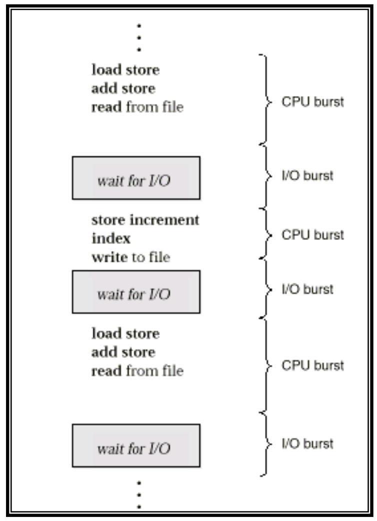
Announcements

- Project #2
  - Is due at 6:00 PM on Friday
- Program #3
  - Posted tomorrow (implements scheduler)
- Reading
  - Chapter 6

Basic Concepts

- CPU–I/O burst cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU burst distribution
  - What are the typical burst sizes of a process's execution?

Burst Cycle



Histogram of Typical CPU-Burst Times

CPU 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*.

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.

Scheduling Criteria

- CPU utilization – % time CPU is in use
- 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 (for interactive environment)

Optimization Criteria

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

Optimization criteria non-performance related

- Predictability, e.g.,
  - job should run in about the same amount of time, regardless of total system load
  - response times should not vary
- Fairness
  - don't starve any processes
- Enforce priorities
  - favor higher priority processes
- Balance resources
  - keep all resources busy

Types of Scheduling

- At least 4 types:
  - long-term - add to pool of processes to be executed
  - medium-term - add to number of processes partially or fully in main memory
  - short-term - which available process will be executed by the processor
  - I/O - which process's pending I/O request will be handled by an available I/O device
- Scheduling changes the ***state*** of a process

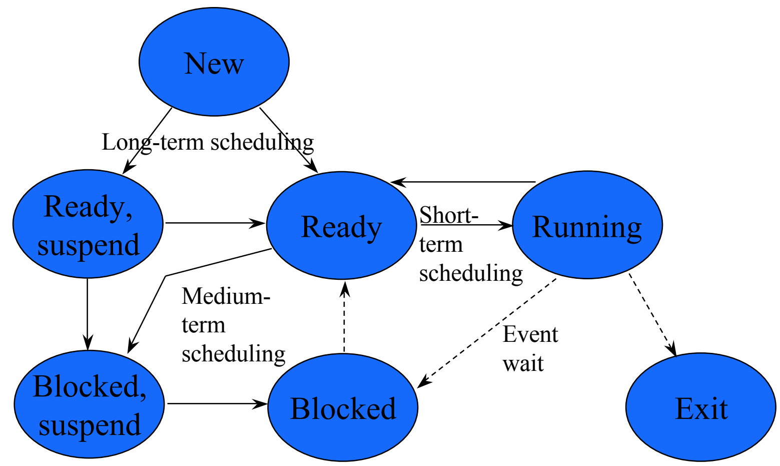
Medium vs. Short Term

- Medium-term scheduling
  - *Swaps* processes between main memory and disk
    - based on how many processes the OS wants available
    - must consider memory management if no virtual memory (VM), so look at memory requirements of swapped out processes
- Short-term scheduling (dispatcher)
  - Executes most frequently, to decide which process to execute next
  - Invoked whenever event occurs that interrupts current process or provides an opportunity to preempt current one in favor of another
  - Events: clock interrupt, I/O interrupt, OS call, signal

Long-term scheduling

- Determine which programs admitted to system for processing
- Once admitted, program becomes a process
  - Queued for short- or medium-term scheduler
- Scheduling batch jobs
  - Can system take a new process?
    - more processes implies less time for each existing one
    - add job(s) when a process terminates, or if percentage of processor idle time is greater than some threshold
  - Which job to schedule?
    - first-come, first-serve (FCFS), or to manage overall system performance (e.g. based on priority, expected execution time, I/O requirements, etc.)

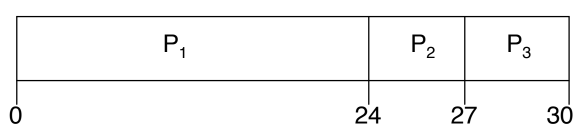
Process State Transitions



First-Come, First-Served (FCFS)

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:



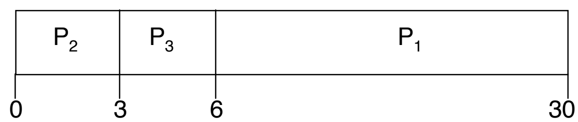
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling

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

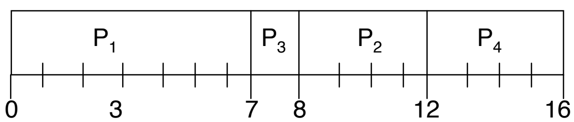
Shortest-Job-First (SJF)

- 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 – process 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. Dubbed Shortest-Remaining-Time-First (SRTF). Should yield better turnaround times.
- SJF is optimal – gives minimum average waiting time for a given set of processes.

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)

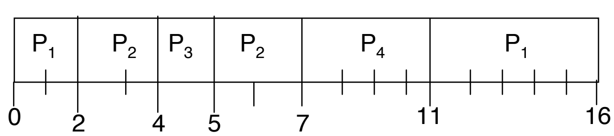


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Preemptive SJF (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

- SJF (preemptive)



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

Length of Next CPU Burst

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

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

t_n	actual length of n^{th} CPU burst
τ_{n+1}	predicted value of $n+1^{st}$ CPU burst
α	history parameter $0 <= \alpha <= 1$

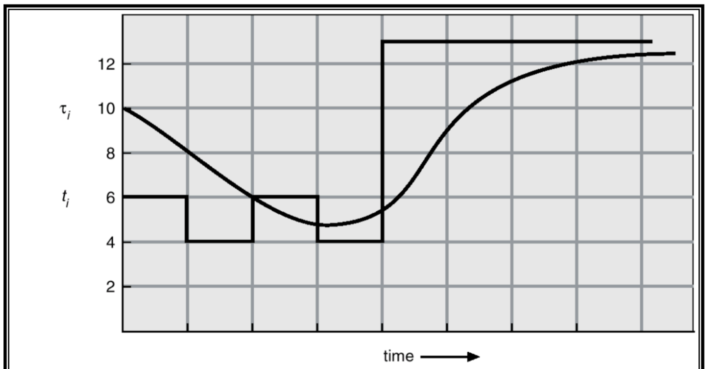
Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} = {} & \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots \\ & + (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots \\ & + (1 - \alpha)^{n=1}\, \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

Predicting the Next CPU Burst Length ($\alpha = {}^1/_2$, $\tau_0 = 10$)



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

Round Robin (RR)

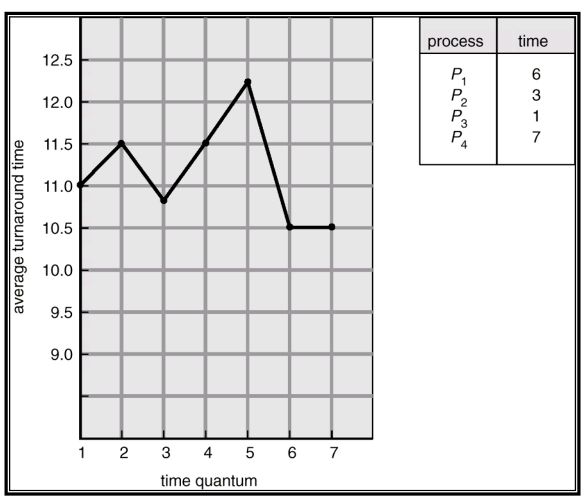
- Each process gets a small unit of CPU time (time quantum).
  - Once this time elapses, the process is preempted and placed on the back of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then no process waits more than *(n-1)q* time units.

11

Choosing the Quantum

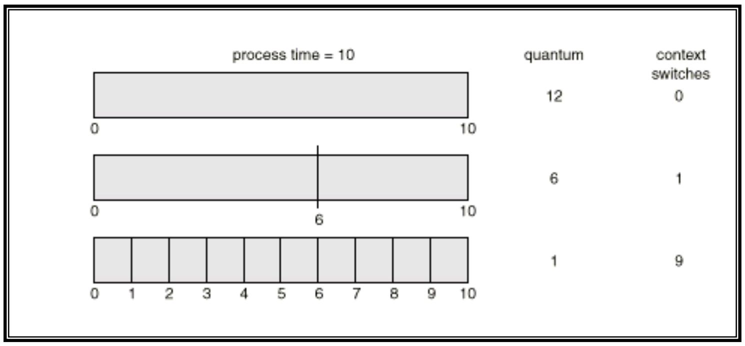
- How to choose *q*?
  - Very large: degenerates to FCFS
  - Very small: dispatch time dominates
  - Guideline: for better turnaround time, quantum should be slightly greater than time of "typical job" CPU burst.

Turnaround Time Varies With The Time Quantum



12

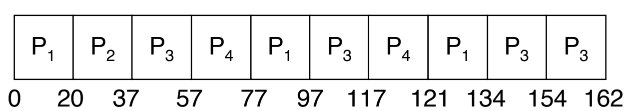
Time Quantum and Context Switch Time



Example RR with q = 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*.

13

Priority Scheduling

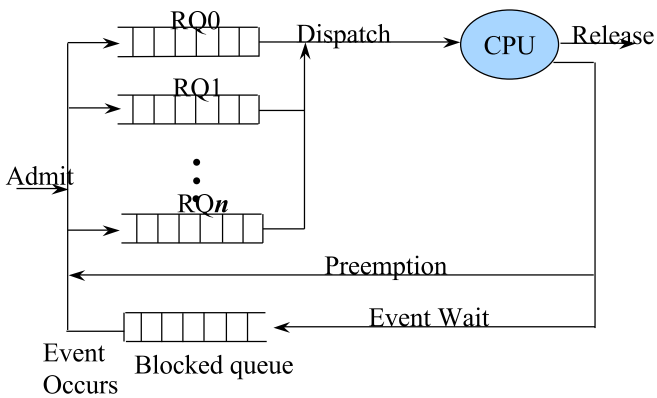
- A priority number (integer) is associated with each process
- OS schedules the process with the highest priority (smallest integer ≡ highest priority).
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
  - Problem ≡ Starvation – low priority processes may never execute.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

Multilevel Priority Queue

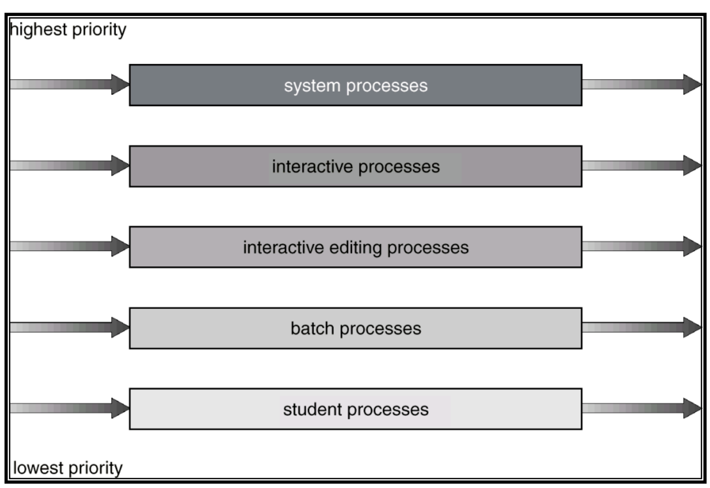
- Ready queue is divided into *n* queues, each with its own scheduling algorithm, e.g.
  - foreground (interactive) - RR
  - background (batch) - FCFS
- Scheduling done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.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

Multilevel Priority Scheduling

- Many ready queues, ordered by priority



Example



15

Multilevel Scheduling Design

PROBLEM: turnaround time for longer processes

- Want to avoid undue increase or starvation when new short jobs regularly enter system

- Solution 1: vary preemption times according to queue
  - processes in lower priority queues have longer time slices
- Solution 2: promote a process to higher priority queue
  - after it spends a certain amount of time waiting for service in its current queue, it moves up
- Solution 3: allocate fixed share of CPU time to jobs
  - if a process doesn't use its share, give it to other processes
  - variation on this idea: lottery scheduling
    - assign a process "tickets" (# of tickets is share)
    - pick random number and run the process with the winning ticket.

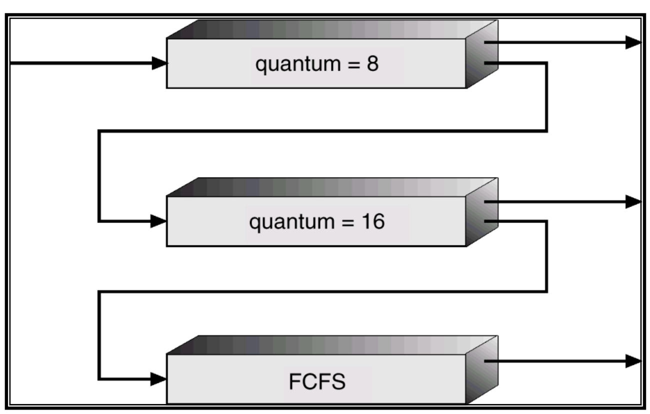
Multilevel Feedback Queue

- A process can move between the various queues, implementing *aging*.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method to determine when to upgrade a process
  - method to determine when to demote a process
  - method 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 RR. 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 does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



Multi-Processor Scheduling

- Multiple processes need to be scheduled together
  - Called gang-scheduling
  - Allowing communicating processes to interact w/o/ waiting
- Try to schedule processes back to same processor
  - Called affinity scheduling
    - Maintain a small ready queue per processor
    - Go to global queue if nothing local is ready

Algorithm Evaluation

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

UNIX System V

- Multilevel feedback, with
  - RR within each priority queue
  - 10ms preemption
  - priority based on process type and execution history, lower value is higher priority
- Priority recomputed once per second, and scheduler selects new process to run

UNIX System V

- Priority P(i) = Base + CPU(i-1)/2 + nice
  - P(i) is priority of process j at interval i
  - Base is base priority of process j
  - CPU(i) = U(i)/2 + CPU(i-1)/2
    - U(i) is CPU use of process j in interval i
    - exponentially weighted average CPU use of process j through interval i
  - nice is user-controllable adjustment factor
- Penalizes CPU-bound processes
  - Targets general-purpose time sharing (and interactive) environment

UNIX System V

- Base priority divides all processes into (non-overlapping) fixed bands of decreasing priority levels
  - swapper, block I/O device control, file manipulation, character I/O device control, user processes
- Bands optimize access to block devices (disk), allow OS to respond quickly to system calls

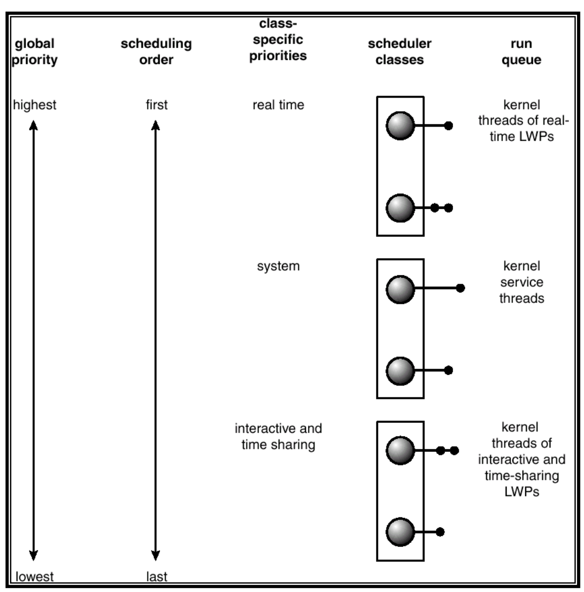
GeekOS (<= project 2)

- Uses priority-based, RR scheduling
  - Each kthread has a priority
    - Level 0 is the "idle" process which "runs" when there is no real work to be done
    - Level 1 is for normal user processes
    - Level 10 is the highest priority
  - Chooses highest-priority thread that is in the ready queue (**`s_runQueue`**).

GeekOS (Project 3)

- Multi-level feedback scheduling
  - Multiple queues, each denoting a higher priority scheduling class (still have priorities within each class)
- Queue placement policy:
  - Thread is demoted to next lower class if it consumes all its quantum.
  - Thread is promoted to the next higher class if it blocks.
  - Idle thread treated specially.

Solaris 2 Scheduling



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

22