Chapter 5:

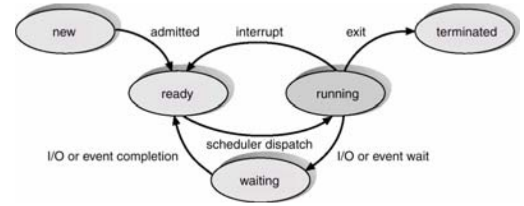
CPU Scheduling

Presented By: Dr. El-Sayed M. El-Alfy

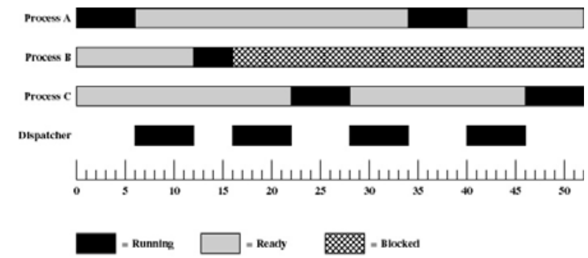
Note: Most of the slides are compiled from the textbook and its complementary resources

Recap: Process states and transitions

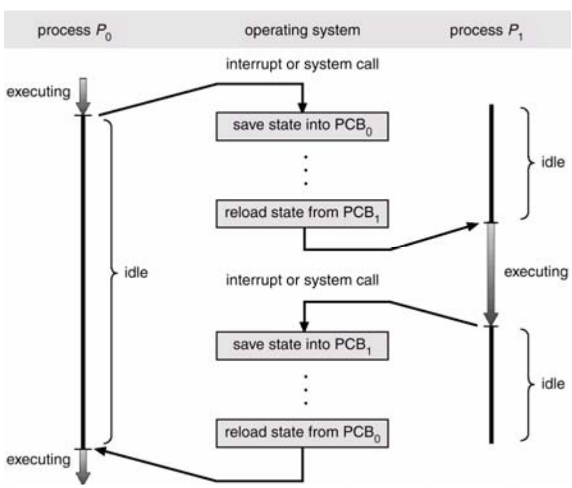
- State diagram



- Example



Recap: Context Switching



Objectives/Outline

Objectives

- Introduce CPU scheduling
- Describe various CPU-scheduling algorithms
- Discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

Outline

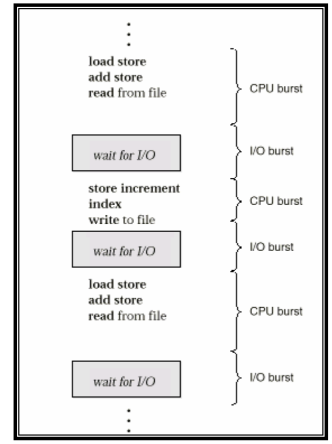
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

Basic Concepts

- Multiprogramming systems provide interleaved execution of several processes to give an illusion of many simultaneously executing processes.
  - Computer can be a single-processor or multi-processor machine.
  - OS must keep track of the state for each active process and make sure that the correct information is properly installed when a process is given control of the CPU.
- By switching CPU among processes can make the computer more productive (i.e. enhance CPU utilization)

Basic Concepts (cont.)

- Nature of Processes
  - Not all processes have an even mix of CPU and I/O usage
  - CPU-BOUND process
    - A number crunching program may do a lot of computation and minimal I/O
  - I/O-BOUND process
    - A data processing job may do little computation and a lot of I/O
- Required OS components:
  - Dispatcher
  - Scheduler



Alternating Sequence of CPU And I/O Bursts

Dispatcher

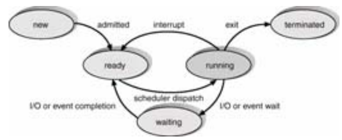
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - Context switch – occurs when a process exchange is made between ready and run queues; OS must save the state of the running process and restore the state of the ready process
  - Switching 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

- Selects from among the processes in memory that are ready to execute, and allocates the CPU cycles to one of them
- Types of Schedulers
  - Job scheduler (long-term scheduler)
    - In a batch system, job scheduler decides as jobs arrive to the system which jobs to let into memory and in which order
    - Occurs less frequently
  - CPU scheduler (short-term scheduler)
    - Decides which process to run next from those waiting in the ready queue
    - Short-term scheduling only deals with jobs that are currently resident in memory
    - Occurs frequently
  - Swapper (medium-term scheduler)
    - Involves suspending or resuming processes by swapping (rolling) them out of or into memory (from disk)

CPU Scheduling

- 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 non-preemptive
- All other scheduling is preemptive
- Non-preemptive scheduler means: a process is never FORCED to give up control of the CPU. The process gives up control of the CPU only
  - If it isn't using the CPU
  - If it is waiting for I/O
  - If it is finished using the CPU
- Preemptive scheduling is forcing a process to give up control of the CPU

Scheduling Criteria

- Scheduling is an optimization task – it is performed in such a way to achieve "good performance" of some criteria
- There are many factors to consider:
  - CPU utilization: percentage of time the CPU is busy; keep the CPU as busy as possible
  - Throughput: number of jobs completed per time unit
  - Total service time (turnaround time): time from submission to completion of a job
  - Waiting Time: amount of time a job spends in the ready queue
  - Response time: time until the system starts to respond to a command
    - much more useful in interactive systems

Additional Scheduling Criteria

- There are also other factors to consider:
  - Priority/Importance of work – hopefully more important work can be done first
  - Fairness – hopefully eventually everybody is served
    - Implement policies to increase priority as we wait longer… (this is known as "priority aging")
  - Deadlines – some processes may have hard or soft deadlines that must be met
  - Overhead – e.g., data kept about execution activity, queue management, context switches
  - Consistency and/or predictability may be a factor as well, especially in interactive systems

Optimization Criteria

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

- In other words, we want to maximize CPU utilization and throughput AND minimize turnaround, waiting, and response times

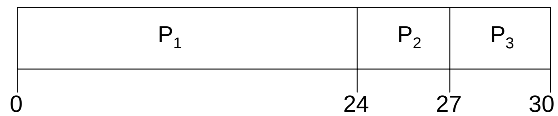
Scheduling Algorithms

- First-Come, First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority
- Round Robin (RR)
- Multi-Level Queue (MLQ)
- Multi-Level Feedback Queue (MLFQ)

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

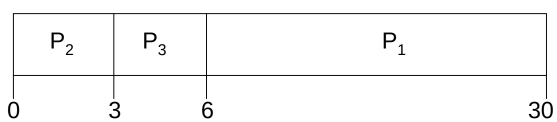
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3 The Gantt Chart for the schedule is:



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

FCFS Scheduling (Cont.)

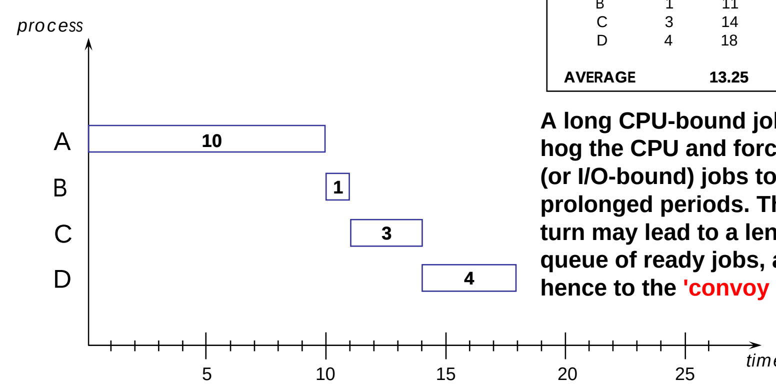
- Suppose that the processes arrive in the order P_2 , P_3 , P_1
- The Gantt chart for the schedule is:



- 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

process	service time t_s	turnaround time t_t	waiting time t_w
A	10	10	0
B	1	11	10
C	3	14	11
D	4	18	14
AVERAGE		**13.25**	**8.75**



A long CPU-bound job may hog the CPU and force shorter (or I/O-bound) jobs to wait for prolonged periods. This in turn may lead to a lengthy queue of ready jobs, and hence to the 'convoy effect'

FCFS Scheduling (Cont.)

- FCFS is:
  - Non-preemptive
  - Ready queue is a FIFO queue
  - Jobs arriving are placed at the end of ready queue
  - First job in ready queue runs to completion of CPU burst
- Advantages: simple, low overhead
- Disadvantages: long waiting time, inappropriate for interactive systems, large fluctuations in average turnaround time are possible

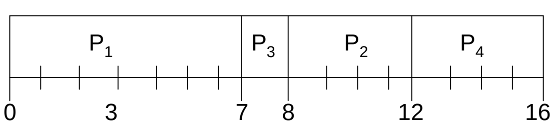
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - Nonpreemptive – once CPU is given to the process it cannot be preempted until the process completes its CPU burst
  - Preemptive – if a new process arrives with CPU burst length less than the 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

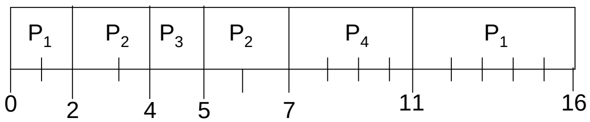
SRTF - Shortest Remaining Time First

- Preemptive version of SJF
- Ready queue ordered on length of time till completion (shortest time to complete first, STCF)
- Arriving jobs inserted at proper position
- shortest job
  - Runs to completion (i.e. CPU burst finishes) or
  - Runs until a job with a shorter remaining time arrives (i.e. placed in the ready queue)

Example of Preemptive SJF (i.e., SRTF)

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

- Preemptive SJF (i.e., SRTF)



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

Shortest-Job-First (SJF) Scheduling

- Ready queue treated as a priority queue based on smallest CPU-time requirement
  - Arriving jobs inserted at proper position in queue
  - Shortest job (1st in queue) runs to completion
- In general, SJF is often used in long-term scheduling
- Advantages: provably optimal w.r.t. average waiting time
- Disadvantages:
  - Starvation is possible!
  - Unimplementable at the level of short-term CPU scheduling.
  - Can do it approximately: use exponential averaging to predict length of next CPU burst
    ==> pick shortest predicted burst next!

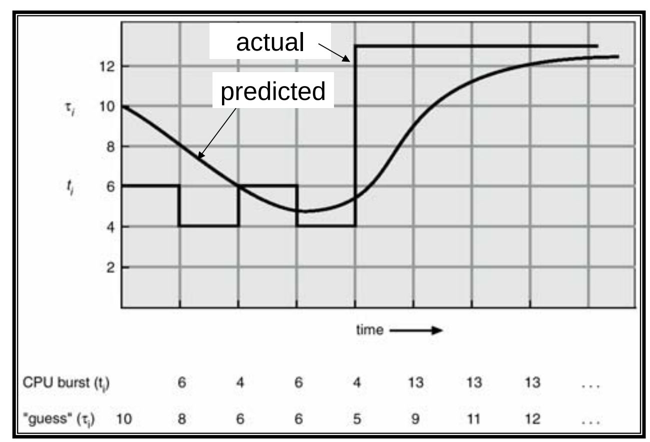
Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, and using exponential averaging
  1. t_n = actual lenght of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next (i.e., $n^{th}+1$) CPU burst
  3. τ_n = predicted value for the n^{th} CPU burst
  4. $\alpha, 0 \le \alpha \le 1$
  5. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$

α = 0 implies making no use of recent history ($\tau_{n+1} = \tau_n$)
α = 1 implies $\tau_{n+1} = t_n$ (past prediction not used)
α = 1/2 equally weighted
Ex. Show that older bursts get less and less weight

Prediction of the Length of the Next CPU Burst

This figure is for $\alpha = 0.5$ and $\tau_0 = 10$

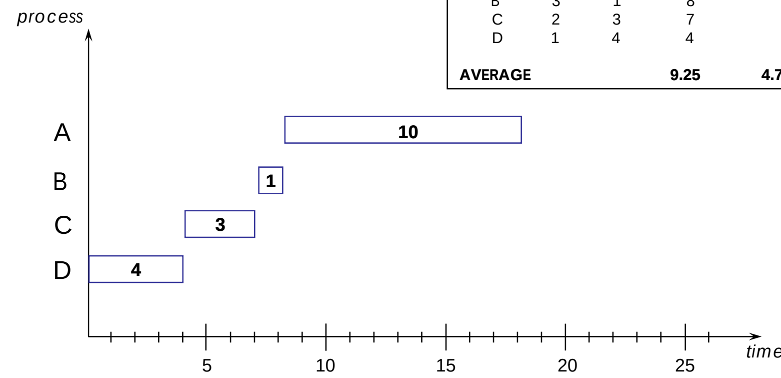


Priority Scheduling

- A priority number (integer) is associated with each process
  - Priority can be internally computed (e.g., may involve time limits, memory usage) or externally (e.g., user type, funds being paid)
  - In SJF, priority is simply the predicted next CPU burst time
- The CPU is allocated to the process with the highest priority (smallest integer might mean highest priority) first
- A priority scheduling mechanism can be
  - Preemptive or Nonpreemptive
- Starvation is a problem, where low priority processes may never execute
- Solution: as time progresses, the priority of the long waiting (starved) processes is increased. This is called priority aging

Priority Scheduling

process	priority	service time t_s	turnaround time t_t	waiting time t_w
A	4	10	18	8
B	3	1	8	7
C	2	3	7	4
D	1	4	4	0
AVERAGE			**9.25**	**4.75**



Round Robin (RR)

- RR is designed especially for time sharing systems
- RR reduces the penalty that short jobs suffer with FCFS by preempting running jobs periodically
- Each process gets a small unit of CPU time (time quantum or time slice), usually 10-100 milliseconds
- The CPU blocks the current job when its reserved time-slice is exhausted
  - The current job is then put at the end of the ready queue if it has not yet completed
  - If the current job is completed, it will exit the system (terminate)

Example of RR with Time Quantum = 20

All arrive at time 0

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

Average waiting time=?

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

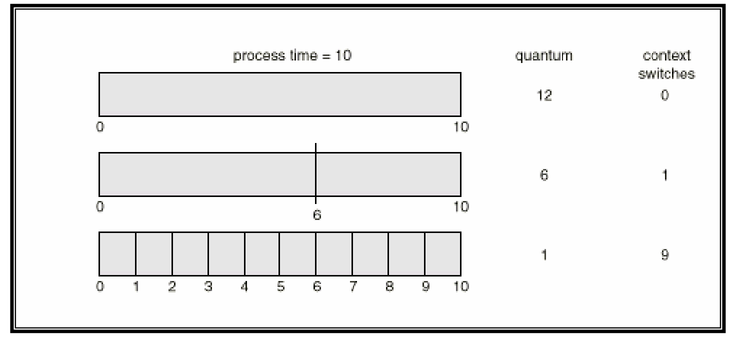
0 20 37 57 77 97 117 121 134 154 162

- Typically, higher turnaround than SJF, but better response time

Round Robin (RR) (cont.)

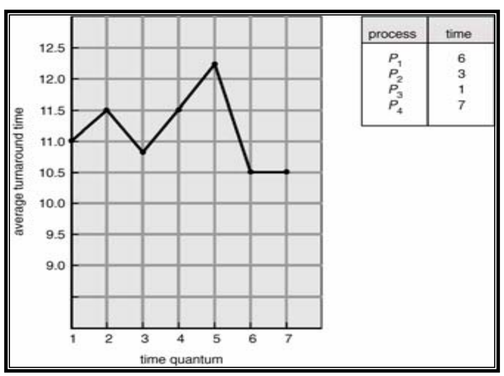
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units
- Performance: the critical issue with the RR policy is the length of the quantum q
  - q is large: RR will behave like FCFS and hence interactive processes will suffer
  - q is small: the CPU will be spending more time on context switching
    - q must be large with respect to context switch, otherwise overhead is too high

Time Quantum and Context Switch Time



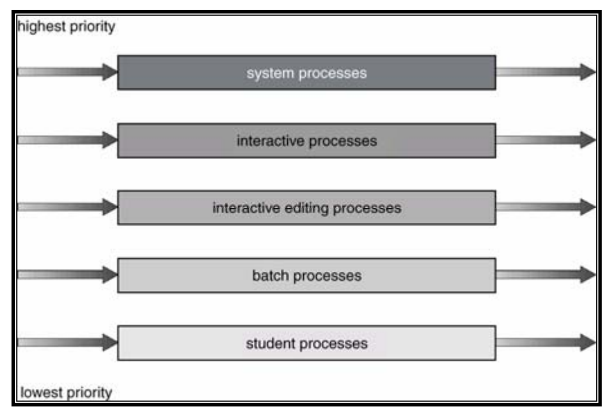
Turnaround Time Varies With The Time Quantum

- Increasing the time quantum does not necessarily improve the average turnaround time!



Multilevel Queue Scheduling

- Used in situations where processes are classified into different groups (with different sch. needs)



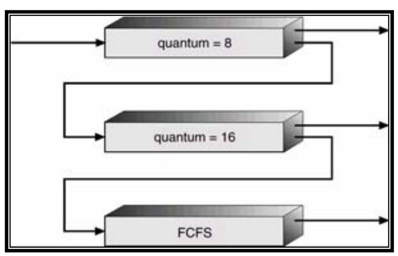
Multilevel Queue Scheduling (cont.)

- Ready queue is partitioned into separate queues and each process is assigned permanently to one queue
  - For example, foreground (interactive) and background (batch)
- Each queue has its own scheduling algorithm:
  - Foreground – RR (better response)
  - Background – FCFS (less overhead)
- Scheduling must be done between the queues
  - Commonly using fixed-priority preemptive scheduling (foreground then background)
    - Possibility of starvation
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes
    - e.g., 80% to foreground in RR and 20% to background in FCFS

Multilevel Feedback Queue

- In Multilevel Queue Scheduling, once a process is assigned to a queue it is not allowed to change (less overhead but inflexible)
  - E.g. a foreground and a background processes
- Multilevel Feedback Queue allows a process to change the queue
  - Allows processes to be separated according to their CPU characteristics
  - E.g. if a process needs more time it can shifted to a lower-priority queue; also if a process waits for long time in a low-priority queue, it can be shuffled to a higher-priority queue
  - More general but more complex

Example of Multilevel Feedback Queue



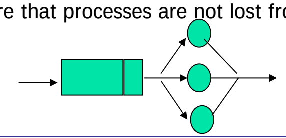
- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, the job receives 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to queue Q_1
  - At Q_1 , the 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

Multiple-Processor Scheduling

- CPU scheduling is more complex when multiple CPUs are available
- A multiprocessor system can have:
  - Homogeneous processors
    - Processors are identical in their functionality. Any available processor can be used to run any of the processes in the ready queue
    - In this class of processors, **load sharing** can occur
  - Heterogeneous processors
    - Processors are not identical. That is, only programs compiled for a given processor's instruction set could be run on that processor

Multiple-Processor Scheduling

- If identical processors are available, then:
  - Can provide a separate ready queue for each processor
  - Can provide a common ready queue
    - Enables load sharing. All processes go into one queue and are scheduled onto any available processor
    - Asymmetric multiprocessing: only one processor accesses the system data structures, alleviating the need for data sharing
      - Master-slave relationship
    - Symmetric multiprocessing: each processor is self-scheduling and may have its own queue
      - We must insure that two processors do not select the same process
      - We must insure that processes are not lost from the ready queue



Real-Time Scheduling

- Real-time computing is divided into two types:
  - 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
- Hard real-time systems:
  - Resource reservation
- Soft real-time systems are less restrictive
  - The dispatch latency must be small
  - The priority of real-time processes must not degrade
    - Disallow aging
  - The priority of non-real-time processes might degrade

Algorithm Evaluation

- How do we select a CPU-scheduling algorithm for a particular system?
- We must define the measures to be used in selecting the CPU scheduler and we must define the relative importance of these measures
- After the selected measures have been defined, than we can evaluate the various algorithms under consideration
- Evaluation methods:
  - Deterministic modeling
  - Queuing models
  - Simulation
  - Implementation

Algorithm Evaluation – Deterministic Modeling

- One type of analytical evaluation
- Deterministic modeling takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Deterministic modeling is
  - Simple and fast
  - Gives exact number allowing algorithms to be compared
- BUT,
  - Requires exact input data
  - Its answers apply to only those input cases
- In general, deterministic modeling is too specific and requires exact knowledge

In class activity

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

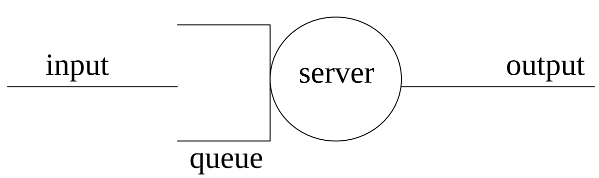
- Average waiting time for each scheduling algorithm: FCFS, SJF, RR

Algorithm Evaluation – Queuing Models

- Since processes running on systems vary with time, there is NO static set of processes to use for deterministic modeling
- We can determine some parameters such as:
  - The distribution of the CPU burst, the distribution of the I/O burst
- These distributions can be measured or estimated
- For example, we have a distribution for CPU burst (service time), arrival time, waiting time, and so on
- Therefore, the computer system can be described by a network of servers

Algorithm Evaluation – Queuing Models

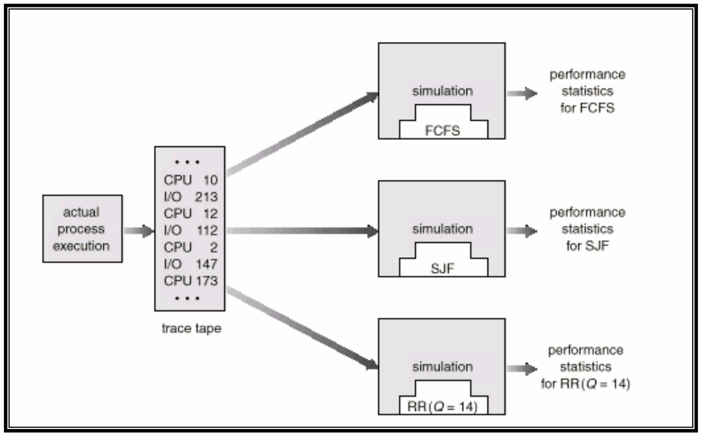
- Queuing analysis can be useful in comparing the performance of scheduling algorithms
- But, queuing analysis is still only an approximation of the real system
  - Since distributions are only estimates of the real pattern



Algorithm Evaluation – Simulations

- Give more accurate results
- Involve programming a model of the computer system
- Software data structures represent the major components of the system; simulator has a variable representing a clock
- The common method to generate the data to drive the simulation is using a random-number generator
  - According to a probability distribution, the random-number generator is programmed to generate processes, CPU burst times, arrivals, etc
- A distribution-driven simulation may be inaccurate
  - The distribution indicates only how many and not the order
  - To resolve this problem, we can use a trace tape obtained by monitoring the real system and recording the sequence of the actual events
- Simulation can be expensive; requires hours of computer time and large amounts of storage space

Evaluation of CPU Schedulers by Simulation



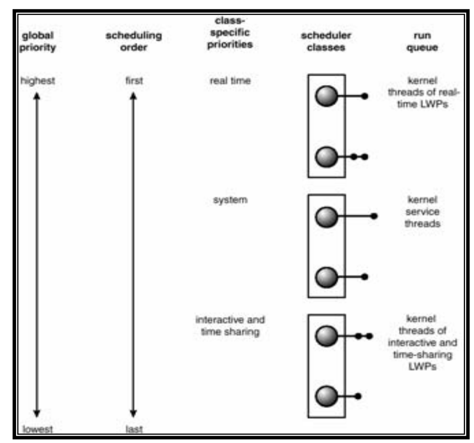
Algorithm Evaluation – Implementation

- The only completely accurate way to evaluate a scheduling algorithm is to code it, deploy it in the real OS, and see how it works
- The major difficulty is the cost involved
  - Coding the algorithm and modifying the OS to support it
  - Users’ reaction to a constantly changing OS may not be accepted
- Another difficulty is that the environment in which the algorithm is used will change as new
  - New programs are coded
  - Performance of the scheduler

Operating Systems Examples

Solaris 2 Scheduling

- Uses priority-based thread scheduling
- Uses four classes, in order of priority:
  - Real time
  - System
  - Time sharing
  - Interactive
- Within each class, there are different priorities and different scheduling algorithms
- Default class for a process is ‘time sharing’
- In time sharing: assign time slices of different lengths using a multilevel feedback queue



Windows 2000 Priorities

	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

Linux Scheduling for time-sharing processes

- When a new task must be chosen, the process with the most credits is selected
- Every time a timer interrupt occurs, the currently running process loses one credit
- When its credit reaches 0 it is suspended and another process gets a chance
- If no runnable process has any credits, every process is re-credited using the formula:

  credits=(credits/2) + priority
- This mixes the process's behaviour history (half its earlier credits) with its priority

End of Chapter 5

Operating System Concepts, 7th Ed. A. Siblerschatz, P. Galvin, and G. Gagne. Addison Wesley, 2005