Lecture Overview

- Introduction to process scheduling
  - Process scheduling and schedulers
  - Process scheduling criteria
  - Process scheduling algorithms
    - First-come, first-serve
    - Shortest-job-first
    - Priority
    - Round-robin
    - Multilevel queue
    - Multilevel feedback queue

Process Scheduling

- Scheduling is a fundamental operating system function
  - Almost all computer system resources are schedule before being used
  - The CPU is the fundament resource that needs to be shared
- Process scheduling deals with selecting the next process to execute on the CPU
- The goal is to obtain maximum CPU utilization using multiprogramming
- *Even though we say "process scheduling" most of the discussion is equally relevant to threads*

Process Schedulers

- Long-term scheduler (or job scheduler) selects which processes should be brought into the ready queue
  - Determines degree of multiprogramming
  - Not invoked very often
  - Does not exist in most timesharing systems
- Medium-term scheduler
  - Swaps processes out to secondary storage
  - We will cover this in a later lecture
- Short-term scheduler (or CPU scheduler) selects which process should be executed next and allocates CPU
  - Invoked frequently so it must be fast
  - This lecture focuses on the short-term scheduler

Process Scheduling

- Processes fall into two categories
  - I/O-*bound processes* spend more time doing I/O than computations, short CPU bursts
  - CPU-*bound processes* spends more time doing computations, long CPU bursts
  - The type and mixture of types of processes has an impact on determining the best approach to process scheduling
- Process exhibit CPU–I/O burst cycle pattern
  - Process execution consists of a *cycle* of CPU execution and I/O waiting
    - CPU burst is longer for CPU-bound processes

Process Scheduling

- As we already learned, the OS keeps track of processes to be scheduled by maintaining various queues
  - *Ready queue* is the set of all processes residing in main memory, ready and waiting to execute
  - *Device queues* are sets of processes waiting for a specific I/O device
  - *Wait queues* are sets of processes waiting for a specific event
- Processes migrate between the various queues as they execute
- The process scheduler in interested in process on the ready queue

Process Scheduling

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions 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*
- Scheduling under 2 and 3 is *preemptive*

Process Dispatching

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

Process 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
- *Wait 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 timesharing environment)

Optimizing Process Scheduling Criteria

- Maximum CPU utilization
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time

Process Scheduling Algorithms

- There are many process scheduling algorithms for many different types of systems, we will examine some of the most common
  - First-come, first-serve
  - Shortest-job-first
  - Priority
  - Round-robin
  - Multilevel queue
  - Multilevel feedback-queue

Process Scheduling Algorithms

- First-come, first-serve
  - The simplest to understand and the simplest to implement
  - The CPU is allocated to processes as they arrive
  - Processes keep the CPU until they are done with it
    - This is a non-preemptive algorithm
  - This is essentially a FIFO queue (i.e., first-in, first-out)
  - Because of its simplicity, FCFS is not very efficient

Process Scheduling Algorithms

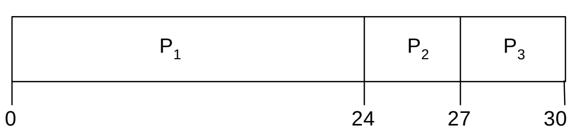
- First-come, first-serve (con't)

Example:

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 is (0 + 24 + 27)/3 = 17

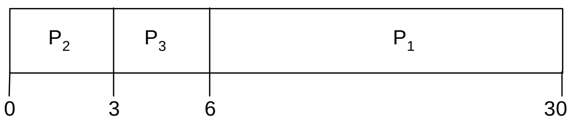
Process Scheduling Algorithms

- First-come, first-serve (con't)

  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 is (6 + 0 + 3)/3 = 3

  Convoy effect short processes behind long process

Process Scheduling Algorithms

- Shortest-job-first
  - The next job to receive the CPU is chosen based which one needs the CPU for the shortest period of time
    - More appropriately, we can associate with each process the length of its next CPU burst
    - Use these lengths to schedule the process with the shortest "next burst" time
  - SJF is provably optimal for average waiting time for a given set of processes
  - A potential starvation problem exists if there are a lot of short jobs, in this case long jobs will never get the CPU

Process Scheduling Algorithms

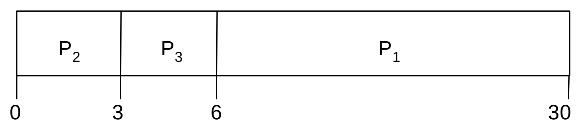
- Shortest-job-first (con't)

Example:

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

SJF



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

Average waiting time is (6 + 0 + 3)/3 = 3

Process Scheduling Algorithms

- Shortest-job-first
  - Two potential schemes for SJF
    - *Non-preemptive* – 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 also known as the *Shortest-Remaining-Time-First (SRTF)*

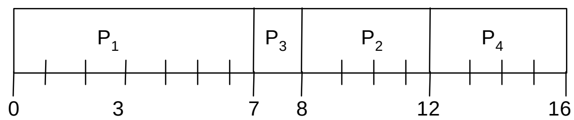
Process Scheduling Algorithms

- Shortest-job-first (con't)

Example:

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 with arrival times)



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

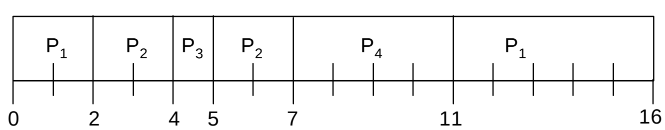
Process Scheduling Algorithms

- Shortest-job-first (con't)

Example:

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 with arrival times)



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

Process Scheduling Algorithms

- Shortest-job-first (con't)
  - What is the real difficulty of SJF?
    - Knowing the length of the next CPU request
      - This is possible for long-term scheduling, but not so easy for short-term scheduling
  - CPU burst length approximation
    - Try to predict the burst length
    - Expect that the next burst will be similar to previous bursts
    - CPU bursts can be predicted as an exponential average of the lengths of previous CPU bursts

Process Scheduling Algorithms

- Shortest-job-first (con't)
  - Exponential average

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\, \tau_n$$

t_n = length of the *n*th CPU burst

τ_n = predicted value historical average

τ_{n+1} = next predicted average value

α = weight of recent history ($0 <= \alpha <= 1$)

Process Scheduling Algorithms

- Priority
  - 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
    - Non-preemptive
  - SJF is a priority scheduling where priority is the predicted next CPU burst time
  - Potential starvation problem
    - Low priority processes may never get to execute
    - One solution is process *aging* – as time progresses increase the priority of the process that have not executed

Process Scheduling Algorithms

- Round-robin
  - Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds
  - After this quantum 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 $(n\text{-}1)q$ time units.
  - Performance
    - q large $\Rightarrow$ FIFO
    - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

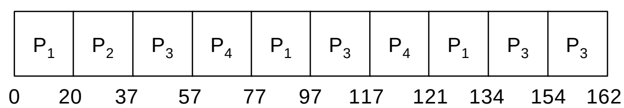
Process Scheduling Algorithms

- Round-robin (con't)

Example: Process Burst Time

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

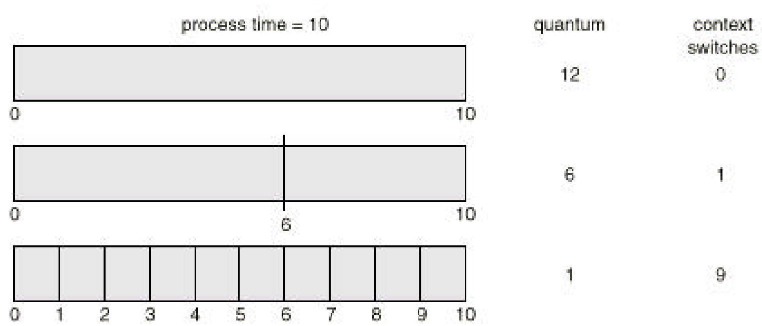
The Gantt chart for these process where the quantum = 20 is:



Typically, higher average turnaround than SJF, but better response

Process Scheduling Algorithms

- Round-robin (con't)
  - Length of time quantum affects context switching



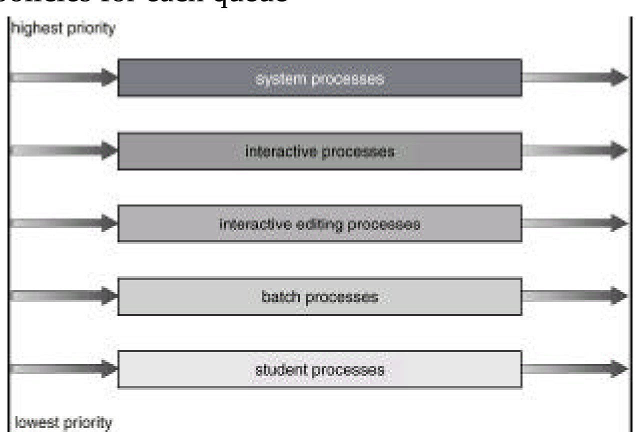
  - The time to perform a context switch is pure overhead (more context switches ==> more overhead)

Process Scheduling Algorithms

- Multilevel queue
  - Ready queue is partitioned into separate queues, for example: foreground (interactive) and background (batch)
  - Each queue can have its own scheduling algorithm, such as round-robin for foreground and FCFS for background
  - Scheduling must be done between the queues
    - Fixed or absolute priority scheduling (i.e., serve all from foreground before any from background)
      - Possibility of starvation
    - Time slicing between queues (i.e., each queue gets a certain amount of CPU time which it can schedule amongst its processes)
      - For example, 80% to foreground and 20% to background

Process Scheduling Algorithms

- Multilevel queue (con't)
  - There may be many queues and associated scheduling policies for each queue

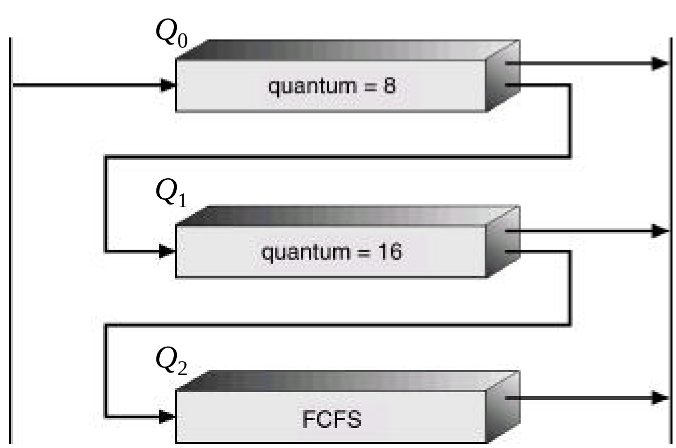


Process Scheduling Algorithms

- 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
  - This is the most general, but most complex algorithm

Process Scheduling Algorithms

- Multilevel feedback-queue (con't)
  - Consider a three level ready queue like this



Process Scheduling Algorithms

- Multilevel feedback-queue (con’t)
  - Three queues
    - Q_0 – time quantum 8 milliseconds round-robin
    - Q_1 – time quantum 16 milliseconds round-robin
    - Q_2 – FCFS
  - Scheduling
    - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, it receives 8 milliseconds
    - If it does not finish in 8 milliseconds, it is moved to queue Q_1
    - At Q_1 it is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is moved to queue Q_2 where it is run FCFS only if other queues are empty and is preempted by the higher level queues

Process Scheduling Algorithms

- Multilevel feedback-queue (con’t)
  - The previous example was just one particular hypothetical implementation of a multilevel feedback-queue
  - The number of queues, the quanta, the scheduling policies, preemption, etc. can vary from one multilevel feedback-queue implementation to the next

Process Scheduling Algorithms

- Scheduling on multiprocessor machines
  - CPU scheduling more complex when multiple CPUs are available
  - Assume homogeneous processors within a multiprocessor
  - *Load sharing* - providing a separate ready queue for each processor
    - Some processors could sit idle
  - *Symmetric multiprocessing (SMP)* – each processor makes its own scheduling decisions
  - *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing

Process Scheduling Algorithms

- Algorithm evaluation
  - *Deterministic modeling* – take a particular predetermined workload and defines the performance of each algorithm for that workload; like we did with the lecture examples
  - *Queuing models* - since queues play an important role in scheduling, if we estimate arrival rates and service rates, it is possible to compute utilization, average queue length, average wait time, etc.
  - *Simulations* - create a model of a computer system and scheduling algorithm(s); data to drive the simulation is created randomly, from mathematical models, or from real system traces
  - *Implementation* - actually implement it and try it in the OS