Contents

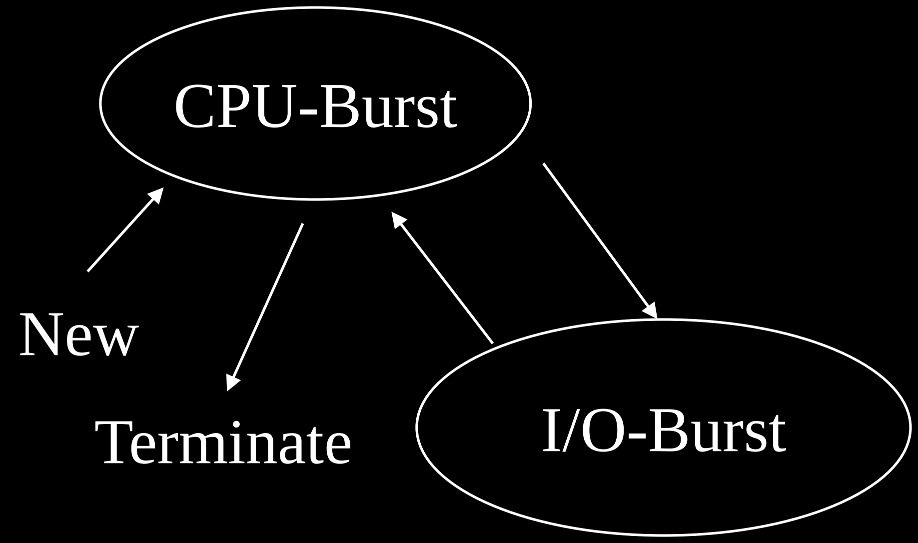
Chapter 5
Process Scheduling

CPU Scheduling

- Objective:
  - Basic Scheduling Concepts
  - CPU Scheduling Algorithms
- Why Multiprogramming?
  - Maximize CPU/Resources Utilization (Based on Some Criteria)

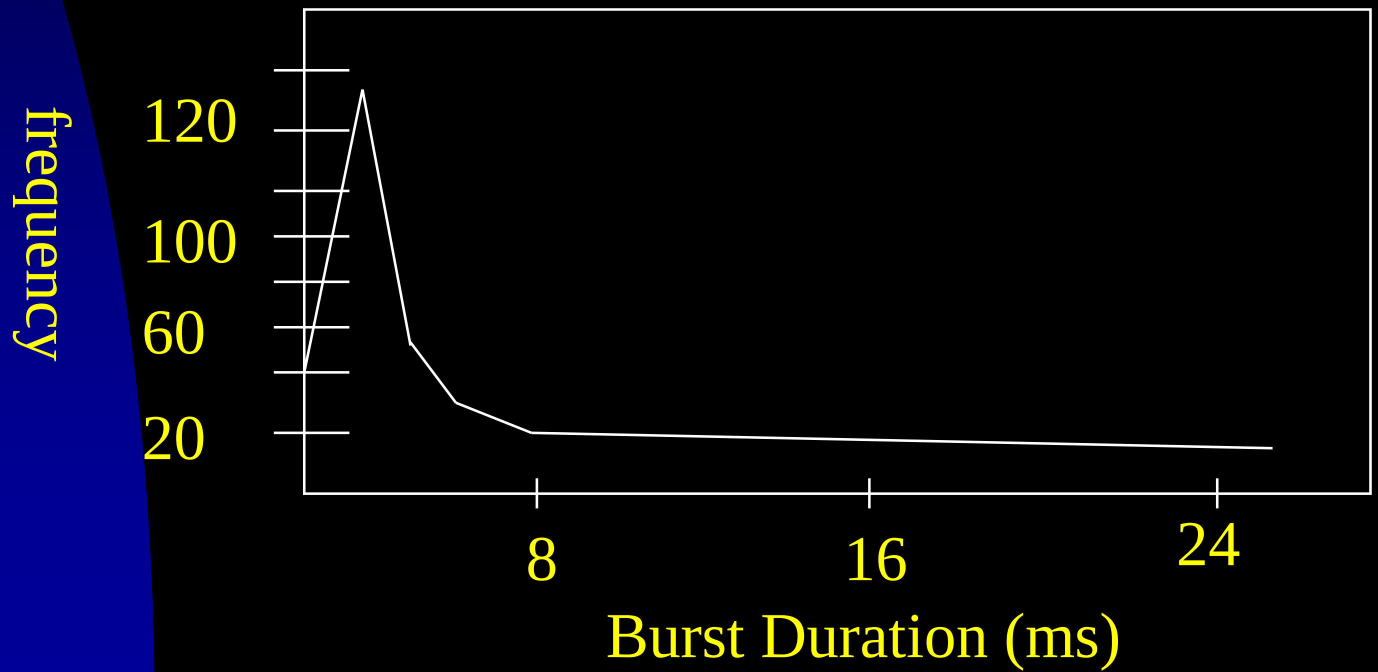
CPU Scheduling

- Process Execution
  - CPU-bound programs tend to have a few very long CPU bursts.
  - IO-bound programs tend to have many very short CPU bursts.



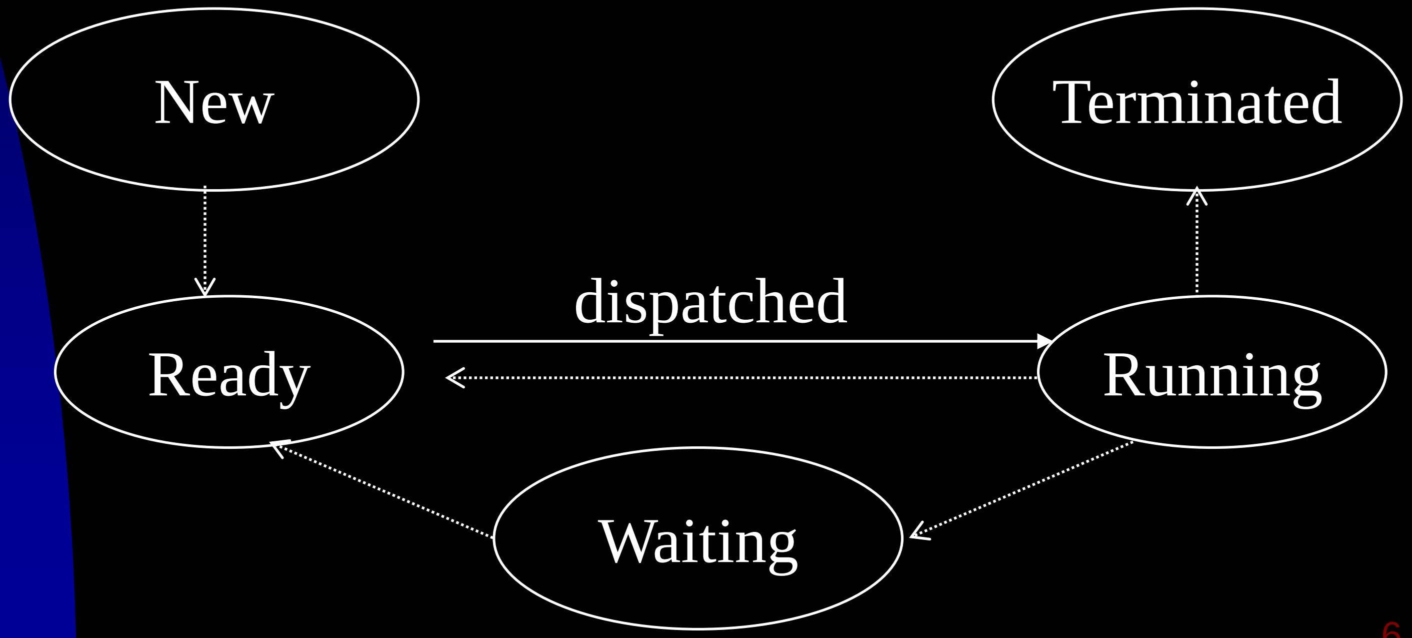
CPU Scheduling

- The distribution can help in selecting an appropriate CPU-scheduling algorithms



CPU Scheduling

- CPU Scheduler – The Selection of Process for Execution
  - A short-term scheduler



CPU Scheduling

- Nonpreemptive Scheduling
  - A running process keeps CPU until it volunteers to release CPU
    - E.g., I/O or termination
  - Advantage
    - Easy to implement (at the cost of service response to other processes)
  - E.g., Windows 3.1

CPU Scheduling

- Preemptive Scheduling
  - Beside the instances for non-preemptive scheduling, CPU scheduling occurs whenever some process becomes ready or the running process leaves the running state!
- Issues involved:
  - Protection of Resources, such as I/O queues or shared data, especially for multiprocessor or real-time systems.
  - Synchronization
    - E.g., Interrupts and System calls

CPU Scheduling

- Dispatcher
  - Functionality:
    - Switching context
    - Switching to user mode
    - Restarting a user program
  - Dispatch Latency:

Must be fast

Stop a process $\longleftrightarrow$ Start a process

Scheduling Criteria

- Why?
  - Different scheduling algorithms may favor one class of processes over another!
- Criteria
  - CPU Utilization
  - Throughput
  - Turnaround Time: CompletionT-StartT
  - Waiting Time: Waiting in the ReadyQ
  - Response Time: FirstResponseTime

Scheduling Criteria

- How to Measure the Performance of CPU Scheduling Algorithms?

- Optimization of what?
  - General Consideration
    - Average Measure
    - Minimum or Maximum Values
  - Variance → Predictable Behavior

Scheduling Algorithms

- First-Come, First-Served Scheduling (FIFO)
- Shortest-Job-First Scheduling (SJF)
- Priority Scheduling
- Round-Robin Scheduling (RR)
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling
- Multiple-Processor Scheduling

First-Come, First-Served Scheduling (FCFS)

- The process which requests the CPU first is allocated the CPU
- Properties:
  - Non-preemptive scheduling
  - CPU might be hold for an extended period.

First-Come, First-Served Scheduling (FCFS)

- Example

Process	CPU Burst Time
P1	24
P2	3
P3	3

Gantt Chart

P1	P2	P3
0 – 24	24 – 27	27 – 30

Average waiting time = (0+24+27)/3 = 17

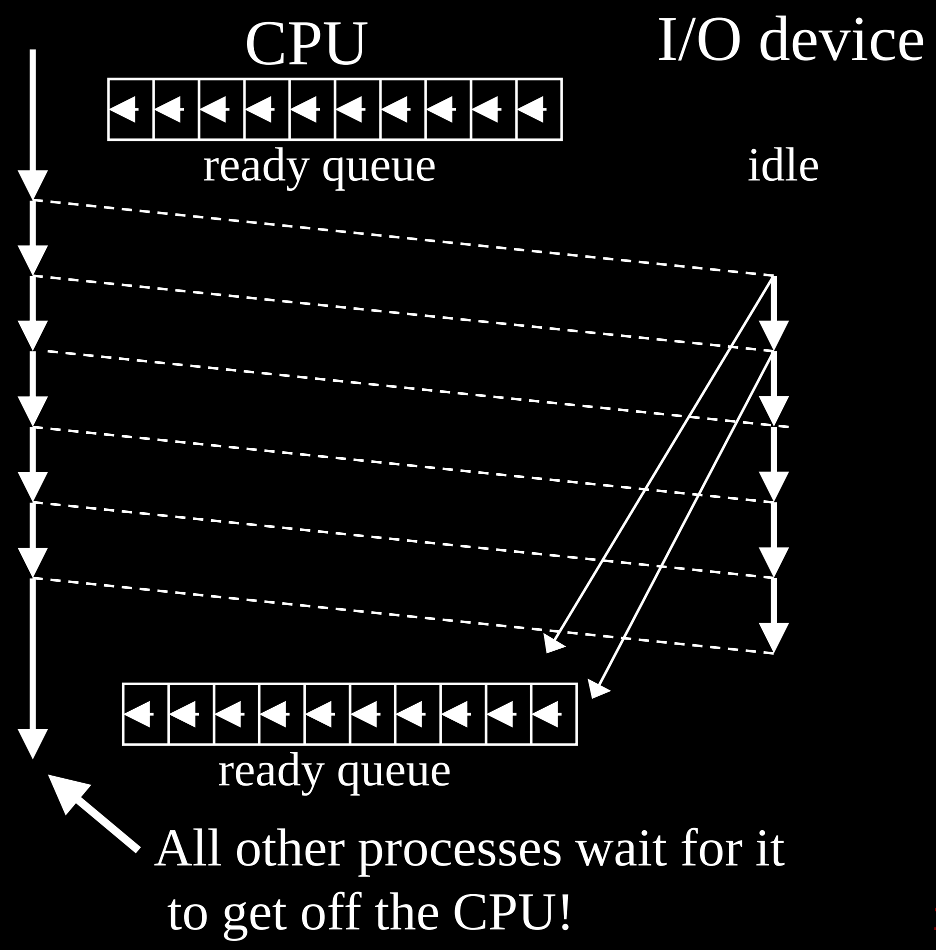
P2	P3	P1
0 – 3	3 – 6	6 – 30

Average waiting time = (6+0+3)/3 = 3

*The average waiting time is highly affected by process CPU burst times !

First-Come, First-Served Scheduling (FCFS)

- Example: Convoy Effect
  - One CPU-bound process + many I/O-bound processes



Shortest-Job-First Scheduling (SJF)

- Non-Preemptive SJF
  - Shortest next CPU burst first

process	CPU burst time
P1	6
P2	8
P3	7
P4	3

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

P4	P1	P3	P2

0 3 9 16 24

Shortest-Job-First Scheduling (SJF)

- Nonpreemptive SJF is optimal when processes are all ready at time 0
  - The minimum average waiting time!
- Prediction of the next CPU burst time?
  - Long-Term Scheduler
    - A specified amount at its submission time
  - Short-Term Scheduler
    - Exponential average ($0 <= \alpha <= 1$)

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

Shortest-Job-First Scheduling (SJF)

- Preemptive SJF
  - Shortest-remaining-time-first

Process	CPU Burst Time	Arrival Time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

P1	P2	P4	P1	P3

0 1 5 10 17 26

Average Waiting Time = ((10-1) + (1-1) + (17-2) + (5-3))/4 = 26/4 = 6.5

Shortest-Job-First Scheduling (SJF)

- Preemptive or Non-preemptive?
  - Criteria such as AWT (Average Waiting Time)

Non-preemptive AWT = (0+(10-1))/2 = 9/2 = 4.5

or

Preemptive AWT = ((2-1)+0) = 0.5

* Context switching cost ~ modeling & analysis

Priority Scheduling

- CPU is assigned to the process with the highest priority – A framework for various scheduling algorithms:
  - FCFS: Equal-Priority with Tie-Breaking by FCFS
  - SFJ: Priority = 1 / next CPU burst length

Priority Scheduling

Process	CPU Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Gantt Graph

Average waiting time
= (6+0+16+18+1)/5 = 8.2

P2	P5	P1	P3	P4

0 1 6 16 18 19

Priority Scheduling

- Priority Assignment
  - Internally defined – use some measurable quantity, such as the # of open files, $\frac{\text{Average CPU Burst}}{\text{Average I/O Burst}}$
  - Externally defined – set by criteria external to the OS, such as the criticality levels of jobs.

Priority Scheduling

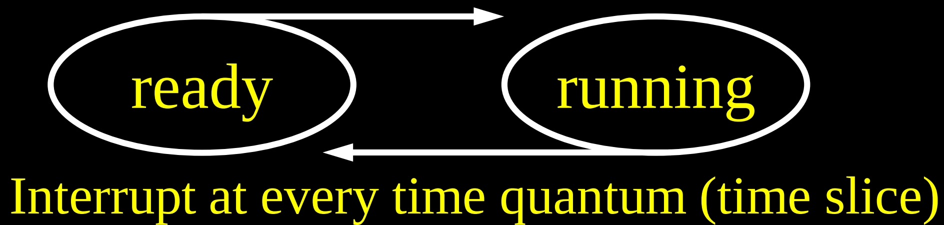
- Preemptive or Non-Preemptive?
  - Preemptive scheduling – CPU scheduling is invoked whenever a process arrives at the ready queue, or the running process relinquishes the CPU.
  - Non-preemptive scheduling – CPU scheduling is invoked only when the running process relinquishes the CPU.

Priority Scheduling

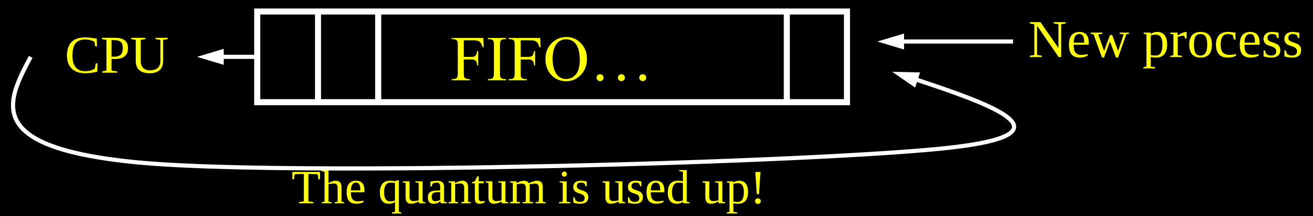
- Major Problem
  - Indefinite Blocking (/Starvation)
    - Low-priority processes could starve to death!
  - A Solution: Aging
    - A technique that increases the priority of processes waiting in the system for a long time.

Round-Robin Scheduling (RR)

- RR is similar to FCFS except that preemption is added to switch between processes.



- Goal: Fairness – Time Sharing



Round-Robin Scheduling (RR)

Process	CPU Burst Time
P1	24
P2	3
P3	3

Time slice = 4

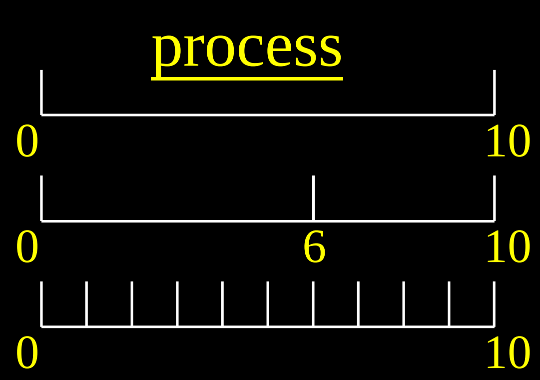
P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22 26 30

AWT = ((10-4) + (4-0) + (7-0))/3
= 17/3 = 5.66

Round-Robin Scheduling (RR)

- Service Size and Interval
  - Time quantum = q → Service interval <= (n-1)*q if n processes are ready.
  - IF q = ∞, then RR → FCFS.
  - IF q = ε, then RR → processor sharing. The # of context switchings increases!



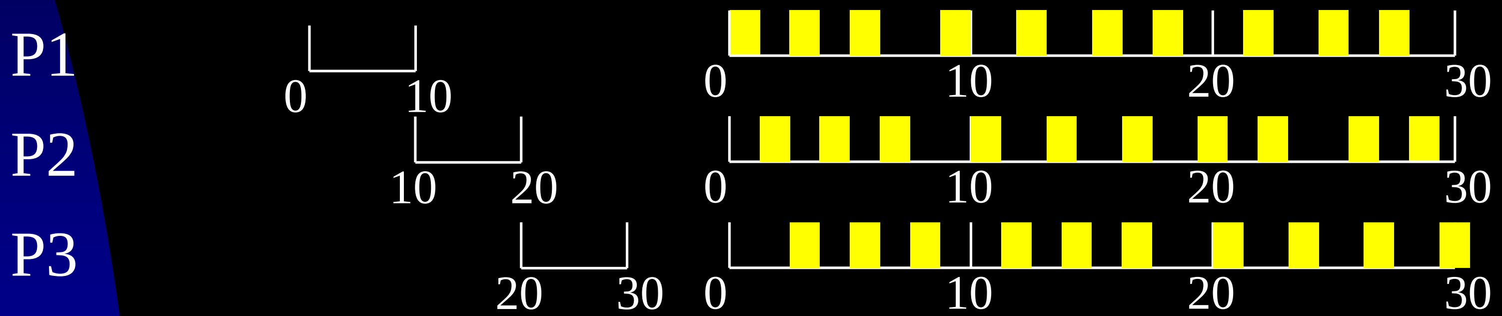
quantum	context switch #
12	0
6	1
1	9

$$\frac{\text{If context switch cost}}{\text{time quantum}} = 10\% \quad \Rightarrow \quad 1/11 \text{ of CPU is wasted!}$$

Round-Robin Scheduling (RR)

- Turnaround Time

process (10ms) quantum = 10 quantum = 1



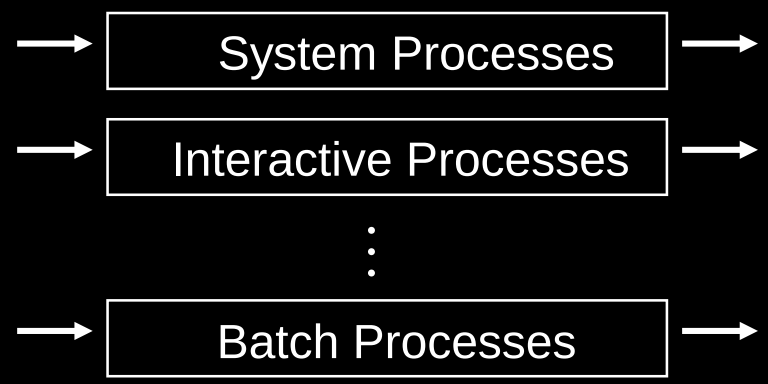
Average Turnaround Time
= (10+20+30)/3 = 20

ATT = (28+29+30)/3 = 29

=> 80% CPU Burst < time slice

Multilevel Queue Scheduling

- Partition the ready queue into several separate queues => Processes can be classified into different groups and permanently assigned to one queue.



Multilevel Queue Scheduling

- Intra-queue scheduling
  - Independent choice of scheduling algorithms.
- Inter-queue scheduling
  - Fixed-priority preemptive scheduling
    - e.g., foreground queues always have absolute priority over the background queues.
  - Time slice between queues
    - e.g., 80% CPU is given to foreground processes, and 20% CPU to background processes.
  - More??

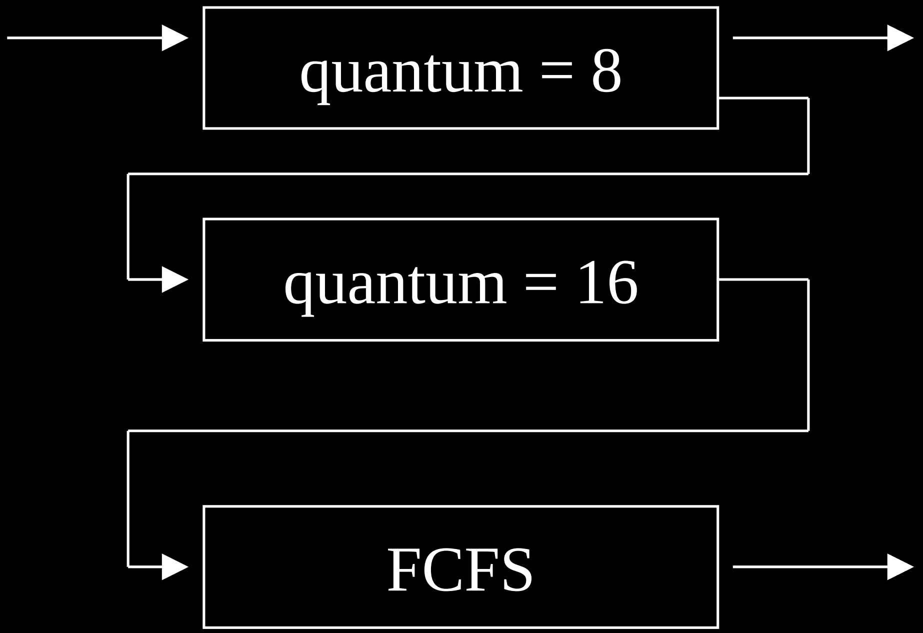
Multilevel Feedback Queue Scheduling

- Different from Multilevel Queue Scheduling by Allowing Processes to Migrate Among Queues.
  - Configurable Parameters:
    - # of queues
    - The scheduling algorithm for each queue
    - The method to determine when to upgrade a process to a higher priority queue.
    - The method to determine when to demote a process to a lower priority queue.
    - The method to determine which queue a newly ready process will enter.

*Inter-queue scheduling: Fixed-priority preemptive?!

Multilevel Feedback Queue Scheduling

- Example



*Idea: Separate processes with different CPU-burst characteristics!

Thread Scheduling

- Two Scopes:
  - Process Contention Scope (PCS): m:1 or m:m
    - Priority-Driven
  - System-Contention Scope (SCS): 1:1
- Pthread Scheduling
  - PCS and SCS

Pthread_attr_setscope(pthread_attr_t *attr, int scope)

Pthread_attr_getscope(pthread_attr_t *attr, int *scope)

Multiple-Processor Scheduling

- CPU scheduling in a system with multiple CPUs
- A Homogeneous System
  - Processes are identical in terms of their functionality.
    - → Can processes run on any processor?
- A Heterogeneous System
  - Programs must be compiled for instructions on proper processors.

Multiple-Processor Scheduling

- Load Sharing – Load Balancing!!
  - A queue for each processor
    - Self-Scheduling – Symmetric Multiprocessing
  - A common ready queue for all processors.
    - Self-Scheduling
      - Need synchronization to access common data structure, e.g., queues.
    - Master-Slave – Asymmetric Multiprocessing
      - One processor accesses the system structures → no need for data sharing

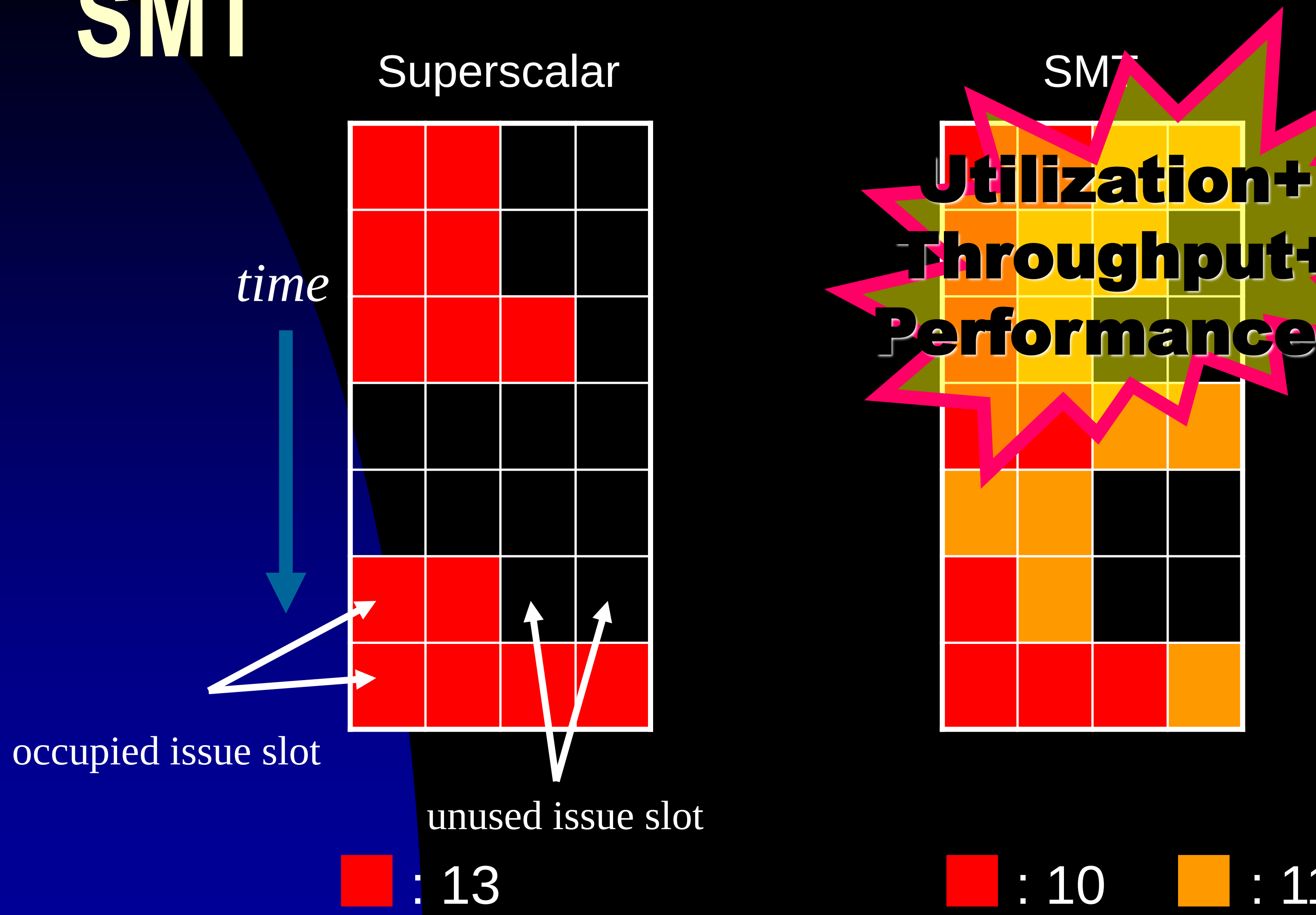
Multiple-Processor Scheduling

- Load Balancing
  - Push migration: A specific task periodically checks for imbalance and migrate tasks
  - Pull migration: An idle processor pulls a waiting task from a busy processor
  - Processor affinity vs imbalance threshold
  - Linux and FreeBSD do both!
- Processor Affinity
  - The system might avoid process migration because of the cost in invalidating or re-populating caches
  - Soft or hard affinity

Multiple-Processor Scheduling

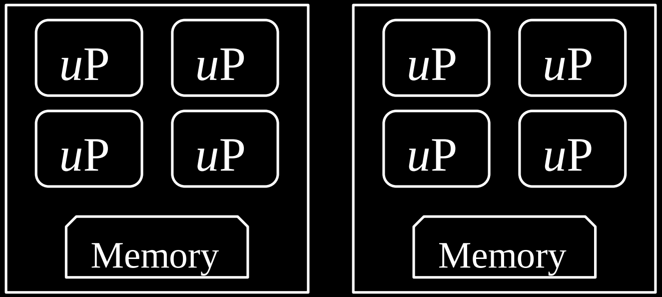
- Symmetric Multithreading (SMT), i.e., Hyperthreading
  - A feature provided by the hardware
  - Several logical processors per physical processor
    - Each has its own architecture state, including registers.
  - Issues: Process Synchronization

Multiple-Processor Scheduling – SMT

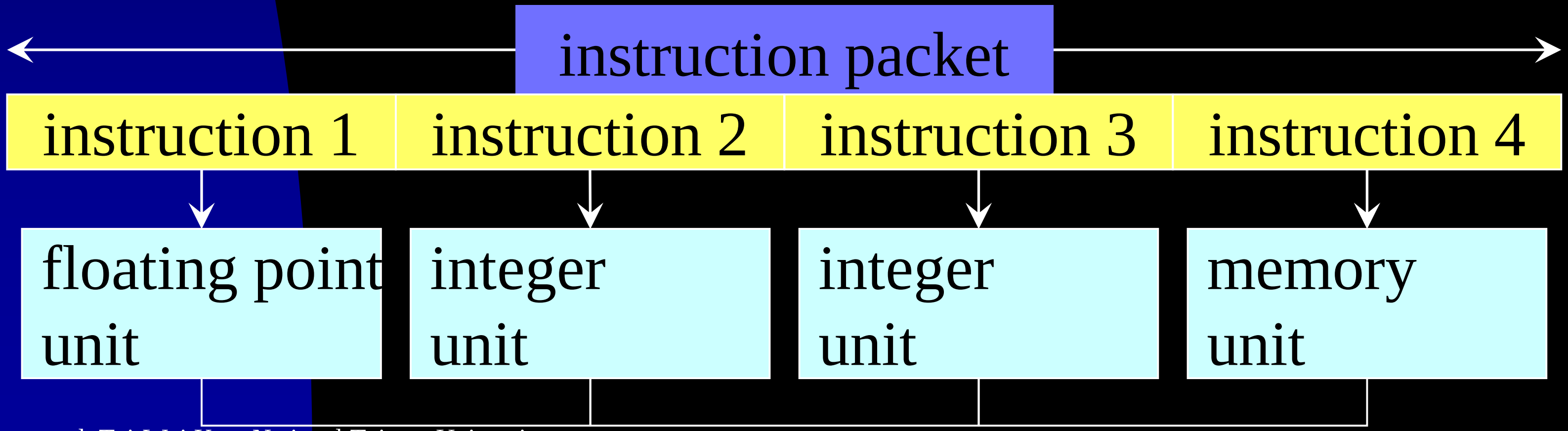


Multiple-Processor Scheduling

- Non-Uniform Memory Access (NUMA)



- Very Long Instruction Word (VLIW)
  - High parallelism and simple architecture



Multicore Processors

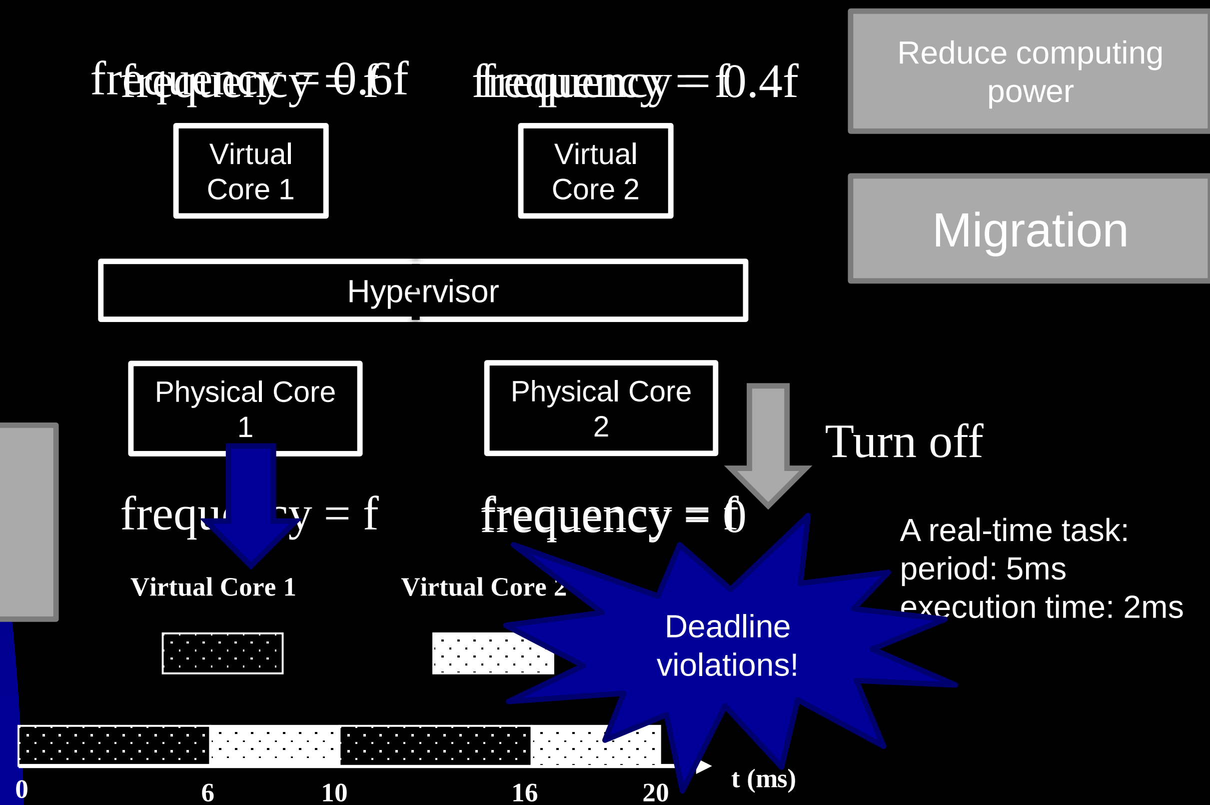
- Multicore Processor: A physical chip with multiple processor cores.
- Scheduling Issues:
  - Memory Stall → Multiple Hardware Threads
    - Coarse-Grained Multithreading
      - Thread execution until a long latency
    - Fine-Grained Multithreading
      - Better architecture design for switching
  - Two-Levels of Scheduling
    - OS chooses to run a software thread.
    - Each core decides to run which hardware thread – round robin (UltraSPARC T1) or dynamic urgency (Itanium)

Virtualization and Scheduling

- The virtualization software presents one or more virtual CPUs to each of the virtual machines running on the system and then schedules the use of the physical CPUs among the machines.
- Potential Scheduling Issues
  - The assumption of a certain amount of progress in a given amount of time has negative impacts by virtualization.
  - Interrupts, such as timer interrupts, could take longer time to receive their attention, compared to dedicated processors

Virtualization and Scheduling

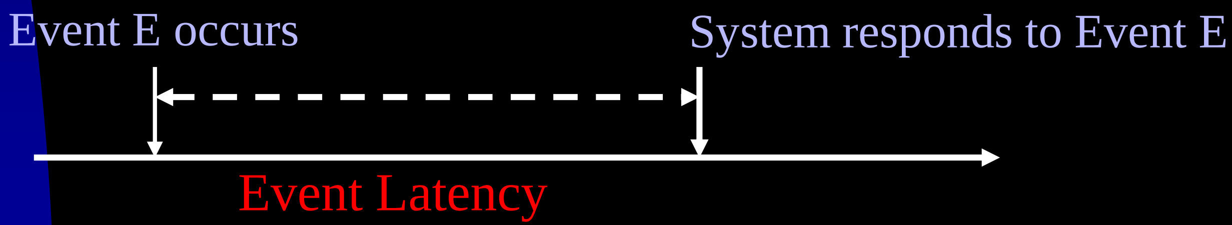
- The progress assumption of a system



Two virtual cores share a physical core

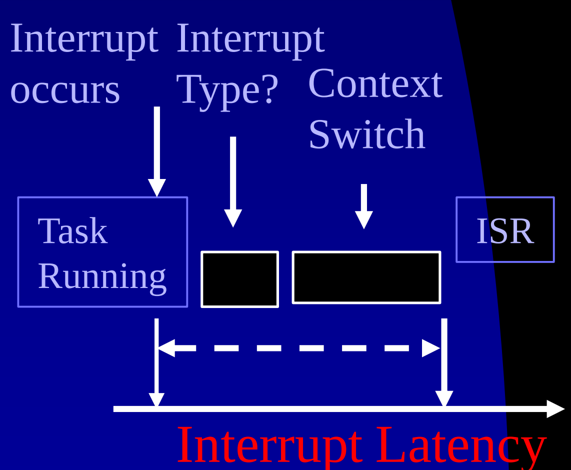
Real-Time CPU Scheduling

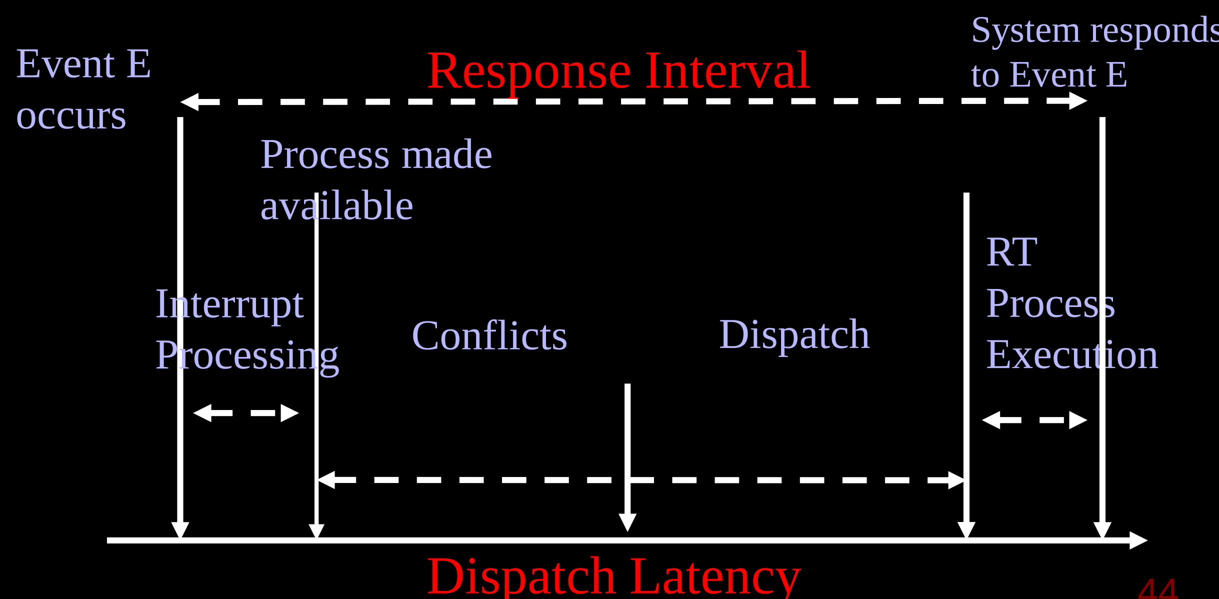
- Definitions
  - Soft Real-Time Systems: Scheduling preference is given to critical processes.
  - Hard Real-time Systems: There is a need to guarantee for deadline satisfaction.
- Different latency requirements to event types:



Real-Time CPU Scheduling

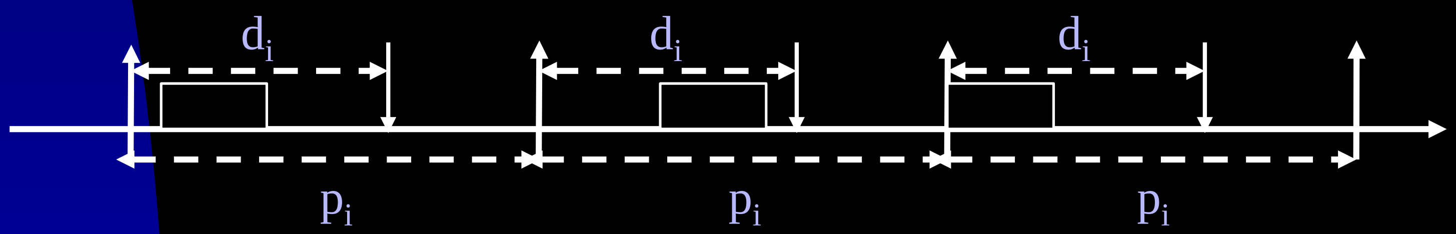
- Types of Latency:
  - Interrupt Latency
  - Dispatch Latency
    - Conflicts: Preemption + Resource Releasing





Real-Time CPU Scheduling

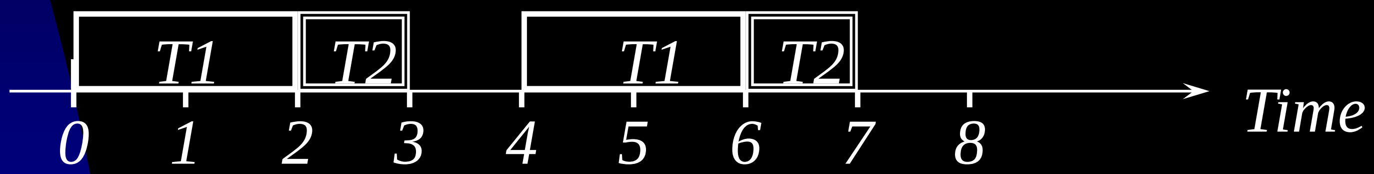
- Priority-Based Preemptive Scheduler
  - It might be ok for soft real-time processes.
  - Hard real-time systems need a guarantee!
- Process Model
  - Periodic Process τ_i: Period p_i, deadline d_i, and required CPU time c_i.
    - Rate = 1/ p_i



- Admission Control vs Deadline-Requirement Announcement

Rate Monotonic Scheduling

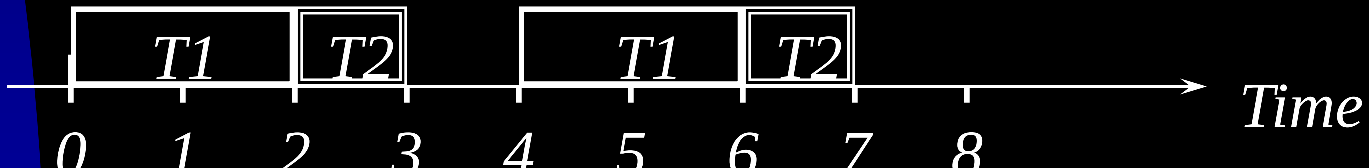
- Fixed Priority Assignment
  - RM priority assignment: priority ~ 1/period.
  - preemptive priority-driven scheduling of periodic processes
- Example: T1 (p1=4, c1=2) and T2 (p2=5, c2=1)



- Properties
  - Optimal Fixed-Priority Scheduler for Independent Processes
  - Achievable Utilization Factor: $N(2^{1/N}-1)$ for N processes, where the utilization of $\tau_i = c_i / p_i$

Rate Monotonic Scheduling

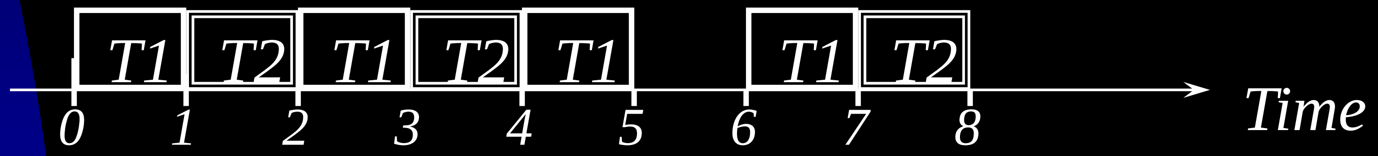
- Properties
  - Critical Instant
    - An instance at which the process is requested simultaneously with requests of all higher priority processes .
  - Fully Utilization of the Processor Power
    - Example: T1 (p1=4, c1=2) and T2 (p2=5, c2=1→2)



[1] *Liu and Layland, "Scheduling Algorithms for multiprogramming in a hard real-time Environment," JACM, vol. 20, no. 1, January 1973, pp. 46-61.*

Earliest Deadline First Scheduling

- Dynamic Priority Assignment
  - EDF priority assignment: priority ~ absolute deadline, i.e., d = arrival time t + relative deadline d_i.
  - preemptive priority-driven scheduling of <u>periodic or aperiodic processes</u>
- Example: T1(c1=1, p1=2), T2(c2=2, p2=7)



- Properties
  - Optimal Scheduler for Independent Processes
  - Achievable Utilization Factor: 100%

Proportional Share Scheduling

- Proportional Share
  - Equal or Proportional Share S_i
  - Admission Control
- Example Real-Time CPU Scheduling
  - Total Bandwidth Server, Constant Utilization Server, Sporadic Server, Deferreable Server

POSIX Real-Time Scheduling

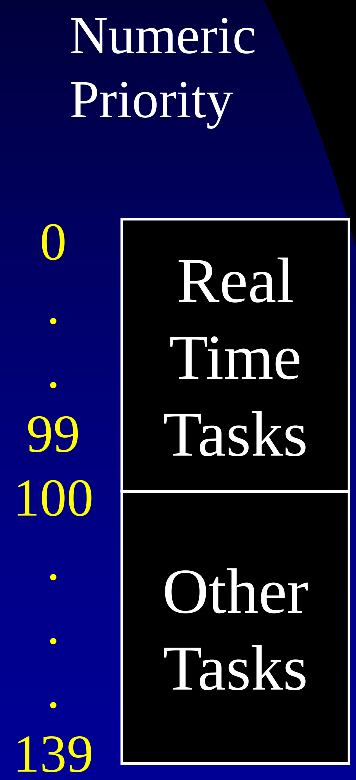
- POSIX.1b – Extensions for Real-Time Computing
  - SCHED_FIFO
  - SCHED_RR
  - SCHED_OTHER
- API

```
pthread_attr_getsched_policy(pthread_attr_t *attr, int *policy)
pthread_attr_setsched_policy(pthread_attr_t *attr, int *policy)
```

Operating System Examples

- Process Local Scheduling
  - E.g., those for user-level threads
  - Thread scheduling is done locally to each application.
- System Global Scheduling
  - E.g., those for kernel-level threads
  - The kernel decides which thread to run.

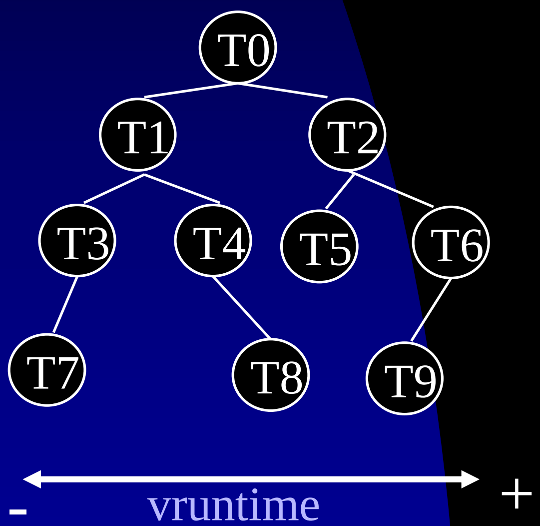
Operating System Examples – Linux Ver. 2.5+



- Scheduling Algorithm
  - O(1)
  - SMP, load balancing, and processor affinity
  - Completely Fair Scheduler (CFS) for V2.6+
    - Priorities 100~139
      - Nice: -20..+19
    - Targeted Latency – An interval to run the task at least once
      - Proportions of the CPU time are allocated from the value

Operating System Examples – Linux Ver. 2.5+

A Red-Black Tree



- Completely Fair Scheduler (CFS) for V2.6+
  - vruntime: How long the task has run.
    - Association with a decay factor based on the task priority, i.e., the vruntime of a higher-priority task being lower than its physical run time
- Real-Time Scheduling Class
  - Adopt POSIX.1b
  - Priorities 1~99

Operating System Examples – Windows

- Priority-Based Preemptive Scheduling
  - Priority Class/Relationship: 0..31
  - Dispatcher: A process runs until
    - It is preempted by a higher-priority process.
    - It terminates
    - Its time quantum ends
    - It calls a blocking system call
  - Idle thread
- A queue per priority level
- Windows 7 introduces user-mode scheduling
  - Thread scheduling without kernel intervention

Operating System Examples – Windows

- Each thread has a base priority that represents a value in the priority range of its class.
- A typical class – Normal_Priority_Class
- Time quantum – thread
  - Increased after some waiting
    - Different for I/O devices.
  - Decreased after some computation
    - The priority is never lowered below the base priority.
  - Favor foreground processes (more time quantum)

Operating System Examples – Windows

A Typical Class

Base Priority

	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

Real-Time Class

Variable Class (1..15)

Operating System Examples – Solaris

- Priority-Based Process Scheduling
  - Real-Time (100-159)
  - System (60-99)
    - Kernel-service processes
  - Time-Sharing (0-59)
    - A default class
  - Interactive (0-59), Fair Share (0-59), Fixed Priority (0-59)
- Each LWP inherits its class from its parent process

low

* Two new classes for Solaris 9: Fair Share and Fixed Priority.

Operating System Examples – Solaris

- Real-Time
  - A guaranteed response
- System
  - The priorities of system processes are fixed.
- Time-Sharing
  - Multilevel feedback queue scheduling – priorities inversely proportional to time slices
- Interactive
  - Prefer windowing process

Operating System Examples – Solaris

- Two New Classes in Solaris 9
  - Fixed Priority (0-59)
    - Non-adjusted priorities in the range of the time-sharing class
  - Fair Sharing (0-59)
    - CPU shares, instead of priorities
- 10 kernel threads are reserved to service interrupts at the priority range 160-169.

Operating System Examples – Solaris

Interactive and time sharing threads

priority	Time quantum	Time quantum exp.	Return from sleep
0 low	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59 high	20	49	59

Operating System Examples – Solaris

- The selected thread runs until one of the following occurs:
  - It blocks.
  - It uses its time slice (if it is not a system thread).
  - It is preempted by a higher-priority thread.
- RR is used when several threads have the same priority.

Algorithm Evaluation

- A General Procedure
  - Select criteria that may include several measures, e.g., maximize CPU utilization while confining the maximum response time to 1 second
  - Evaluate various algorithms
- Evaluation Methods:
  - Deterministic modeling
  - Queuing models
  - Simulation
  - Implementation

Deterministic Modeling

- A Typical Type of Analytic Evaluation
  - Take a particular predetermined workload and defines the performance of each algorithm for that workload
- Properties
  - Simple and fast
  - Through excessive executions of a number of examples, trends might be identified
  - But it needs exact numbers for inputs, and its answers only apply to those cases
    - Being too specific and requires too exact knowledge to be useful!

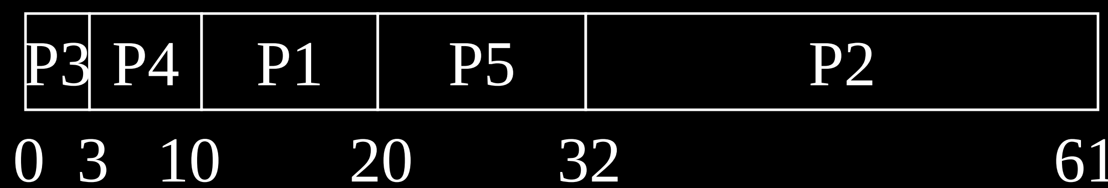
Deterministic Modeling

process	CPU Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

FCFS

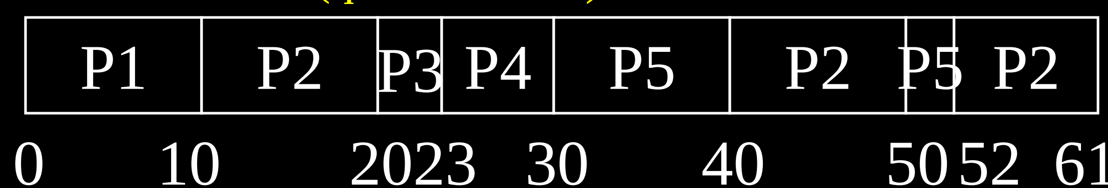
Average Waiting Time (AWT)=(0+10+39+42+49)/5=28

Nonpreemptive Shortest Job First



AWT=(10+32+0+3+20)/5=13

Round Robin (quantum =10)



AWT=(0+(10+20+2)+20+23+(30+10))/5=23

Queueing Models

- Motivation:
  - Workloads vary, and there is no static set of processes
- Models (~ Queueing-Network Analysis)
  - Workload:
    - Arrival rate: the distribution of times when processes arrive.
    - The distributions of CPU & I/O bursts
  - Service rate

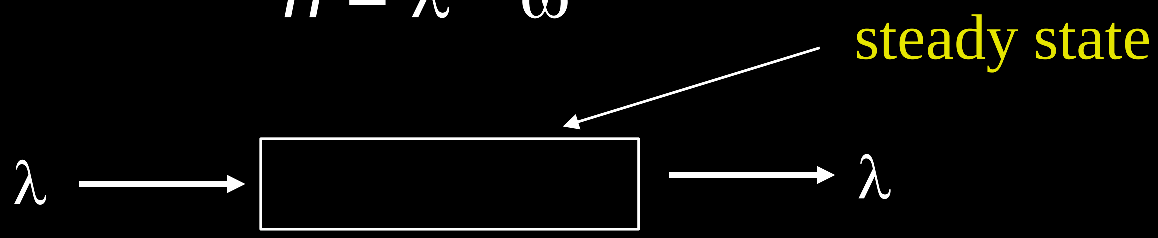
Queueing Models

- Model a computer system as a network of servers. Each server has a queue of waiting processes
  - Compute average queue length, waiting time, and so on.
- Properties:
  - Generally useful but with limited application to the classes of algorithms & distributions
  - Assumptions are made to make problems solvable => inaccurate results

Queueing Models

- Example: Little's formula

$$n = \lambda * \omega$$



n = # of processes in the queue

λ = arrival rate

ω = average waiting time in the queue

  - If n =14 & λ =7 processes/sec, then w = 2 seconds.

Simulation

- Motivation:
  - Get a more accurate evaluation.
- Procedures:
  - Program a model of the computer system
  - Drive the simulation with various data sets
    - Randomly generated according to some probability distributions

      => inaccuracy occurs because of only the occurrence frequency of events. Miss the order & the relationships of events.
    - Trace tapes: monitor the real system & record the sequence of actual events.

Simulation

- Properties:
  - Accurate results can be gotten, but it could be expensive in terms of computation time and storage space.
  - The coding, design, and debugging of a simulator can be a big job.

Implementation

- Motivation:
  - Get more accurate results than a simulation!
- Procedure:
  - Code scheduling algorithms
  - Put them in the OS
  - Evaluate the real behaviors

Implementation

- Difficulties:
  - Cost in coding algorithms and modifying the OS
  - Reaction of users to a constantly changing the OS
  - The environment in which algorithms are used will change
    - For example, users may adjust their behaviors according to the selected algorithms

  => Separation of the policy and mechanism!