OPERATING SYSTEMS

SCHEDULING

Jerry Breecher

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

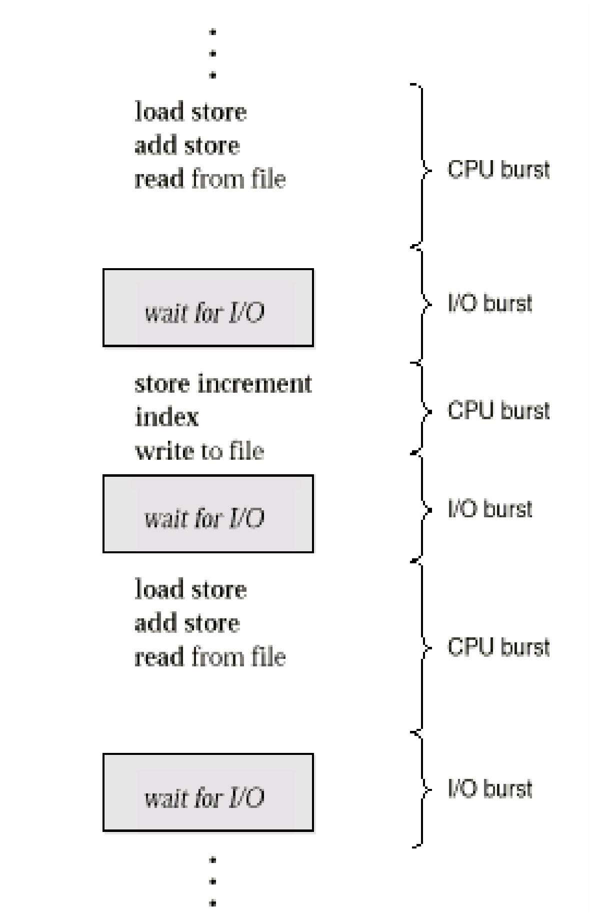
What Is In This Chapter?

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Operating Systems Examples
- Java Thread Scheduling
- Algorithm Evaluation

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.



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*

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

CPU SCHEDULING

Criteria For Performance Evaluation

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

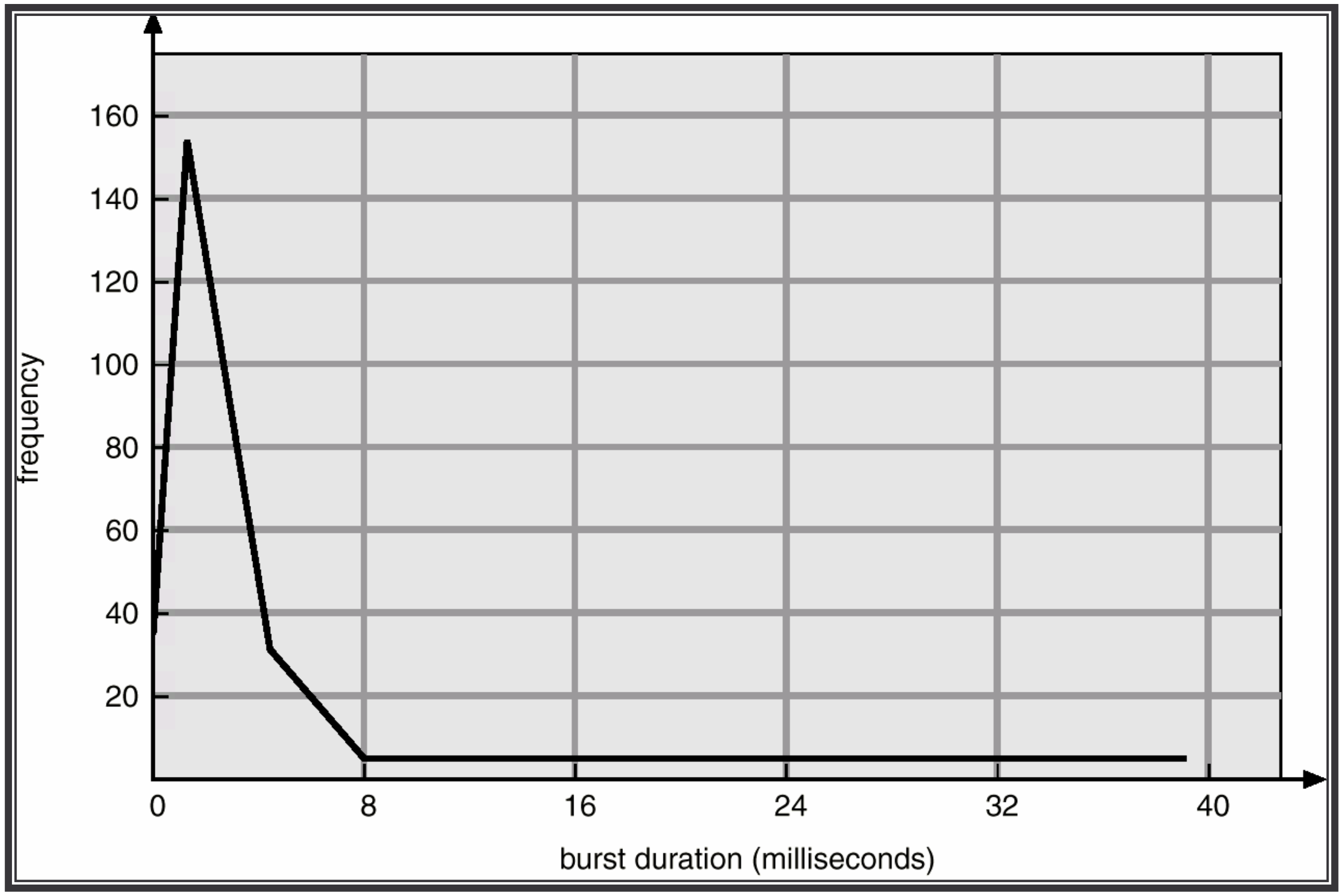
UTILIZATION	The fraction of time a device is in use. (ratio of in-use time / total observation time)
THROUGHPUT	The number of job completions in a period of time. (jobs / second)
SERVICE TIME	The time required by a device to handle a request. (seconds)
QUEUEING TIME	Time on a queue waiting for service 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.)

CPU SCHEDULING

Scheduling Behavior

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?

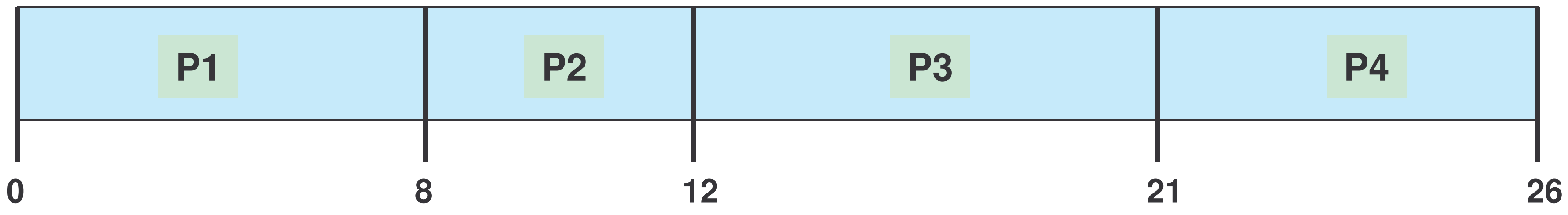
CPU SCHEDULING

Scheduling Algorithms

EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

FCFS



Average wait = ((8-0) + (12-1) + (21-2) + (26-3))/4 = 61/4 = 15.25

Residence Time at the CPU

CPU SCHEDULING

Scheduling Algorithms

SHORTEST JOB FIRST:

- Optimal for minimizing queueing time, but impossible to implement. Tries to predict the process to schedule based on previous history.
- Predicting the time the process will use on its next schedule:

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

Here:	t(n+1)	is time of next burst.
	t(n)	is time of current burst.
	T(n)	is average of all previous bursts .
	W	is a weighting factor emphasizing current or previous bursts.

CPU SCHEDULING

Scheduling Algorithms

PREEMPTIVE ALGORITHMS:

- Yank the CPU away from the currently executing process when a higher priority process is ready.
- Can be applied to both Shortest Job First or to Priority scheduling.
- Avoids "hogging" of the CPU
- On time sharing machines, this type of scheme is required because the CPU must be protected from a run-away low priority process.
- Give short jobs a higher priority – perceived response time is thus better.
- What are average queueing and residence times? Compare with FCFS.

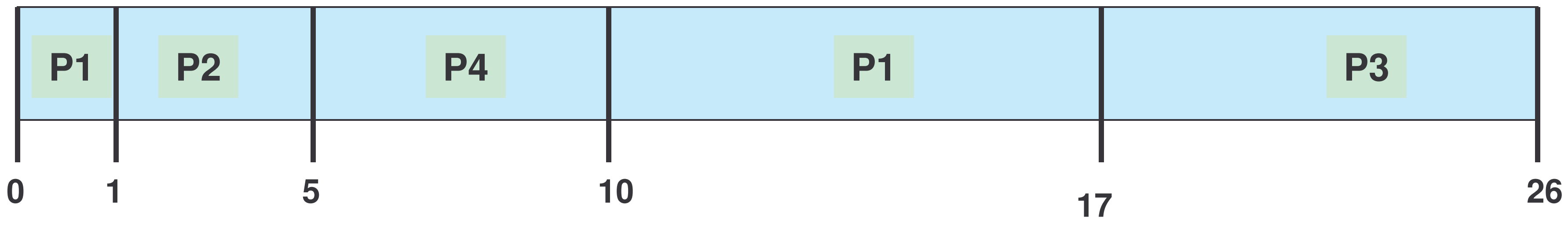
CPU SCHEDULING

Scheduling Algorithms

EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

Preemptive Shortest Job First



Average wait = ((5-1) + (10-3) + (17-0) + (26-2))/4 = 52/4 = 13.0

CPU SCHEDULING

Scheduling Algorithms

PRIORITY BASED SCHEDULING:

- Assign each process a priority. Schedule highest priority first. All processes within same priority are FCFS.
- Priority may be determined by user or by some default mechanism. The system may determine the priority based on memory requirements, time limits, or other resource usage.
- **Starvation** occurs if a low priority process never runs. Solution: build aging into a variable priority.
- Delicate balance between giving favorable response for interactive jobs, but not starving batch jobs.

CPU SCHEDULING

Scheduling Algorithms

ROUND ROBIN:

- Use a timer to cause an interrupt after a predetermined time. Preempts if task exceeds it’s quantum.

- Train of events
  Dispatch
  Time slice occurs OR process suspends on event
  Put process on some queue and dispatch next

- Use numbers in last example to find queueing and residence times. (Use quantum = 4 sec.)

- Definitions:
  - **Context Switch** Changing the processor from running one task (or process) to another. Implies changing memory.
  - **Processor Sharing** Use of a small quantum such that each process runs frequently at speed 1/n.
  - **Reschedule latency** How long it takes from when a process requests to run, until it finally gets control of the CPU.

CPU SCHEDULING

Scheduling Algorithms

ROUND ROBIN:

- Choosing a time quantum
  - Too short - inordinate fraction of the time is spent in context switches.
  - Too long - reschedule latency is too great. If many processes want the CPU, then it's a long time before a particular process can get the CPU. This then acts like FCFS.
  - Adjust so most processes won't use their slice. As processors have become faster, this is less of an issue.

CPU SCHEDULING

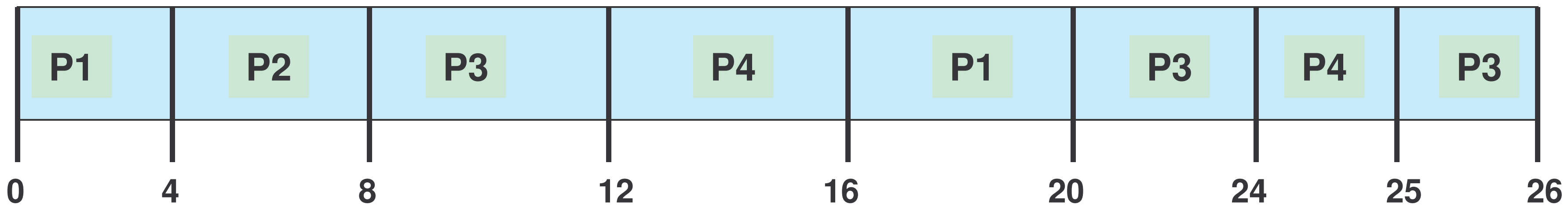
Scheduling Algorithms

EXAMPLE DATA:

Process	Arrival Time	Service Time
1	0	8
2	1	4
3	2	9
4	3	5

Note:
Example violates rules for quantum size since most processes don't finish in one quantum.

Round Robin, quantum = 4, no priority-based preemption



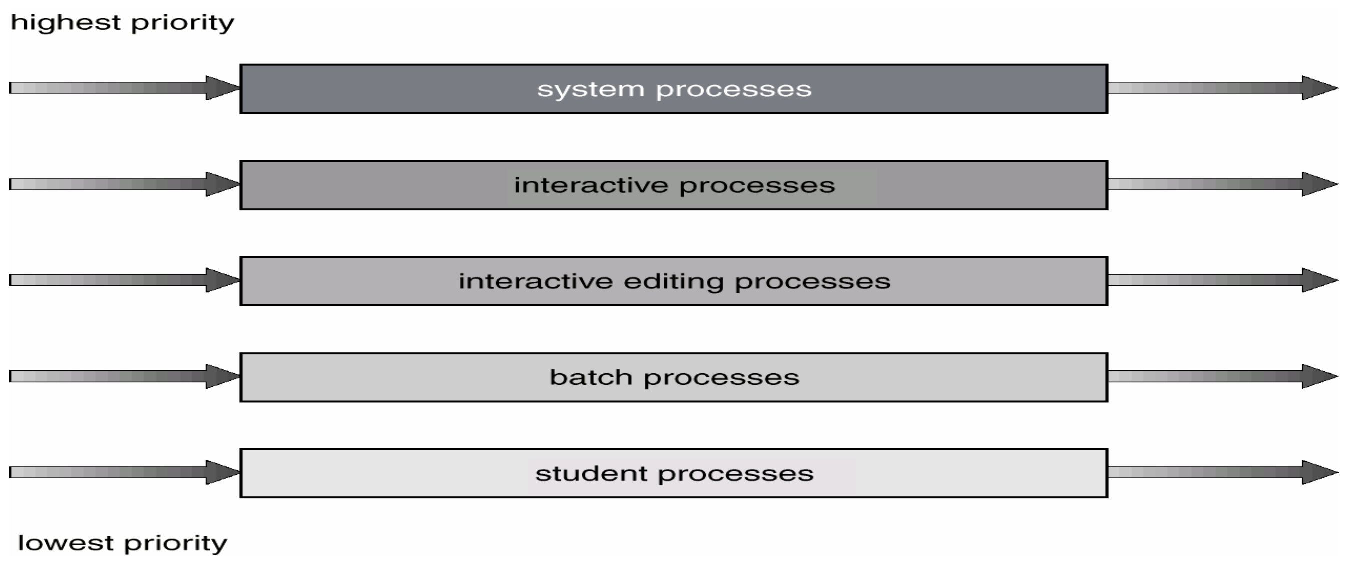
Average wait = ((20-0) + (8-1) + (26-2) + (25-3))/4 = 74/4 = 18.5

CPU SCHEDULING

Scheduling Algorithms

MULTI-LEVEL QUEUES:

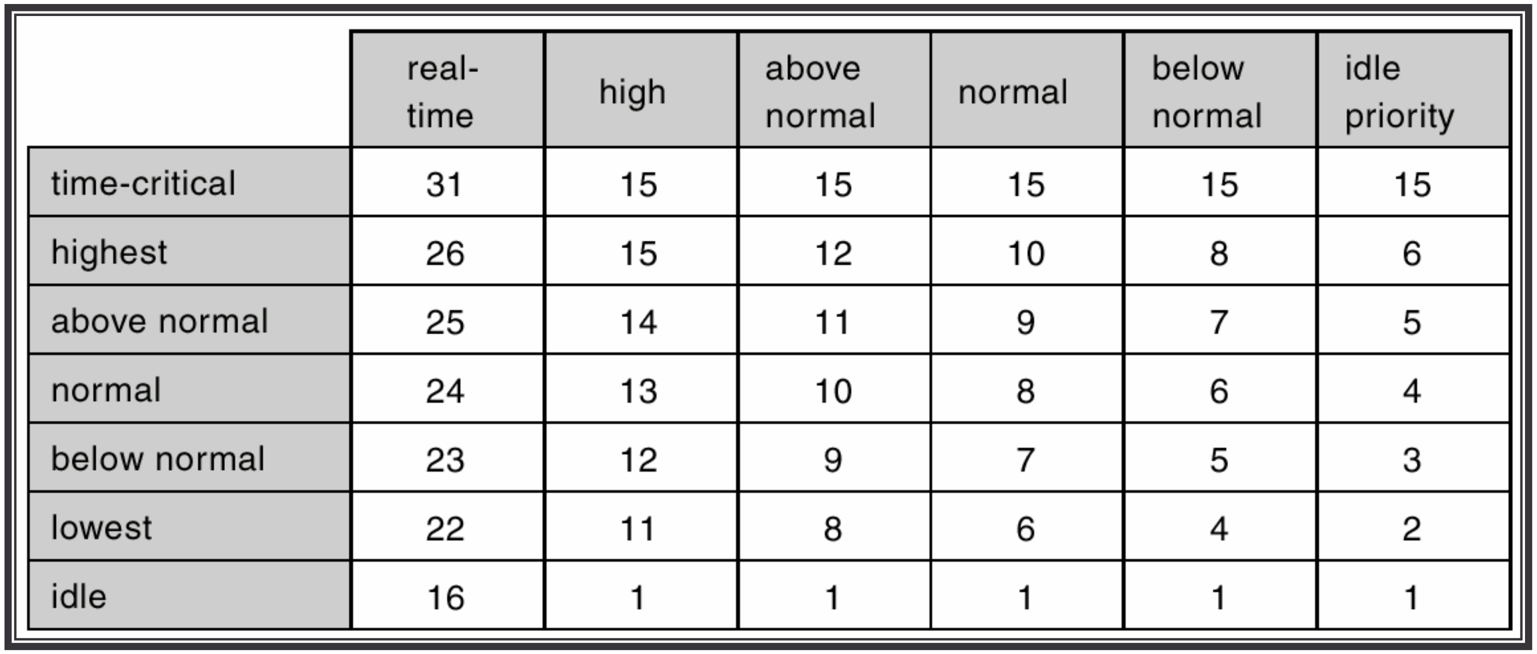
- Each queue has its scheduling algorithm.
- Then some other algorithm (perhaps priority based) arbitrates between queues.
- Can use feedback to move between queues
- Method is complex but flexible.
- For example, could separate system processes, interactive, batch, favored, unfavored processes



CPU SCHEDULING

Using Priorities

Here's how the priorities are used in Windows



	real-time	high	above normal	normal	below normal	idle priority
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

CPU SCHEDULING

Scheduling Algorithms

MULTIPLE PROCESSOR SCHEDULING:

- Different rules for homogeneous or heterogeneous processors.
- Load sharing in the distribution of work, such that all processors have an equal amount to do.
- Each processor can schedule from a common ready queue (equal machines) OR can use a master slave arrangement.

Real Time Scheduling:

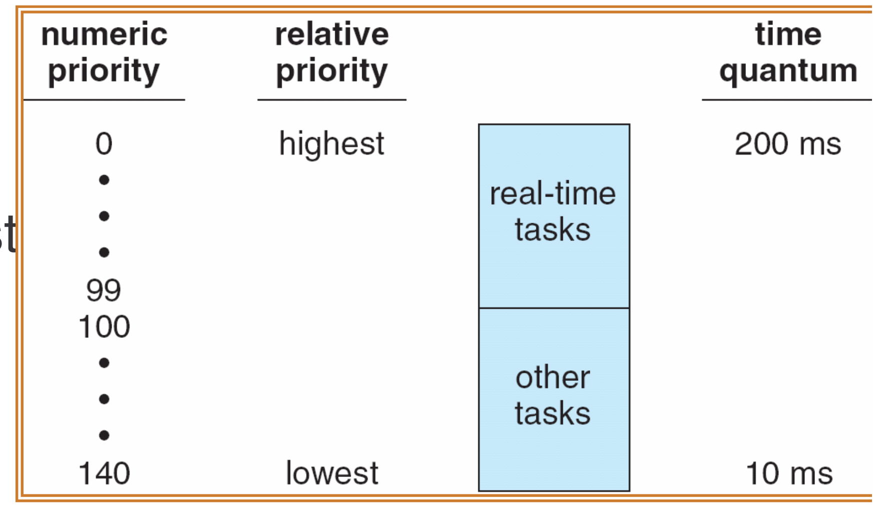
- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

CPU SCHEDULING

Linux Scheduling

Two algorithms: time-sharing and real-time

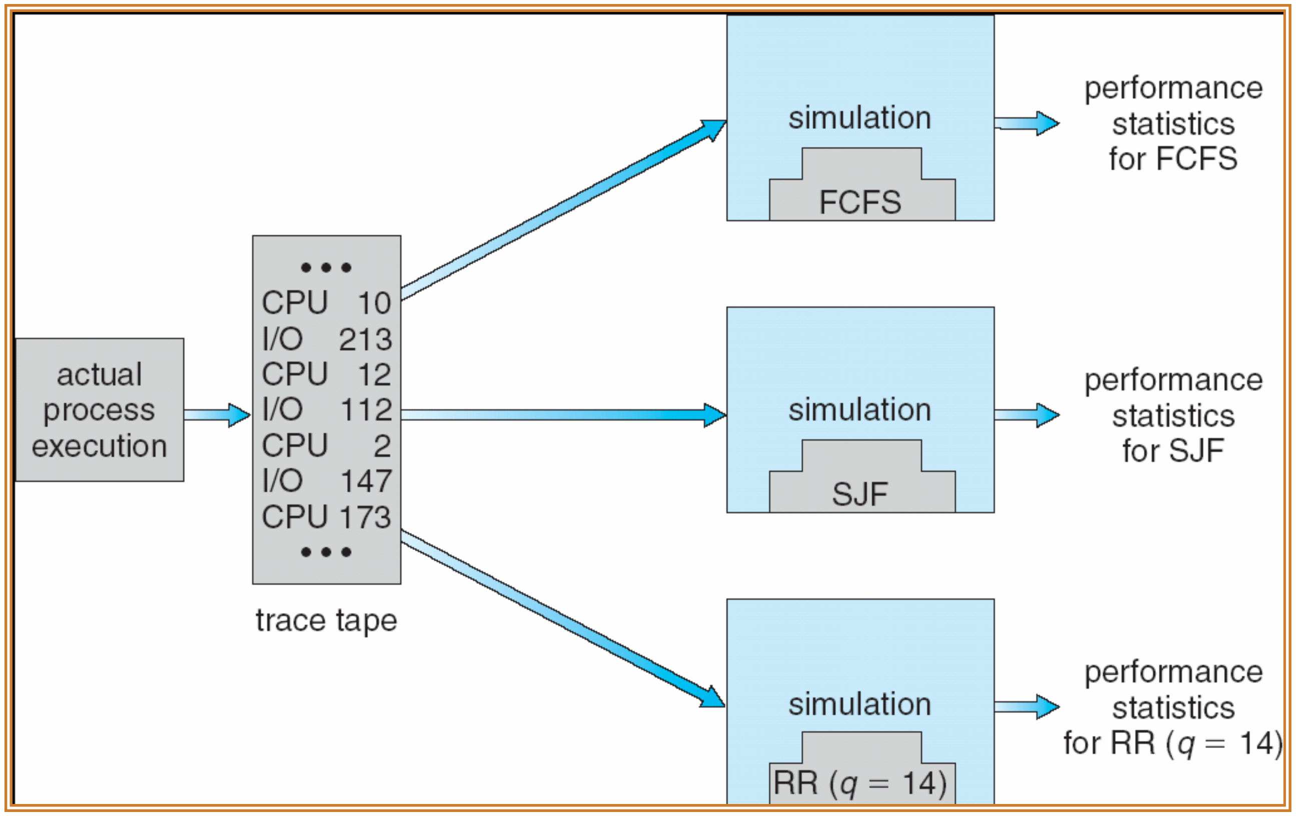
- **Time-sharing**
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all processes have credit = 0, recrediting occurs
    - Based on factors including priority and history
- **Real-time**
  - Soft real-time
  - Posix.1b compliant – two classes
    - FCFS and RR
    - Highest priority process runs first



CPU SCHEDULING Algorithm Evaluation

How do we decide which algorithm is best for a particular environment?

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



CPU SCHEDULING

WRAPUP

We've looked at a number of different scheduling algorithms.

Which one works the best is application dependent.

General purpose OS will use priority based, round robin, preemptive

Real Time OS will use priority, no preemption.