CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 14

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Chapter 6: CPU Scheduling

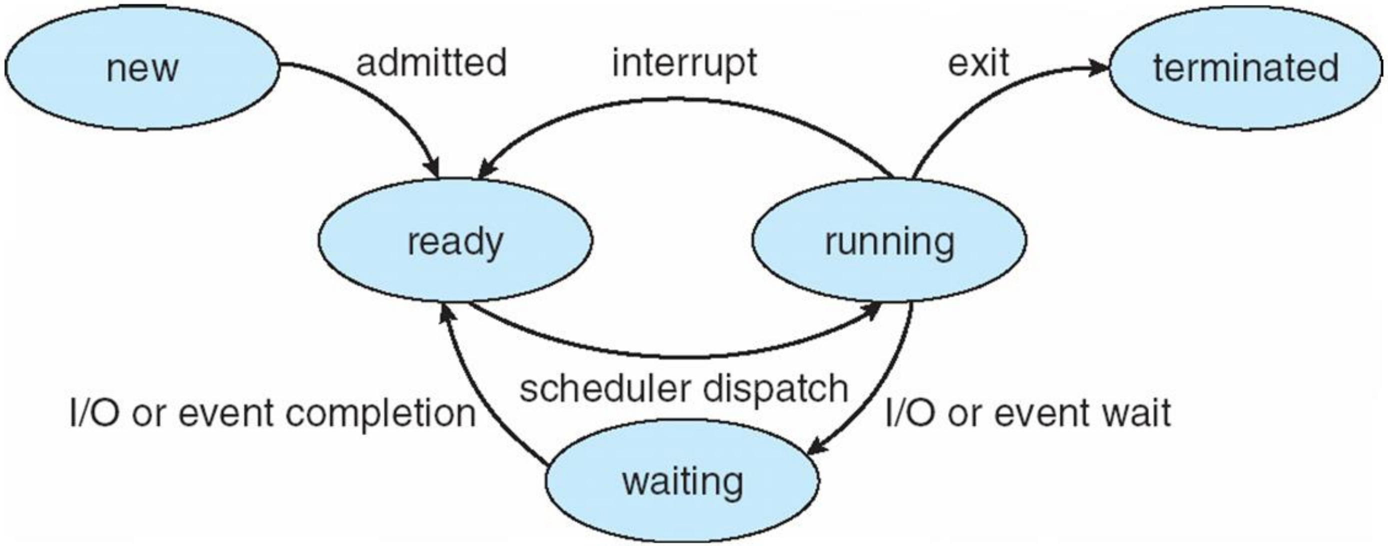
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation

You should have received and invitation for 370 Piazza. Use Piazza for now posts. Canvas discussion still accessible.

FAQ

- Time Quantum review
- Turnaround time using time quantum review
- Context switching when there is only a single process? No. But it is unlikely situation in most modern computers.
- We assume we know the true process burst time. What if guess is wrong? SJF, pSJF
- Idea: table for comparing turnaround time etc.
- Idea: compare function call, forking a child, starting a thread

Diagram of Process State



Ready to Running: scheduled by scheduler
Running to Ready: scheduler picks another process, back in ready queue
Running to Waiting (Blocked) : process blocks for input/output
Waiting to Ready: Input available

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible: Maximize
- **Throughput** – # of processes that complete their execution per time unit: Maximize
- **Turnaround time** –time to execute a process from submission to completion: Minimize
- **Waiting time** – amount of time a process has been waiting in the ready queue: Minimize
- **Response time** –time it takes from when a request was submitted until the **first** response is produced, not output (for time-sharing environment): Minimize

Notes

- **Nonpreemptive scheduling:** Process keeps CPU until it relinquishes it when **It terminates or switches to the waiting state**
- **Preemptive** scheduling: preempted with time quantum expires (or other reasons)
- Assumptions for simple examples:
  - Single processor
  - We know the execution time of a process
  - Scheduling a single burst

First- Come, First-Served (FCFS) Scheduling

Who was Gantt?

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 *but almost the same time*.
  The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Throughput: 3/30 = 0.1 per unit

Example of SJF

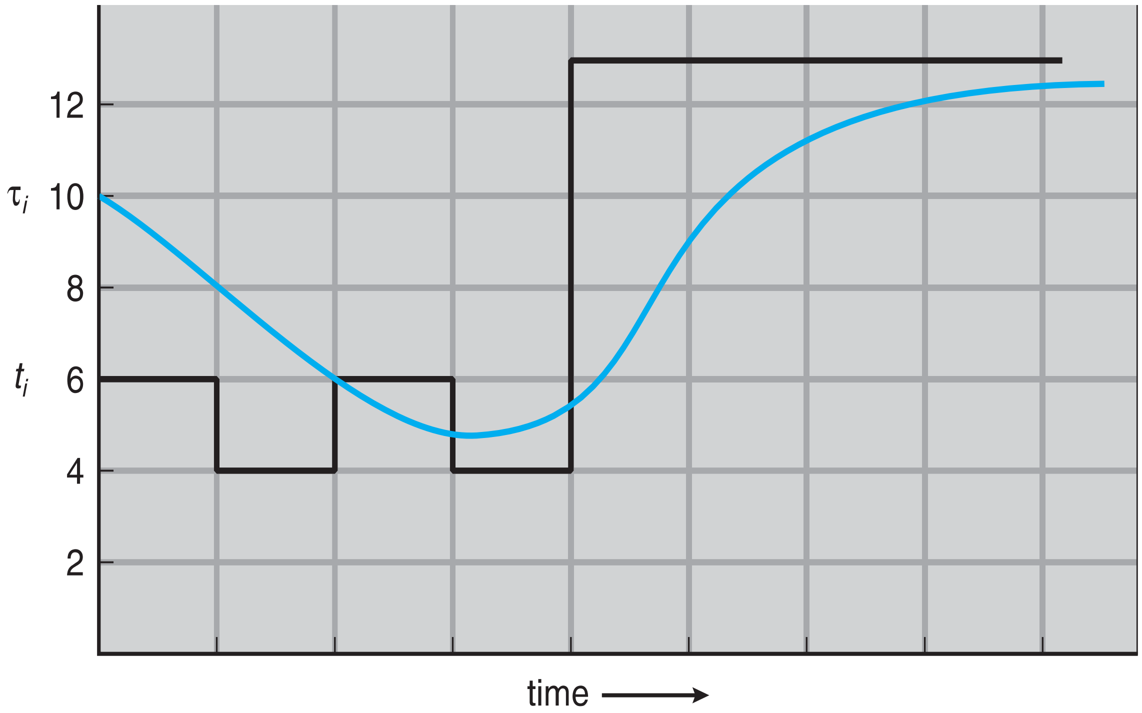
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- All arrive at time 0.
- SJF scheduling chart

P_4	P_1	P_3	P_2
0–3	3–9	9–16	16–24

- Average waiting time for P_1,P_2,P_3,P_4 = (3 + 16 + 9 + 0) / 4 = 7

Prediction of the Length of the Next CPU Burst



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

Blue points: guess
Black points: actual
$\alpha = 0.5$

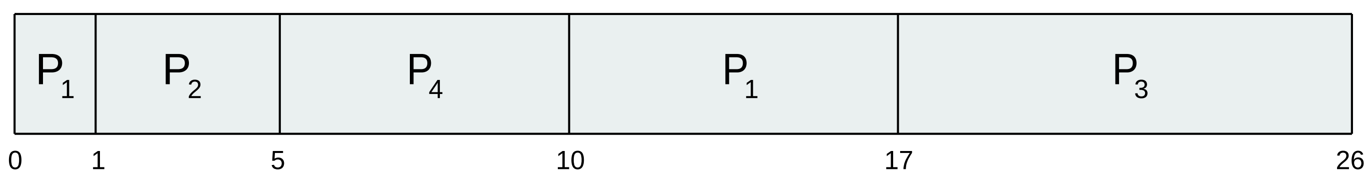
Ex:
0.5x6 +(1-0.5)x10 = 8

Shortest-remaining-time-first (**preemptive** SJF)

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time for P1,P2,P3,P4

= [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1 (highest)
P_3	2	4
P_4	1	5
P_5	5	2

- Arrived at time 0 in order P1,P2, P3, P4,P5
- Priority scheduling Gantt Chart

- Average waiting time for P1, .. P5: (6+0+16+18+1)/5 = 8.2 msec

Colorado State University

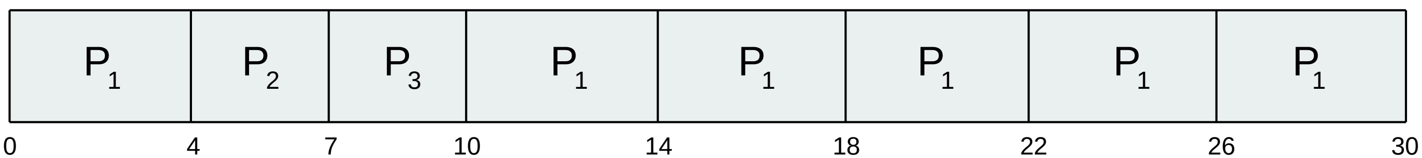
Round Robin (RR) with time quantum

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 1-10 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high (overhead *typically in 0.5% range*)

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Arrive a time 0 in order P1, P2, P3: The Gantt chart is:

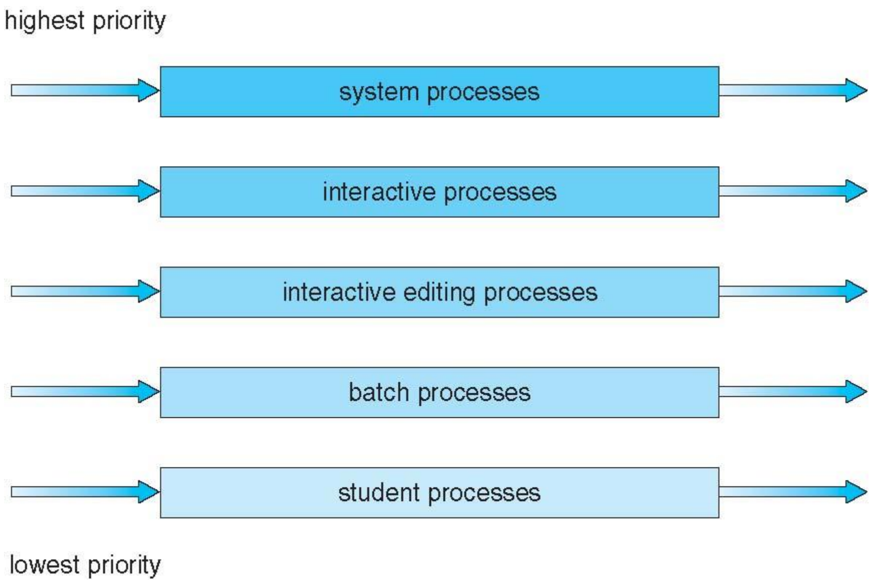


- Waiting times: 10-4 =6, 4, 7, average 17/3 = 5.66 units
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Multilevel Queue

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

Multilevel Queue Scheduling



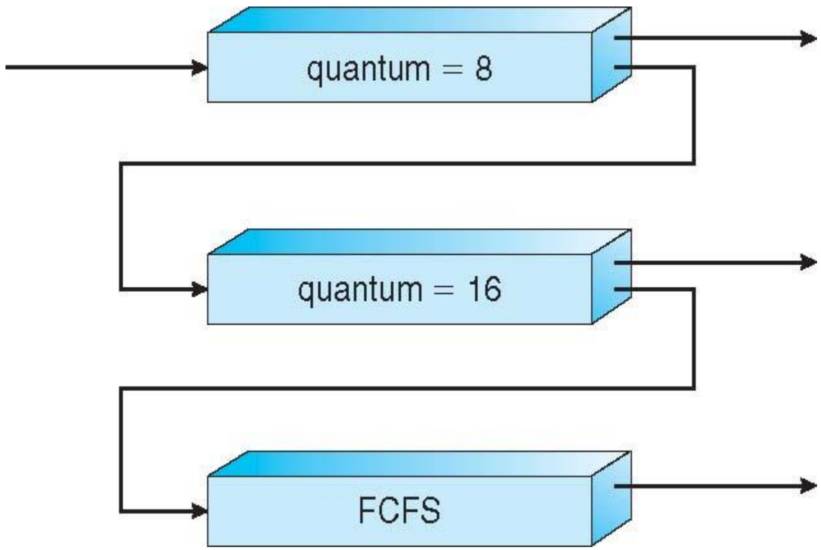
Single CPU.
Lower priority queue started
When higher priority queue done.

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

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS (no time quantum limit)
- 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



CPU-bound: priority falls, quantum raised,
I/O-bound: priority rises, quantum lowered

Thread Scheduling

- Thread scheduling is similar
- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes

Scheduling competition

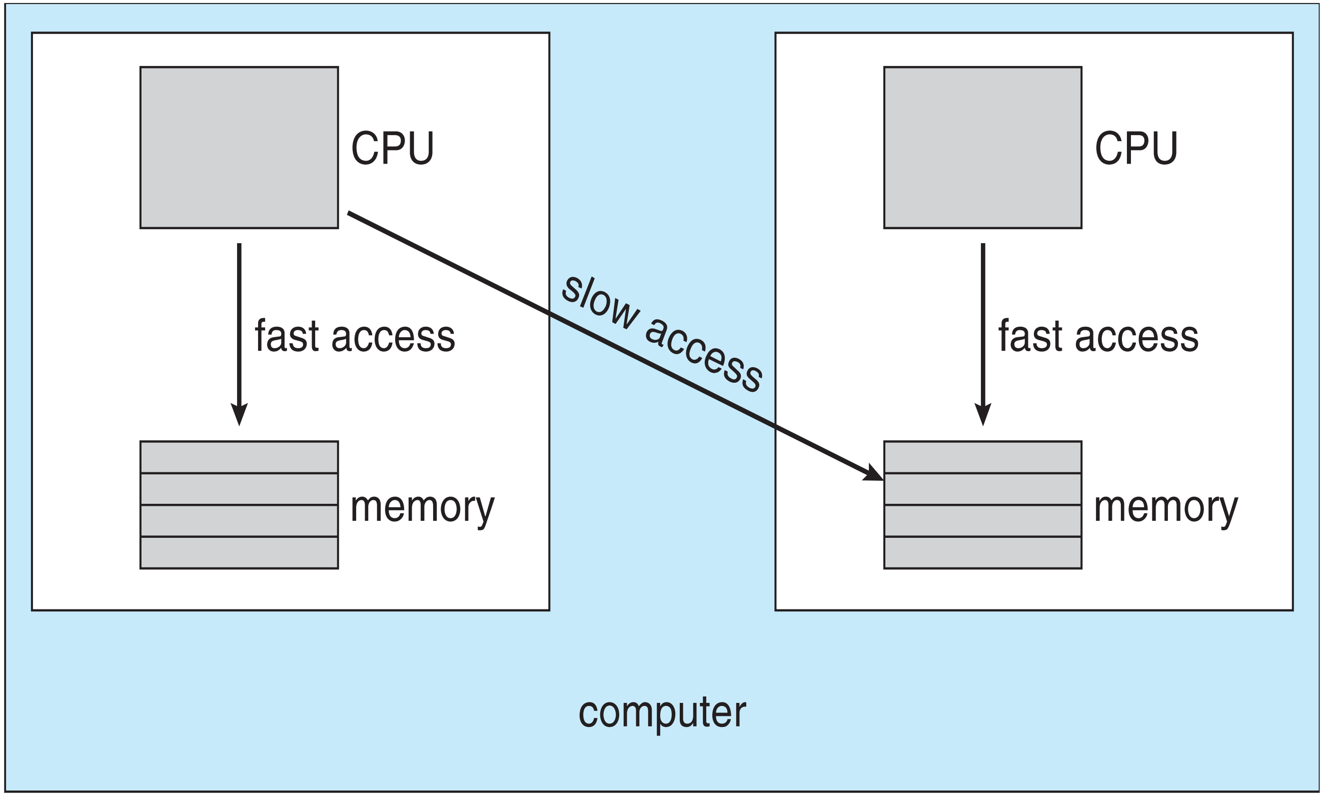
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

LWP layer between kernel threads and user threads in some older Oss. Not for one-to-one.

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- **Assume Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – individual processors can be dedicated to specific tasks at design time
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling,
  - all processes in common ready queue, **or**
  - each has its own private queue of ready processes
    - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running because of info in cache
  - **soft affinity: try but no guarantee**
  - **hard affinity can specify processor sets**

NUMA and CPU Scheduling



Note that memory-placement algorithms can also consider affinity
Non-uniform memory access (NUMA), in which a CPU has faster access to some parts of main memory.

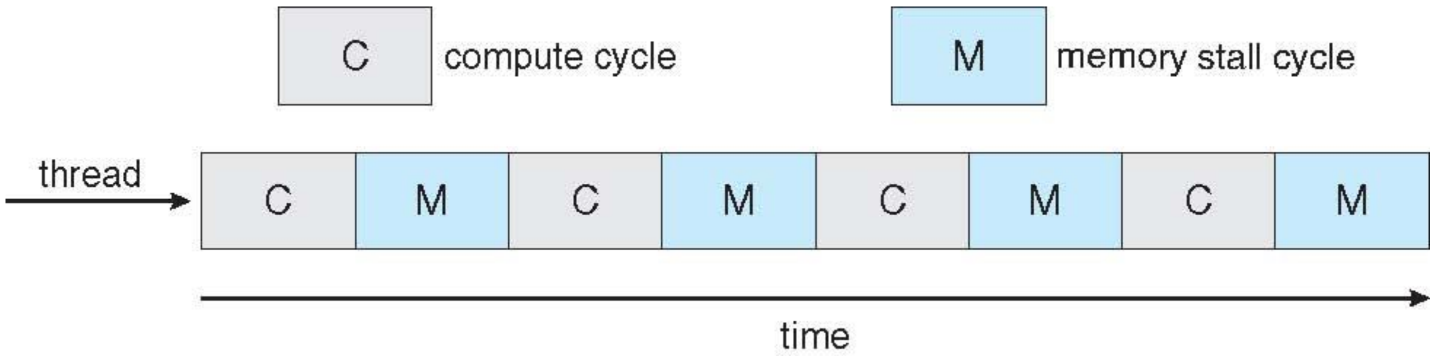
- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
  - **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
  - **Pull migration** – idle processors pulls waiting task from busy processor
  - Combination of push/pull may be used.

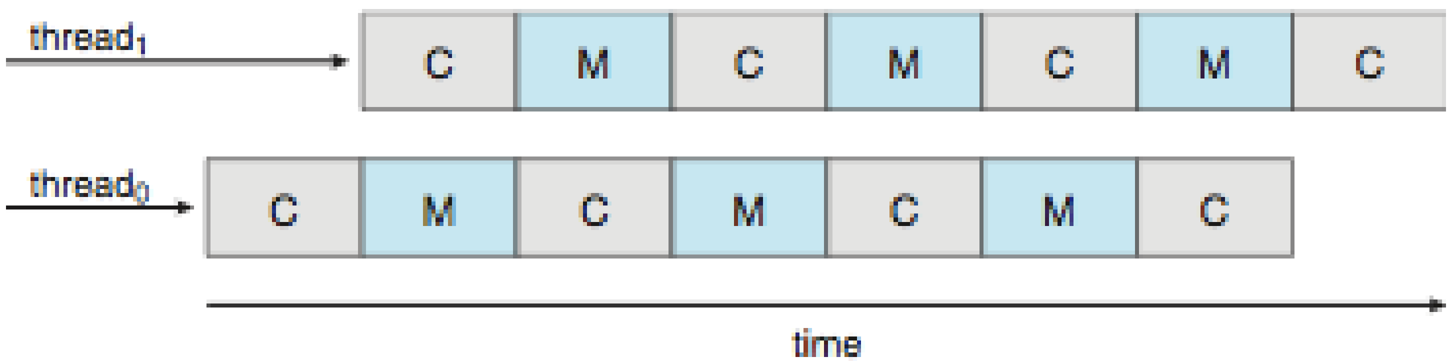
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core now common
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens
  - See next

Multithreaded Multicore System

Memory stalls due to cache miss





This is temporal multithreading. Simultaneous multithreading allows threads to computer in parallel

Real-Time CPU Scheduling

- Can present obvious challenges
  - **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
  - **Hard real-time systems** – task must be serviced by its deadline
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
  - **periodic** ones require CPU at constant intervals

Virtualization and Scheduling

- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
  - Not knowing it doesn't own the CPUs
  - Can effect time-of-day clocks in guests
- VMM has its own scheduler
- Various approaches have been used
  - Workload aware, Guest OS cooperation, etc.

Operating System Examples

- Solaris scheduling: 6 classes, Inverse relationship between priorities and time quantum
- Windows XP scheduling: 32 priority levels (real-time, not real-time levels)
- Linux scheduling: newer - Completely fair scheduler (CFS):
  - 140 priority levels
  - variable timeslice, number and priority of the tasks in the queue

- Approaches evolve.

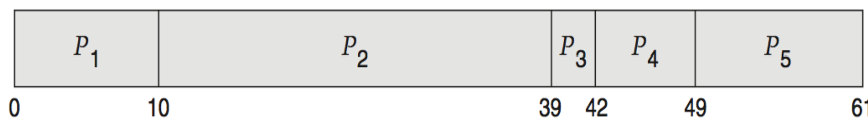
Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

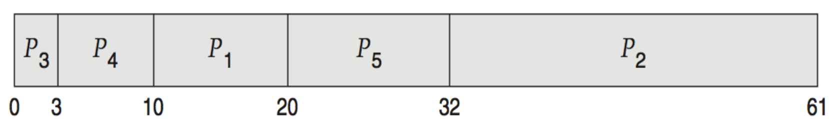
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Deterministic Evaluation

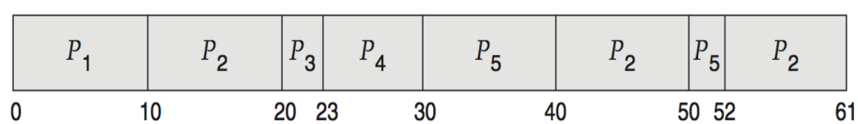
- For each algorithm, calculate minimum average waiting time
- Simple, *but requires exact numbers for input, applies only to those inputs*
  - FCS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc
- Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc

Little's Formula

- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

  $n = \lambda \times W$

  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Simulations

- Queueing models limited
- **Simulations** more flexible
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - "Trace tapes" record sequences of real events in real systems

Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system