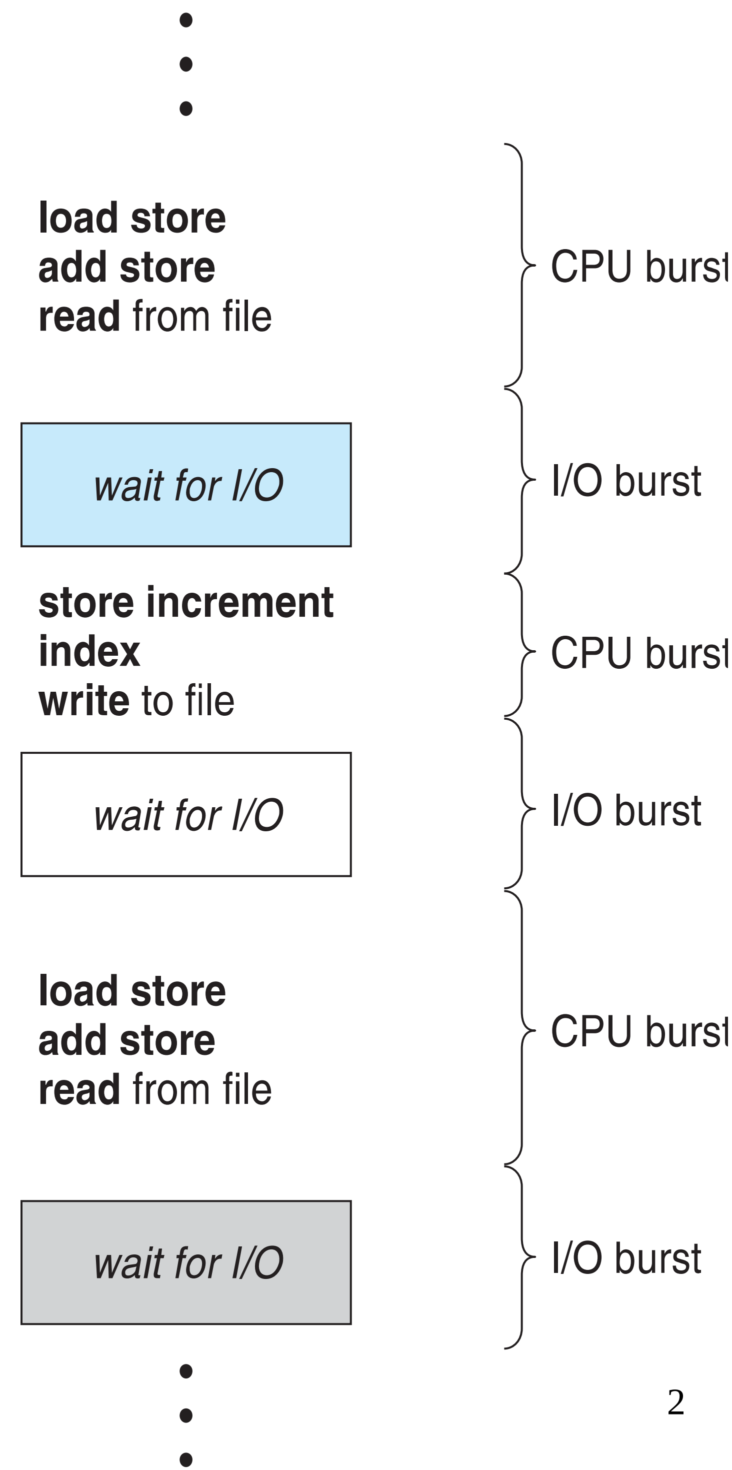
CPU Scheduling

Moumita Patra
July-Nov 2019
Galvin-Gagne

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU-I/O burst cycle- Process execution consists of a cycle of CPU execution and I/O wait
- CPU burst followed by I/O burst
- CPU burst distribution is of main concern



CPU Scheduler

- Short term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  - Switches from running to waiting state- nonpreemptive
  - Switches from running to ready state- preemptive
  - Switches from waiting to ready- preemptive
  - Terminates- nonpreemptive

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- **Dispatch latency**- time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

- **CPU utilization**- keep the CPU as busy as possible
- **Throughput**- number of processes that complete their execution per time unit
- **Turnaround time**- amount of time to execute a particular process
- **Waiting time**- amount of time a process has been waiting in the ready queue
- **Response time**- amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First Come First Served Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:

Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

The Gantt chart for the schedule is:

Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$

Average waiting time: $(6 + 0 + 3)/3 = 3$

Much better than previous case

Convoy effect - short process behind long process

Consider one CPU-bound and many I/O-bound processes

Shortest Job First (SJF) Scheduling

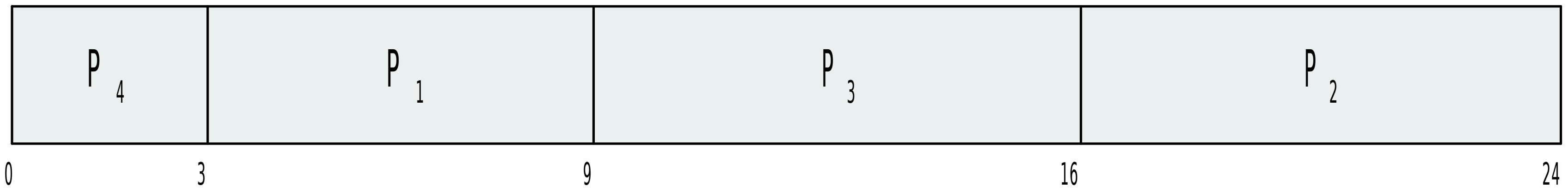
- Associate with each process the length of its next CPU burst

  -> Use these lengths to schedule the process with the shortest time
- SJF is optimal- gives minimum average waiting time for a given set of processes
- The difficulty is knowing the length of the next CPU request
- Could ask the users

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



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

Determining Length of Next CPU Burst

Can only estimate the length – should be similar to the previous one

Then pick process with shortest predicted next CPU burst

Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

Commonly, α set to ½

Preemptive version called **shortest-remaining-time-first**

Examples of Exponential Averaging

$\alpha = 0$

$\tau_{n+1} = \tau_n$

Recent history does not count

$\alpha = 1$

$\tau_{n+1} = \alpha \, t_n$

Only the actual last CPU burst counts

If we expand the formula, we get:

$$\tau_{n+1} = \alpha \, t_n + (1 - \alpha)\alpha \, t_{n-1} + \dots$$
$$+ (1 - \alpha)^j \alpha \, t_{n-j} + \dots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

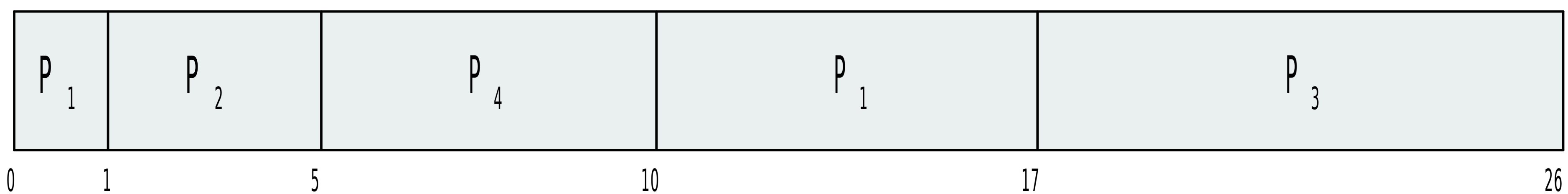
Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Example of Shortest Remaining Time First

Now we add the concepts of varying arrival times and preemption to the analysis

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

Preemptive SJF Gantt Chart



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

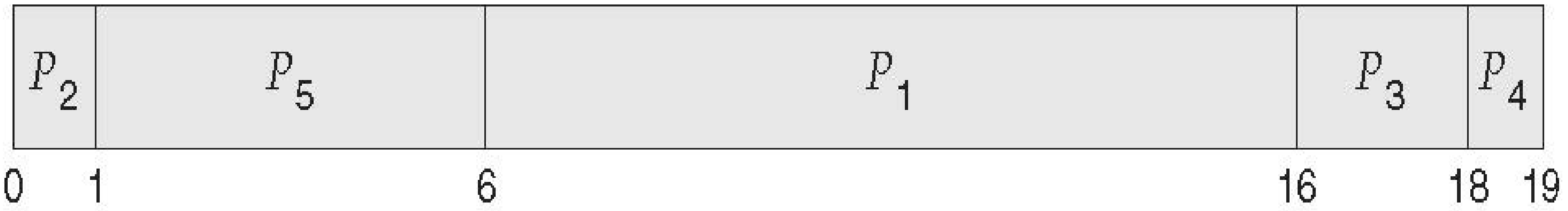
Priority Scheduling

- A priority number (integer) associated with each process
- The CPU is allocated to the process with the highest priority
- SJF is priority scheduling where priority is inverse of next CPU burst time
- Problem- Starving
- Solution- Aging

Example of Priority Scheduling

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

Priority scheduling Gantt Chart



Average waiting time = 8.2 msec

Round Robin (RR)

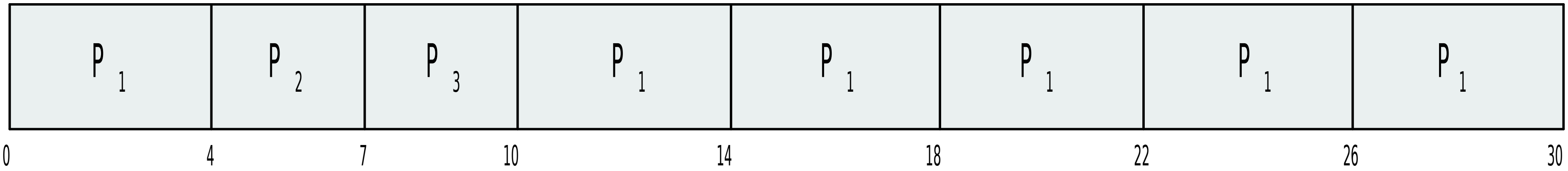
- Each process gets a small unit of CPU time.
- Process is preempted when time is elapsed and is added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once.
- No process waits more than (n-1)q time units.
- Timer interrupts every quantum to schedule next process.
- Performance
  - q large => FIFO
  - q small => q must be large with respect to context switch, otherwise overhead is too high

Example of RR

Time quantum =4

Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:

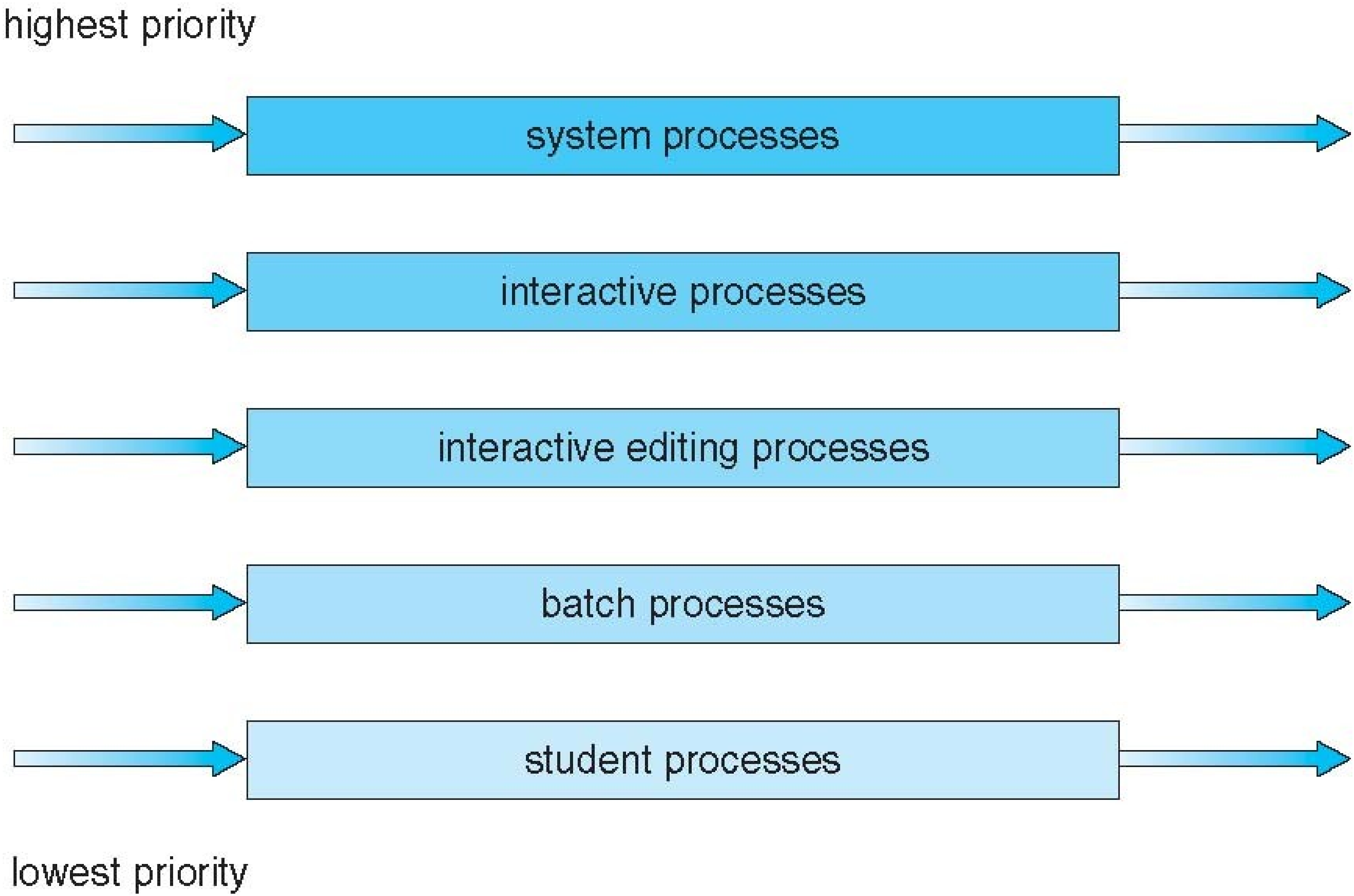


- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Multilevel Queue

- Ready queue is partitioned into separate queues- foreground and background
- Process permanently in a given queue
- Each queue has its own scheduling algorithm
- Scheduling must be done between queues:
  - Fixed priority scheduling
  - Time slice

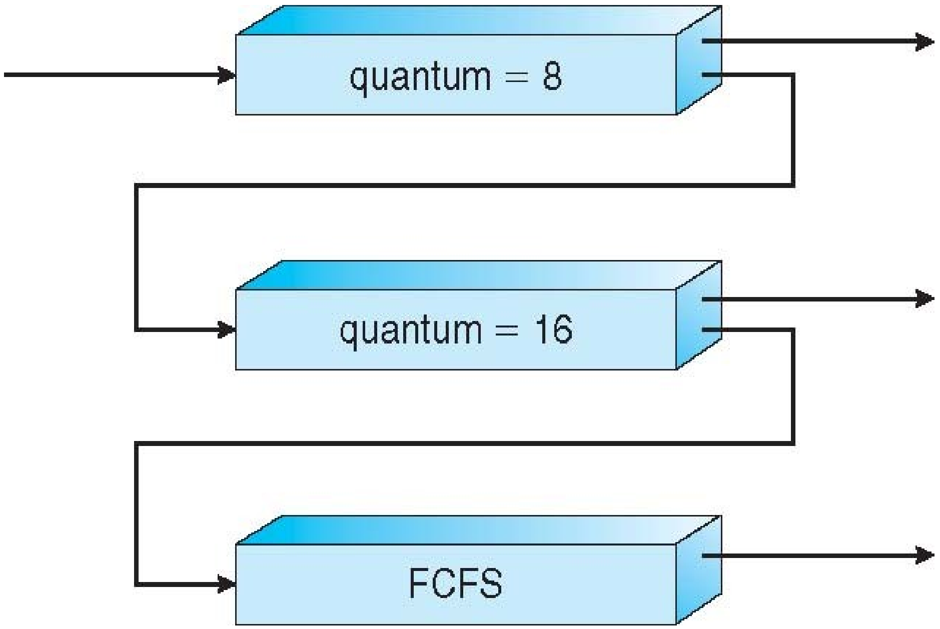
Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues
- Multi-level feedback queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service

Example of Multiple Feedback Queue



Three queues:

Q_0 – RR with time quantum 8 milliseconds

Q_1 – RR time quantum 16 milliseconds

Q_2 – FCFS

Scheduling

→ A new job enters queue Q_0 which is served FCFS

→ When it gains CPU, job receives 8 milliseconds

→ If it does not finish in 8 milliseconds, job is moved to queue Q_1

→ At Q_1 job is again served FCFS and receives 16 additional milliseconds

→ If it still does not complete, it is preempted and moved to queue Q_2