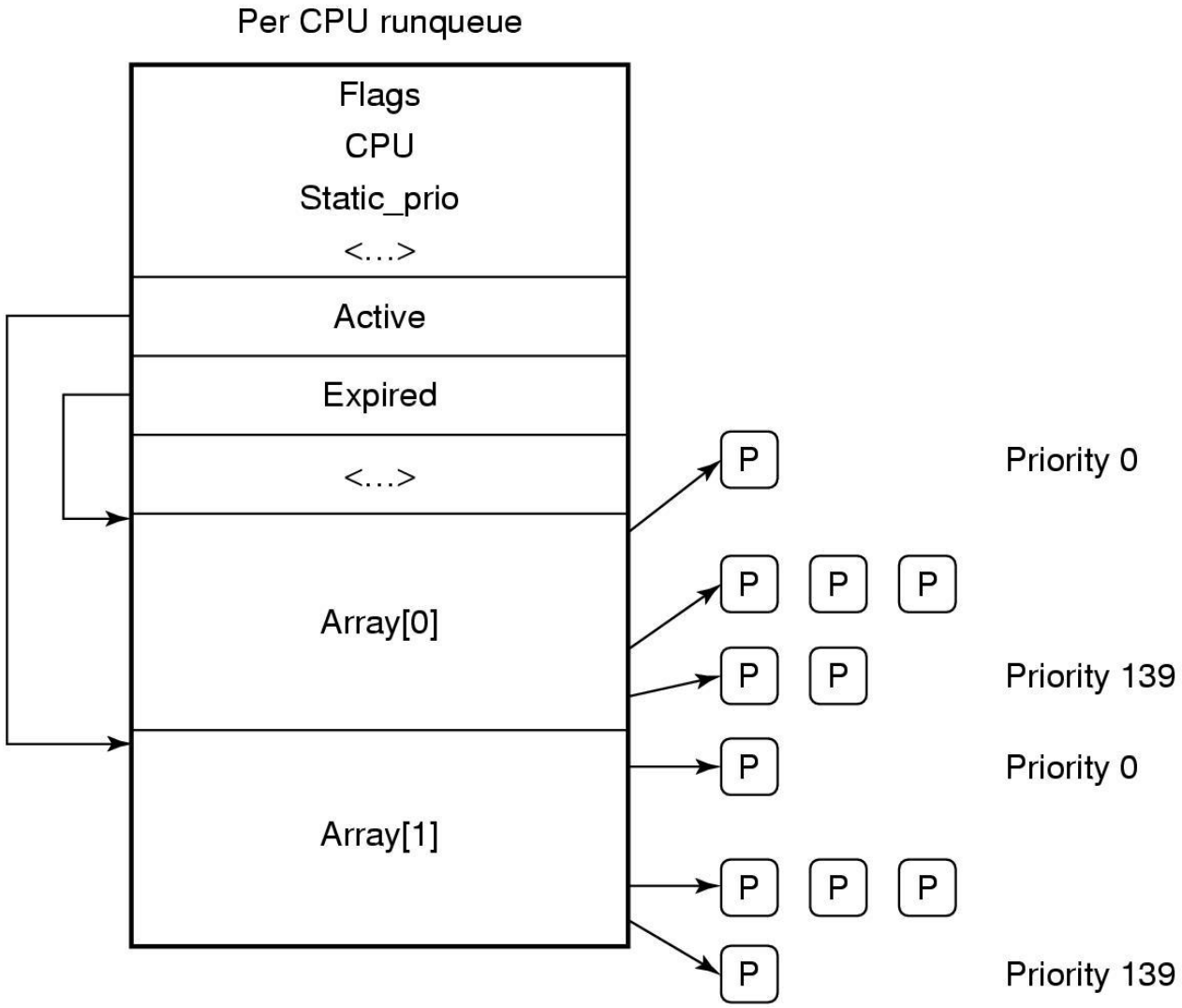
Квант на процес

- При създаване child получава ½ от timeslice на родителя
- Високия приоритет получава по-голям timeslice
  - Нисък приоритет/ по-малко интерактивен
  - Висок приоритет/ повече интерактивен
- Когато timeslice→ 0, процесът става *expired*

Квант за конвенционални процеси

Description	Static priority	Base Time quantum (timeslice)
Highest static priority	100	800 ms
High static priority	110	600 ms
Default static priority	120	100 ms
Low static priority	130	50 ms
Lowest static priority	139	5 ms

Опашки за изпълнение



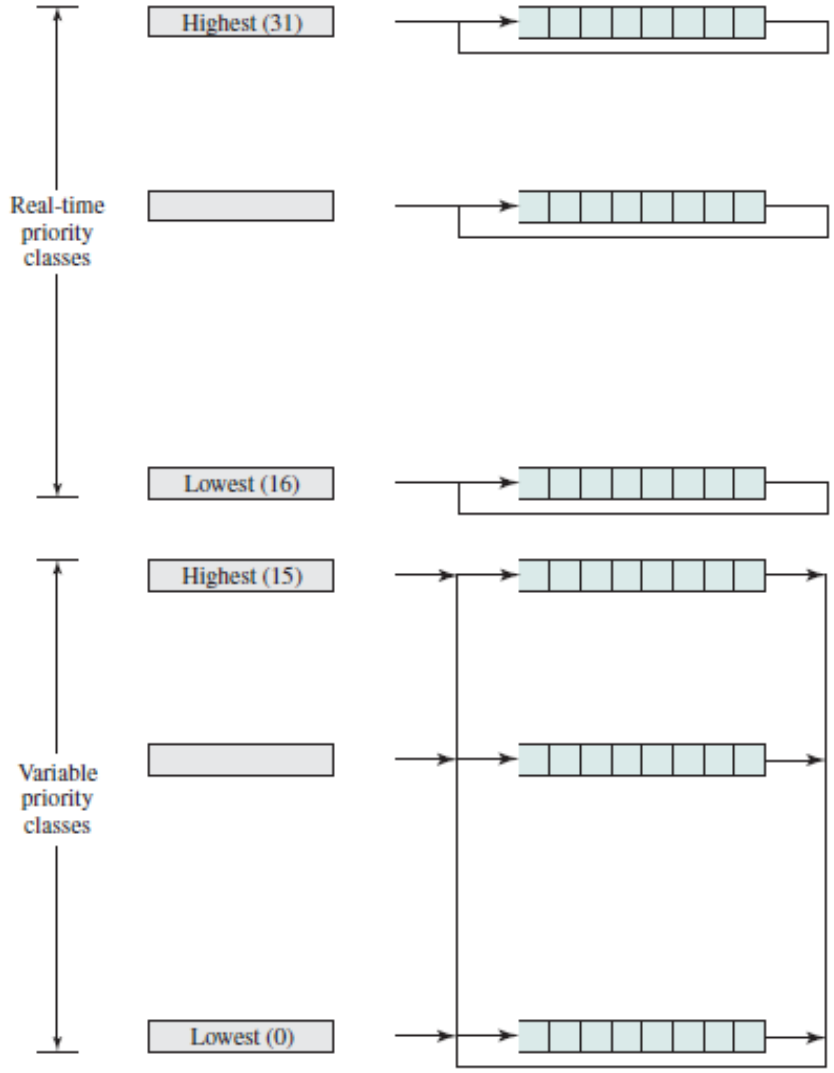
Планиране в Windows

Нива от 0 до 32

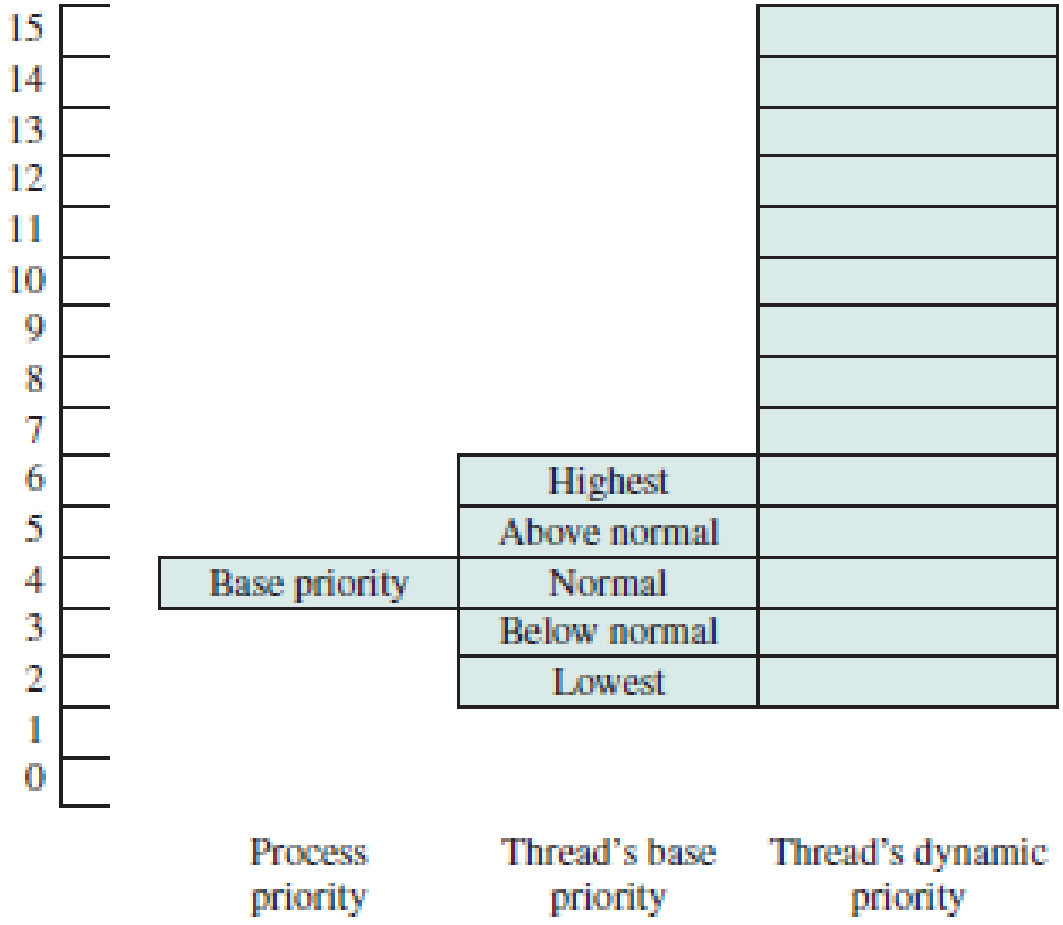
Два класа приоритети:

- Real-time (16-31)
- Variable priority (1-15)

Приоритети в Windows



Приоритети в Windows



Планиране в Windows

- Windows 3.1 – non-preemptive
- Windows 95 – опростено preemptive
- Windows NT – Multi-level Feedback Queue
- Windows XT – priority preemptive с приоритетни нива и RR за всяко от тях
- Windows 7 – User-mode scheduling
- Windows 8 – WinJS
- Widows 10 – Multi-level Feedback Queue

Въпроси ?