Chapter 5: CPU Scheduling

CPU Scheduling

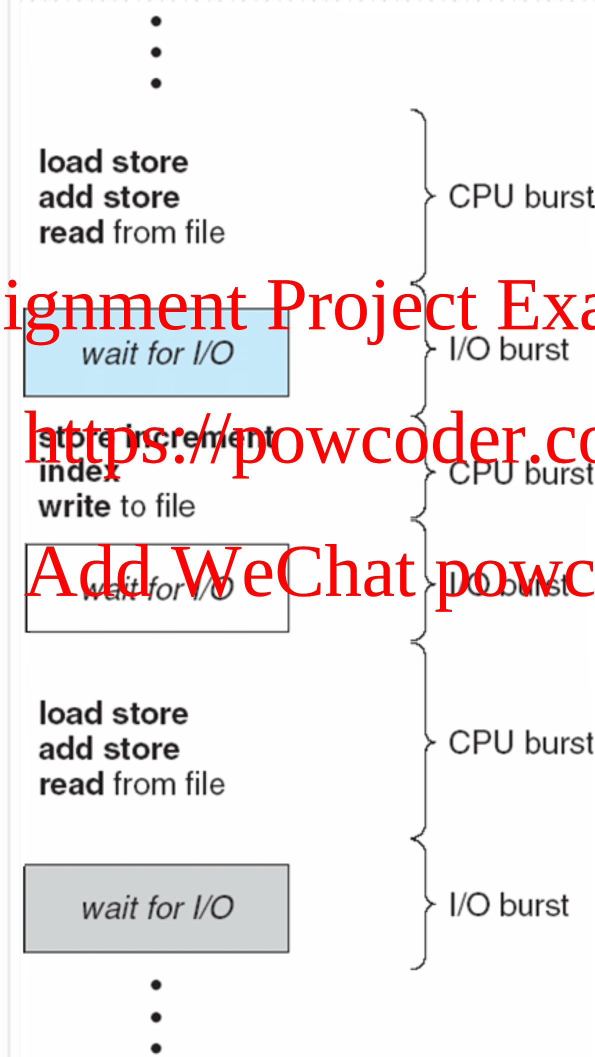
CPU Scheduling:

- The CPU Scheduler selects one process among all processes that are in the *ready state*, and allocates the CPU to it.
- Often CPU Schedulers use a ready queue, where the records in the ready queue are PCBs of the processes that are in the ready state. The ready queue is not necessarily a FIFO queue.

CPU-I/O Burst Cycle

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.

Alternating Sequence of CPU And I/O Bursts



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 (e.g. I/O request).
  2. Switches from running to ready state (e.g. interrupt).
  3. Switches from waiting to ready (e.g. completion of I/O).
  4. Terminates.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive*.

Preemptive Scheduling

- The CPU Scheduler may take away the CPU from a process during the course of that process' execution, and allocate the CPU to another process
- A process may be preempted:

  - When the CPU scheduler performs preemptive scheduling

  - When interrupts happen (which can happen at any time).
- When accessing shared data, may need to prohibit preemptions, in order to prevent simultaneous access to the shared data.

Nonpreemptive Scheduling

Once the CPU has been allocated to a process, that process keeps the CPU until it voluntarily releases the CPU.

- Easier to implement, does not require the special hardware (e.g. timer) needed for preemptive scheduling.
- When a single CPU is used, prevents simultaneous access to shared data.
- Provides less flexibility in scheduling processes to meet timing constraints.

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 – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – the interval from the time of submission of a process to the time of completion (including I/O, thus depends on speed of I/O).
- Waiting time – sum of the amounts of time that a process has spent 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** the time it takes to output that response) - for interactive systems.

Scheduling Algorithm Optimization Criteria

- Most frequently used measure for comparing CPU scheduling algorithms is the **average waiting time of all the processes.**

- There are circumstances when it is desirable to optimize the minimum or maximum values, rather than the average. E.g. Minimize the maximum response time, to guarantee that all users get good service.

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

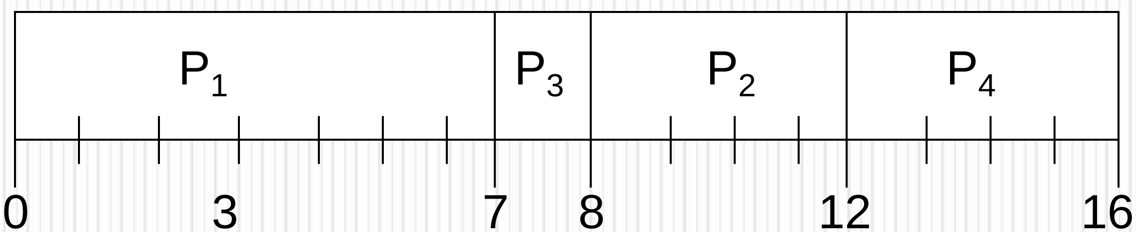
Shortest-Job-First (SJF) Scheduling

- 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 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 Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.
  - The difficulty is knowing the length of the next CPU request

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

- Non-Preemptive also called Shortest-Remaining-Time-First Scheduling

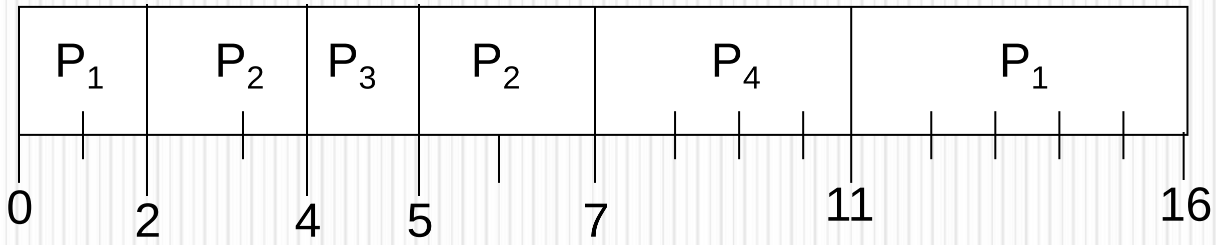


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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

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

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

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

Examples of 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.

- $\alpha = 1/2$

  – Recent history and past history equally weighted

CPU burst (t_i)		6	4	6	4	13
"guess" (τ_n)	10	8	6	6	5	9

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), 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 $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

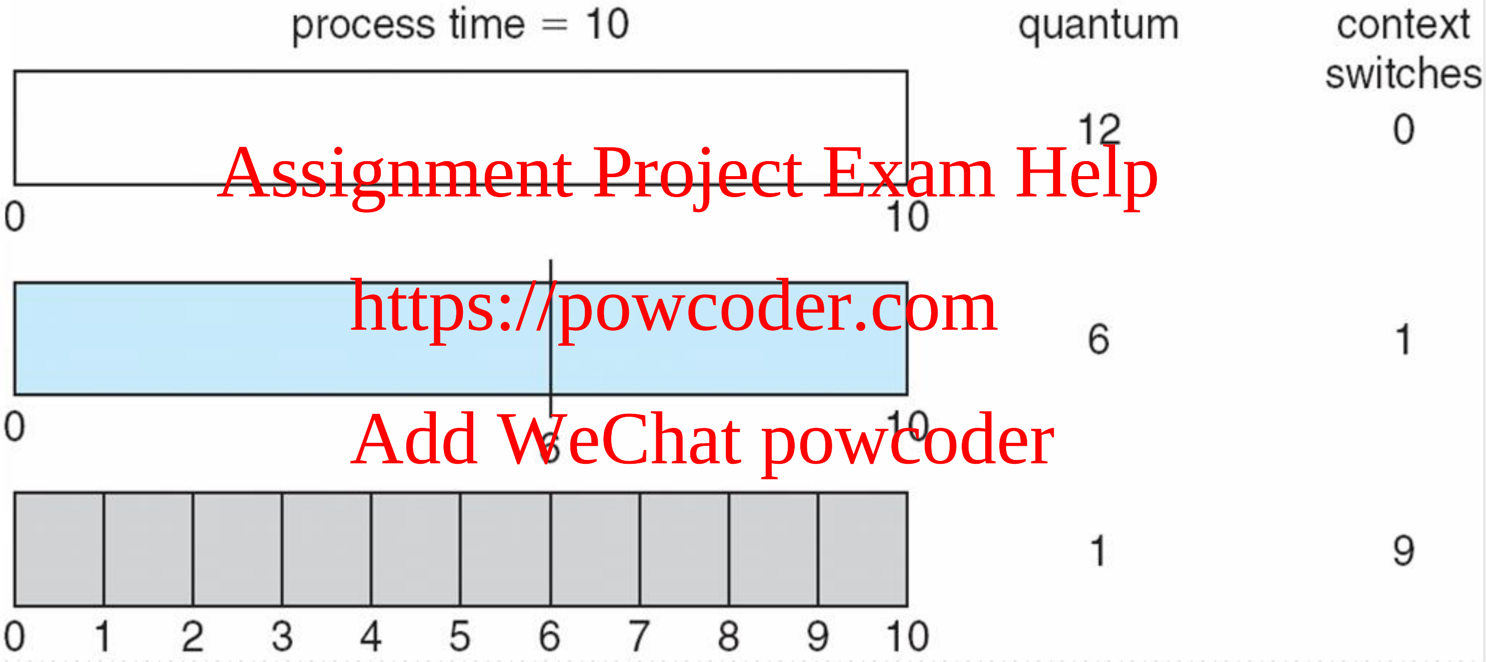
P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

Average waiting time = ((10-4) + 4 + 7)/3 = 17/3 = 5.66

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

Time Quantum and Context Switch Time

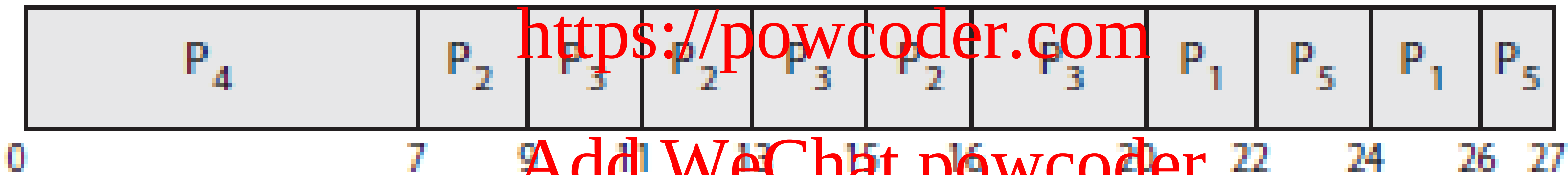


Priority Scheduling

- 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

Priority Scheduling Example

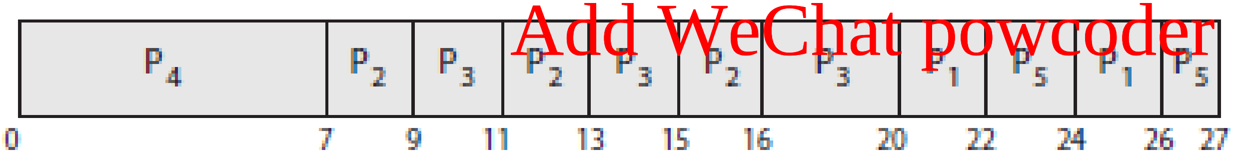
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3



Combining Priority Scheduling With Round-Robin Scheduling Example

Assume that the system executes the highest-priority process and runs processes with the same priority using round-robin scheduling with a time-quantum of 2.

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3



Process P4 has the highest priority, so it will run to completion. Processes P2 and P3 have the next-highest priority, and theywill execute in a round-robin fashion. Notice that when process P2 finishes at time 16, process P3 is the highest-priority process, so it will run until it completes execution. Now, only processes P1 and P5 remain, and as they have equal priority, they will execute in round-robin order until they complete.

Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm
  - 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.
  - 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

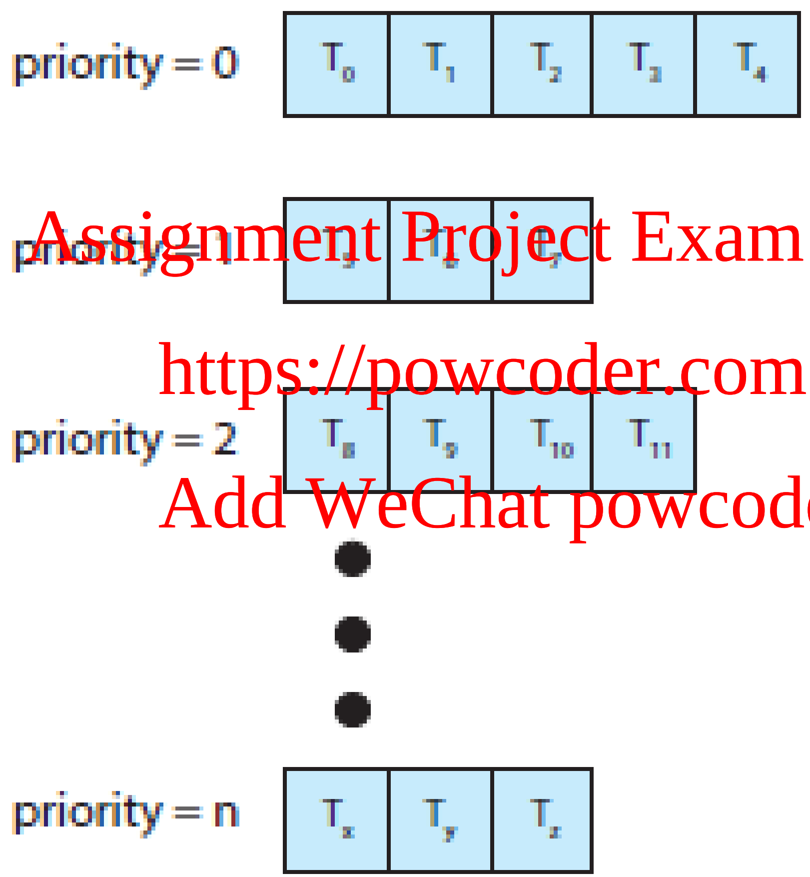
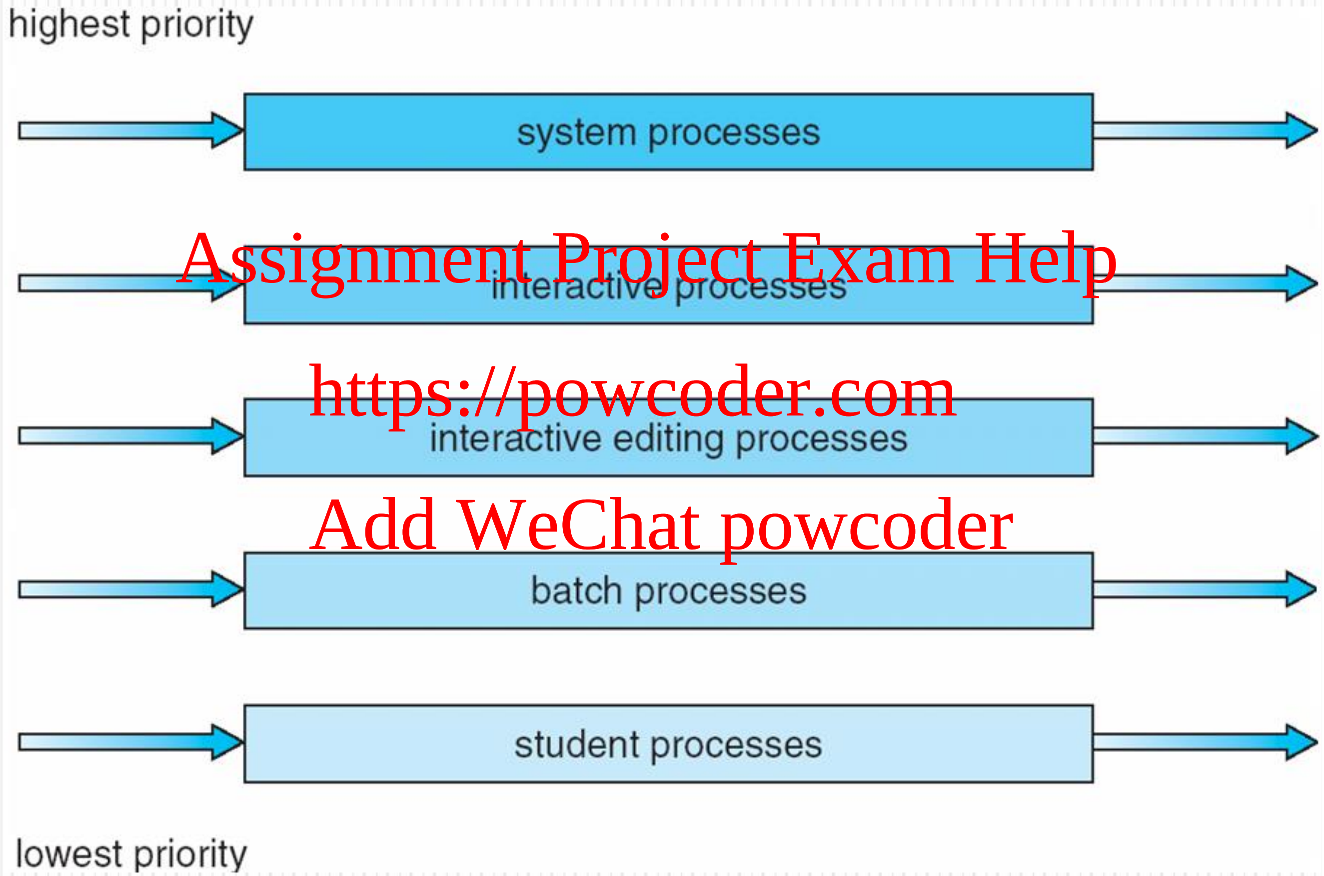


Figure 5.7 Separate queues for each priority.

Multilevel Queue Scheduling



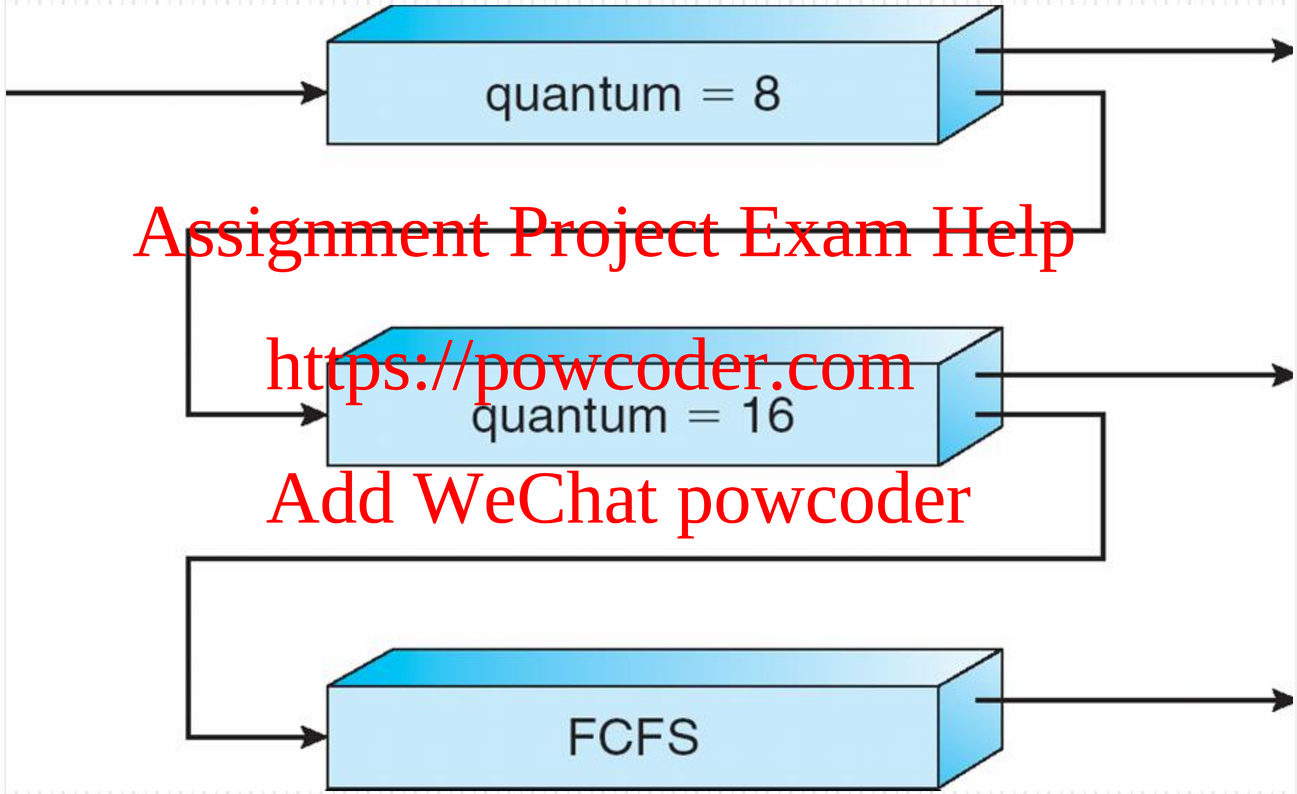
Multilevel Feedback Queue

- 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

- 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



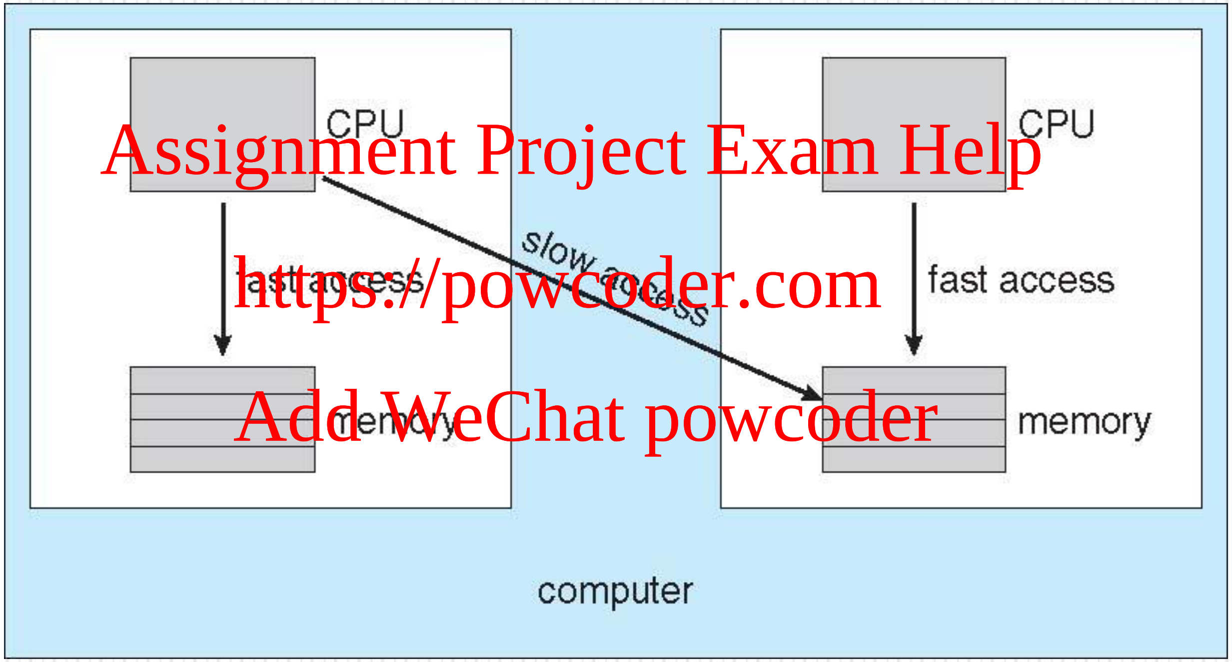
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing

- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes

- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity**
  - **hard affinity**

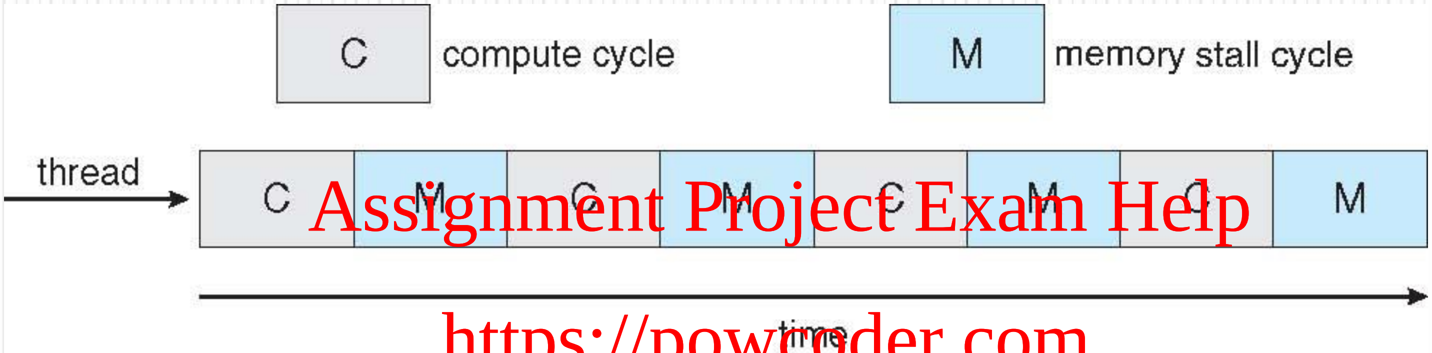
NUMA and CPU Scheduling



Multicore Processors

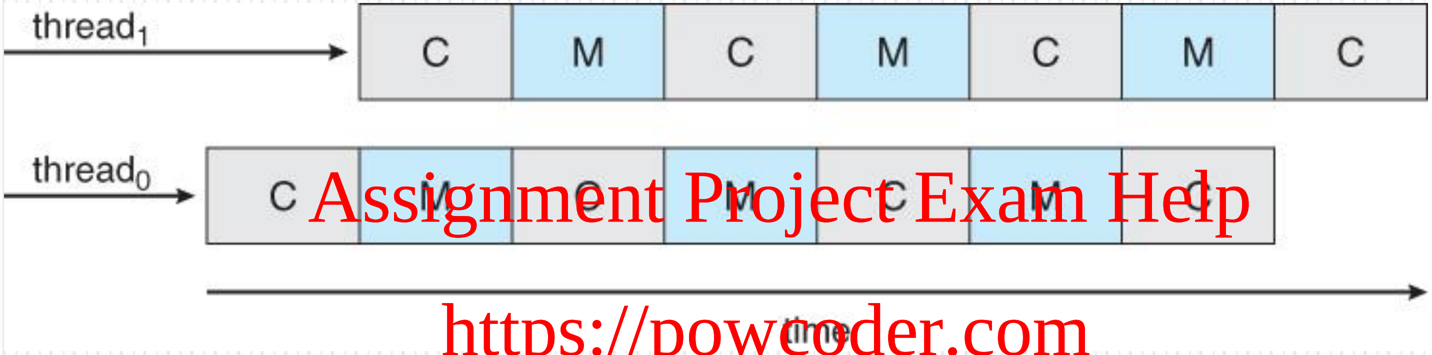
- Recent trend to place multiple processor cores on same physical chip
- Faster and consume less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Memory stall



Memory stall: when a thread accesses memory, it may need to wait for the data to become available from memory.

Multithreaded multicore system



In a multithreaded multicore system, two or more threads can be assigned to each core. If one thread stalls while waiting for data to become available from memory, the core can be switched to another thread.

Dual-threaded multicore system

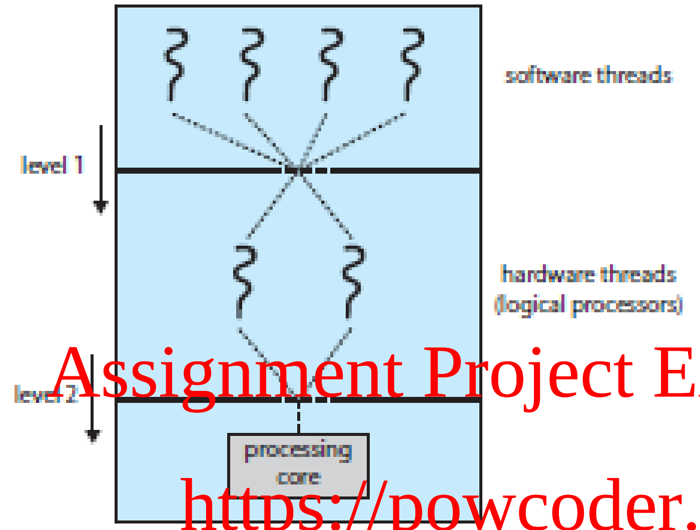


Figure 5.15 Two levels of scheduling.

On one level are the scheduling decisions that must be made by the operating system as it chooses which software thread to run on each hardware thread. For this level of scheduling, the operating system may choose any scheduling algorithm described earlier.

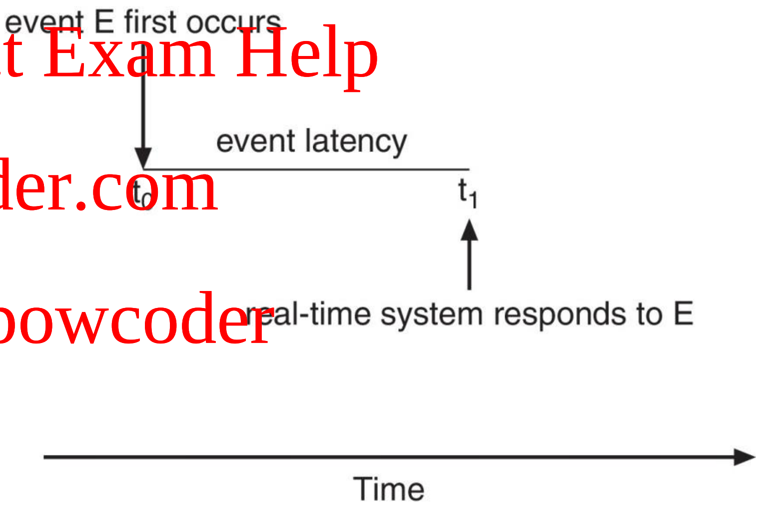
A second level of scheduling specifies how each core decides which hardware thread to run. For example, use a simple round-robin algorithm to schedule a hardware thread to the processing core.

Real-Time CPU Scheduling

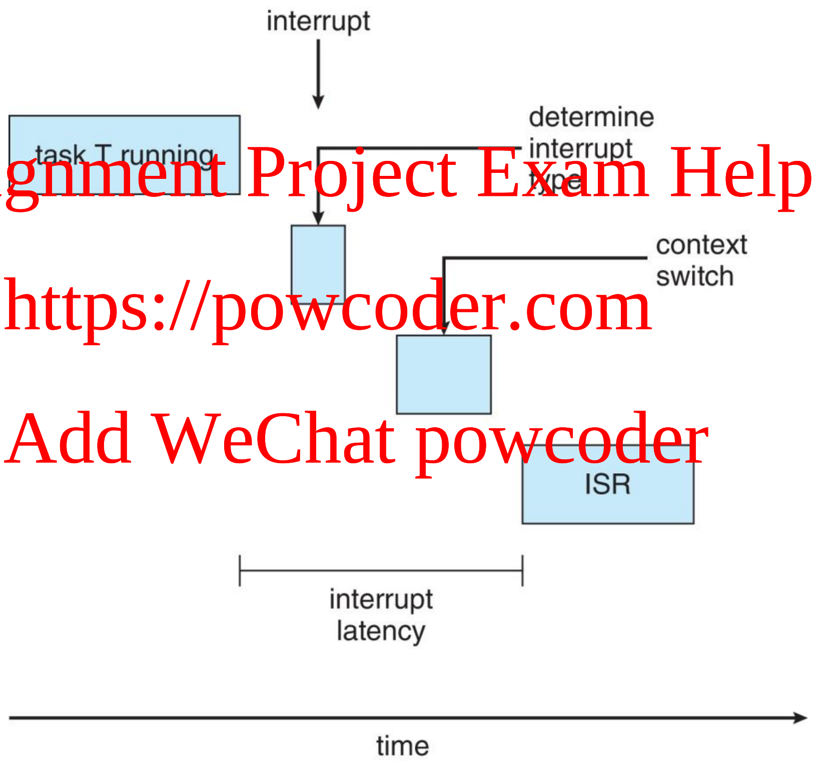
- Can present obvious challenges
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline

Real-Time CPU Scheduling

- Event latency – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  1. **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  2. **Dispatch latency** – time for schedule to take current process off CPU and switch to another

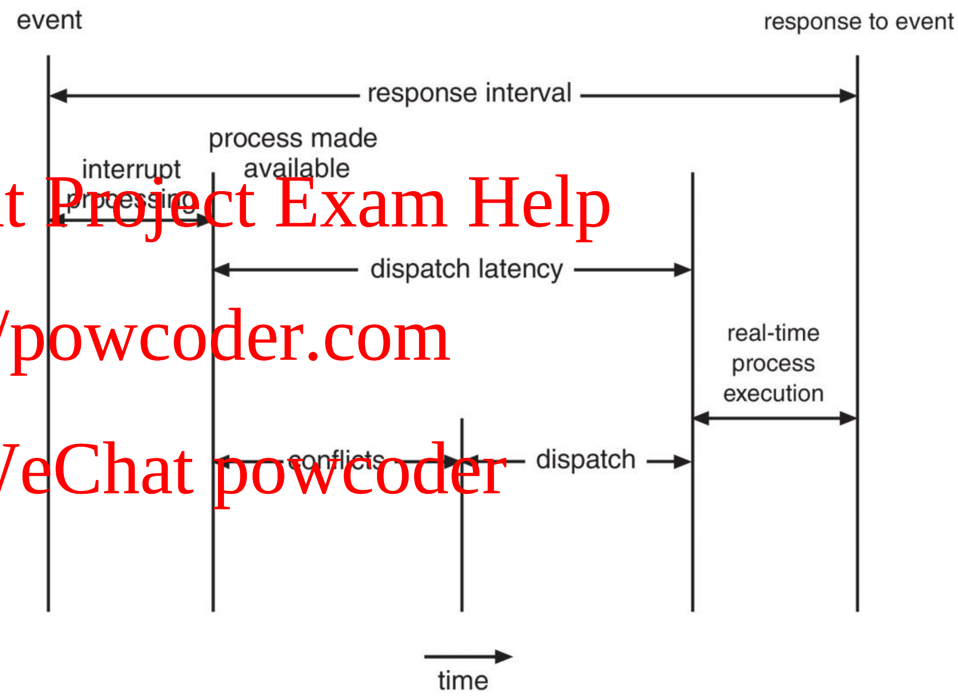


Interrupt Latency



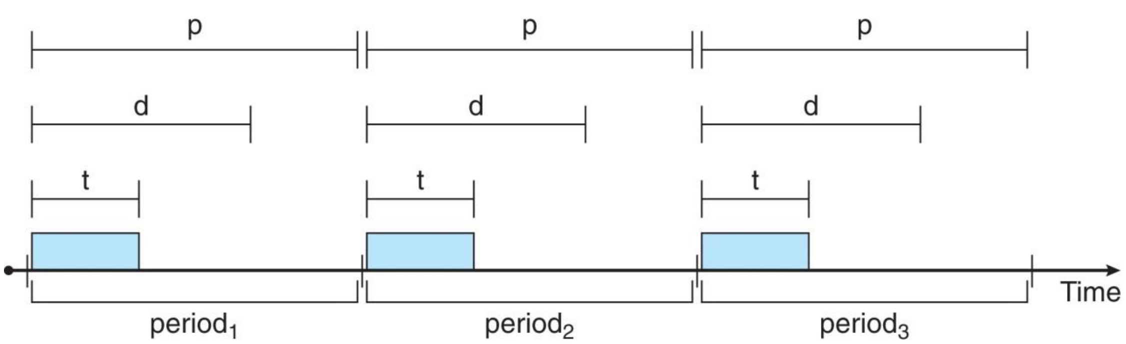
Dispatch Latency

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



Priority-based Scheduling

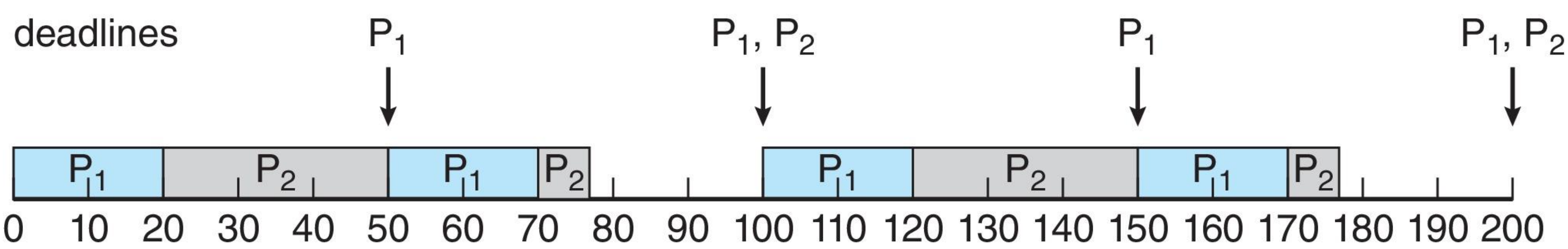
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time t, deadline d, period p
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$



Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

Process	Processing Time t	Deadline d	Period p
P1	20	50	50
P2	35	50	50

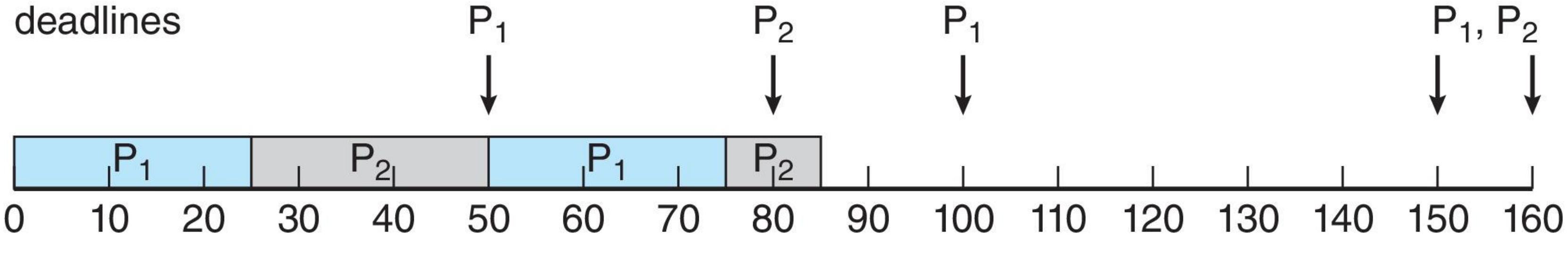


Missed Deadlines with Rate Monotonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

Process	Processing Time t	Deadline d	Period p
P1	25	50	50
P2	35	80	80

Process P2 misses finishing its deadline at time 80



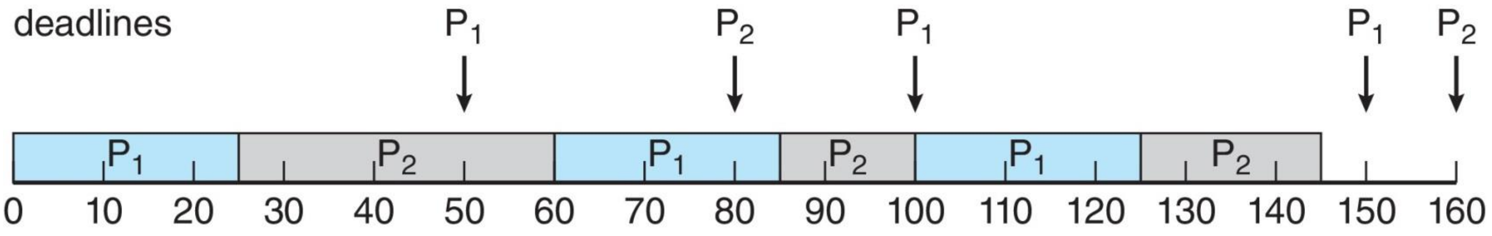
Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:

  the earlier the deadline, the higher the priority;

  the later the deadline, the lower the priority

Process	Processing_Time t	Deadline d	Period p
P1	25	50	50
P2	35	80	80



Algorithm Evaluation

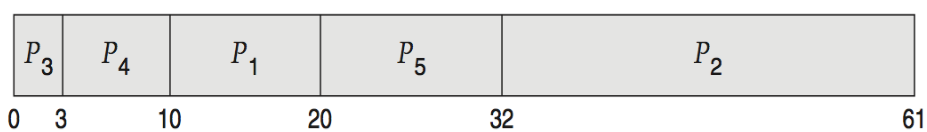
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving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

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

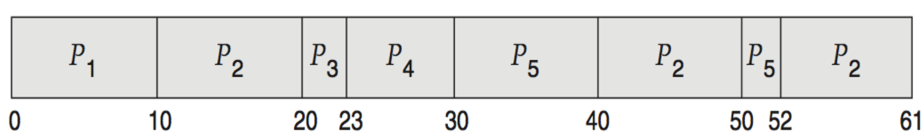
Deterministic Evaluation

- For each algorithm, calculate minimum average waiting time
- Simple and fast, 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:

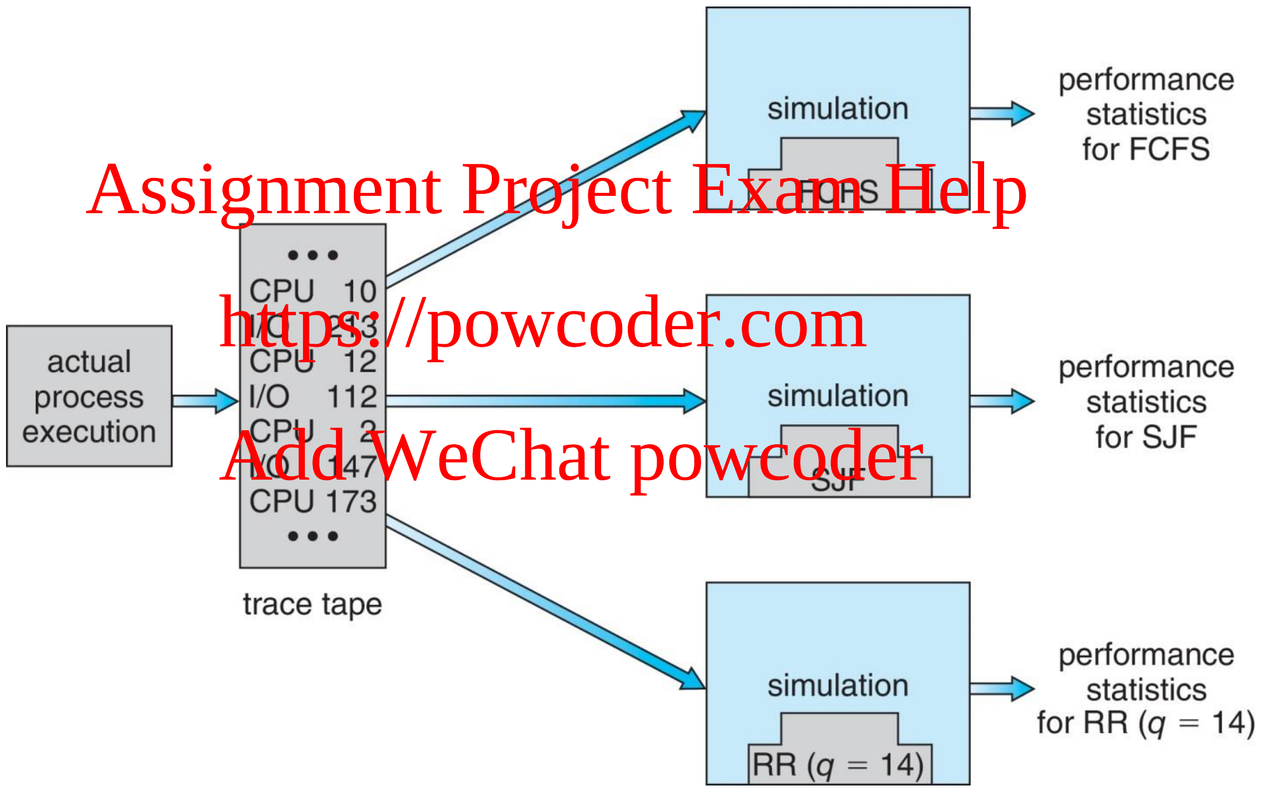
  $$n = \lambda \times W$$

  - 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 accurate
  - 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

Evaluation of CPU Schedulers by Simulation



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
- Or APIs to modify priorities
- But again environments vary

End of Chapter 5