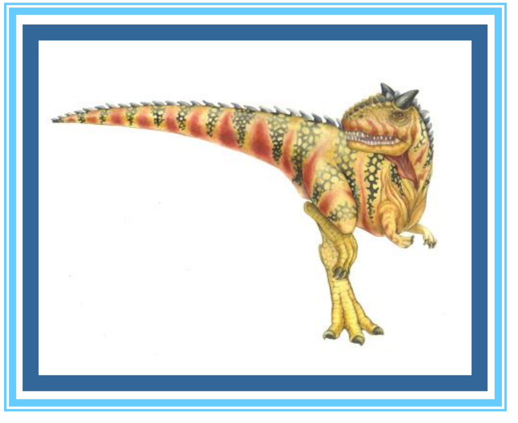
Chapter 5: CPU Scheduling



Outline

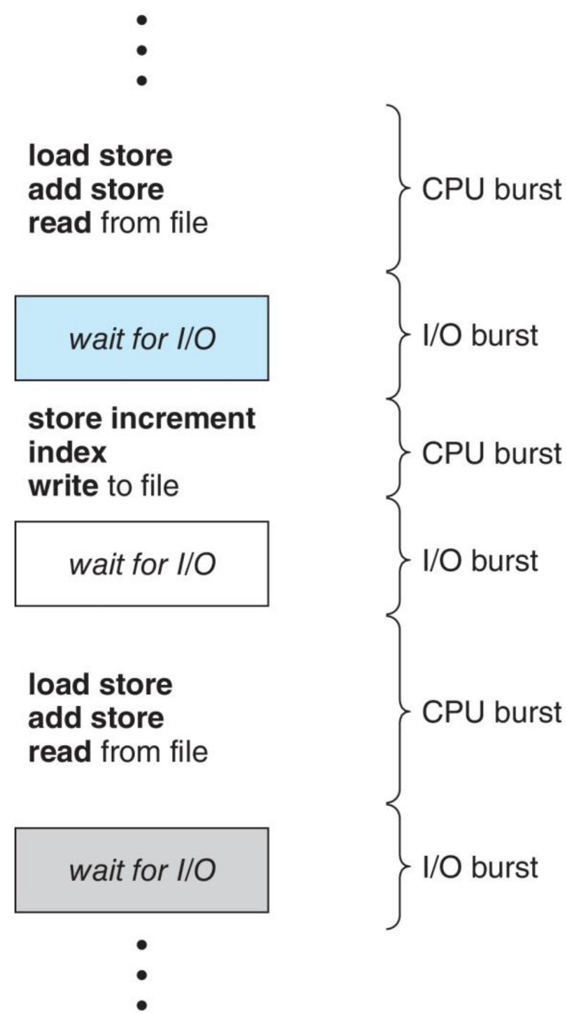
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Algorithm Evaluation

Objectives

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms

Basic Concepts

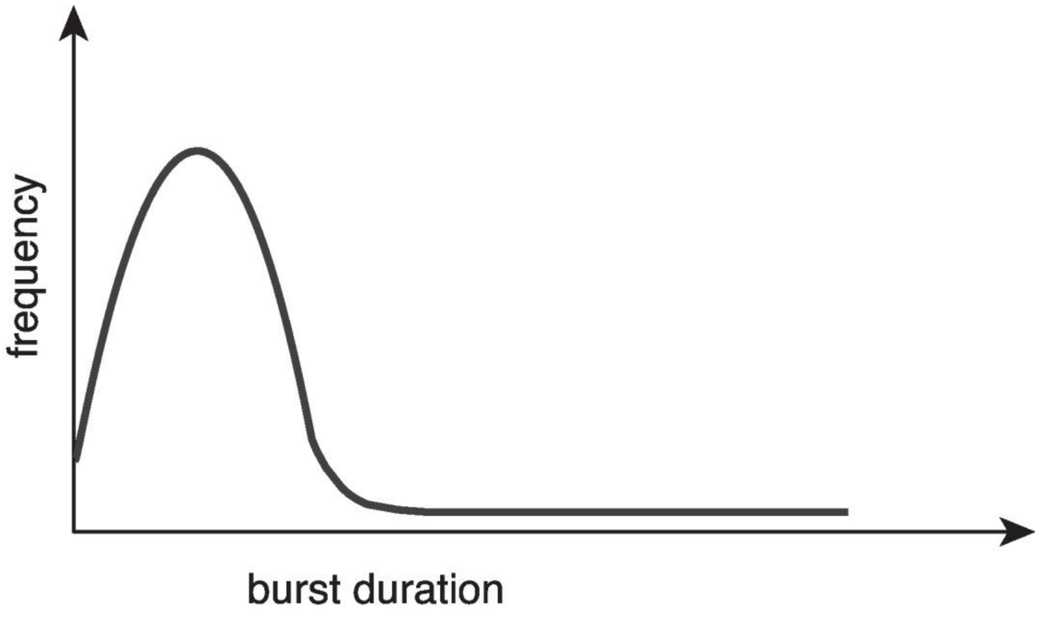
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



Histogram of CPU-burst Times

Large number of short bursts

Small number of longer bursts



CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core to one of them
  - Queue may be ordered in various ways
- 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
- For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
- For situations 2 and 3, however, there is a choice.

Preemptive and Nonpreemptive Scheduling

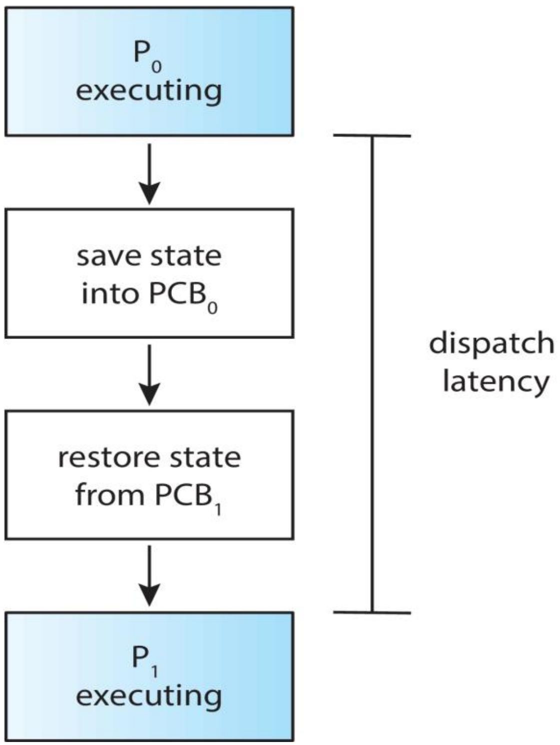
- When scheduling takes place only under circumstances 1 and 4, the scheduling scheme is **nonpreemptive**.
- Otherwise, it is **preemptive**.
- Under Nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.
- Virtually all modern operating systems including Windows, MacOS, Linux, and UNIX use preemptive scheduling algorithms.

Preemptive Scheduling and Race Conditions

- Preemptive scheduling can result in race conditions when data are shared among several processes.
- Consider the case of two processes that share data. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.
- This issue will be explored in detail in Chapter 6.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU 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** – 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.

Recall: Useful metrics

Waiting time for P

Time before *P* got scheduled

Average waiting time

Average of all processes’ wait time

Completion time

Waiting time + Run time

Average completion time

Average of all processes' completion time

Important Performance Metrics

Fairness

Equality in the performance perceived by one task

Starvation

The lack of progress for one task, due to resources being allocated to different tasks

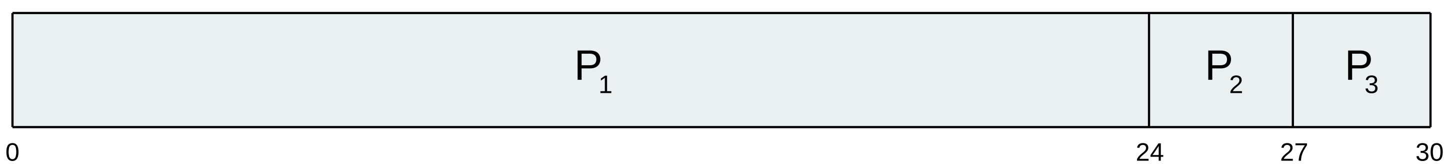
Scheduling Algorithm Optimization Criteria

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

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

FCFS/FIFO Summary

<u>The good</u>

Simple
Low Overhead
No Starvation

<u>The bad</u>

Sensitive to arrival order
(poor predictability)

<u>The ugly</u>

Convoy Effect.
Bad for Interactive Tasks

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
- SJF is optimal – gives minimum average waiting time for a given set of processes
- Preemptive version called **shortest-remaining-time-first**

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

P_4	P_1	P_3	P_2

0 3 9 16 24

- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Shortest Remaining Time First Scheduling

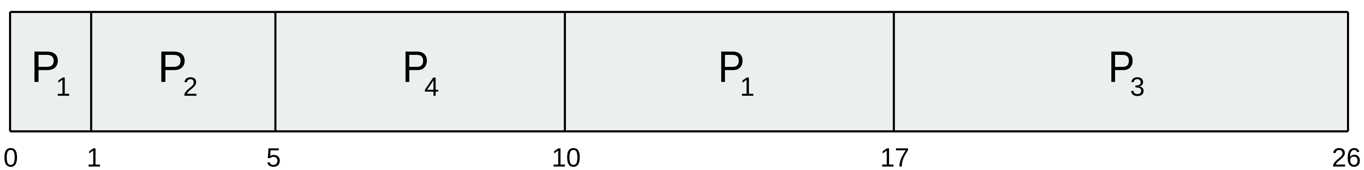
- Preemptive version of SJN
- Whenever a new process arrives in the ready queue, the decision on which process to schedule next is redone using the SJN algorithm.

Example of Shortest-remaining-time-first

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

SJF Summary

The good

Optimal Average Completion Time when jobs arrive simultaneously

The bad

Still subject to convoy effect

The ugly

Can lead to starvation!

Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** q), 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO (FCFS)
  - q small $\Rightarrow$ RR
- Note that 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

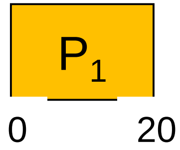
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
  - q usually 10 milliseconds to 100 milliseconds,
  - Context switch < 10 microseconds

RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

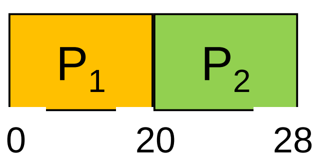
RR with Time Quantum = 20

Process	Burst Time
P_1	53 => 33
P_2	8
P_3	68
P_4	24



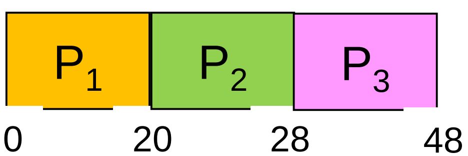
RR with Time Quantum = 20

Process	Burst Time
P_1	33
P_2	8 => 0
P_3	68
P_4	24



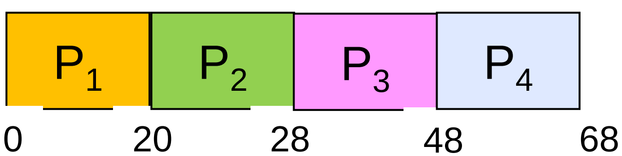
RR with Time Quantum = 20

Process	Burst Time
P_1	33
P_2	0
P_3	68 => 48
P_4	24



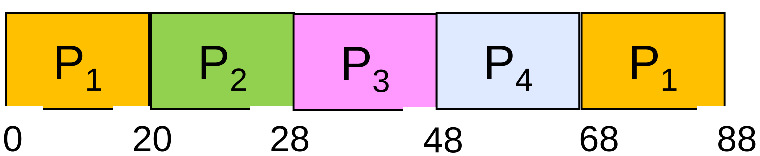
RR with Time Quantum = 20

Process	Burst Time
P_1	33
P_2	0
P_3	48
P_4	24 => 4



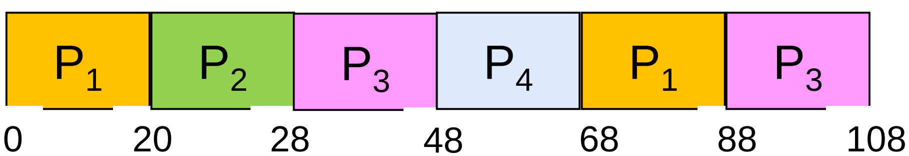
RR with Time Quantum = 20

Process	Burst Time
P_1	33 => 13
P_2	0
P_3	48
P_4	4



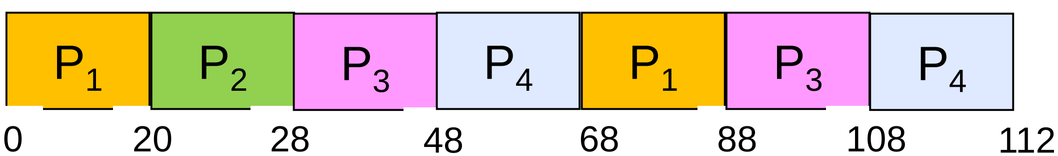
RR with Time Quantum = 20

Process	Burst Time
P_1	13
P_2	0
P_3	48 => 28
P_4	4



RR with Time Quantum = 20

Process	Burst Time
P_1	13
P_2	0
P_3	28
P_4	4 => 0



RR with Time Quantum = 20

Waiting time

- P_1= 0 + (68-20)+(112-88)=72
- P_2=(20-0)=20
- P_3=(28-0)+(88-48)+(125-108)+0=85
- P_4=(48-0)+(108-68)=88

Average waiting time

$$\left(\frac{72+20+85+88}{4} = 66.25\right)$$

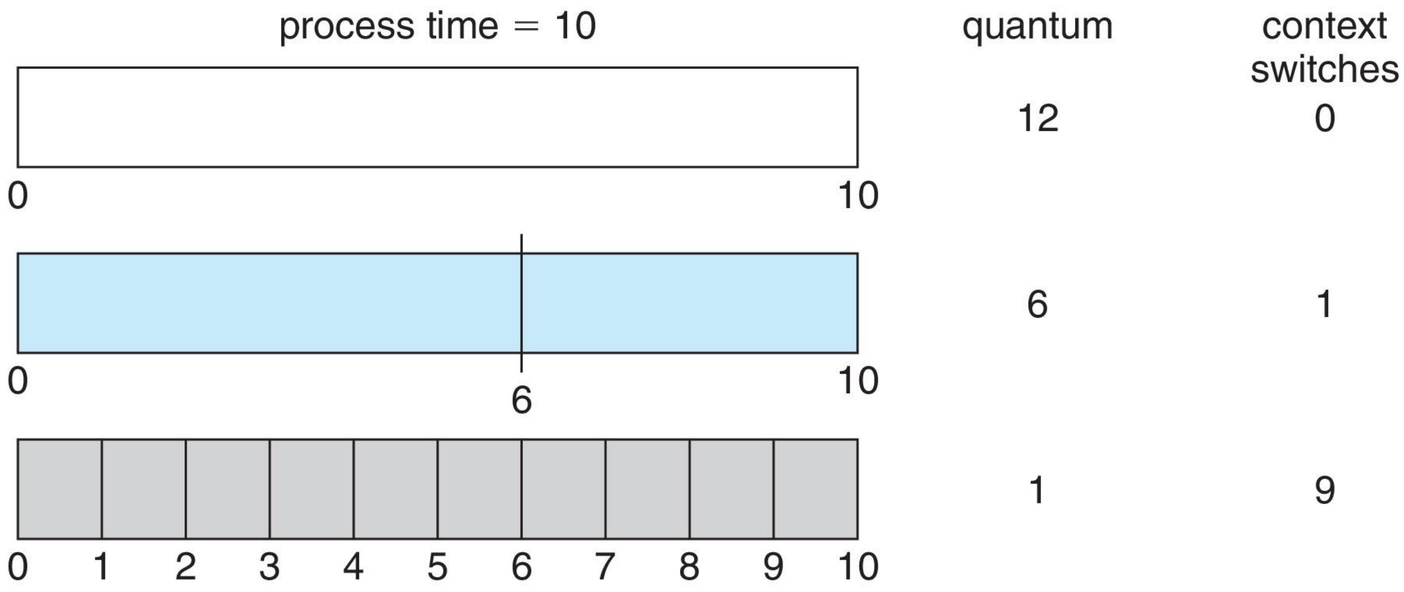
Average completion time

$$\left(\frac{125+28+153+112}{4} = 104.25\right)$$

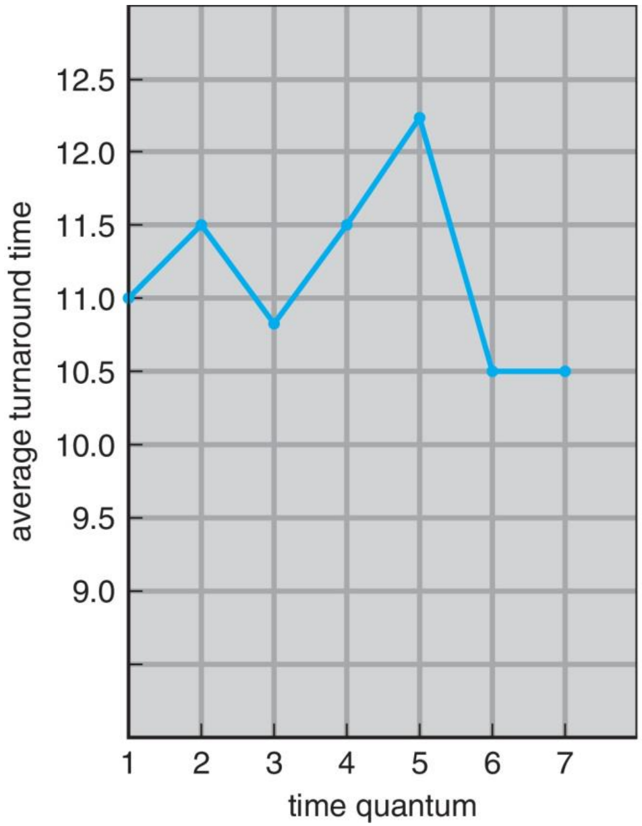
P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

The magic number

What should the time slice be?

If increase the time slice:

Average Completion Time ↓

Average Waiting Time ↑

If decrease the time slice:

Average Completion Time ↑

Average Waiting Time ↓

a) **Completion Up, Response Down**

b) **Completion** Down, Response Up

Switching is not free!

Small scheduling quantas lead to
frequent context switches
- Mode switch overhead

q must be large with respect to context switch,
otherwise overhead is too high

Are we done?

Can RR lead to starvation?

No

No process waits more than $(n\text{-}1)q$ time units

Are we done?

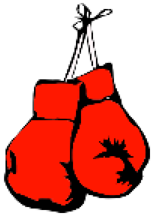
Can RR suffer from convoy effect?

No

Only run a time-slice at a time

FIFO vs RR Showdown

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24



Impact of different time quantum

Best FCFS:

P_2 [8]	P_4 [24]	P_1 [53]	P_3 [68]	
0	8	32	85	153

	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	61¼
	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

RR Summary

The good

Bounded wait time

The bad

Completion time can be high (stretches out long jobs)

The ugly

Overhead of context switching

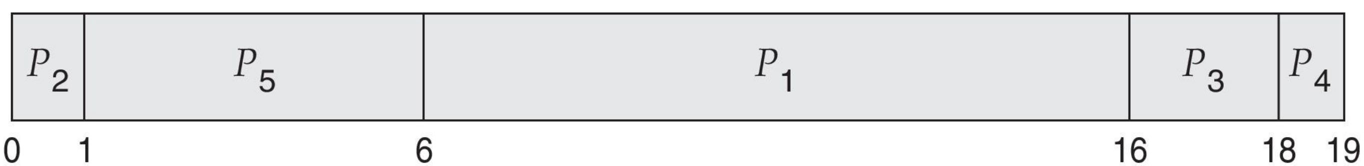
Priority Scheduling

- 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
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

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

- Priority scheduling Gantt Chart



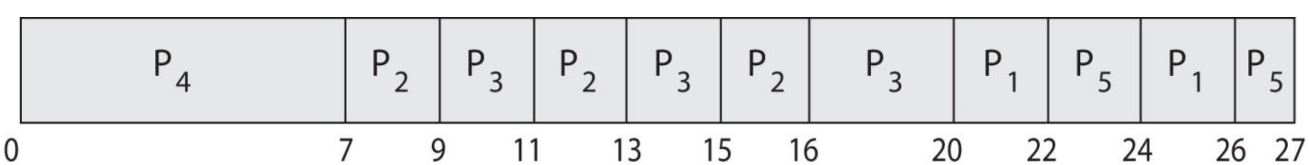
- Average waiting time = 8.2

Priority Scheduling w/ Round-Robin

- Run the process with the highest priority. Processes with the same priority run round-robin
- 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

- Gantt Chart with time quantum = 2

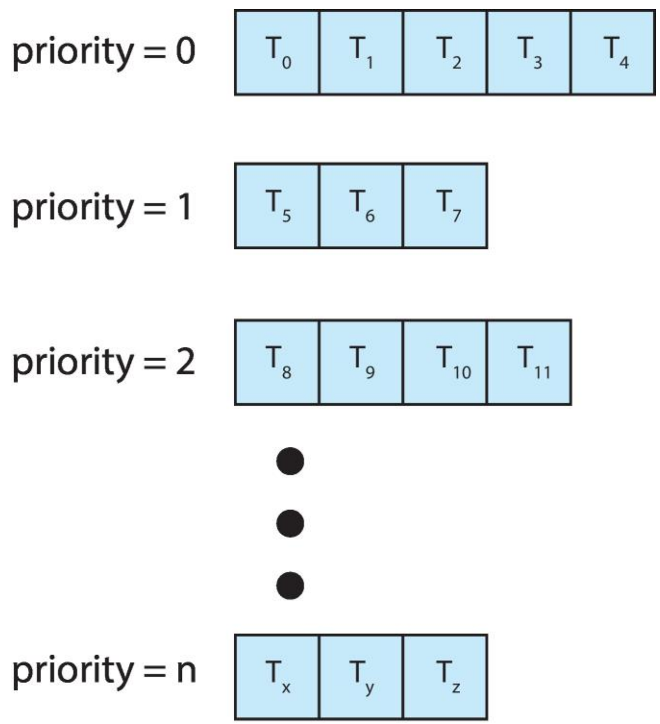


Multilevel Queue

- The ready queue consists of multiple queues
- Multilevel queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Scheduling among the queues

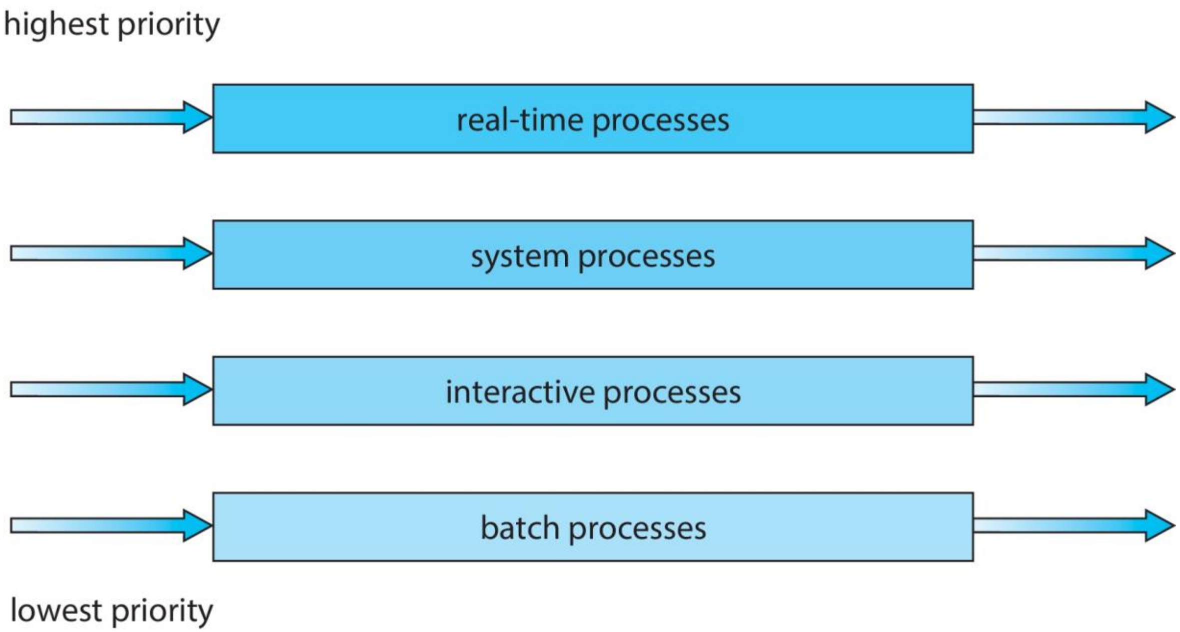
Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



Multilevel Queue

- Prioritization based upon process type

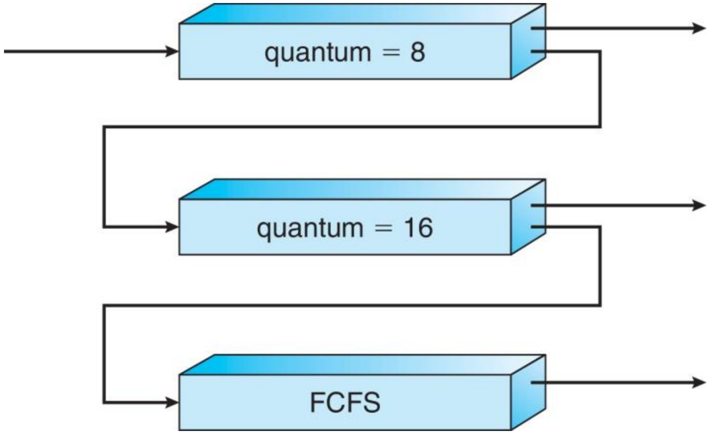


Multilevel Feedback Queue

- A process can move between the various queues.
- 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
- Aging can be implemented using multilevel feedback queue

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
- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, the process is moved to queue Q_1
  - At Q_1 job is again served in RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

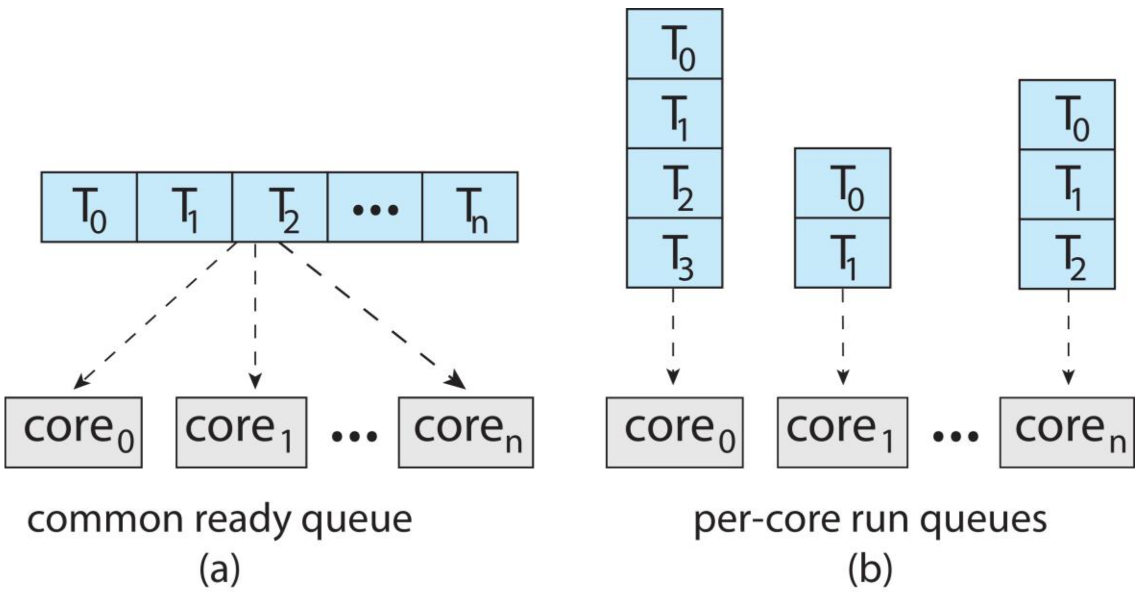


Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing

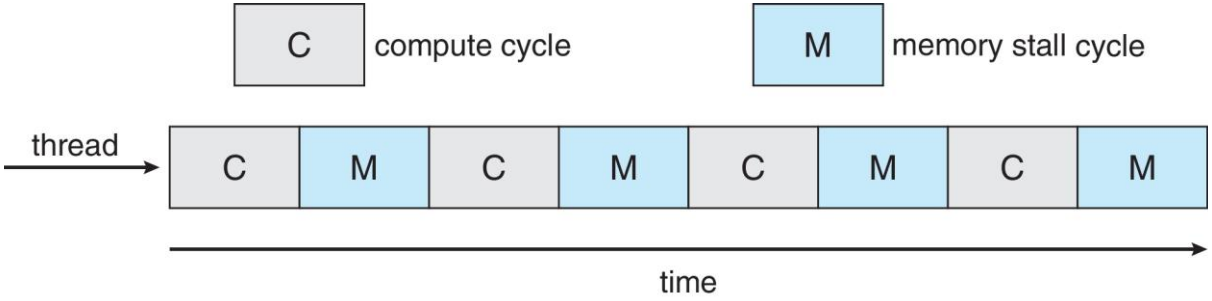
Multiple-Processor Scheduling

- Symmetric multiprocessing (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)



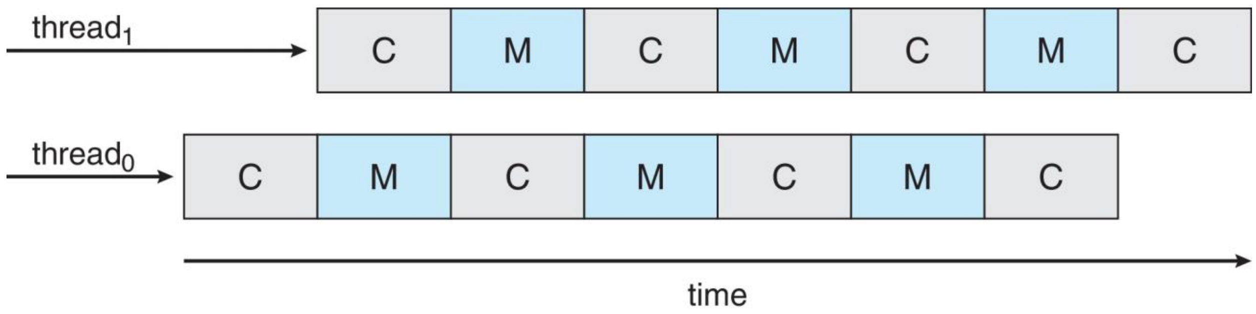
Multicore Processors

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



Multithreaded Multicore System

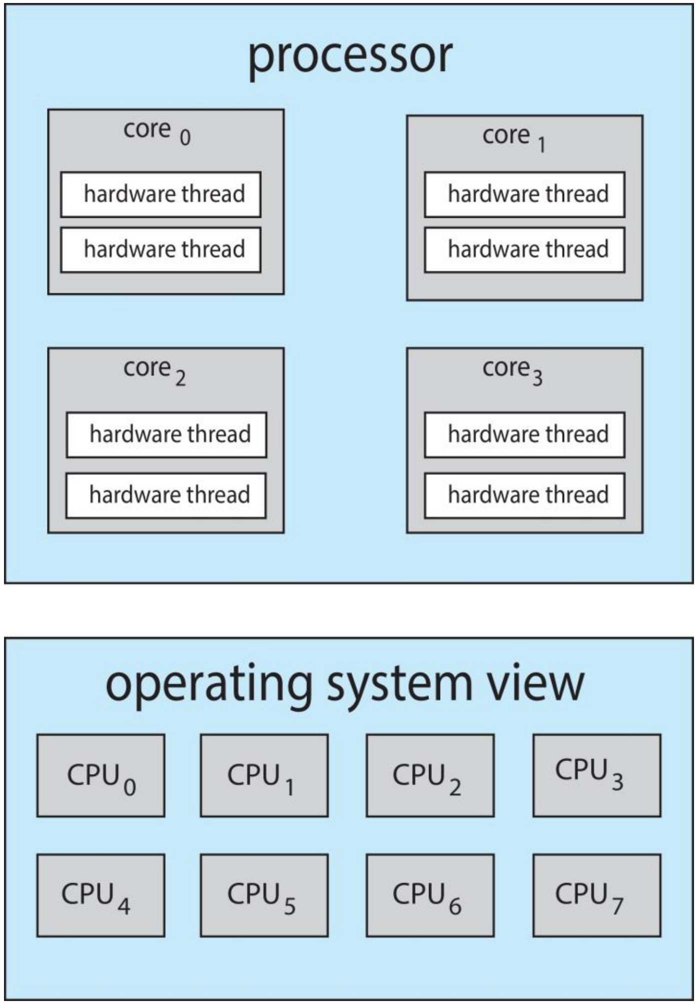
- Each core has > 1 hardware threads.
- If one thread has a memory stall, switch to another thread!
- Figure



Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)

- On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.

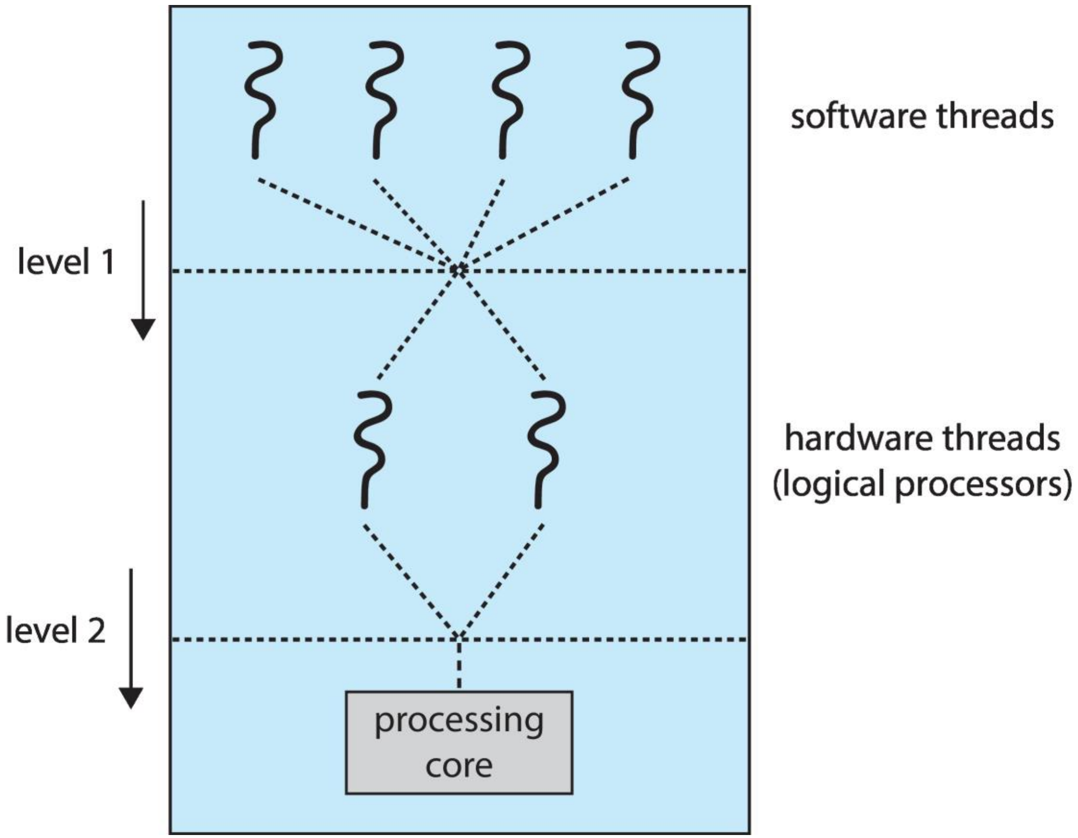


Multithreaded Multicore System

- Two levels of scheduling:

  1. The operating system deciding which software thread to run on a logical CPU

  2. How each core decides which hardware thread to run on the physical core.



Multiple-Processor Scheduling – Load Balancing

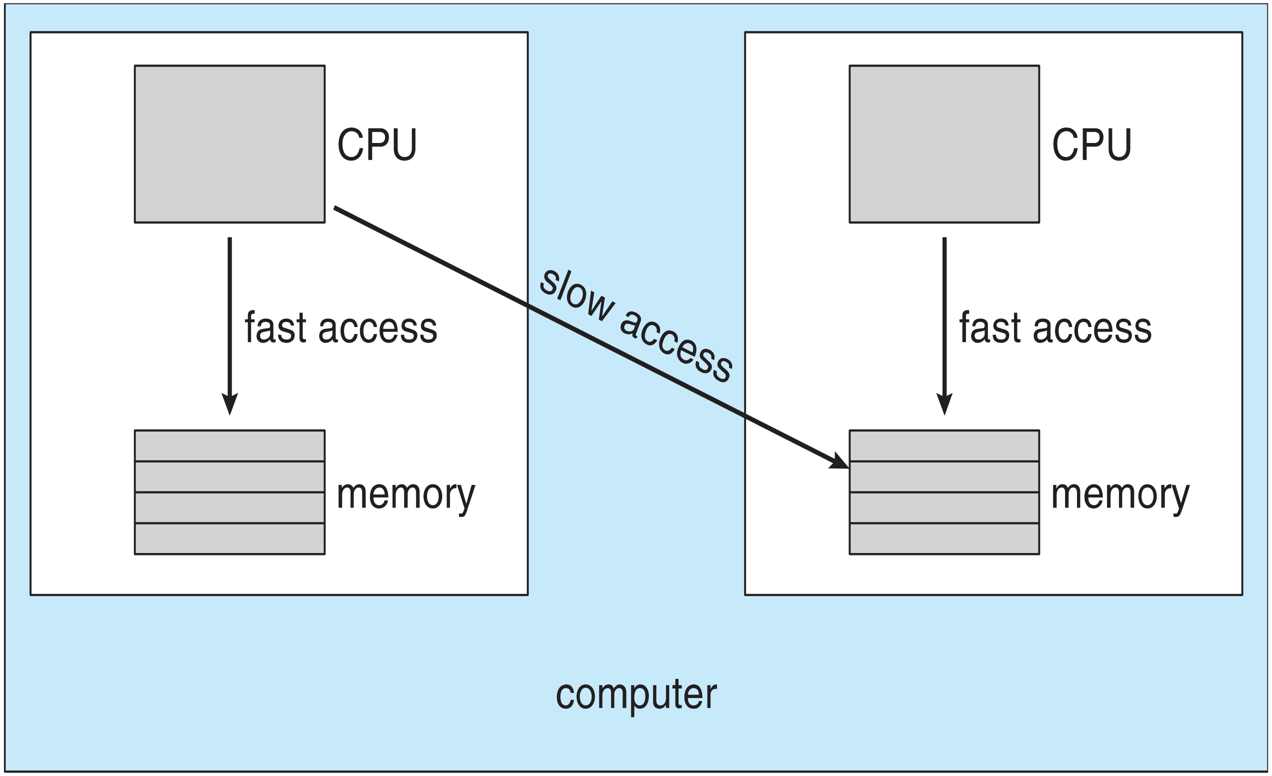
- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task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

Multiple-Processor Scheduling – Processor Affinity

- When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- We refer to this as a thread having affinity for a processor (i.e., "processor affinity")
- Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off of.
- **Soft affinity** – the operating system attempts to keep a thread running on the same processor, but no guarantees.
- **Hard affinity** – allows a process to specify a set of processors it may run on.

NUMA and CPU Scheduling

If the operating system is **NUMA-aware**, it will assign memory closes to the CPU the thread is running on.

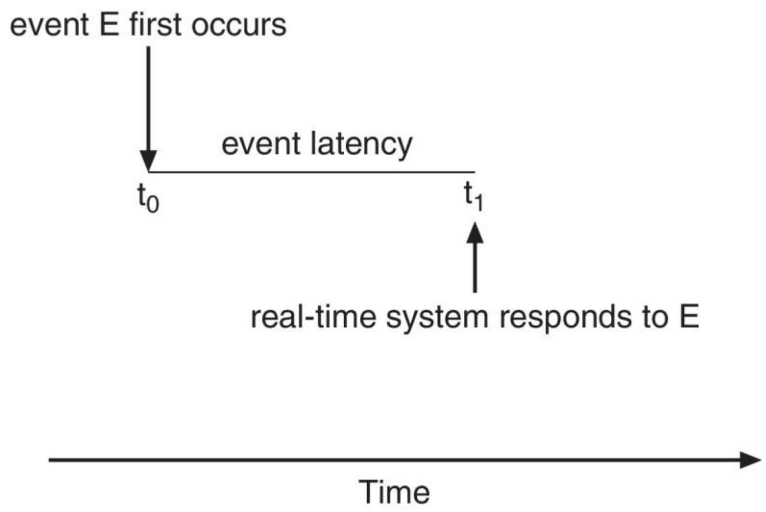


Real-Time CPU Scheduling

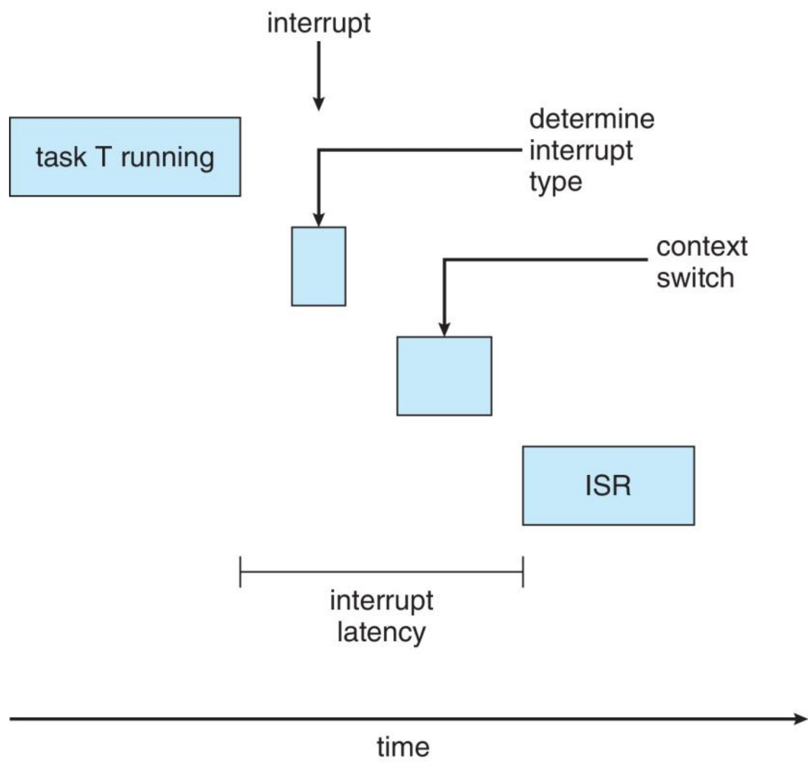
- Can present obvious challenges
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems – task must be serviced by its deadline**

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

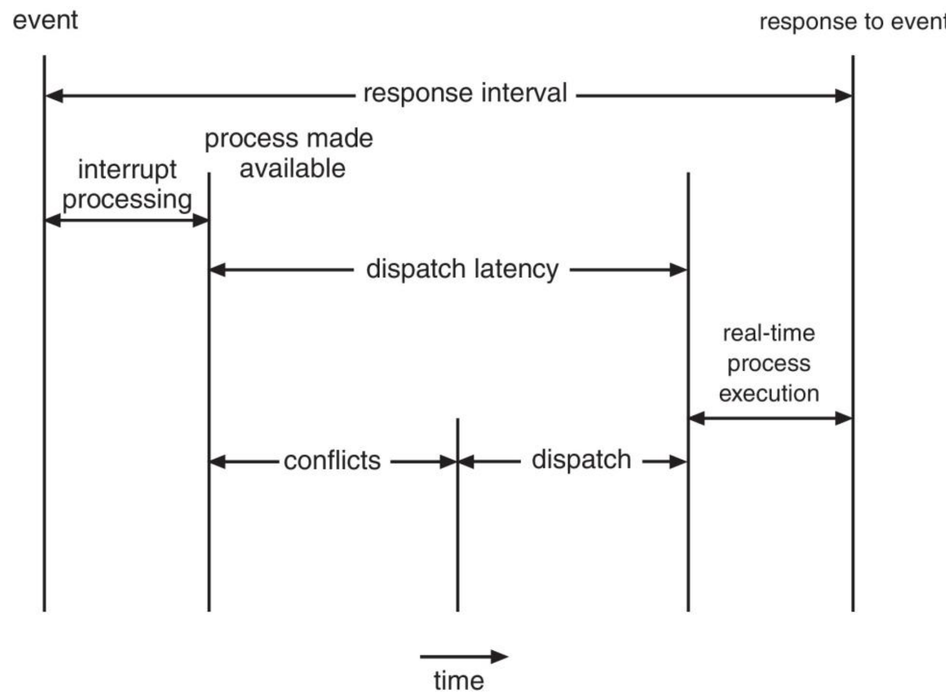


Interrupt Latency



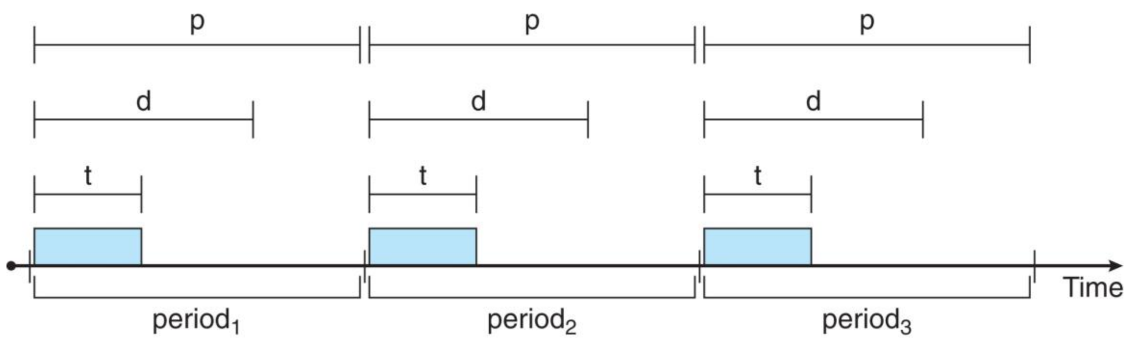
Dispatch Latency

- 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



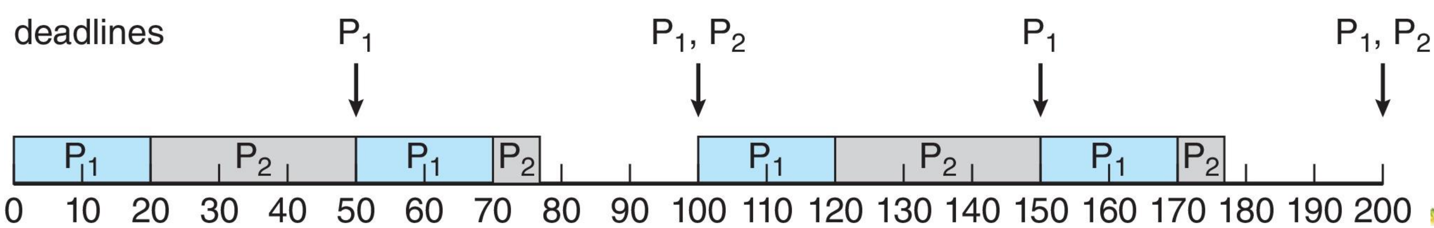
Priority-based Scheduling

- 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 $1/p$



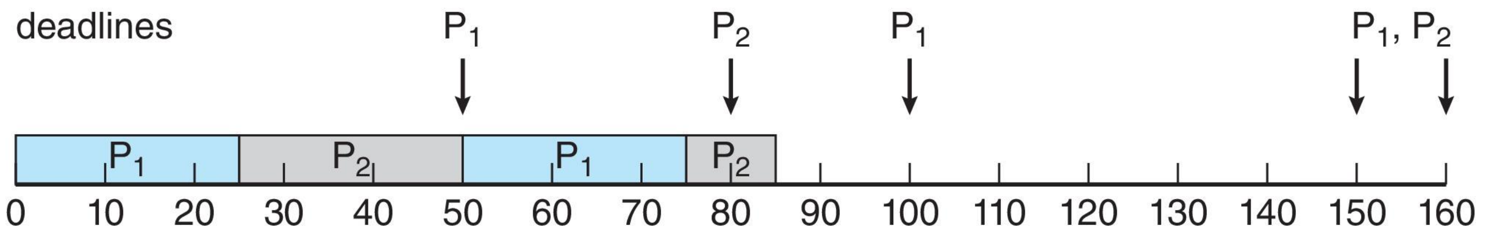
Rate Monotonic Scheduling

- 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.
- p1 = 50 and p2 = 100 , t1 = 20 for P1 and t2 = 35 for P2
- deadline for each process requires that it complete its CPU burst by the start of its next period
- CPU utilization of P1 is 20/50 = 0.40 and that of P2 is 35/100 = 0.35, for a total CPU utilization of 75 percent
- Rate-monotonic scheduling is considered optimal in that if a set of processes cannot be scheduled by this algorithm, it cannot be scheduled by any other algorithm that assigns static priorities.



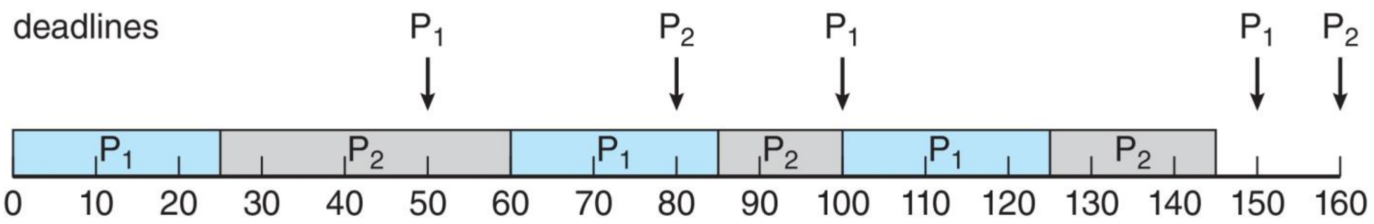
Missed Deadlines with Rate Monotonic Scheduling

- p1 = 50 and t1 = 25 , p2 = 80 and t2 = 35
- Process P_2 misses finishing its deadline at time 80
- The total CPU utilization of the two processes is (25/50) + (35/80) = 0.94
- CPU utilization is bounded, and it is not always possible to maximize CPU resources fully.
- The worst-case CPU utilization for scheduling *N* processes is $N(2^{1/N} - 1)$
- With two processes, CPU utilization is bounded at about 83 percent



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
- p1 = 50 and t1 = 25 , p2 = 80 and t2 = 35



Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where *N* < *T*
- This ensures each application will receive ***N*** / *T* of the total processor time
- Assume that a total of T = 100 shares is to be divided among three processes, A, B, and C. A is assigned 50 shares, B is assigned 15 shares, and C is assigned 20 shares.
- we have allocated 50 + 15 + 20 = 85 shares of the total of 100 shares.
- If a new process D requested 30 shares, the admission controller would deny D entry into the system

Algorithm Evaluation

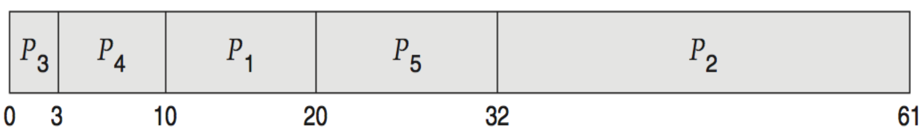
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving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

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

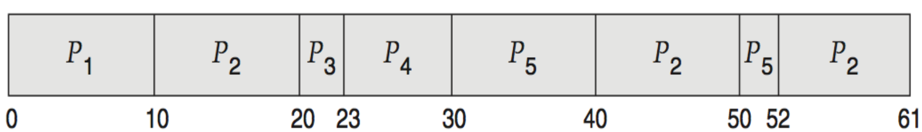
Deterministic Evaluation

- 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 SJF is 13ms:



  - RR is 23ms:



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

True, False Questions

- A **FIFO scheduler** has lower average turnaround time when long jobs arrive after short jobs, compared to when short jobs arrive after long jobs.
- True. If the long jobs arrive after the short jobs, then they will be scheduled after the short jobs; moving a long job after a short job reduces average turnaround time (just like what SFJ does).
- An **SJF scheduler** may preempt the currently running job.
- False. SJF is non-preemptive; a job has to complete or relinquish the CPU before a different job is schedule
- An **SJF scheduler** can suffer from the convoy effect
- True. Assume a very long job is scheduled and then shortly thereafter many short jobs arrive; all of those short jobs will have to wait for the one long job.

True, False Questions

- An **RR scheduler** may preempt the currently running job.
- True. RR preempts after a time slice has expired.
- If all jobs arrive at the same point in time, an SJF and an STCF scheduler will behave the same
- True, they only behave differently if a job arrives while some jobs are already running (in which case, STCF may preempt)
- With an **MLFQ scheduler**, jobs run to completion as long as there is not a higher priority job.
- False; multiple jobs at the same priority level will be scheduled with RR
- With an **MLFQ scheduler**, while a job is waiting for I/O to complete, the job remains in the READY state without changing priority.
- False; jobs move to the WAITING state when waiting for an event to complete that doesn't need the CPU.

Question

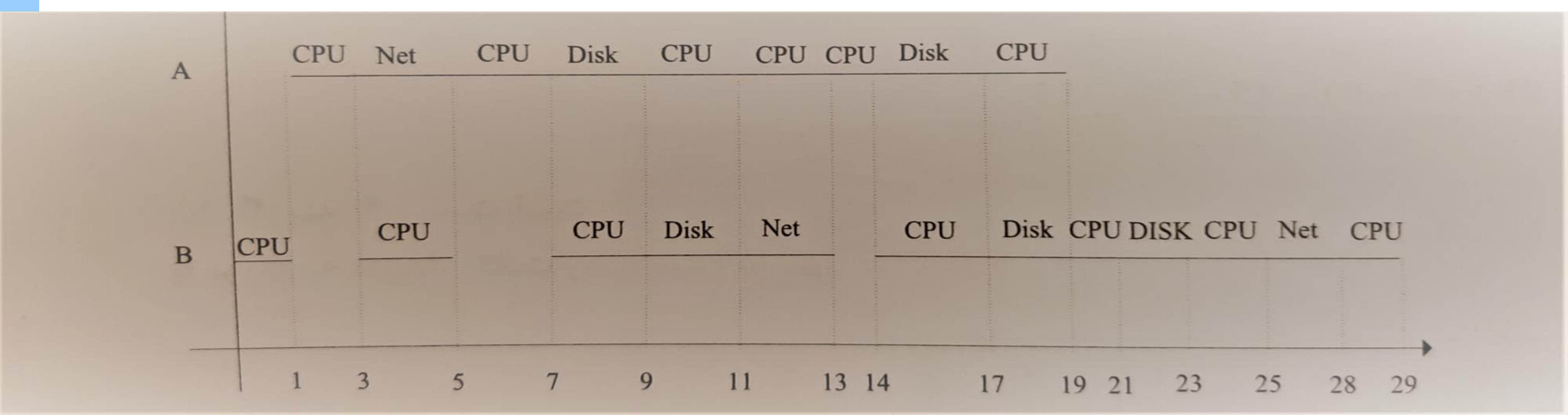
- We have 2 process
- Execution time on CPU,DISK,NET are shown
- Arrival time for proc A: 1 , Arrival time for proc B: 0
- DISK, NET are non preemptive
- SJF Preemtive is done for CPU. (Priority A is higher for equal process)
- CPU utilization?

A	B
CPU 2	CPU 5
NET 2	DISK 2
CPU 2	NET 2
DISK 2	CPU 5
CPU 5	DISK 2
DISK 3	CPU 2
CPU 2	NET 3
	CPU 1

Answer

- Idle times: 21-23
- 25-28
- $U = \frac{29-5}{29} \times 100\% \approx 83\%$

A	B
CPU 2	CPU 5
NET 2	DISK 2
CPU 2	NET 2
DISK 2	CPU 5
CPU 5	DISK 2
DISK 3	CPU 2
CPU 2	NET 3
	CPU 1



Question

We have MLFQ, q0 -> 8ms , q1->16ms, q3->FCFS

We have 6 process that inters at time 0.

Time execution: 4,7,12,20,25,30

Average Turnaround time?

Average Waiting time?

Answer

4	7	12	20	25	30	8
4	11	19	27	35	43	
		4	12	17	22	16
		47	59	75	91	
				1	6	FCFS
				92	98	

p1	p2	p3	p4	p5	p6	p3	p4	p5	p6	p5	p6

ATT = 4+11+47+59+92+98/6= 51.83

AWT = 0 + (11-7)+(47-12)+(59-20)+(92-25)+(98-30)/6 = 35.5

End of Chapter 5