Operating Systems

Lecture 9: CPU Scheduling

Department of Computer Science & Technology
Tsinghua University

Outline

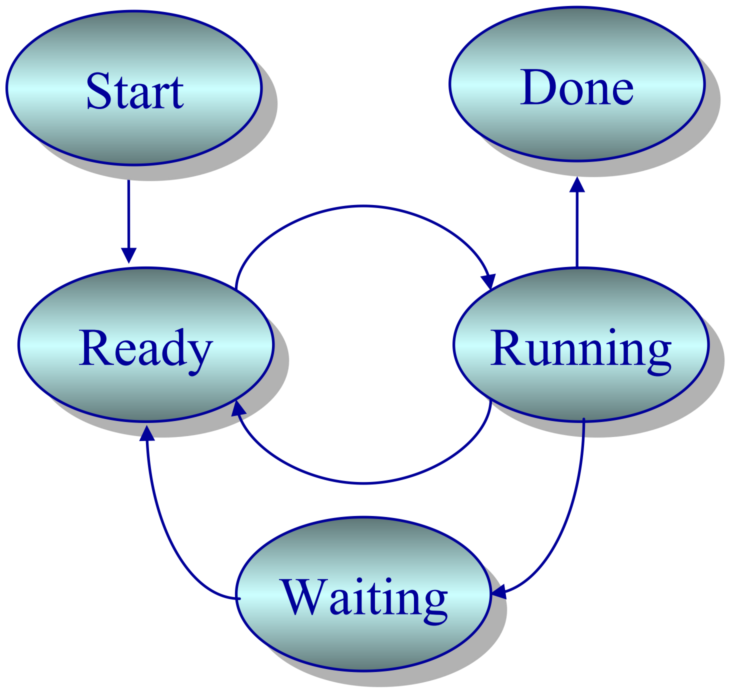
- Background
  - CPU scheduling
  - CPU Scheduling Time
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
- Multiprocessor Scheduling
- Priority Inversion

OS Mechanisms for Time Multiplexing

- Context switch
  - Switch CPU's current task, from one process/thread to another
  - Save the execution context (CPU state) of the current process/thread in PCB/TCB
  - Load the context of the next process/thread
- CPU scheduling
  - Pick a process/thread from the Ready queue to execute next in CPU
  - Scheduler: a kernel function that returns the pick (according to some scheduling policy)
  - When?

When do you call scheduler?

- When in the process/thread life cycle?



CPU Scheduling Time

- The kernel runs the scheduler at least when
  - a process switches from running to waiting,
  - a process is terminated.
- If non-preemptive
  - The scheduler must wait for one of these events
- If preemptive
  - The scheduler runs after and interrupt is serviced
  - Current process from running to ready, or a process from waiting to ready
  - Current running process can be switched out

Outline

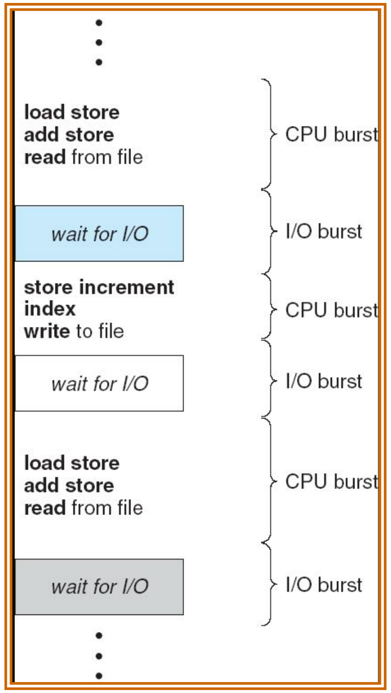
- Background
- Scheduling Criteria
  - Scheduling Policy
  - Program Execution Model
  - Criteria for Comparing Scheduling Algorithms
  - Throughput vs. Latency
  - The “Fairness” Goal
- Scheduling Algorithms
- Real-Time Scheduling
- Multiprocessor Scheduling
- Priority Inversion

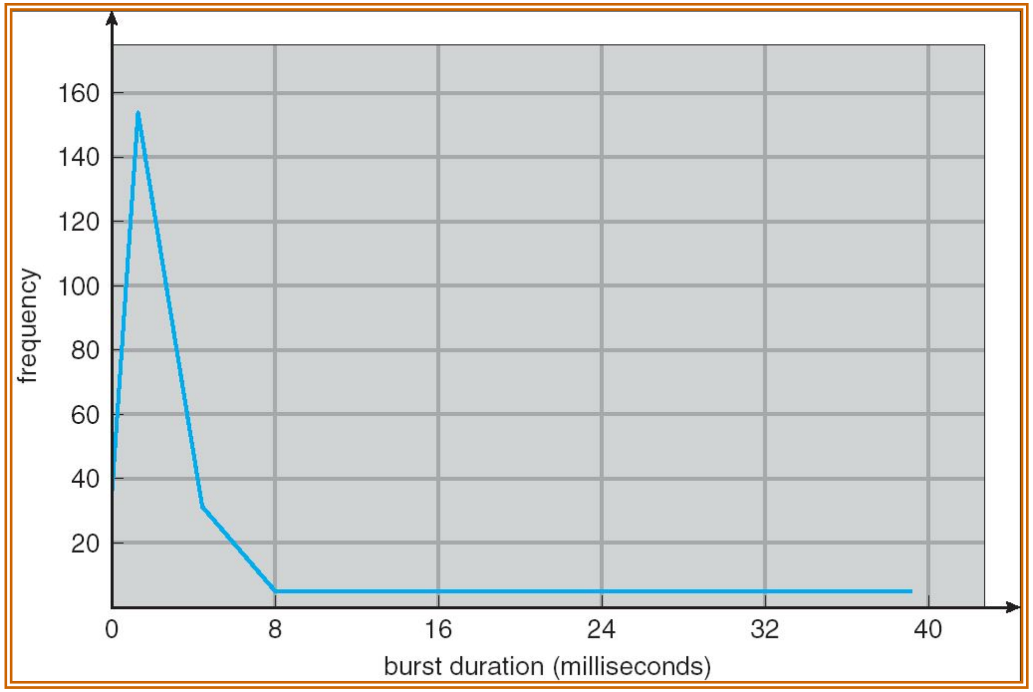
Scheduling Policy

- Which one (in the Ready queue) to pick?
  - The first one? Or according to some criteria?
- Scheduling policy
  - Determines how the OS should select a process from the ready queue to execute?
  - Goal and options
- Scheduling algorithm
  - Implementation of a policy in CPU scheduler

- Which policy/algorithm is better?

The CPU Bursts Model





- Execution model: programs alternate between bursts of CPU and I/O
  - Each scheduling decision is about which job to give to the CPU for use by its next CPU burst
  - With timeslicing, thread may be forced to give up CPU before finishing current CPU burst

Criteria for Comparing Scheduling Algorithms

- CPU Utilization
  - The percentage of time that the CPU is busy
- Throughput
  - The number of processes completing in a unit of time
- Turnaround time
  - The length of time it takes to run a process from initialization to termination, including all the waiting time
- Waiting time
  - The total amount of time that a process is in the ready queue
- Response time
  - Amount of time it takes from when a request was submitted until the first response is produced.

Throughput vs. Latency

- People often say they want “faster" service.
- What is faster?
  - If they transfer files, then they want large bandwidth
  - If they play games, they probably want low latency
  - These two factors are separate
- Analogy to water pipes
  - Low latency: if I want a drink, I want water to come out of the spout as soon as I turn it on
  - High bandwidth: if I wan to fill up a swimming pool, I want a lot of water coming out of that spout at the same time, and I don't care if it takes long before I see the first drop

CPU Scheduling Policy Goals

- Minimize response time
  - provide output to the user as quickly as possible and process their input as soon as it is received.
- Minimize variance of average response time
  - in an interactive system, predictability may be more important than a low average with a high variance.
- Maximize throughput - two components
  - minimize overhead (OS overhead, context switching)
  - efficient use of system resources (CPU, I/O devices)
- Minimize waiting time
  - Minimize the time each process waits for its turn

Other considerations

- Scheduling for low latency maximizes interactive performance
  - This is good because if my mouse doesn't move, I might reboot the machine
- But the OS needs to make sure that throughput does not suffer
  - I want my long running programming to finish, so the OS must schedule it occasionally, even if there are many interactive jobs
- Throughput is computational bandwidth.
- Response time is computational latency.

The "Fairness" Goal

- What is the definition of fairness
- Example
  - Ensuring each process occupies same amount of CPU time
  - Fair? What if a user runs more processes than another?
- Example
  - Ensuring each process waits the same amount of time
- Fairness often increases average response time

Outline

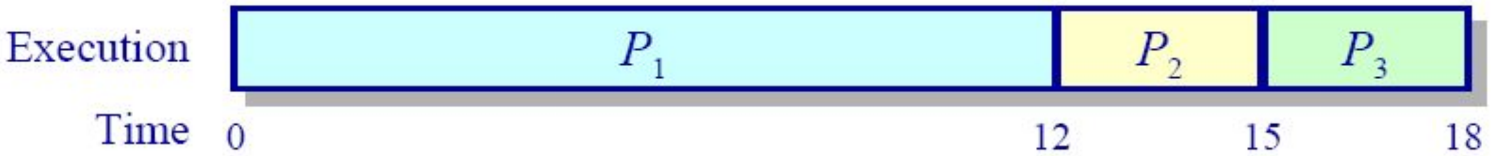
- Background
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
- Multiprocessor Scheduling
- Priority Inversion

Scheduling Algorithms

- FCFS
  - First Come, First Served
- SJF
  - Shortest Job First
- Priority Scheduling
  - User denotes process priority
- Round Robin
  - Use a time slice and preemption to alternate jobs.
- Multilevel Feedback Queues
  - Round robin on priority queue.
- Lottery Scheduling *
  - Jobs get tickets and scheduler randomly picks winning ticket.
- Stride Scheduling *
  - Jobs get tickets and scheduler determinately picks winning ticket.
- WFQ *
  - Weighted Fair Queuing

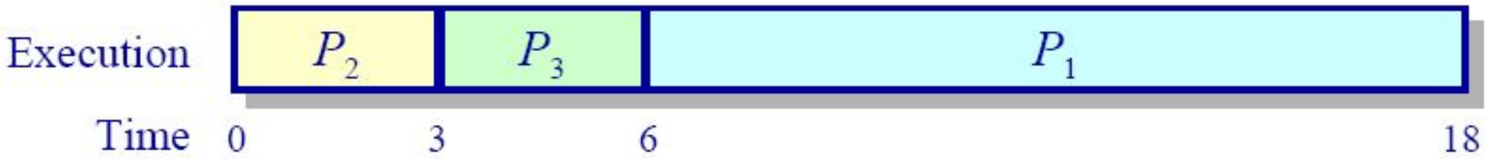
First-Come-First-Served (FCFS)

- The discipline corresponding to FIFO queuing
  - If a process blocks while executing, CPU is given to next in queue
- Example — 3 processes w/ compute times 12, 3, 3
  - Job arrival order P_1, P_2, P_3



Average response time = (12 + 15 + 18)/3 = 15

- Job arrival order P_2, P_3, P_1



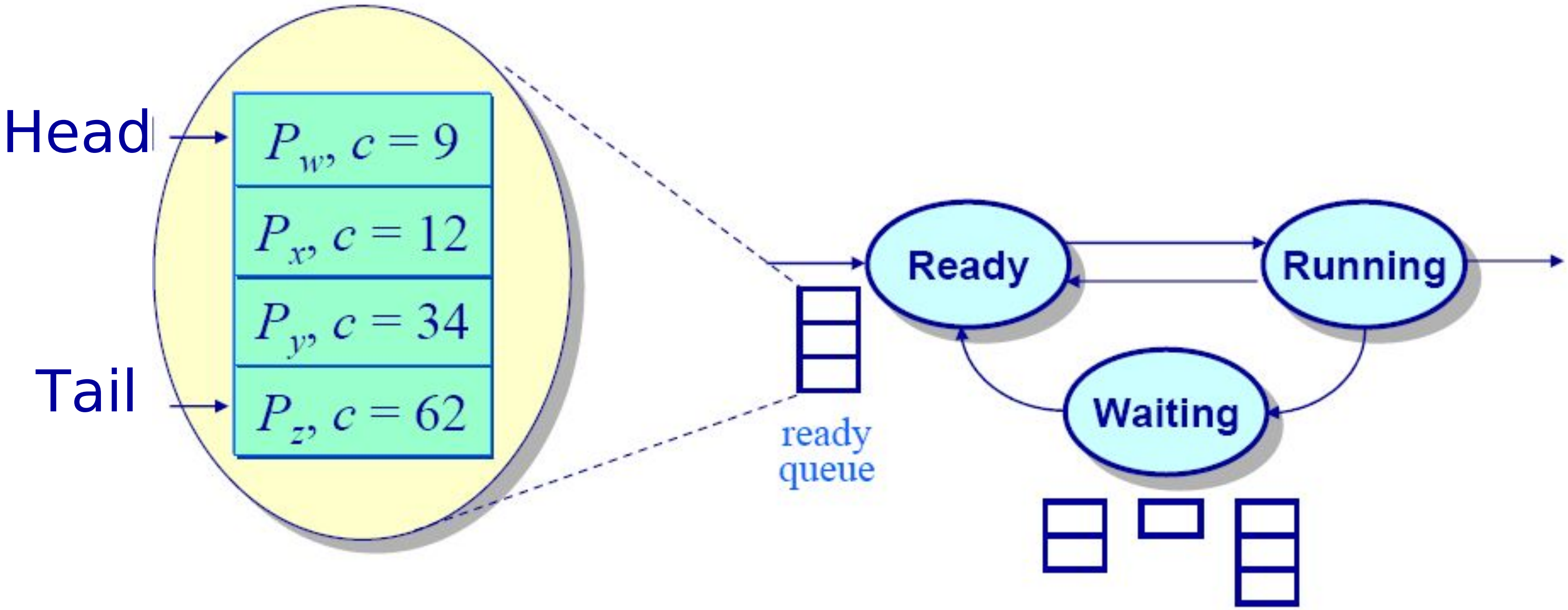
Average response time = (3 + 6 + 18) /3 = 9

FCFS (cont'd)

- Pro
  - Simple!
- Con
  - Average waiting time is highly variable
  - Short jobs may wait behind long ones !!
  - May lead to poor overlap between I/O and CPU processing
    - CPU bound processes will make I/O bounds processes to wait, when I/O devices remain idle

Shortest-Job-First (SJF)

- Select the shortest job first
  - Enqueue jobs in order of estimated completion time

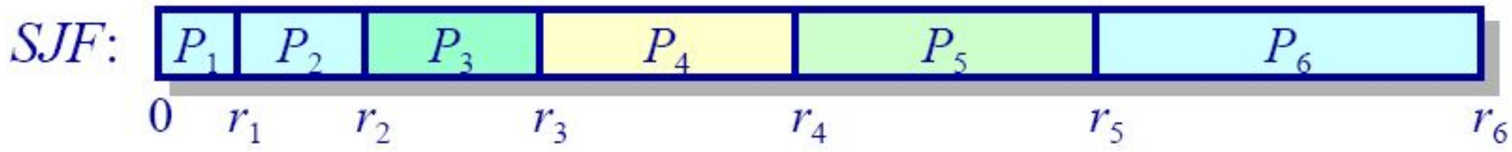


- Can be preemptive or non-preemptive
  - Preemptive: aka. Shortest-Remaining-Time-First

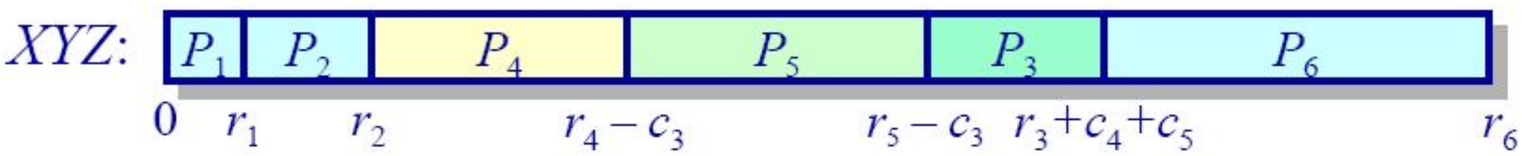
SJF - Advantage

- Provably optimal mean waiting time
  - Consider an SJF execution of a set of processes

$$\text{Average response time} = (r_1 + r_2 + r_3 + r_4 + r_5 + r_6)/6$$



Can switching the execution order reduce response time?



$$\begin{aligned}\text{Average response time} &= (r_1 + r_2 + r_4 - c_3 + r_5 - c_3 + r_4 + c_4 + c_5 + r_6)/6 \\ &= (r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + (c_4 + c_5 - 2c_3))/6\end{aligned}$$

SJF - Disadvantage

- Possible starvation
  - Continuous stream of short jobs will starve long jobs
  - Any CPU time to long jobs when short jobs are available will always degrade average waiting time
- Need to know the future
  - How do you estimate the duration of next CPU burst?
  - Simple solution: ask the user! (Yeah, right!!)
  - Kill the process if the user cheats
  - What if the user doesn’t know?

SJF - Estimating Execution Time

- Recent history is a good indicator of the near future

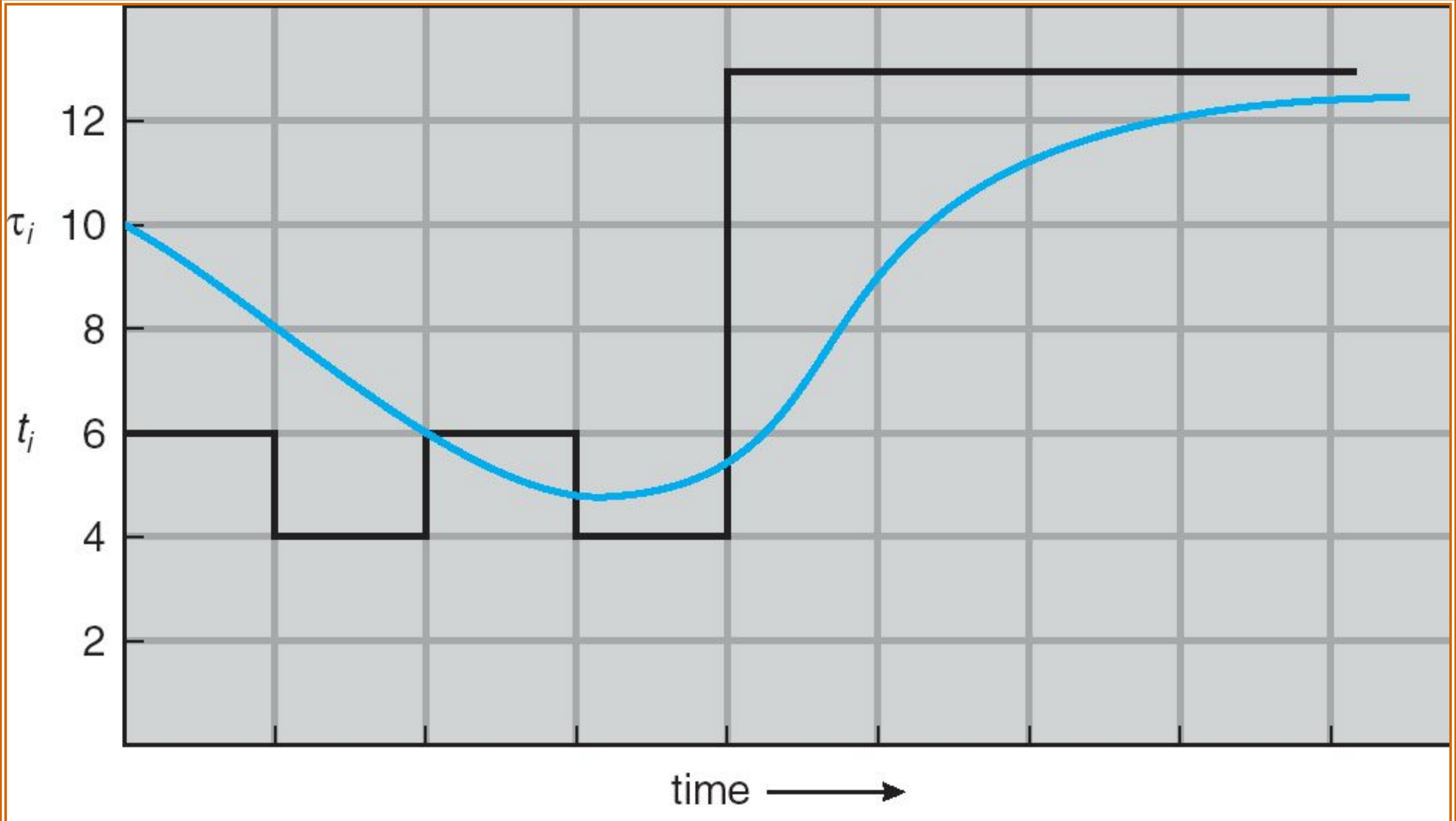
```
process P
begin
   loop
      <read input from user>
      <process input>
   end loop
end P
```

t_n — duration of the n^{th} CPU burst

τ_{n+1} — predicted duration of the $n+1^{st}$ CPU burst

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, \quad \text{for } 0 \le \alpha \le 1$$

Estimating Execution Time



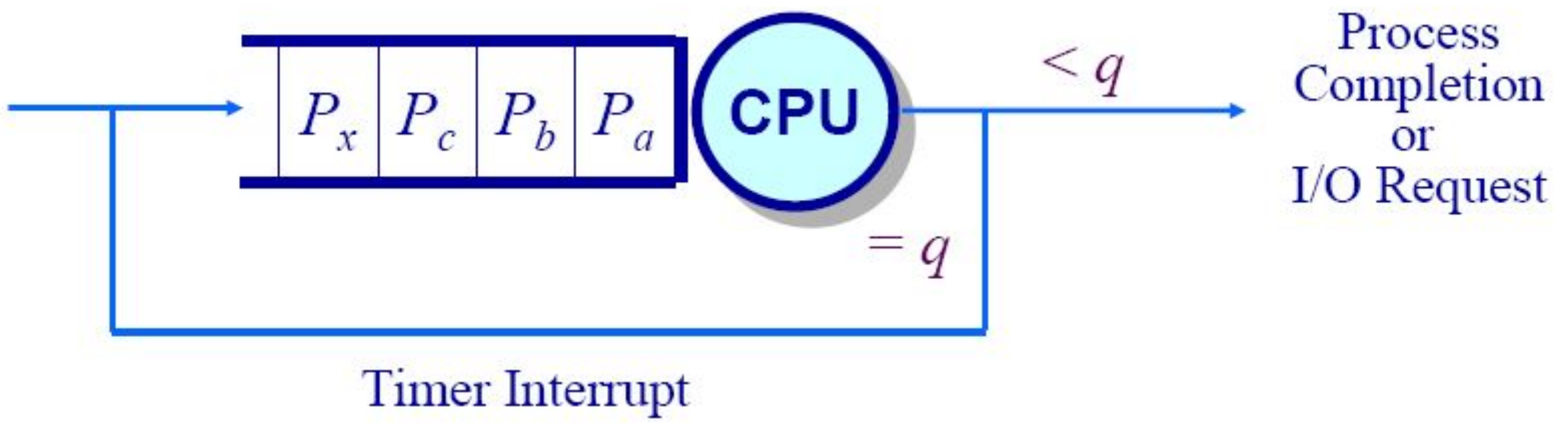
CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Priority Scheduling (PS)

- Assign a priority (a number) to each job and schedule jobs in order of priority
  - Typically low priority numeric values = "high priority"
  - If priority is τ_n, then a priority scheduler becomes a SJF
- Disadvantage: Starvation
  - Low priority processes may never execute
- Aging: avoiding starvation
  - Gradually increase a process's priority (decrease its priority numeric value) over time

Round-Robin (RR)

- Allocate the processor in discrete unit called quantum (or timeslice)
- Switch to the next ready process at the end of each quantum
  - Processes execute every (n – 1)q time units



Example RR with Time Quantum = 20

- Example:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - The Gantt chart is:

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

  - Waiting time for $P_1=(68-20)+(112-88)=72$

    $P_2=(20-0)=20$

    $P_3=(28-0)+(88-48)+(125-108)=85$

    $P_4=(48-0)+(108-68)=88$

  - Average waiting time = (72+20+85+88)/4=66¼

RR – Selecting a Time Quantum

- RR overhead: additional context switches
- Time quantum too large
  - Long waiting time
  - Degenerates to FCFS in the limit
- Time quantum too small
  - Responsive, but …
  - Throughput suffers due to large context switch overhead
- Goal:
  - Select a time quantum that balances this tradeoff
  - Rule of thumb: maintain context switch overhead to <1%

Exercise: Comparing FCFS and RR

- Example: Process Burst Time

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Assuming context-switch time is zero
  - What is the average wait time under FCFS or RR?

Quantum	P_1	P_2	P_3	P_4	Average
RR (q=1)					
RR (q=5)					
RR (q=8)					
RR (q=10)					
RR (q=20)					
Best FCFS					
Worst FCFS					

Exercise: Comparing FCFS and RR

- Example: ProcessBurst Time

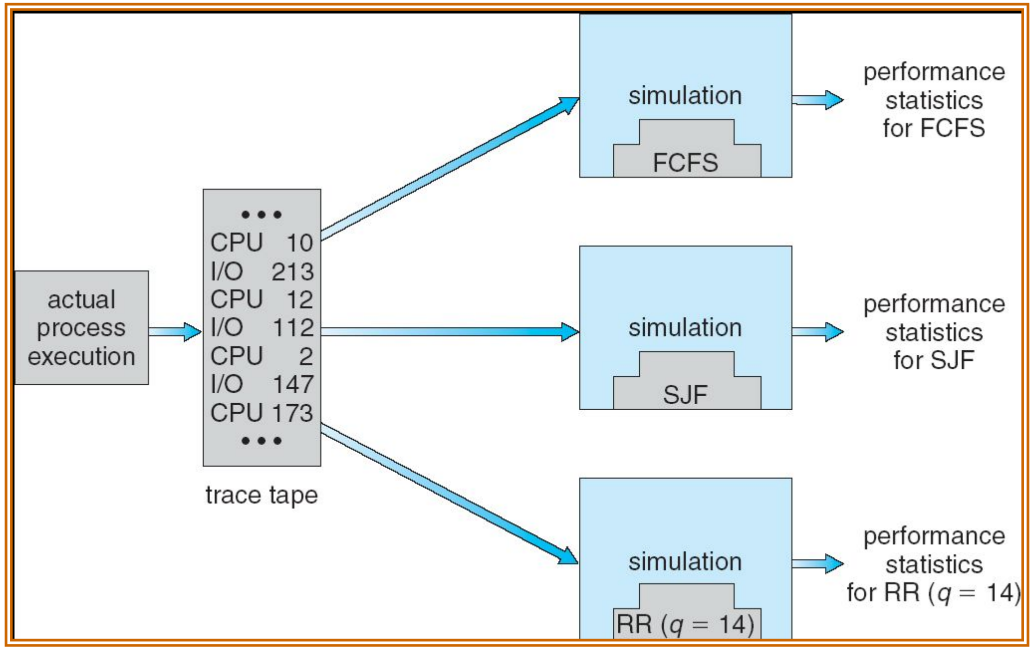
Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  - Assuming context-switch time is zero
  - What is the average wait time under FCFS or RR?

Quantum	P_1	P_2	P_3	P_4	Average
RR (q=1)	84	22	85	57	62
RR (q=5)	82	20	85	58	61.25
RR (q=8)	80	8	85	56	57.25
RR (q=10)	82	10	85	68	61.25
RR (q=20)	72	20	85	88	66.25
Best FCFS	32	0	85	8	31.25
Worst FCFS	68	145	0	121	83.5

How to Evaluate a Scheduling Algorithm

- Deterministic modeling
  - takes a predetermined workload and compute the performance of each algorithm for that workload
- Queuing models
  - Mathematical approach for handling stochastic workloads
- Implementation/Simulation:
  - Build system which allows actual algorithms to be run against actual data. Most flexible/general.

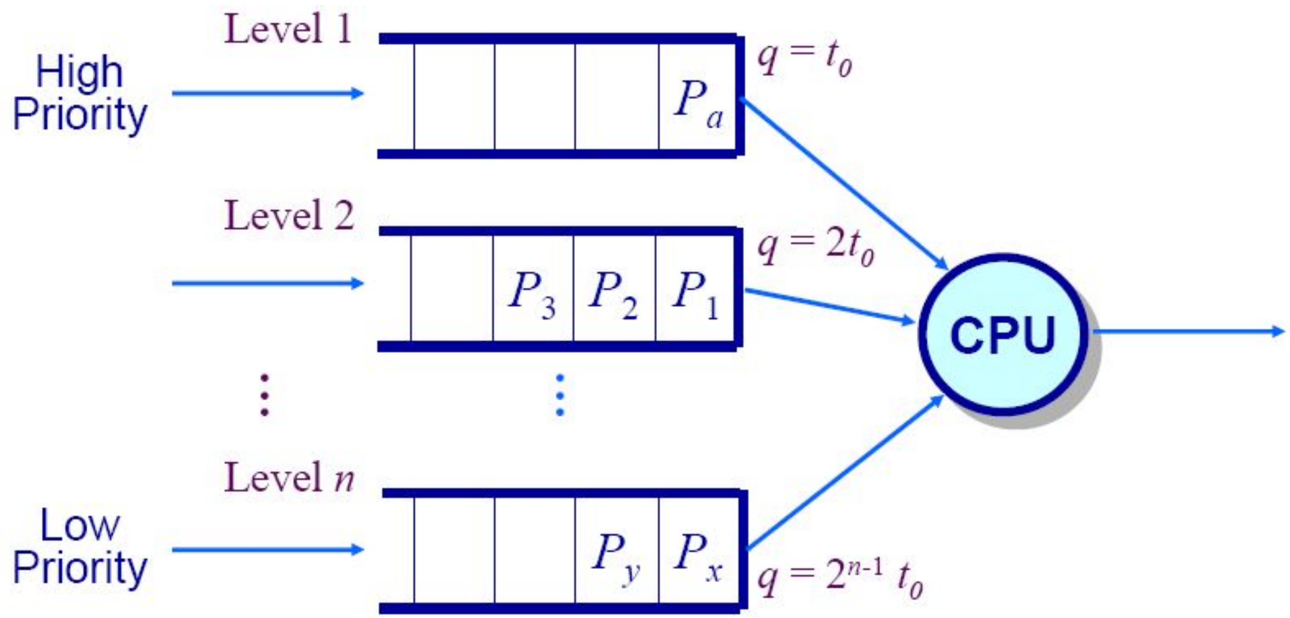


Multi-level Queues

- Ready queue is partitioned into separate queues:
  - E.g. foreground (interactive), background (batch)
- Each queue has its own scheduling algorithm
  - E.g. foreground – RR, background – FCFS
- Scheduling must be done between the queues
  - Fixed priority
    - 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

Multi-level Feedback Queues (MLFQ)

- A process can move between the various queues
- Example: n priority levels — priority scheduling between levels, round-robin within a level
  - Quantum size increases with priority level
  - Jobs are demoted to next priority levels if they don't complete within the current quantum



Approximating SJF with MLF

- CPU bound jobs drop quickly in priority
- I/O bound jobs stay at a high priority

Lottery Scheduling

- Give every job some number of lottery tickets
- On each time slice, randomly pick a winning ticket
- On average, CPU time is proportional to the number of tickets given to each job
- To approximate SJF
  - Assign tickets by giving the most to short running jobs, and fewer to long running jobs
  - To avoid starvation, every job gets at least one ticket.
- Degrades gracefully as load changes
  - Adding or deleting a job affects all jobs proportionately, independent of the number of tickets a job has

Lottery Scheduling Example

- Assume short jobs get 10 tickets, long jobs get 1 ticket

# short jobs/ # long jobs	% of CPU each short jobs gets	% of CPU each long jobs gets
1/1	91%	9%
0/2	N/A	50%
2/0	50%	N/A
10/1	9.9%	0.99%
1/10	50%	5%

Summary of Traditional Scheduling Algorithms

- FCFS
  - Not fair, and poor average waiting times
- SJF/SRTF/Priority Scheduling
  - Not fair, but average waiting time is minimized
  - Requires accurate prediction of computation times
  - Starvation is possible
- Priority Scheduling
  - Represent user intention
- Round Robin
  - Fair, but poor average waiting times
- MLFQ
  - An approximation to SJF
- Lottery Scheduling
  - Fairer with a low average waiting time, but less predictable
- Stride Scheduling
  - A deterministic solution for fairness

Fair Scheduling Algorithms

- Processes are assigned weights
- **Fairness**: processes receive CPU in proportion to their weights

$$\left|\frac{W_p(\Delta)}{r_p} - \frac{W_q(\Delta)}{r_q}\right| = 0$$

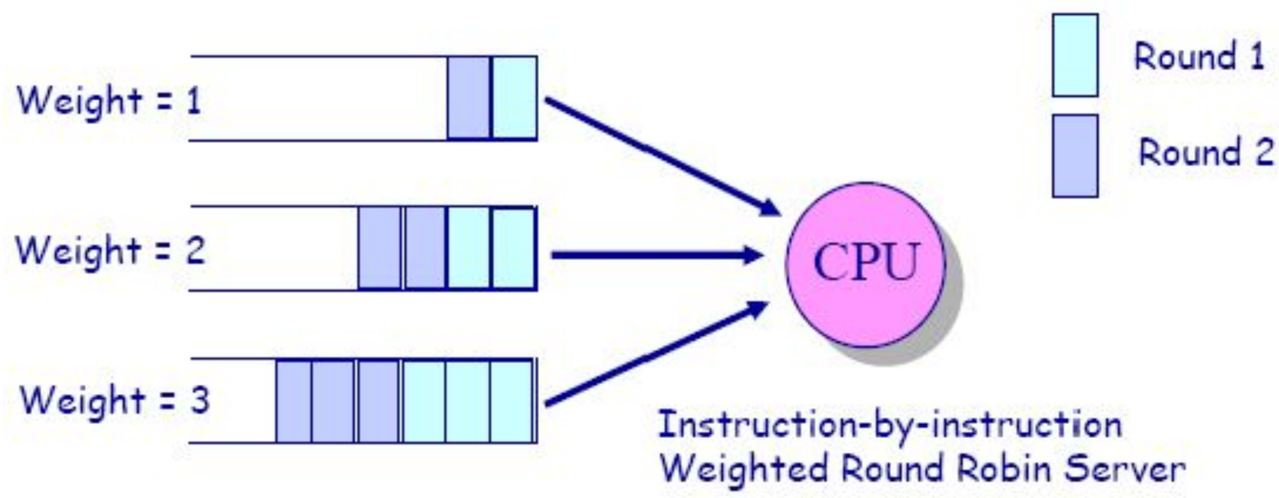
- Quantum-based CPU scheduling:

$$\left|\frac{W_p(\Delta)}{r_p} - \frac{W_q(\Delta)}{r_q}\right| \le U(p,q)$$

  - where U(p,q) is the unfairness measure
  - Objective: achieve small unfairness measure

Weighted Fair Queuing (WFQ)

- Emulate a fluid-flow model using an instruction-by-instruction weighted round robin server



- Schedule processes in the finish order in the weighted round robin server
- Caveat: emulation is expensive !

Outline

- Background
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
  - Real-Time Systems
  - Schedulability
  - Rate Monotonic(RM)
  - Earliest Deadline First(EDF)
- Multiprocessor Scheduling
- Priority Inversion

Real-Time Systems

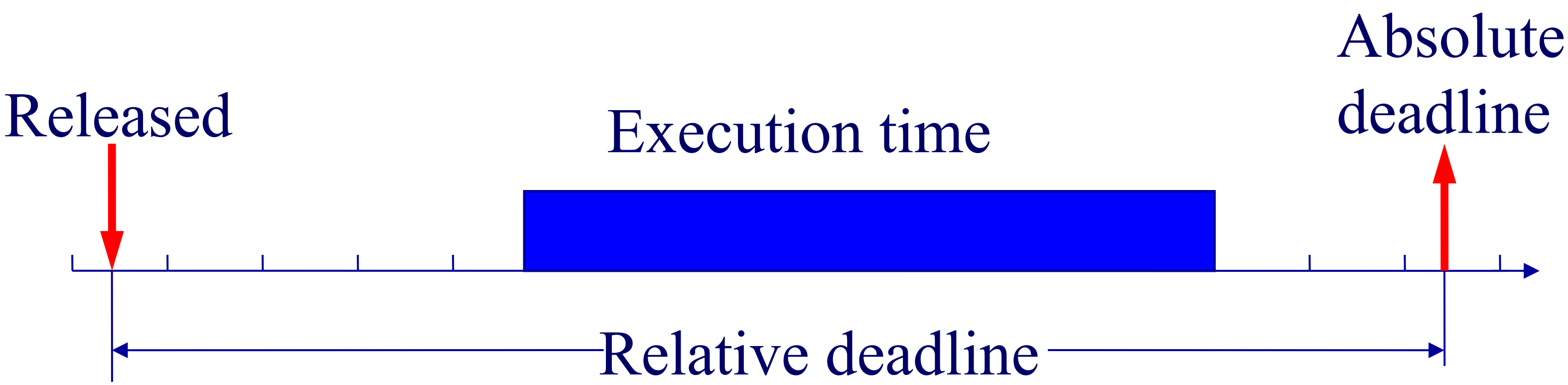
- Definition
  - Systems whose correctness depends on their temporal aspects as well as their functional aspects
- Performance measure
  - Timeliness on timing constraints (deadlines)
  - Speed/average case performance are less significant.
- Key property
  - Predictability on timing constraints

Real-Time Scheduling

- *Hard real-time* systems
  - required to complete a critical task within a guaranteed amount of time
- *Soft real-time* computing
  - requires that critical processes receive priority over less fortunate ones

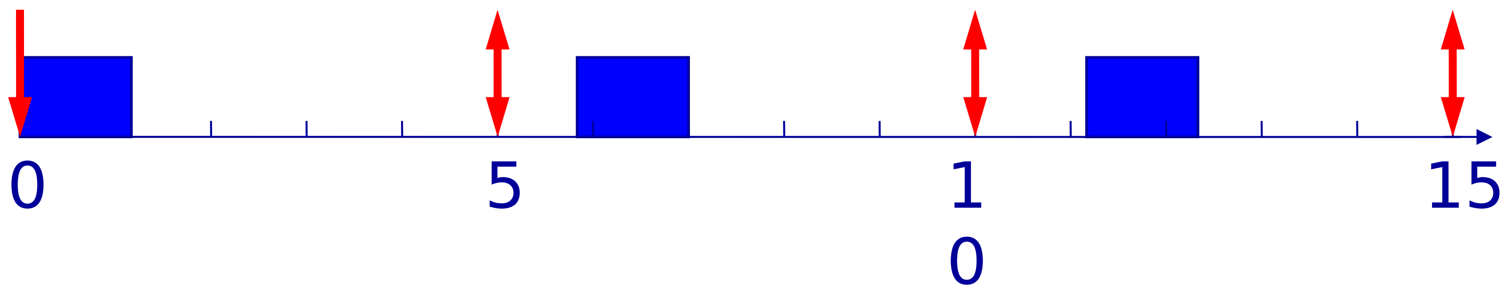
Real-Time Workload

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters



Real-Time Task

- Task : a sequence of similar jobs
  - Periodic task (*p*,*e*)
    - Its jobs repeat regularly
    - Period *p* = inter-release time (0 < *p*)
    - Execution time *e* = maximum execution time (0 < *e* < *p*)
    - Utilization U = e/p



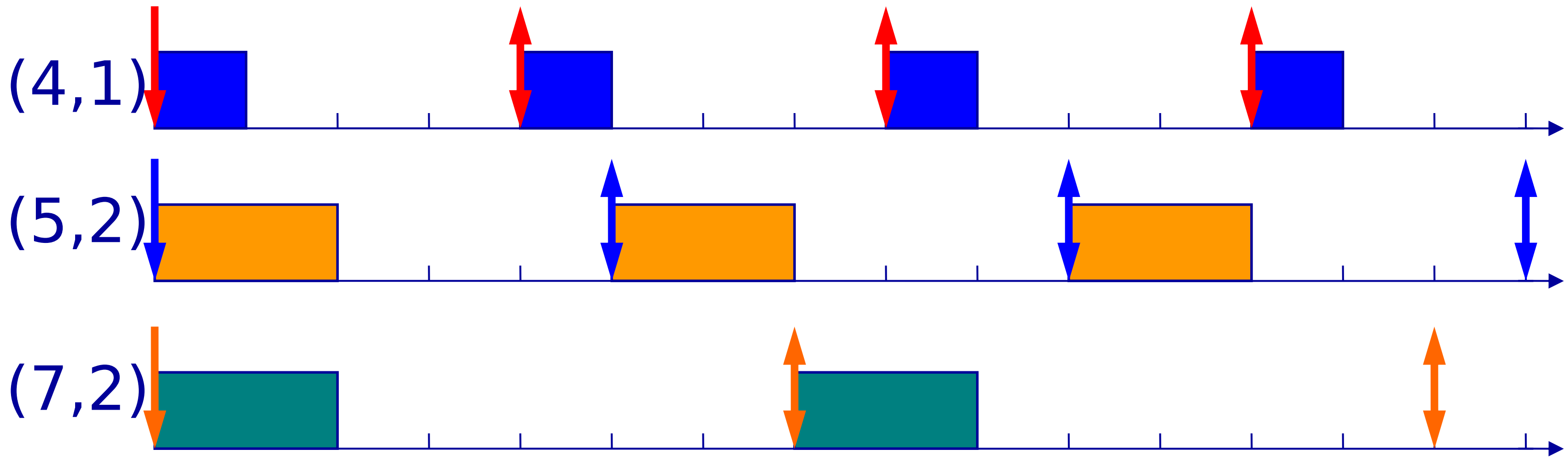
Deadlines: Hard vs. Soft

- Hard deadline
  - Disastrous or very serious consequences may occur if the deadline is missed
  - Validation is essential : can all the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees

- Soft deadline
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees

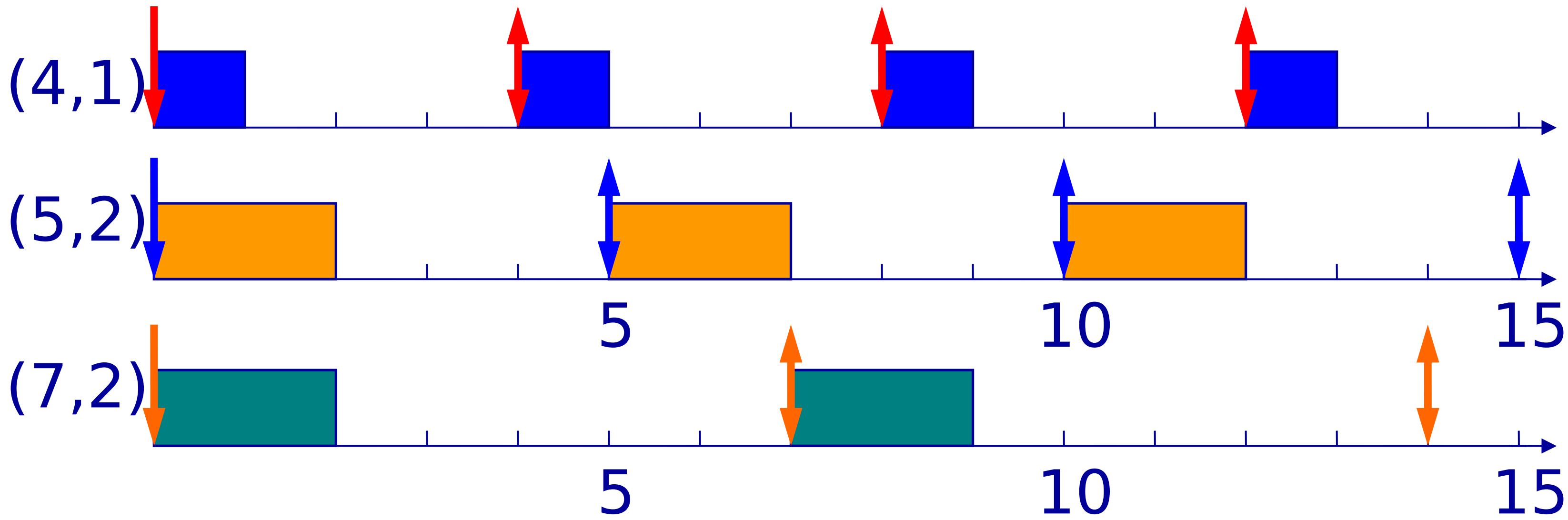
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines



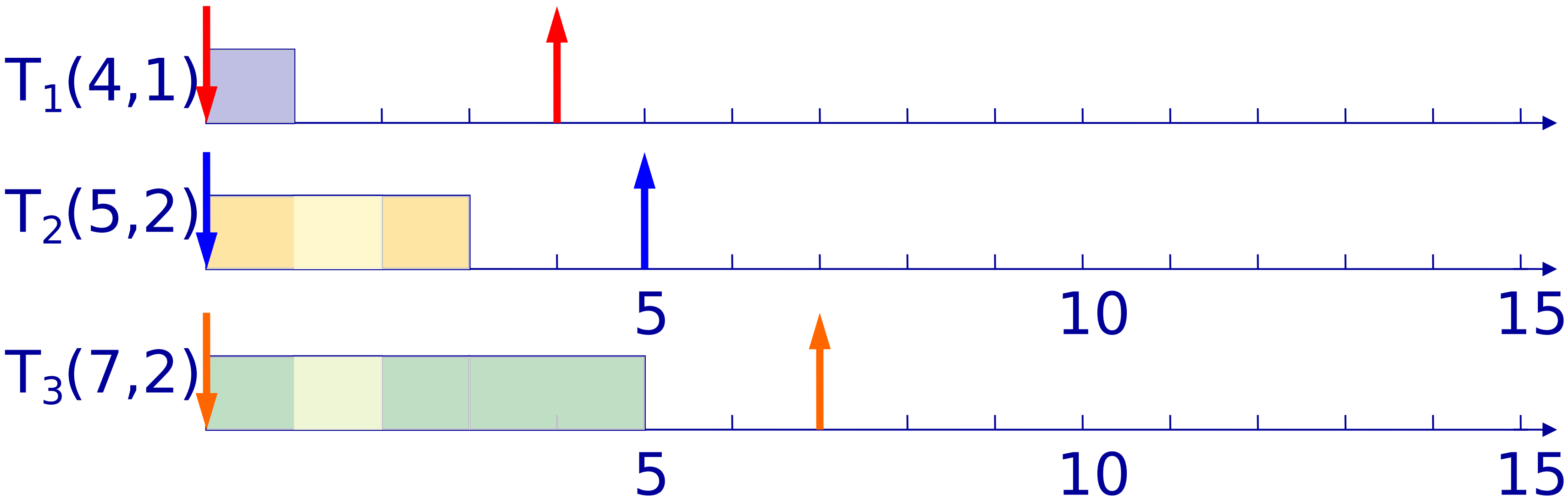
Real-Time Scheduling

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling



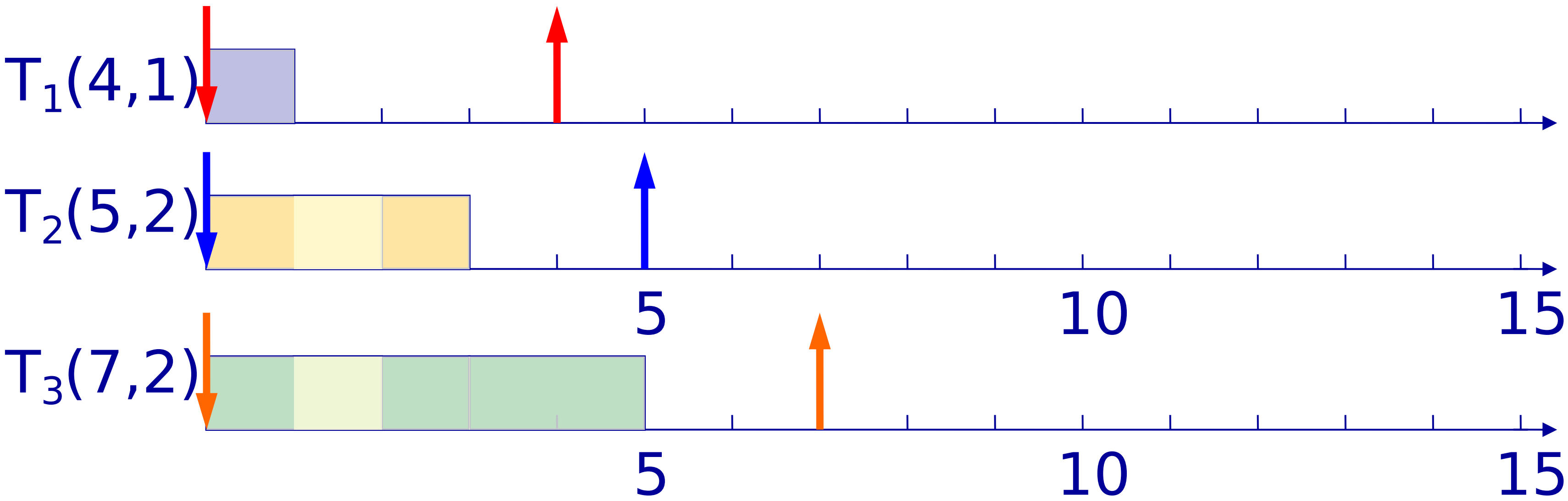
RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period



EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



RM vs. EDF

- Rate Monotonic
  - Simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines)
  - Predictability for the highest priority tasks
- EDF
  - Full processor utilization
  - Misbehavior during overload conditions
- For more details: Buttazzo, “Rate monotonic vs. EDF: Judgement Day”, EMSOFT 2003.

Outline

- Background
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
- Multiprocessor Scheduling
- Priority Inversion

Multiprocessor Scheduling

- CPU scheduling more complex for multiprocessors
  - Homogeneous processors within a multiprocessor
  - Benefit: load sharing
- Asymmetric multiprocessing – only one processor runs the kernel, others run user mode programs
  - Only one CPU accesses the system data structures, alleviating the need for data sharing
- Symmetric multiprocessing (SMP)
  - Each processor runs own scheduler
  - Need synchronization among schedulers
- Symmetric multithreading
  - Create multiple logical processors on the same physical processor (sounds like two threads)

Aim of Multiprocessor Scheduling

- Assignment of processes to processors
- Use of multiprogramming on individual processors
- Actual dispatching of a process

Assignment of Processes to Processors

- Treat processors as a pooled resource and assign process to processors on demand
- Permanently assign process to a processor
  - Known as group or gang scheduling
  - Dedicate short-term queue for each processor
  - Less overhead
  - Processor could be idle while another processor has a backlog

Outline

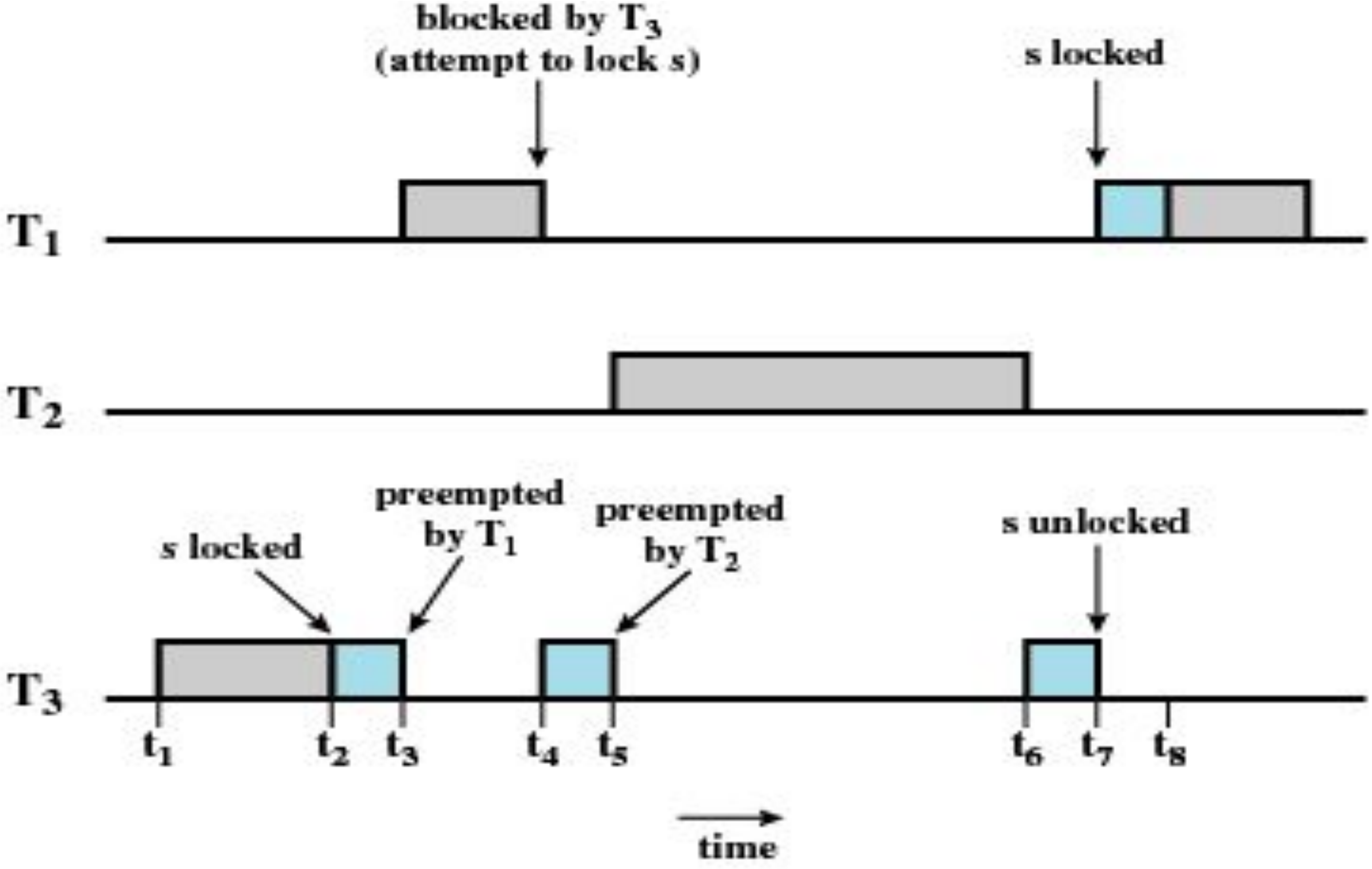
- Background
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
- Multiprocessor Scheduling
- Priority Inversion

Priority Inversion

- Can occur in any priority-based preemptive scheduling scheme
- Occurs when circumstances within the system force a higher priority task to wait for a lower priority task

Unbounded Priority Inversion

- Duration of a priority inversion depends on unpredictable actions of other unrelated tasks



(a) Unbounded priority inversion

Priority Inheritance

- Lower-priority task inherits the priority of any higher priority task pending on a resource they share

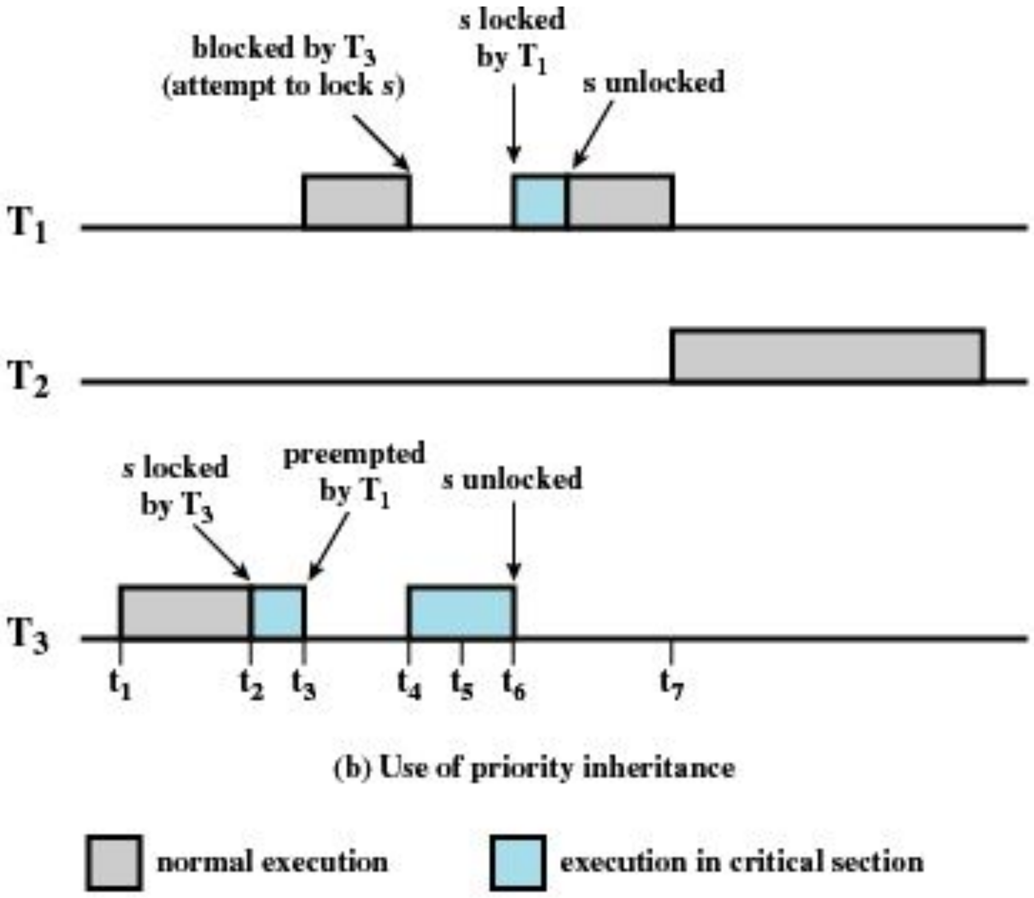


Figure 10.9 Priority Inversion

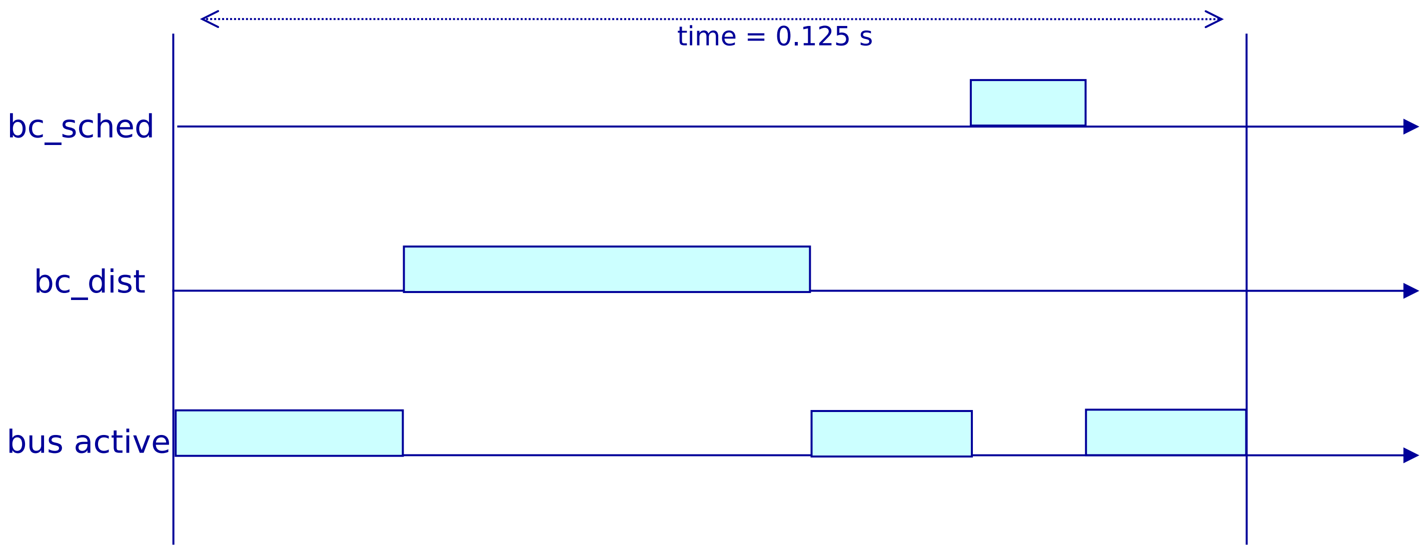
Priority Ceiling Protocol

- Priority Ceiling: a binary semaphore is the highest priority of all of the tasks that may lock it.
- A task attempting to a execute critical section is blocked unless its priority is higher than the priority ceilings of all of the locked semaphores in the system.
- The task holding the lock on the highest priority ceiling semaphore inherits the priorities of tasks blocked in this way.

What really happened on Mars?

(the first time)

- Two tasks were critical for controlling communication on the lander's communication bus, the scheduler task (bc_sched) and the distribution task (bc_dist).
- Each of these tasks checked each cycle to be sure that the other had run successfully.



Mars Pathfinder: The Problem

- bc_dist was blocked by a much lower priority meteorological science task (ASI/MET)
- ASI/MET was preempted by several medium priority processes such as accelerometers and radar altimeters.
- bc_sched started and discovered that bc_dist had not completed. Under these circumstances, bc_sched reacted by reinitializing the lander's hardware and software and terminating all ground command activities.

Mars Pathfinder: Resolution

- "Faster, better, cheaper" had NASA and JPL using "shrink-wrap" hardware (IBM RS6000) and software (Wind River vxWorks RTOS).
- Logging designed into vxWorks enabled NASA and Wind River to reproduce the failure on Earth. This reproduction made the priority inversion obvious.
- NASA patched the lander's software to enable priority inheritance.

Mini Lesson on System Design

1. Separate mechanism from policy
   - In this case: thread *mechanism* should allow context switch at any time, so we can use any policy we want
2. Know your goals
   - There must be trade-off of one goal against another
   - Explicitly write down your goals
3. Compare against optimal (even if you don't know how to build optimal for real system)
   - Provides reference to compare against (don't waste your time if you are already at 99% of optimal)
   - Provides insight used to understand other algorithms (under what circumstances will I not be optimal?)