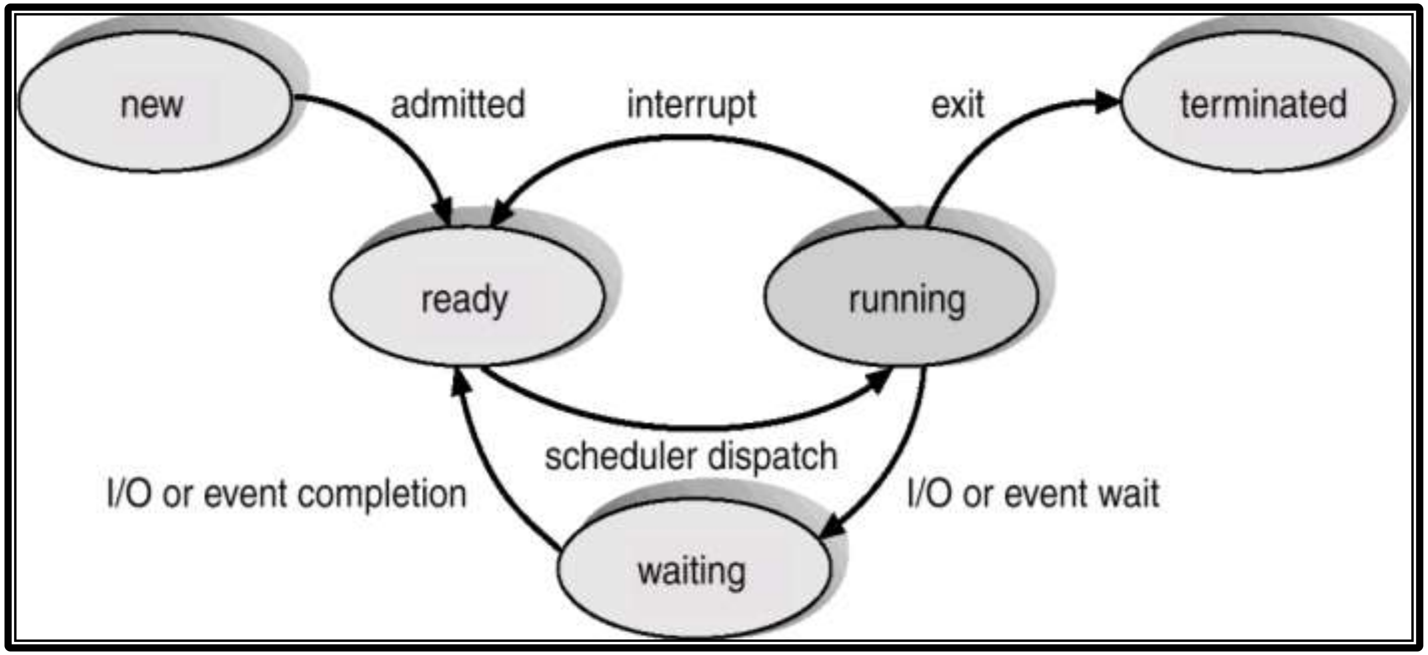
Process State

- As a process executes, it changes *state*
  - **new**: The process is being created.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur.
  - **ready**: The process is waiting to be assigned to a process.
  - **terminated**: The process has finished execution

Diagram of Process State

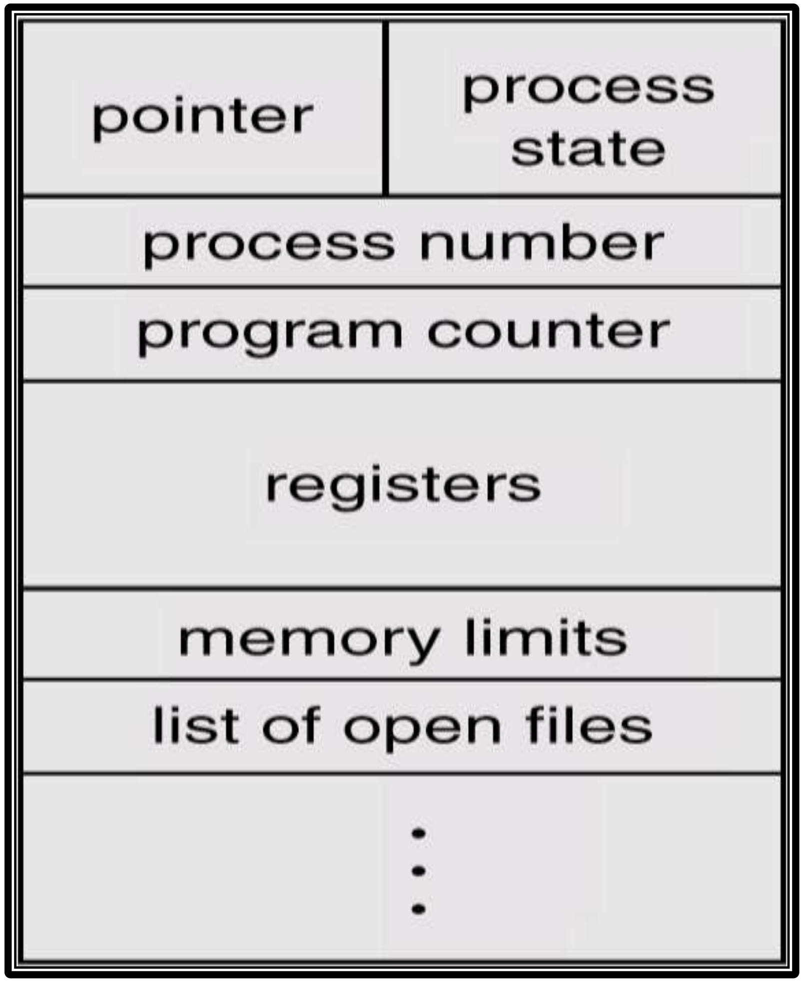


Process Control Block (PCB)

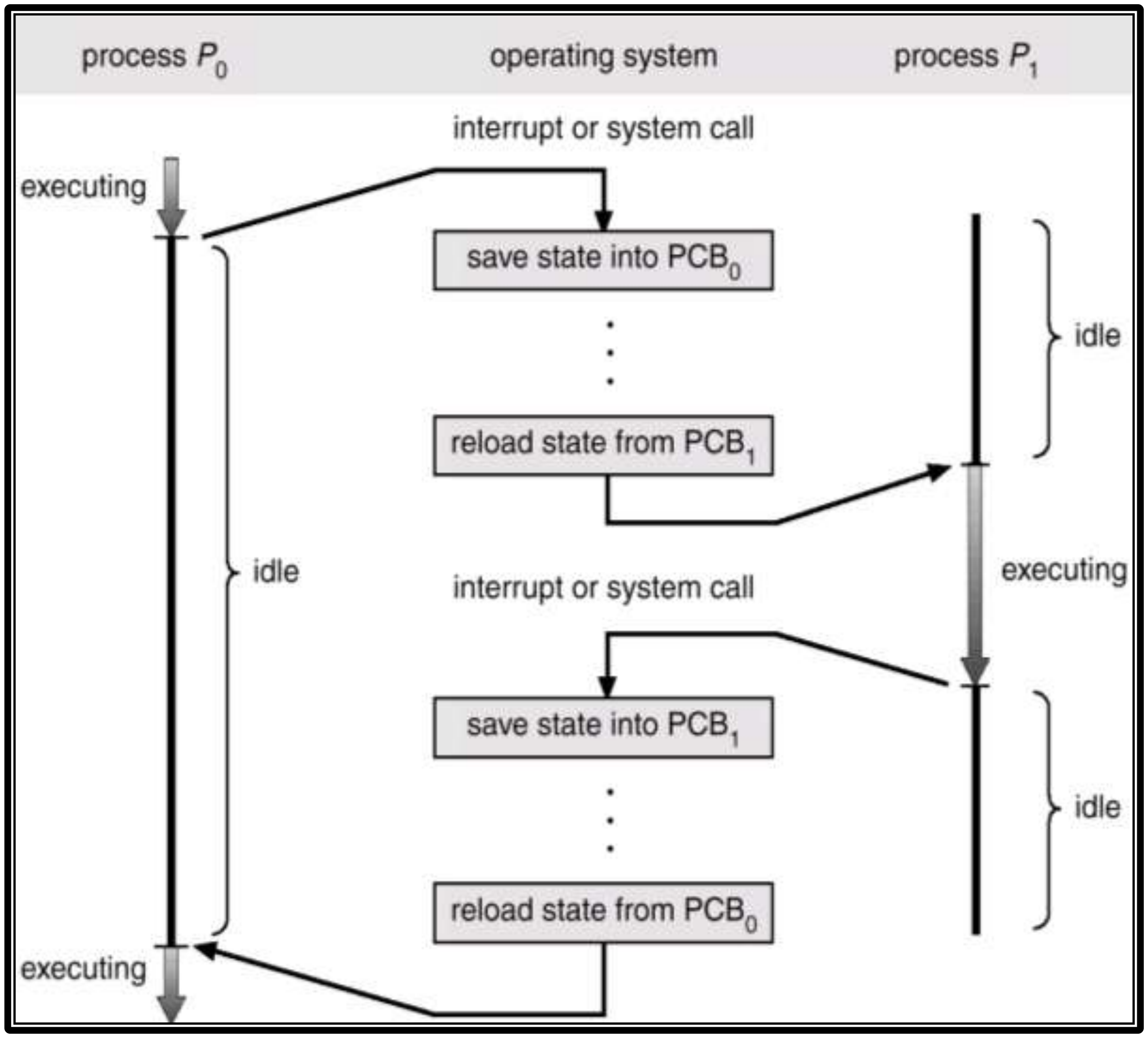
Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



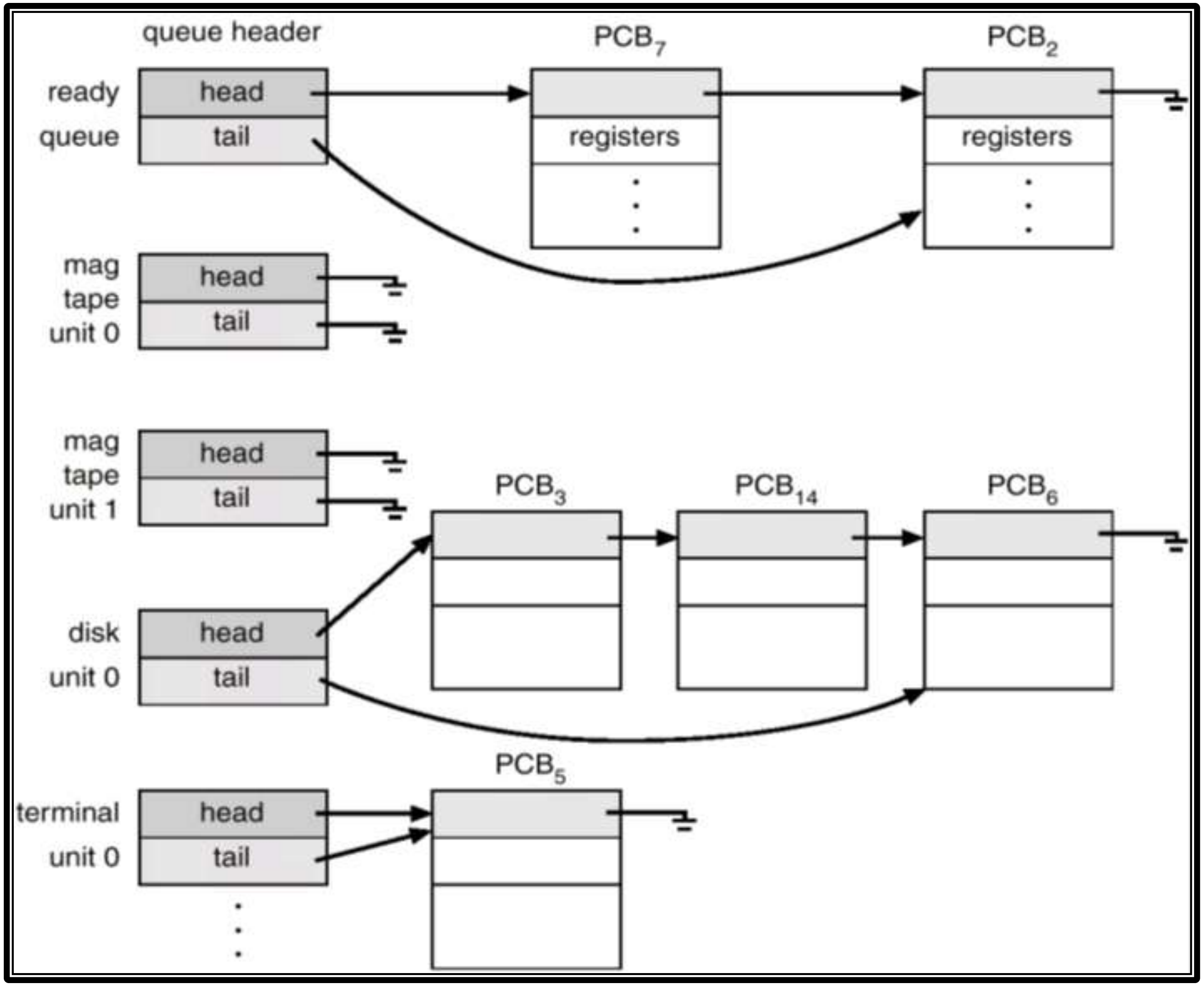
CPU Switch From Process to Process



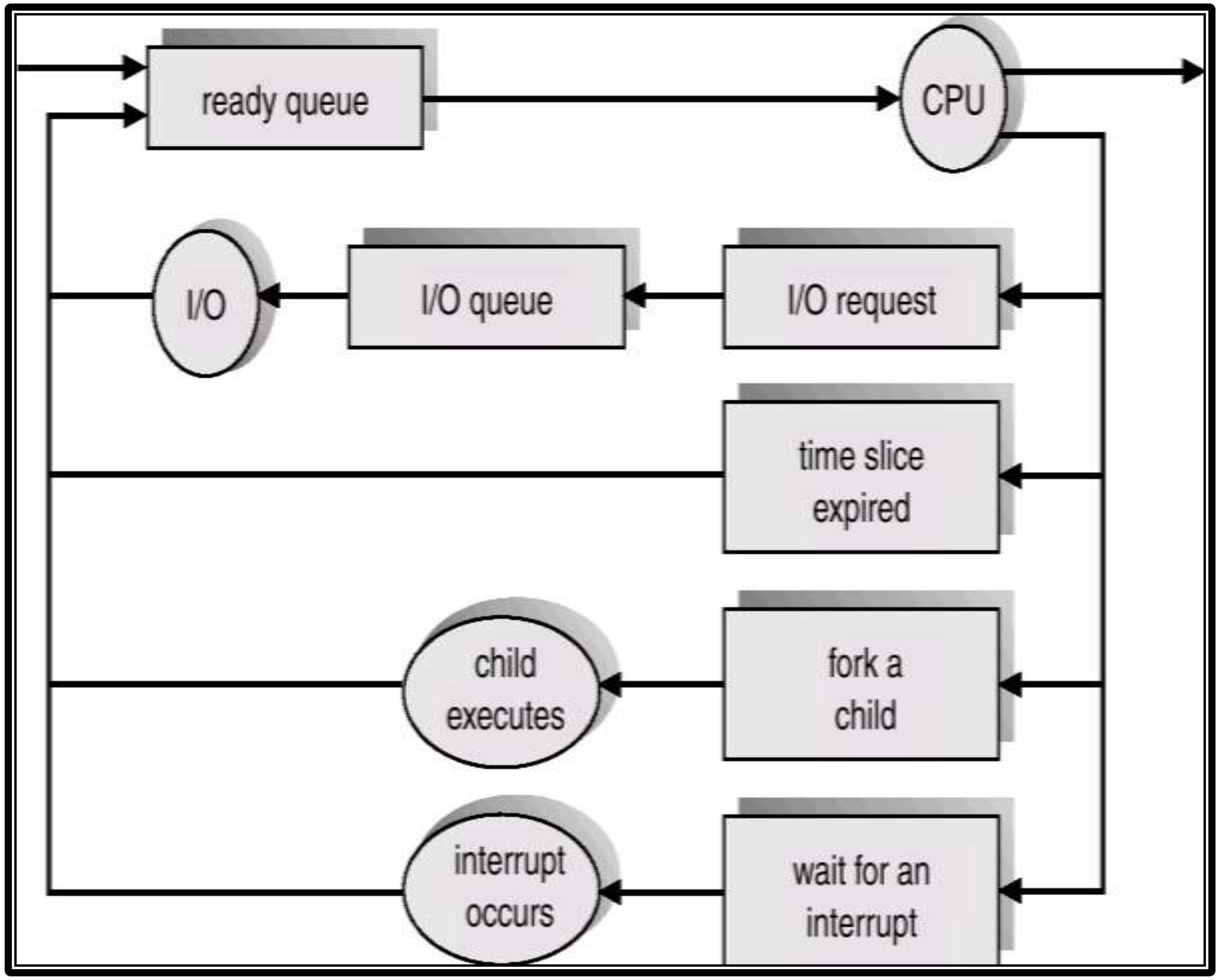
Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

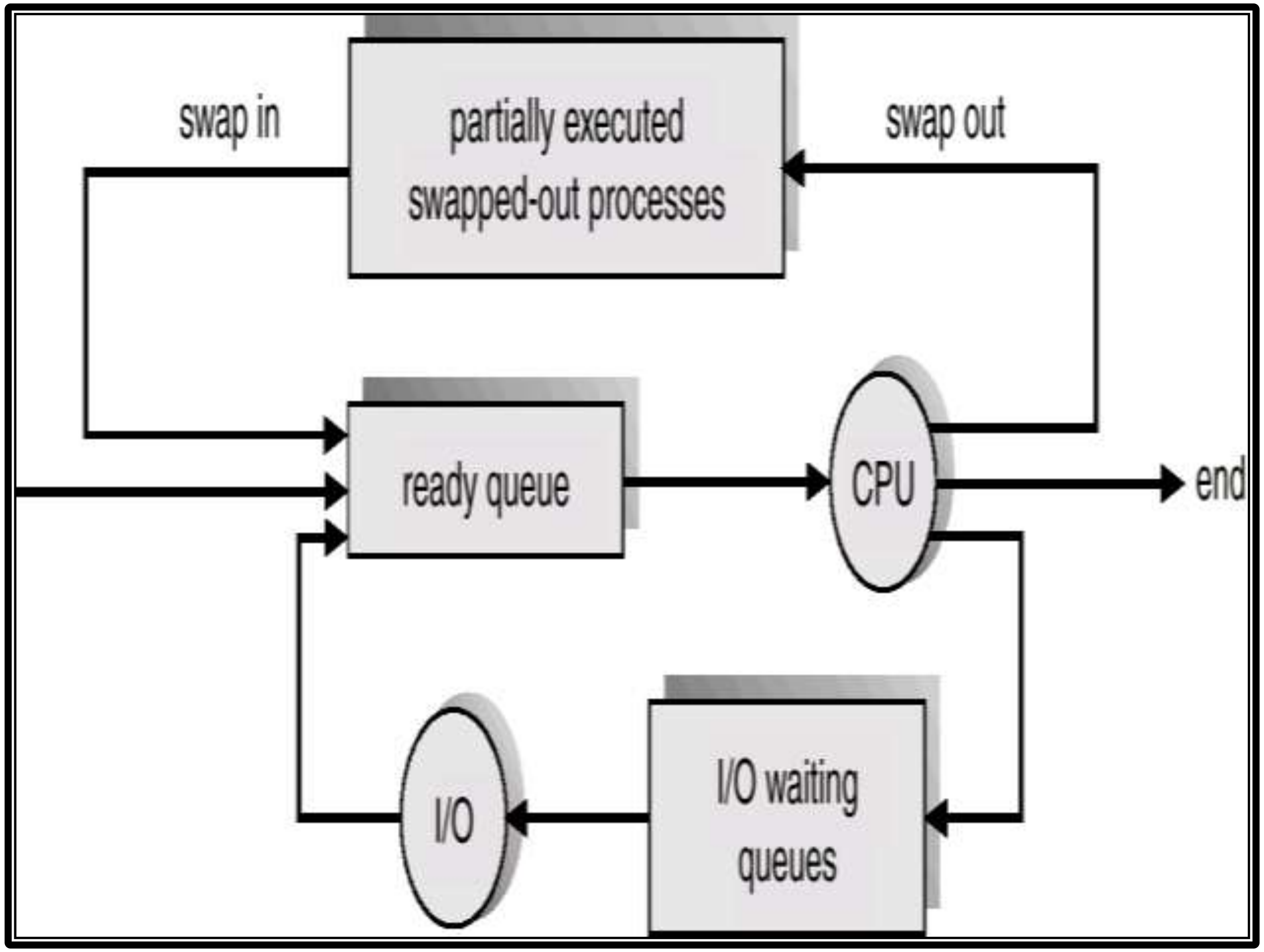


Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.

- The mid-term scheduler, present in all systems with virtual memory, temporarily removes processes from main memory and places them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as "swapping out" or "swapping in" (also incorrectly as "paging out" or "paging in"). The mid-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is page faulting frequently, or a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available, or when the process has been unblocked and is no longer waiting for a resource. [Stallings, 396] [Stallings, 370]
- In many systems today (those that support mapping virtual address space to secondary storage other than the swap file), the mid-term scheduler may actually perform the role of the long-term scheduler.

Addition of Medium Term Scheduling



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming.*
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts.

CPU Scheduling

What Is In This Chapter?

- This chapter is about how to get a process attached to a processor.
- It centers around efficient algorithms that perform well.
- The design of a scheduler is concerned with making sure all users get their fair share of the resources.

CPU Scheduling

What Is In This Chapter?

CPU SCHEDULING

Scheduling Concepts

Multiprogramming A number of programs can be in memory at the same time. Allows overlap of CPU and I/O.

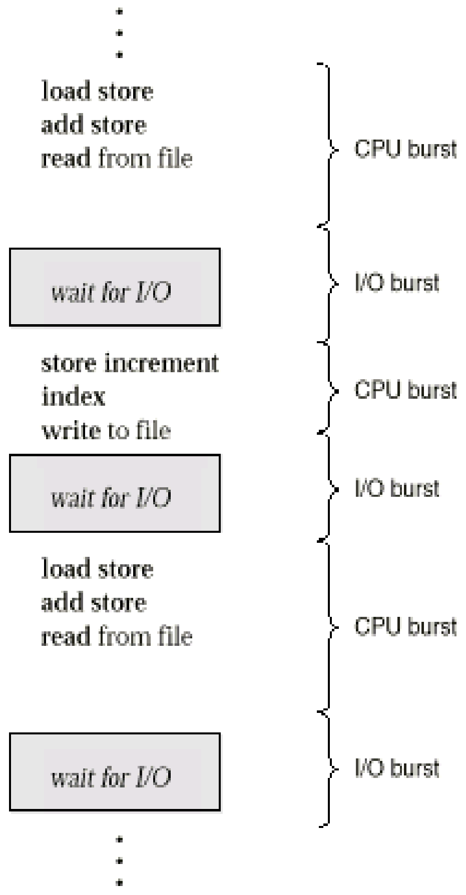
Jobs (batch) are programs that run without user interaction.

User (time shared) are programs that may have user interaction.

Process is the common name for both.

CPU - I/O burst cycle Characterizes process execution, which alternates, between CPU and I/O activity. CPU times are generally much shorter than I/O times.

Preemptive Scheduling An interrupt causes currently running process to give up the CPU and be replaced by another process.



CPU SCHEDULING

The Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is *nonpreemptive*
- All other scheduling is *preemptive*

CPU SCHEDULING

The 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

Criteria For Performance Evaluation

Note usage of the words **DEVICE, SYSTEM, REQUEST, JOB.**

UTILIZATION	The fraction of time a device is in use. (ratio of in-use time / total observation time)
THROUGHPUT	The number of job completions in a period of time. (jobs / second)
SERVICE TIME	The time required by a device to handle a request. (seconds)
QUEUEING TIME	Time on a queue waiting for service from the device. (seconds)
RESIDENCE TIME	The time spent by a request at a device. RESIDENCE TIME = SERVICE TIME + QUEUEING TIME.
RESPONSE TIME	Time used by a system to respond to a User Job. (seconds)
THINK TIME	The time spent by the user of an interactive system to figure out the next request. (seconds)

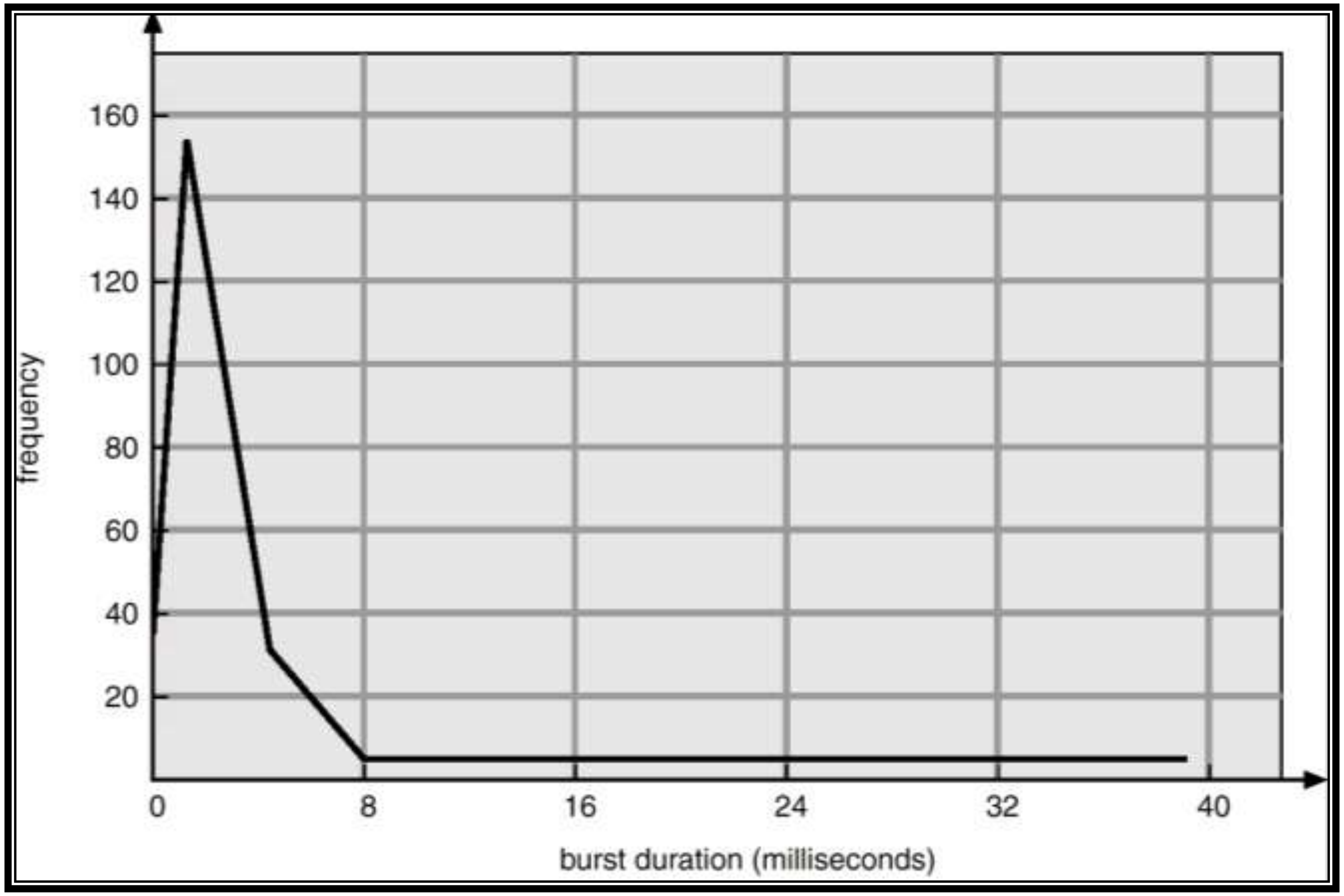
The goal is to optimize both the average and the amount of variation. (but beware the ogre predictability.)

Performance Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # 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)

CPU SCHEDULING

Most Processes Don't Use Up Their Scheduling Quantum!



CPU SCHEDULING

Scheduling Algorithms

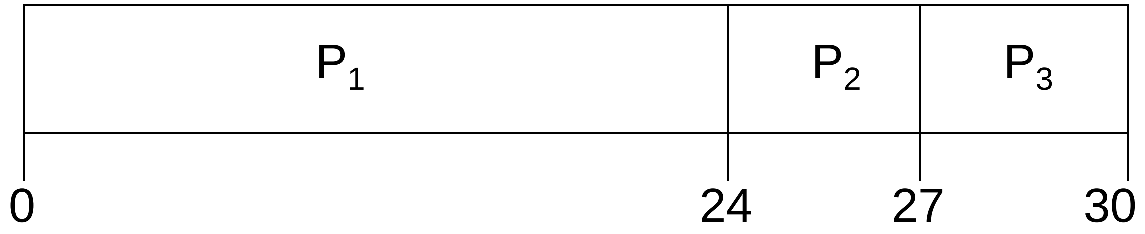
FIRST-COME, FIRST SERVED:

- (FCFS) same as FIFO
- Simple, fair, but poor performance.   Average queueing time may be long.
- What are the average queueing and residence times for this scenario?
- How do average queueing and residence times depend on ordering of these processes in the queue?

First-Come, First-Served (FCFS) 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:



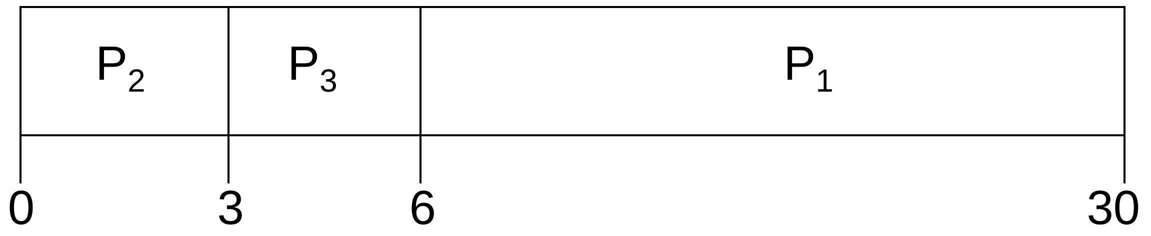
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

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

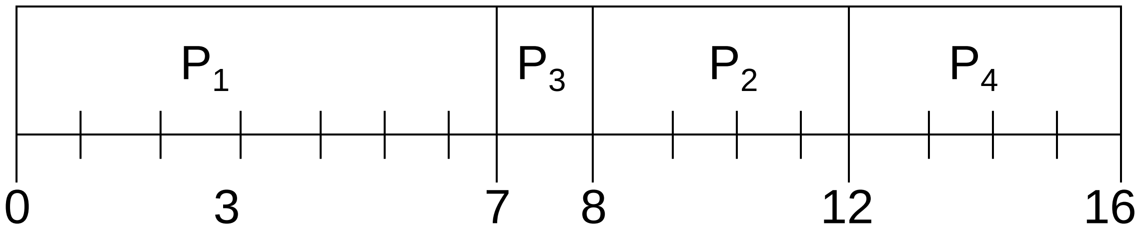
Shortest-Job-First (SJR) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

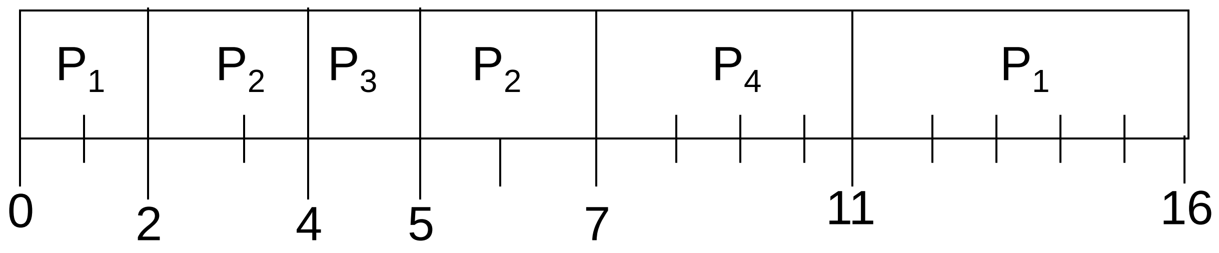


- Average waiting time = (0 + 6 + 3 + 7)/4 - 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



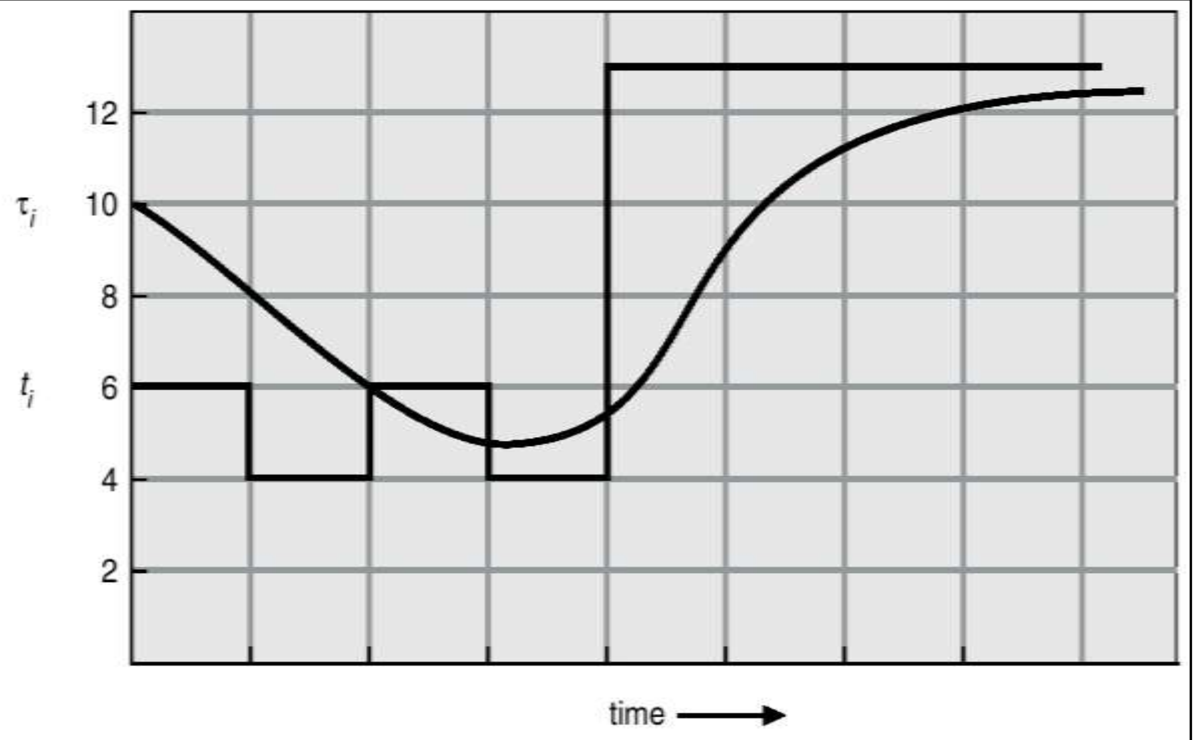
- Average waiting time = (9 + 1 + 0 +2)/4 - 3

Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

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

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem ≡ Starvation – low priority processes may never execute.
- Solution ≡ Aging – as time progresses increase the priority of the process.

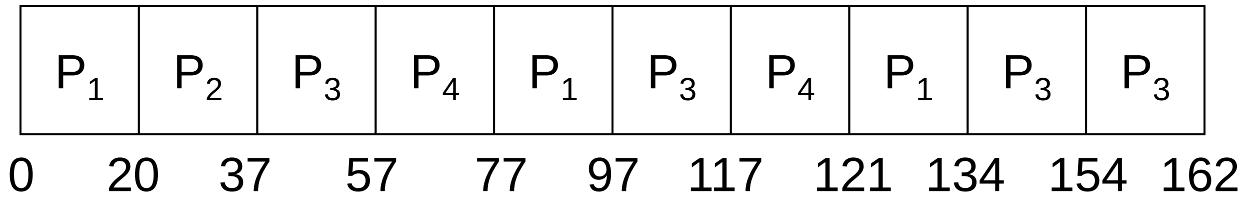
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and 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.
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum = 20

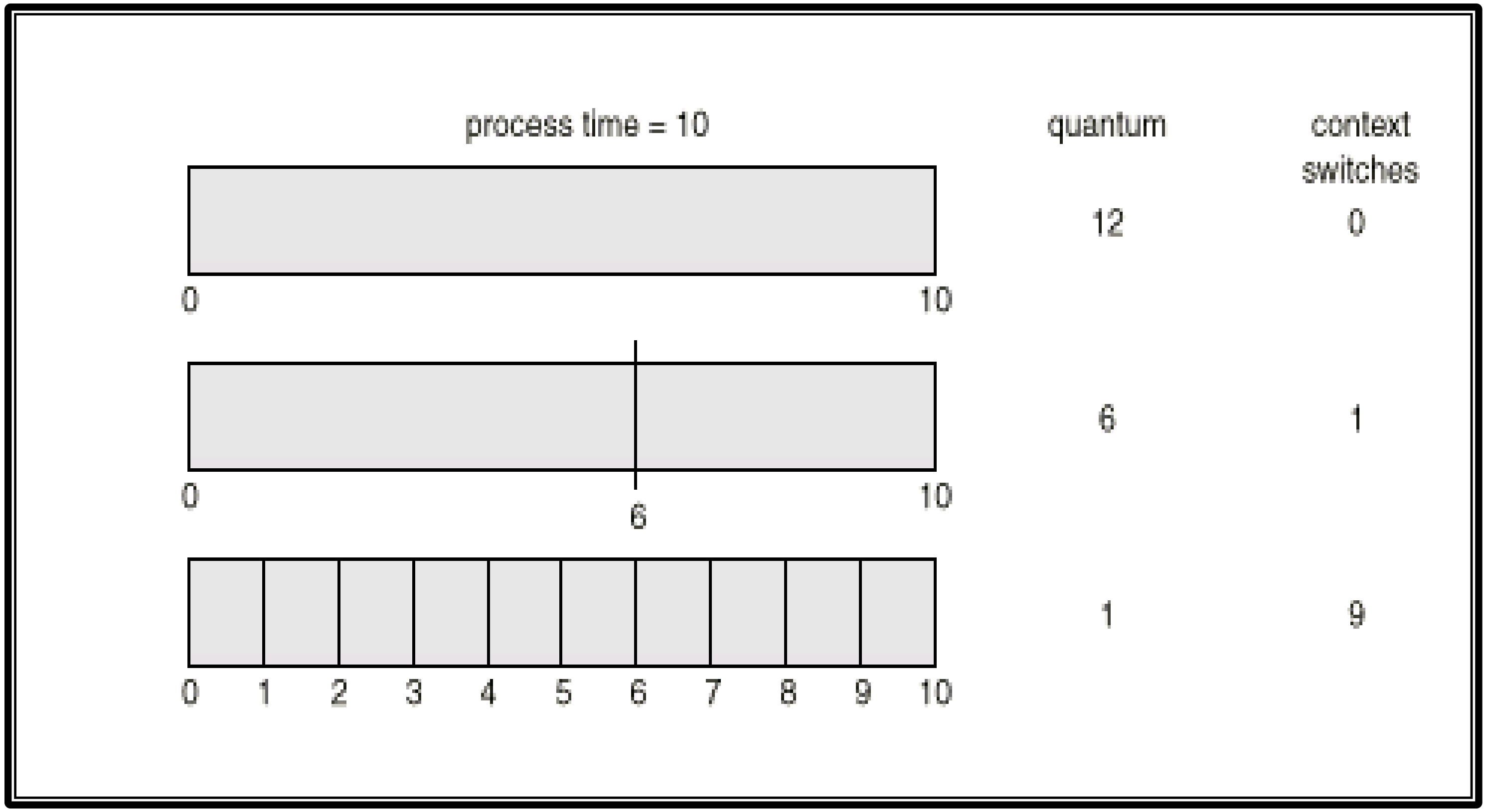
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

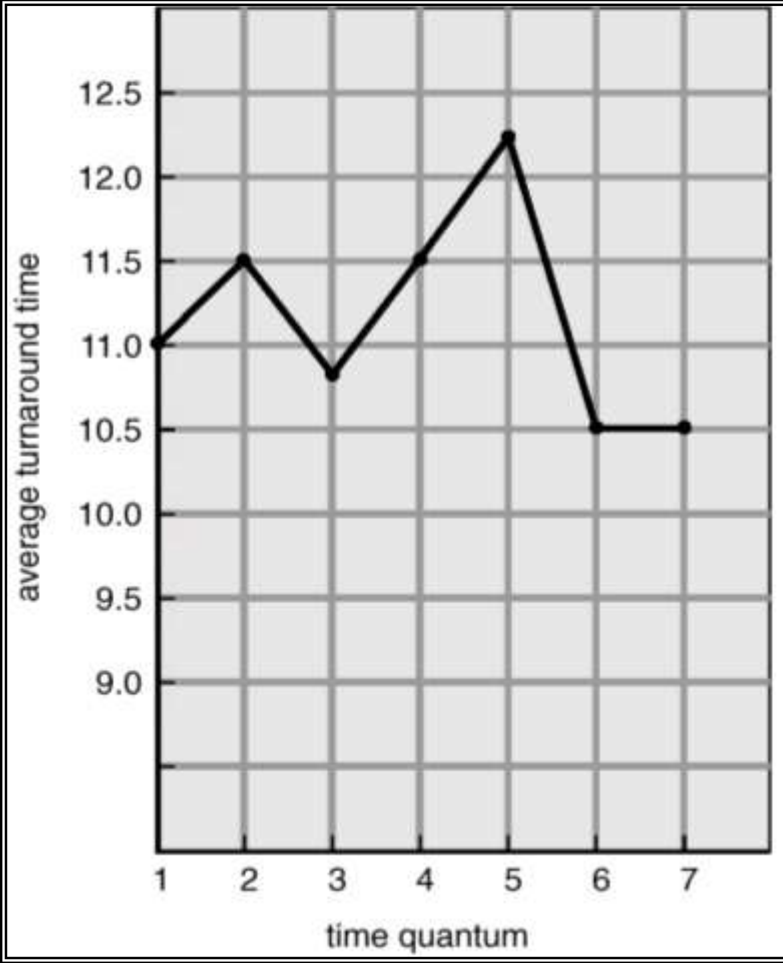


- Typically, higher average turnaround than SJF, but better *response*.

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum

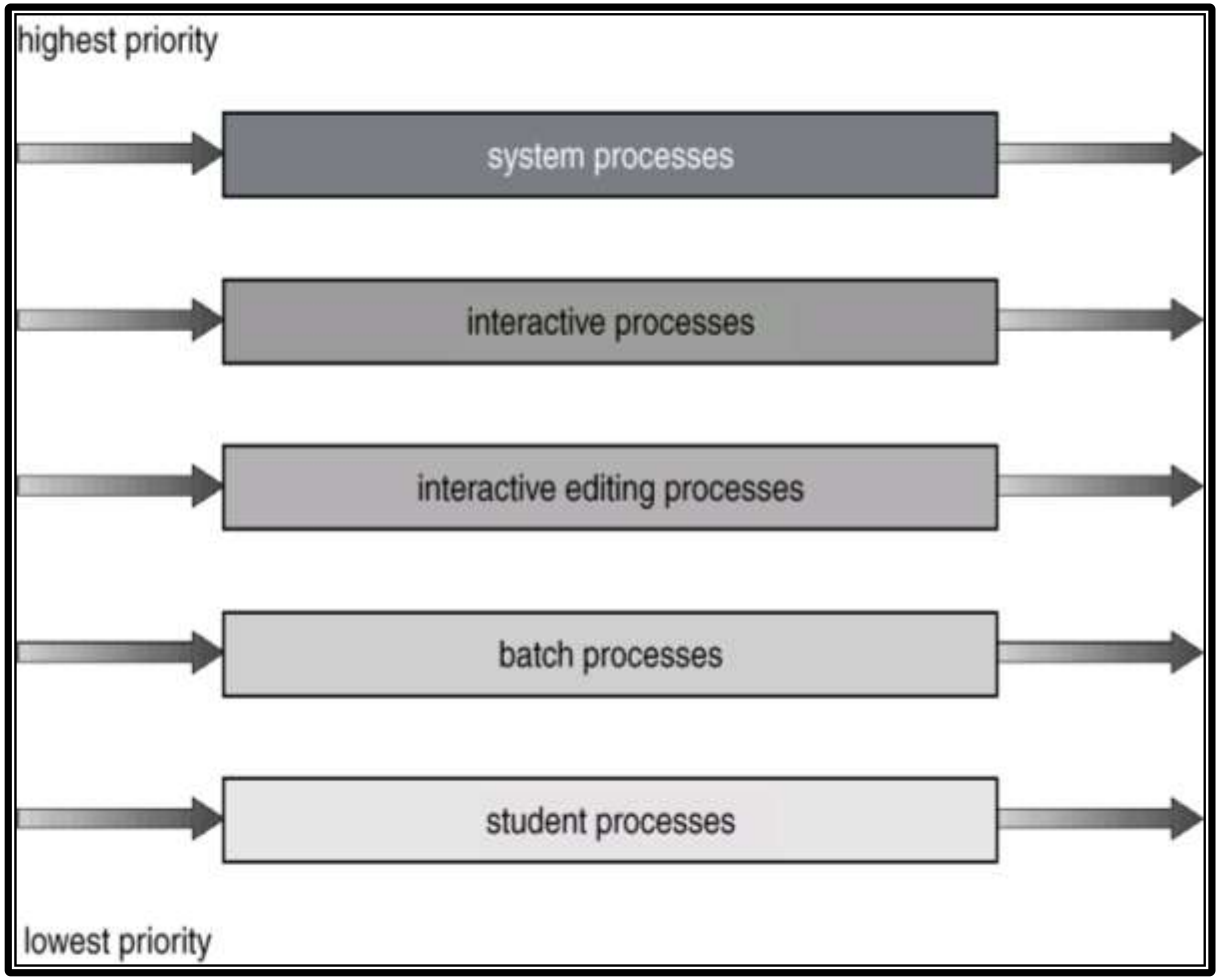


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm,
  foreground – RR
  background – FCFS
- Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



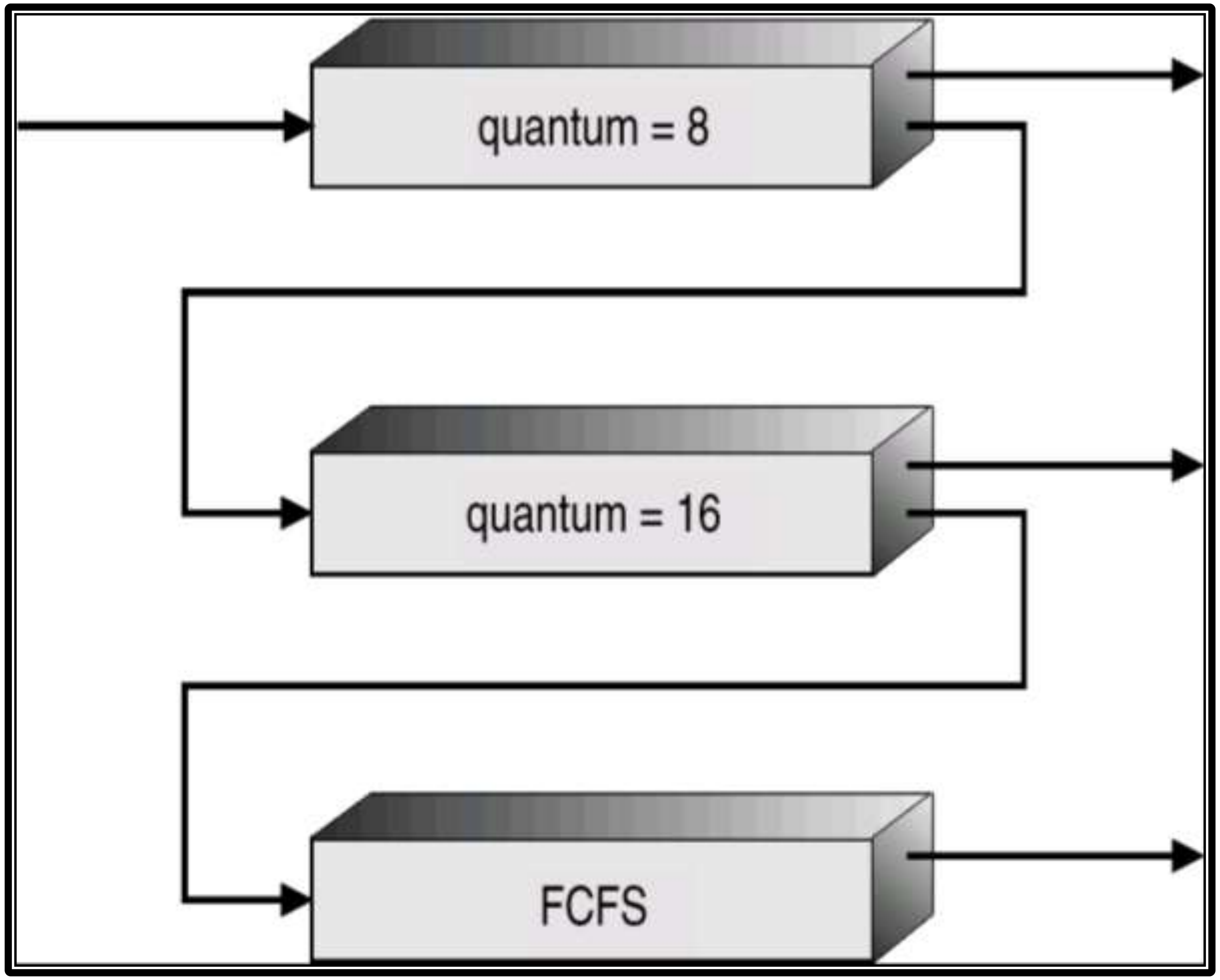
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-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 Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – 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.

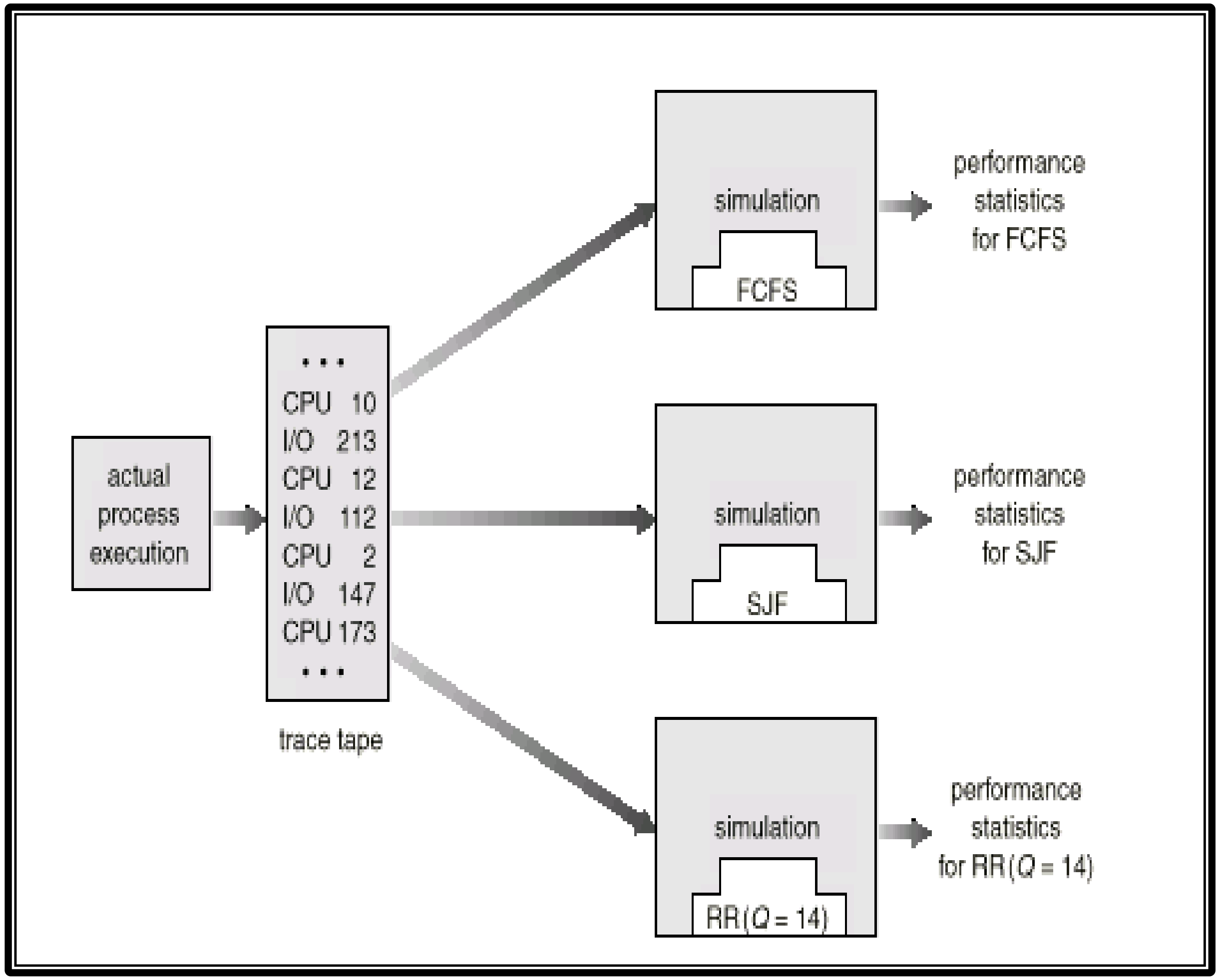
Multilevel Feedback Queues



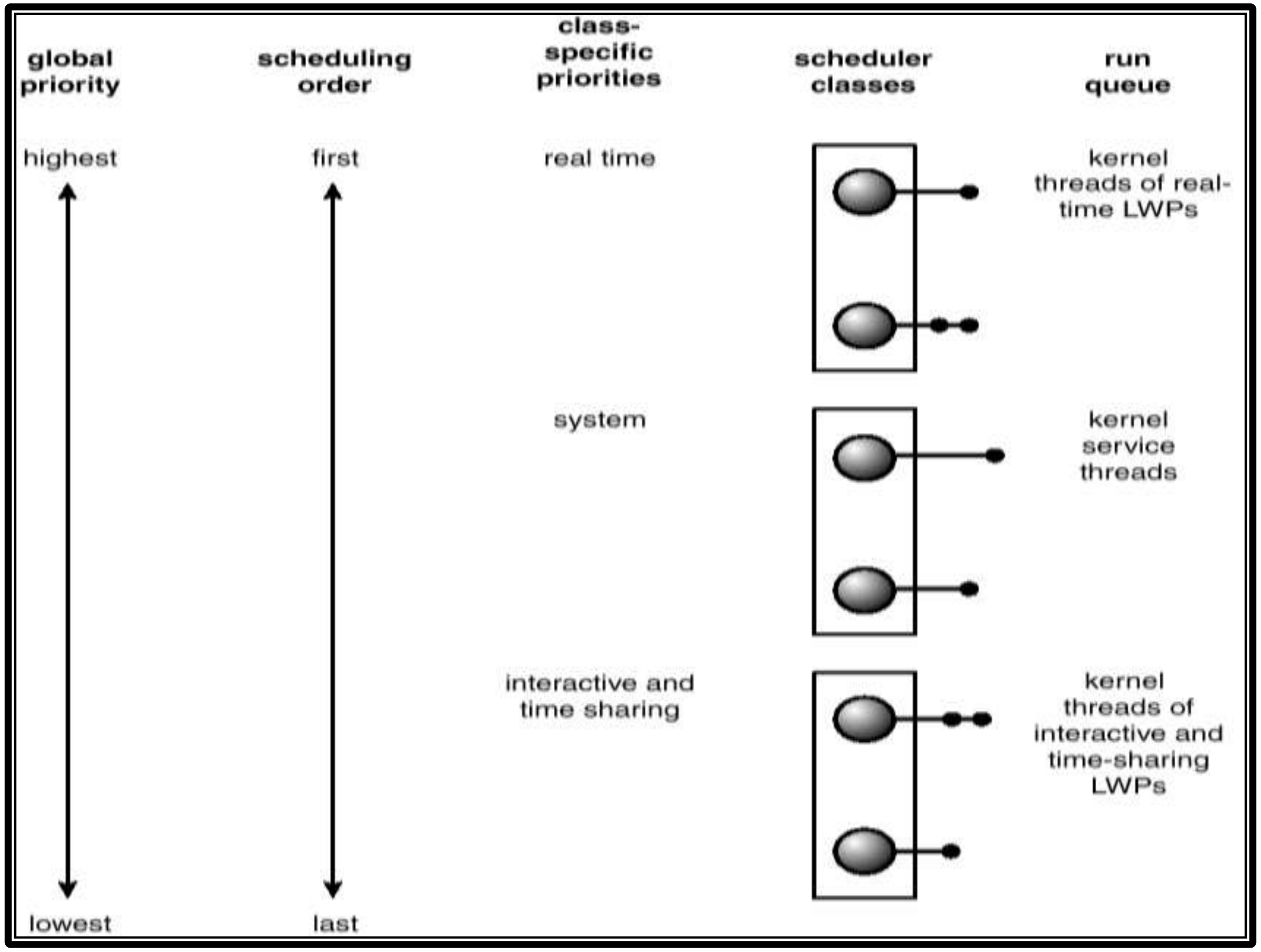
Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queueing models
- Implementation

Evaluation of CPU Schedulers by Simulation



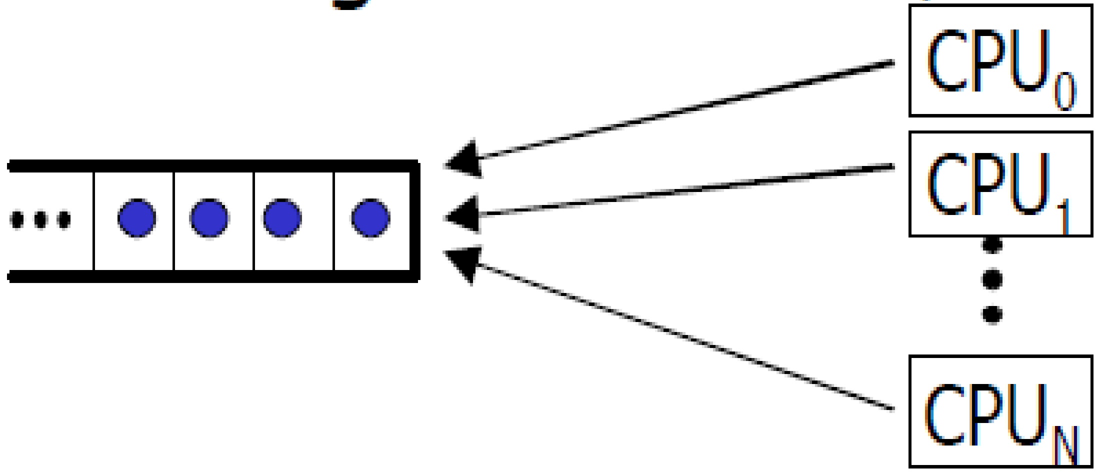
Solaris 2 Scheduling


global priority
scheduling order
class-specific priorities
scheduler classes
run queue
highest
first
real time
kernel threads of real-time LWPs
system
kernel service threads
interactive and time sharing
kernel threads of interactive and time-sharing LWPs
lowest
last

Multiple-Processor Scheduling

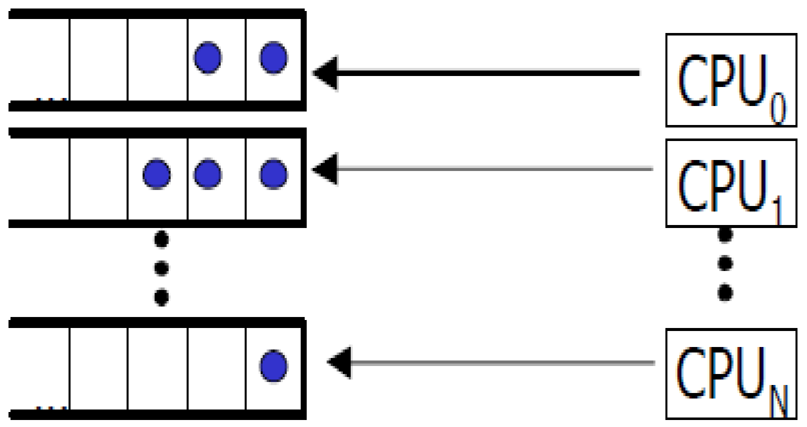
- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* :Processors are identical in their functionality.
- *Load sharing* is possible with homogenous systems.

Option 1: Single Shared Queue



- Common ready queue for the processes and are scheduled onto any available processor.
- Two scheduling approaches may be used:
- Each processor is self scheduling: Each processor examines the common ready queue and selects a process to execute.
- Appoint one processor as a scheduler for the other processor.
- Some system carry this structure one step further, by having all scheduling decisions, I/O processing and other system activities are handled by one single processor-the master server. The other processors only execute user code. This is asymmetric multiprocessing.

Option 2: Per-CPU Ready Queue



- Separate queue for each processor.
- In this case it may possible that one processor is idle having empty queue while others are very busy.