Chapter 6: CPU Scheduling

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

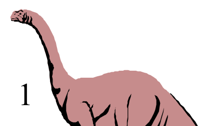
6.1 Basic Concepts

- Maximum CPU utilization obtained with multiprogramming.
- In a uniprocessor system, only one process may run at a time; any other processes must wait until the CPU is free and can be rescheduled.

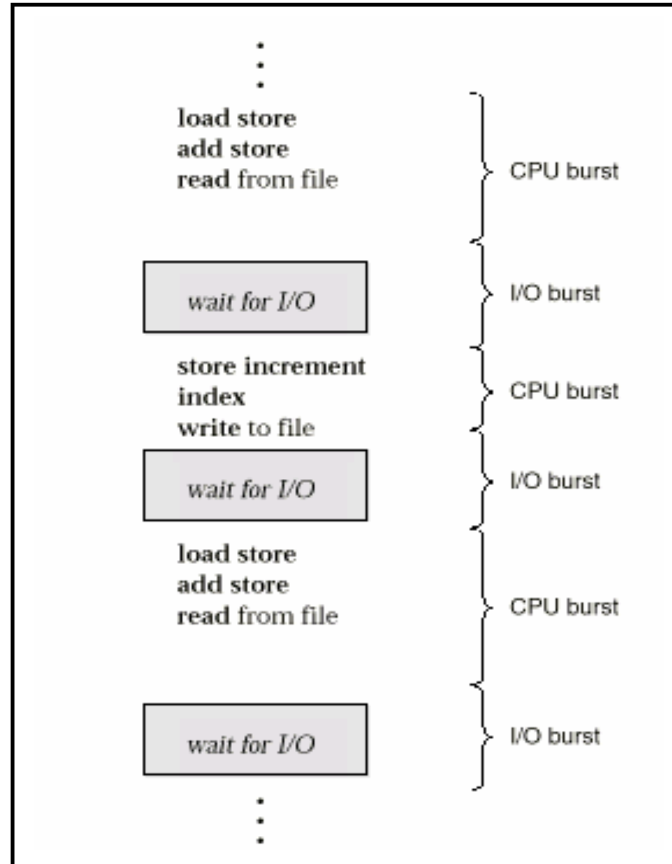
CPU–I/O Burst Cycle

- Process execution consists of a *cycle* of CPU execution and I/O wait.
- Processes alternate between these two states.
- Process execution begins with a **CPU burst**, followed by an **I/O burst**, then another CPU burst … etc
- The last CPU burst will end with a system request to terminate execution rather than with another I/O burst.

- The duration of these CPU burst have been measured.
- An I/O-bound program would typically have many short CPU bursts, A CPU-bound program might have a few very long CPU bursts.
- this can help to select an appropriate CPU-scheduling algorithm.

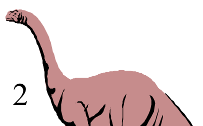


- Alternating Sequence of CPU And I/O Bursts



CPU Scheduler

- Selects from among the processes in memory that are ready to execute (from a ready queue), and allocates the CPU to one of them.
- The selection process is carried out by the short-term scheduler (or CPU scheduler)
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (I/O request).
  2. Switches from running to ready state (when interrupt occurs).
  3. Switches from waiting to ready (completion of I/O).
  4. Terminates.
- Scheduling under 1 and 4 is ***nonpreemptive*** ***(once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. Such as win 3.1 or Apple Macintosh)***.
- All other scheduling is ***preemptive*** ***(incurs a cost, assume that 2 processes sharing data. One may be in the middle of updating the data when it is preempted and the second process is running. The second process may try to read the data which are currently in an inconsistent state).***



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this function 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.

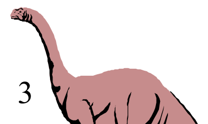
6.2 Scheduling Criteria

Different CPU-scheduling algorithms have different properties and characteristics and may favor one class of processes over another. The characteristics used for comparison can make a substantial difference in the determination of the bets algorithm. The criteria include :-

- CPU utilization – keep the CPU as busy as possible (40 % for a lightly loaded system to 90 % for heavily used system)
- Throughput – # of processes that complete their execution per time unit (1 process / hour for long processes, 10 process / second for short transactions)
- Turnaround time – amount of time it takes to execute a particular process (from time of submission to time of completion)
- 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)

Optimization Criteria

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

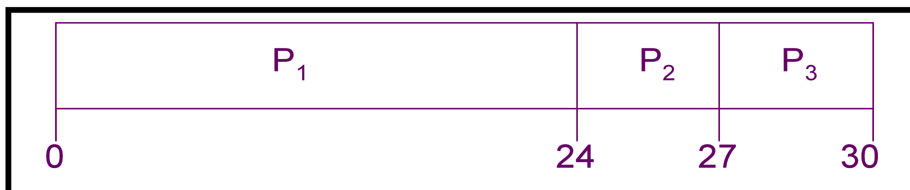


6.3 CPU Scheduling

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

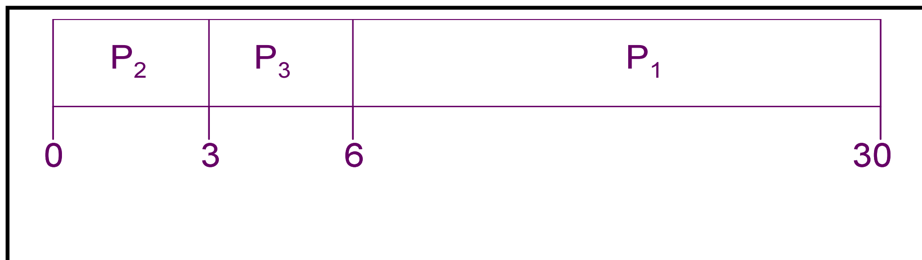
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



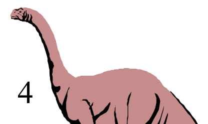
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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



- Average waiting time is often quite long.
- It is nonpreemptive.
- This algorithm has problems when it comes time-sharing systems
- Implemented using FIFO queue.

Look @ the scenario page 158 ?????

Shortest-Job-First (SJR) Scheduling

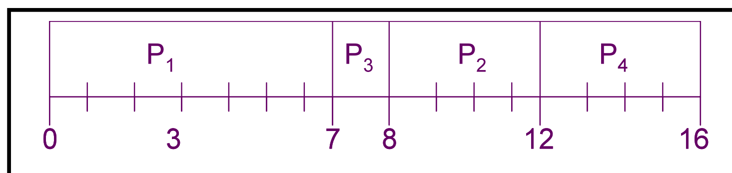
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **preemptive** – if a new process arrives with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

What happens when two jobs arrive with the same length of CPU burst ?????

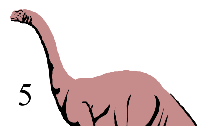
Example of Non-Preemptive SJF

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)



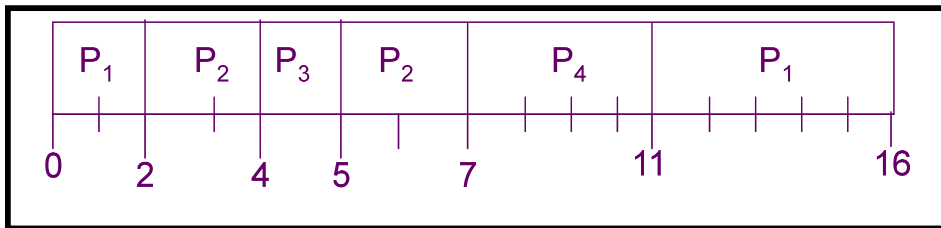
. Average waiting time = (0 + 6 + 3 + 7)/4 = 4



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)

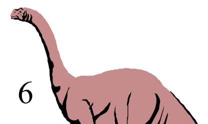


. Average waiting time = (9 + 1 + 0 +2)/4 = 3

What would be the average waiting time if you were using FCFS algorithm ????

Difficulty >>> 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.
- Thus, it is used frequently in long-term scheduling.



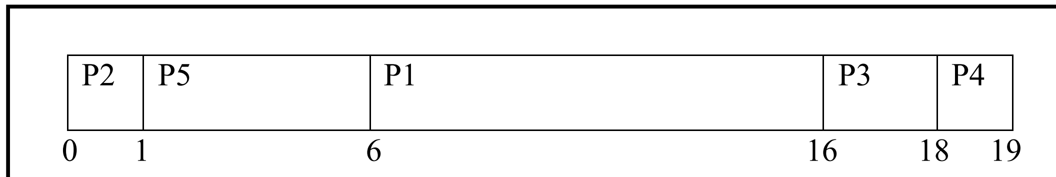
Priority Scheduling

- 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.

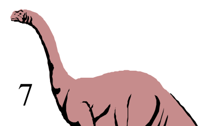
None Preemptive Priority Scheduling

Process	Burst Time	Priority
P1	*10*	*3*
P2	*1*	*1*
P3	*2*	*4*
P4	*1*	*5*
P5	*5*	*2*

The Gantt Chart for the schedule is:



Average Waiting Time for processes = (6 + 0 + 16 + 18 + 1) / 5 = 8.2 milliseconds



Round Robin (RR) Scheduling

- Designed especially for time-sharing systems.
- Same as FCFS scheduling, but preemption is added to switch between processes.
- Each process gets a small unit of CPU time (***time quantum or time slice***), 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.

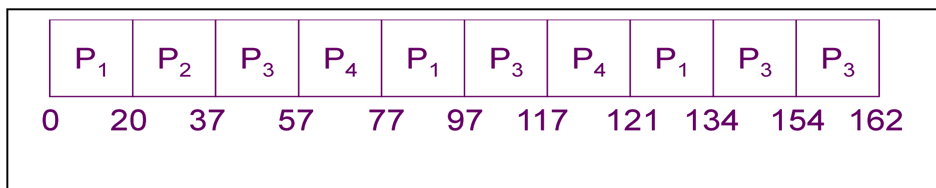
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high.

- The average wait under the RR policy is often quite long.

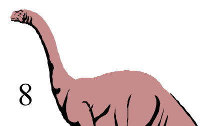
Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



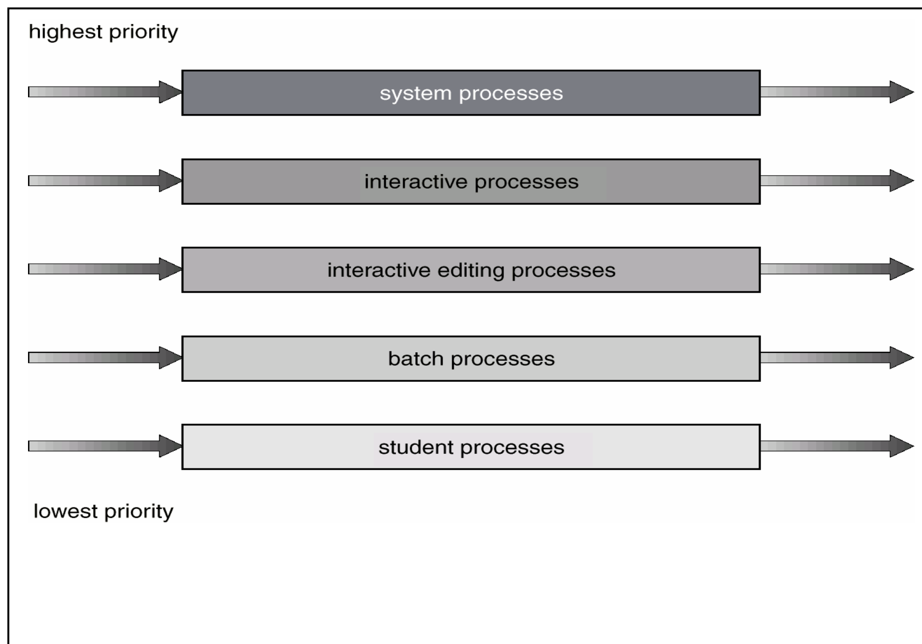
- Typically, higher average turnaround than SJF, but better *response*.

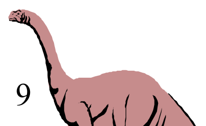


Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues :
  foreground process (interactive)
  background process (batch)
- Each queue has its own scheduling algorithm,
  foreground – RR
  background – FCFS
- Processes are permanently assigned to one queue based on some of property of the process such as memory size, process type , process priority.
- Processes can not move between queues.
- Scheduling must be done between the queues.
  - **Fixed priority scheduling**; (i.e., serve all from foreground first 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% is given to foreground queue in RR while 20% is given to background queue in FCFS

Multilevel Queue Scheduling





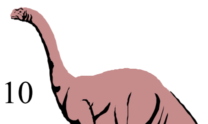
Student Processes queue can not be processed unless all other queues are empty. (Solaris 2 uses this form of algorithm)

Multilevel Feedback Queue Scheduling

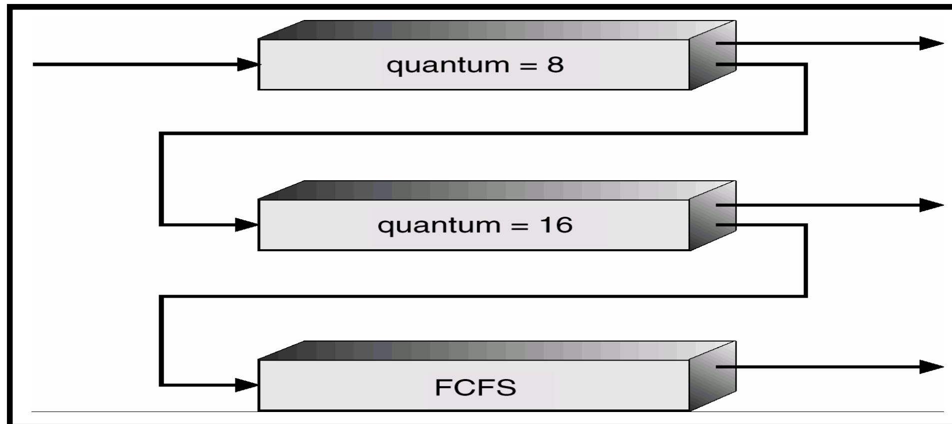
- A process can move between the various queues; aging can be implemented this way.
- If the process uses too much CPU time, it will be moved to a lower – priority queue.
- If the process waits too long in a lower-priority queue, may be moved to a higher-priority queue.
- This form of aging prevents starvation.
- 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

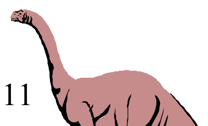


6.4 Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous (Identical) processors* within a multiprocessor, l*oad sharing* can occur.
- Provide separate queue for each processor. In this case, one processor could be idle with an empty queue, while another processor will be very busy.
- Could use a common ready queue, all processes go into one queue and are scheduled onto any available processor.
- Scheduling approaches are :-
    - Each processor is self-scheduling – each processor examines the common ready queue and selects a process to execute
    - One processor is as a scheduler for the other processors, master-slave structure.
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.

6.5 Real-Time Scheduling

- ***Hard real-time*** systems – required to complete a critical task within a guaranteed amount of time. Process submits with a statement of the amount of time needed to complete or perform I/O operation. The scheduler then either admits the process, guaranteeing that the process will complete on time, or reject the request ***(resource reservation)***
- ***Soft real-time*** computing – requires that critical processes receive priority over less fortunate ones.



6.6 Algorithm Evaluation

How do we select a CPU-scheduling algorithm for a particular system ?

One problem is define a criteria in selecting an algorithm

- CPU criteria are max CPU utilization , max throughput , min turnaround time , min waiting time , min response time.

Once the criteria is defined, evaluate various algorithms under consideration .

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queuing models - Simulation
- Implementation – code it, put in the operating system and see how it works – expensive to code , support, maintain

Evaluation of CPU Schedulers by Simulation