Cyber-Physical Systems

Scheduling

UNIVERSITY AT ALBANY
State University of New York

IECE 553/453– Fall 2019

Prof. Dola Saha

Scheduler

- A scheduler makes the decision about what to do next at certain points in time
- When a processor becomes available, which process will be executed

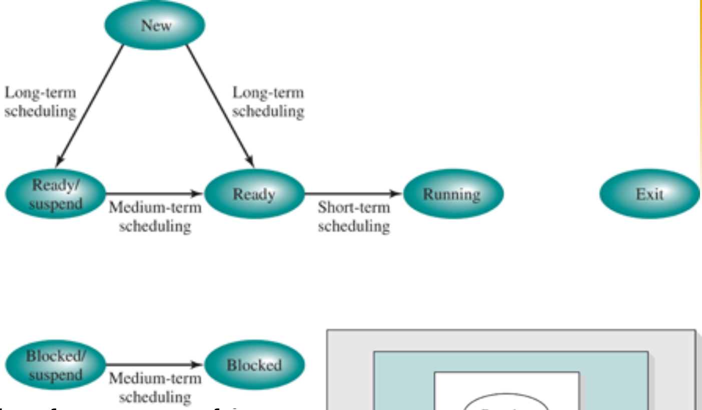
Scheduler Policy

- Different schedulers will have different goals
  - Maximize throughput
  - Minimize latency
  - Prevent indefinite postponement
  - Complete process by given deadline
  - Maximize processor utilization

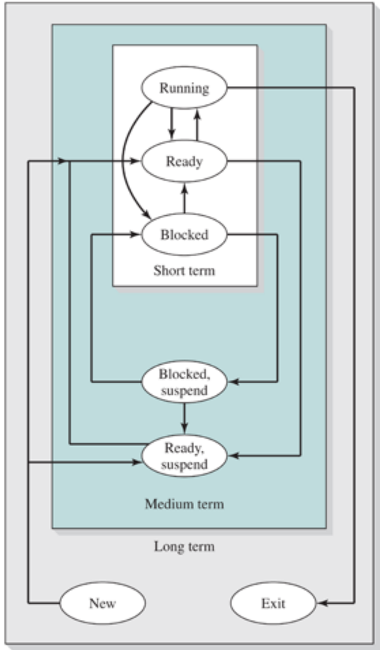
Scheduler Levels

- High-level scheduling
  - Determines which jobs can compete for resources
  - Controls number of processes in system at one time
- Intermediate-level scheduling
  - Determines which processes can compete for processors
  - Responds to fluctuations in system load
- Low-level scheduling
  - Assigns priorities
  - Assigns processors to processes

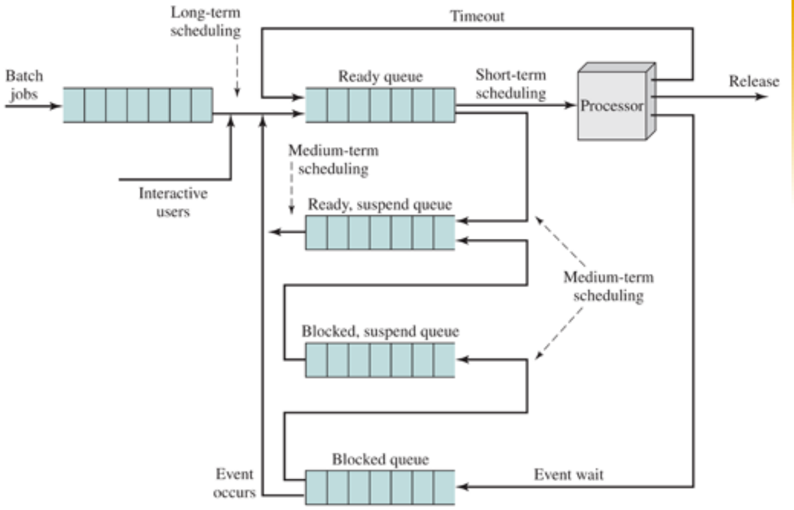
Types of Processor Scheduling



- Long-term scheduling
  - when a new process is created
  - adds the new process to the set of processes that are active
- Medium-term scheduling
  - swapping function, adds a process to those that are at least partially in main memory and therefore available for execution
- Short-term scheduling
  - actual decision of which ready process to execute next.



Queuing Diagram



- Long Term (Infrequently)
  - Controls degree of multiprogramming
- Medium Term
  - swapping-in decision will consider the memory requirements of the swapped-out processes
- Short Term (Frequently)
  - Clock interrupts, I/O interrupts, Operating system calls, Signals (e.g., semaphores)

Priorities

- Static priorities
  - Priority assigned to a process does not change
  - Easy to implement
  - Low overhead
  - Not responsive to changes in environment
- Dynamic priorities
  - Responsive to change
  - Promote smooth interactivity
  - Incur more overhead, justified by increased responsiveness

How to decide which thread to schedule?

- Considerations:
  - Preemptive vs. non-preemptive scheduling
  - Periodic vs. aperiodic tasks
  - Fixed priority vs. dynamic priority
  - Priority inversion anomalies
  - Other scheduling anomalies

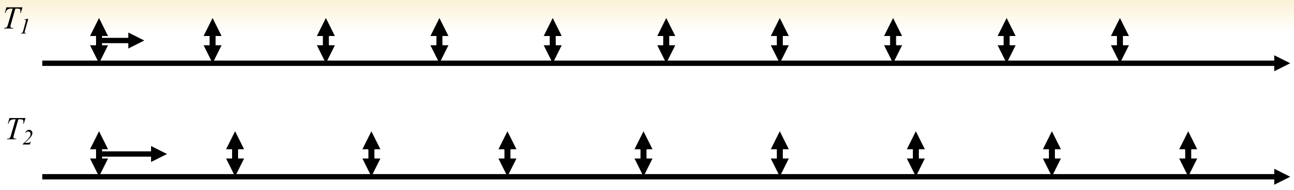
Non-Preemptive vs Preemptive

- Non-Preemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to ready state by the OS
  - Decision to preempt may be performed
    - when a new process arrives,
    - when an interrupt occurs that places a blocked process in the Ready state, or
    - periodically, based on a clock interrupt

Preemptive Scheduling

- Assume all threads have priorities
  - either statically assigned (constant for the duration of the thread)
  - or dynamically assigned (can vary).
- Assume that the kernel keeps track of which threads are “enabled”
- Preemptive scheduling:
  - At any instant, the enabled thread with the highest priority is executing.
  - Whenever any thread changes priority or enabled status, the kernel can dispatch a new thread.

Periodic scheduling



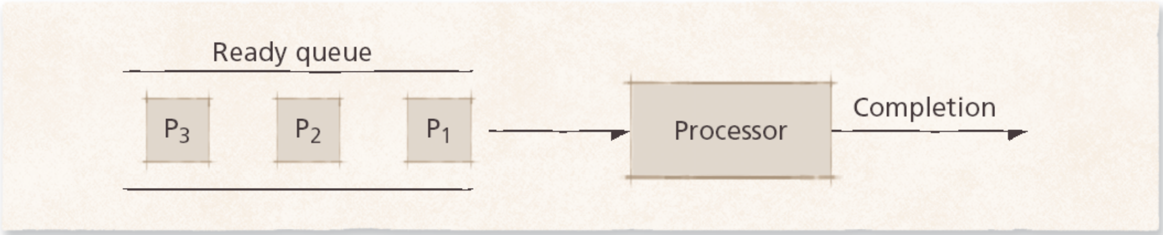
- Each execution instance of a task is called a job.
- For periodic scheduling, the best that we can do is to design an algorithm which will always find a schedule if one exists.
- A scheduler is defined to be optimal iff it will find a schedule if one exists.

Scheduling Policies

- First Come First Serve
- Round Robin
- Shortest Process Next
- Shortest Remaining Time Next
- Highest Response Ratio Next
- Feedback Scheduler
- Fair Share Scheduler

First Come First Serve (FCFS)

- Processes dispatched according to arrival time
- Simplest scheme
- Nonpreemptible

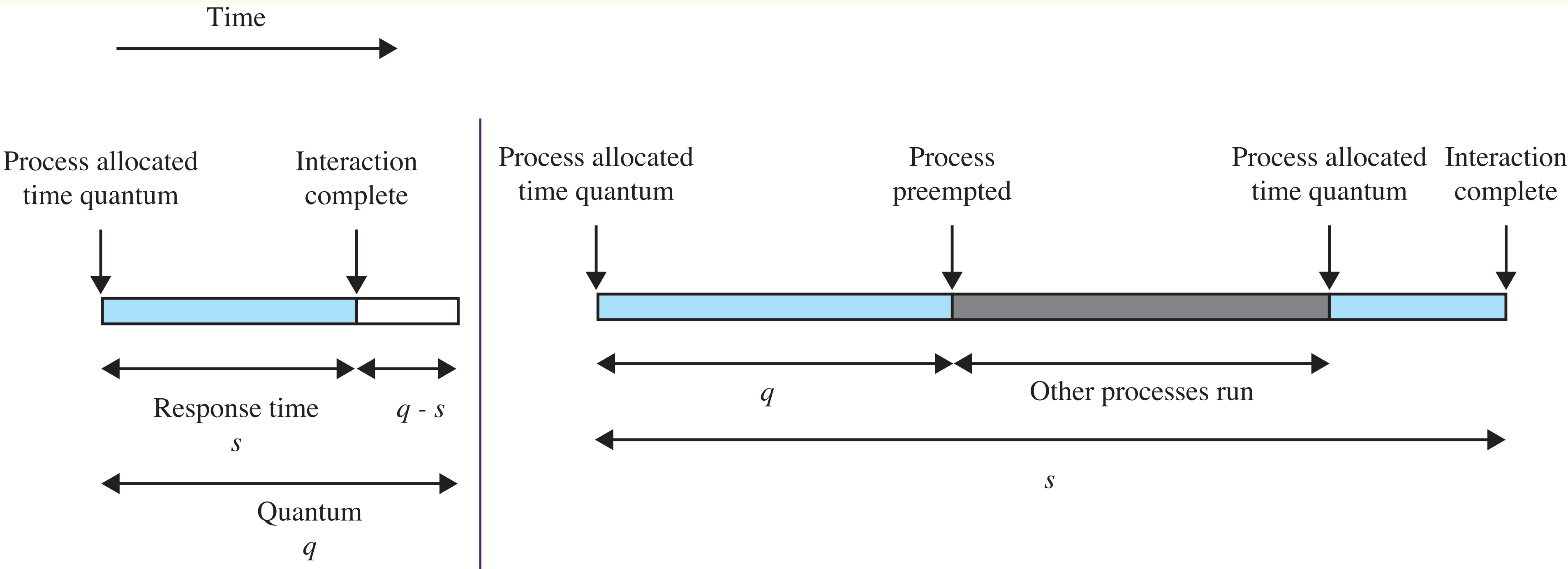


- Rarely used as primary scheduling algorithm
- Implemented using FIFO
- Tends to favor processor-bound processes over I/O-bound processes

Round Robin

- Based on FIFO
- Processes run only for a limited amount of time called a time slice or a quantum
- Preemptible
- Requires the system to maintain several processes in memory to minimize overhead
- Often used as part of more complex algorithms

Effect of Quantum Size (Principal Design Issue)



q < Typical Interaction Time

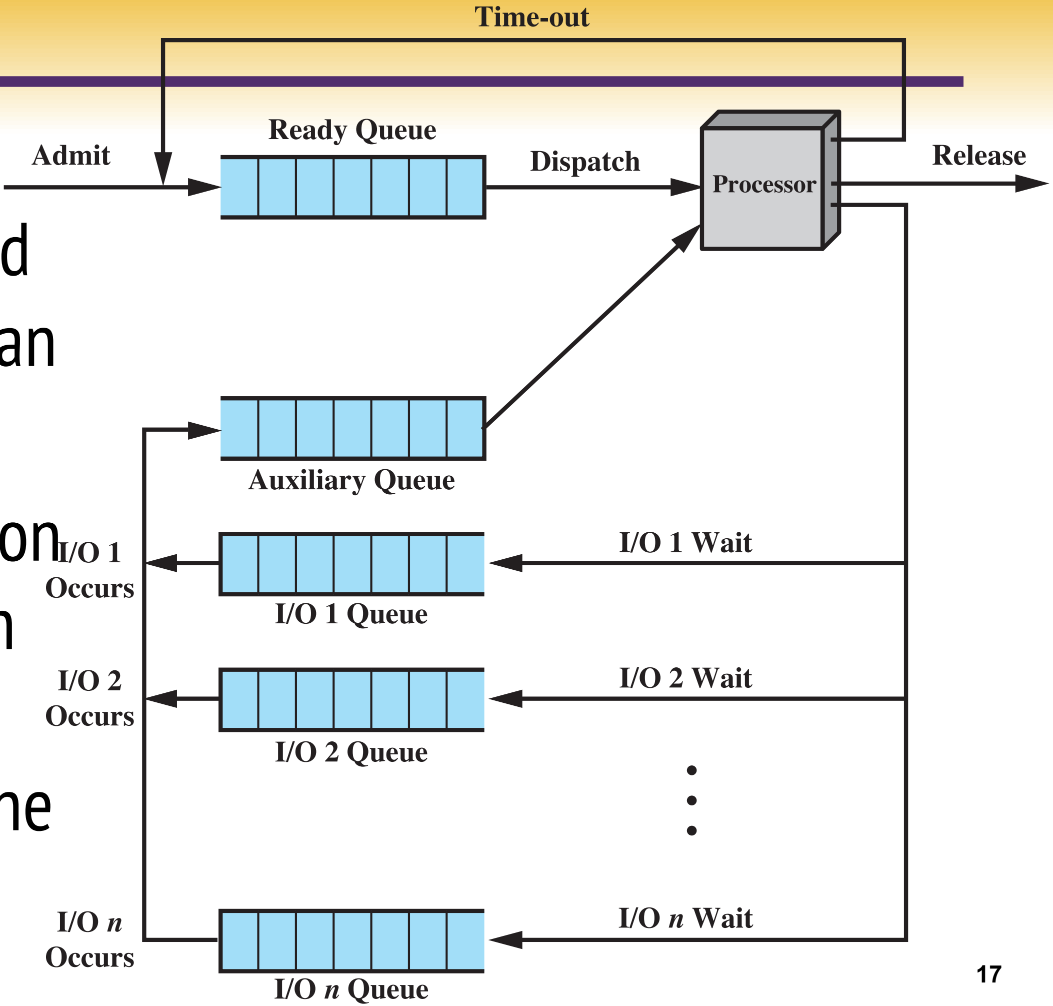
q > Typical Interaction Time

Quantum Size

- Determines response time to interactive requests
- Very large quantum size
  - Processes run for long periods
  - Degenerates to FIFO
- Very small quantum size
  - System spends more time context switching than running processes
- Middle-ground
  - Long enough for interactive processes to issue I/O request
  - Batch processes still get majority of processor time

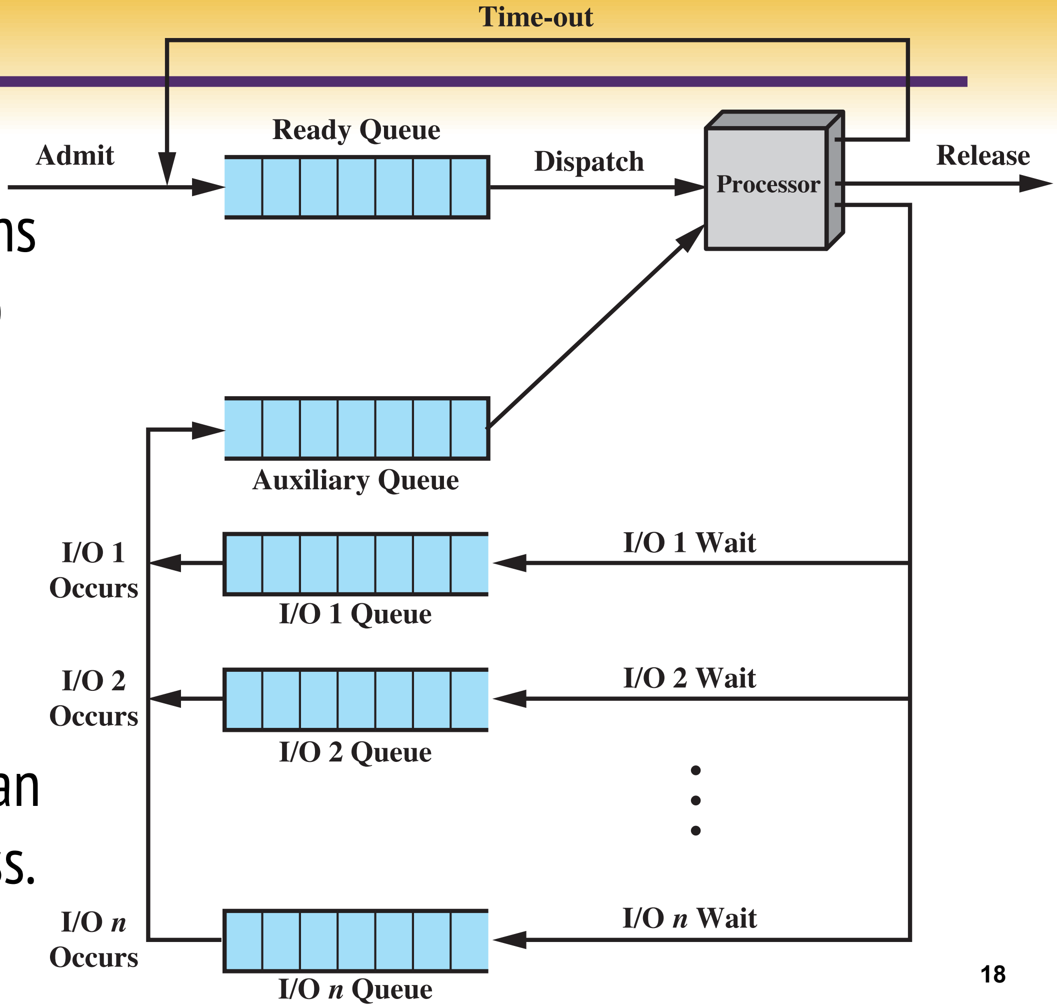
Virtual Round Robin

- FCFS auxiliary queue to which processes are moved after being released from an I/O block.
- When a dispatching decision is to be made, processes in the auxiliary queue get preference over those in the main ready queue.



Virtual Round Robin

- When a process is dispatched from the auxiliary queue, it runs no longer than a time equal to the basic time quantum minus the total time spent running since it was last selected from the main ready queue.
- Performance studies indicate that this approach is better than round robin in terms of fairness.



Shortest Process Next (SPN) Scheduling

- Scheduler selects process with smallest time to finish
  - Lower average wait time than FIFO
    - Reduces the number of waiting processes
  - Potentially large variance in wait times, starvation for longer processes
  - Nonpreemptive
    - Results in slow response times to arriving interactive requests
  - Relies on estimates of time-to-completion
    - Can be inaccurate
  - Unsuitable for use in modern interactive systems

How to predict execution time in SPN ?

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$$

T_i =processor execution time for the *i*th instance of this process (total execution time for batch job; processor burst time for interactive job),

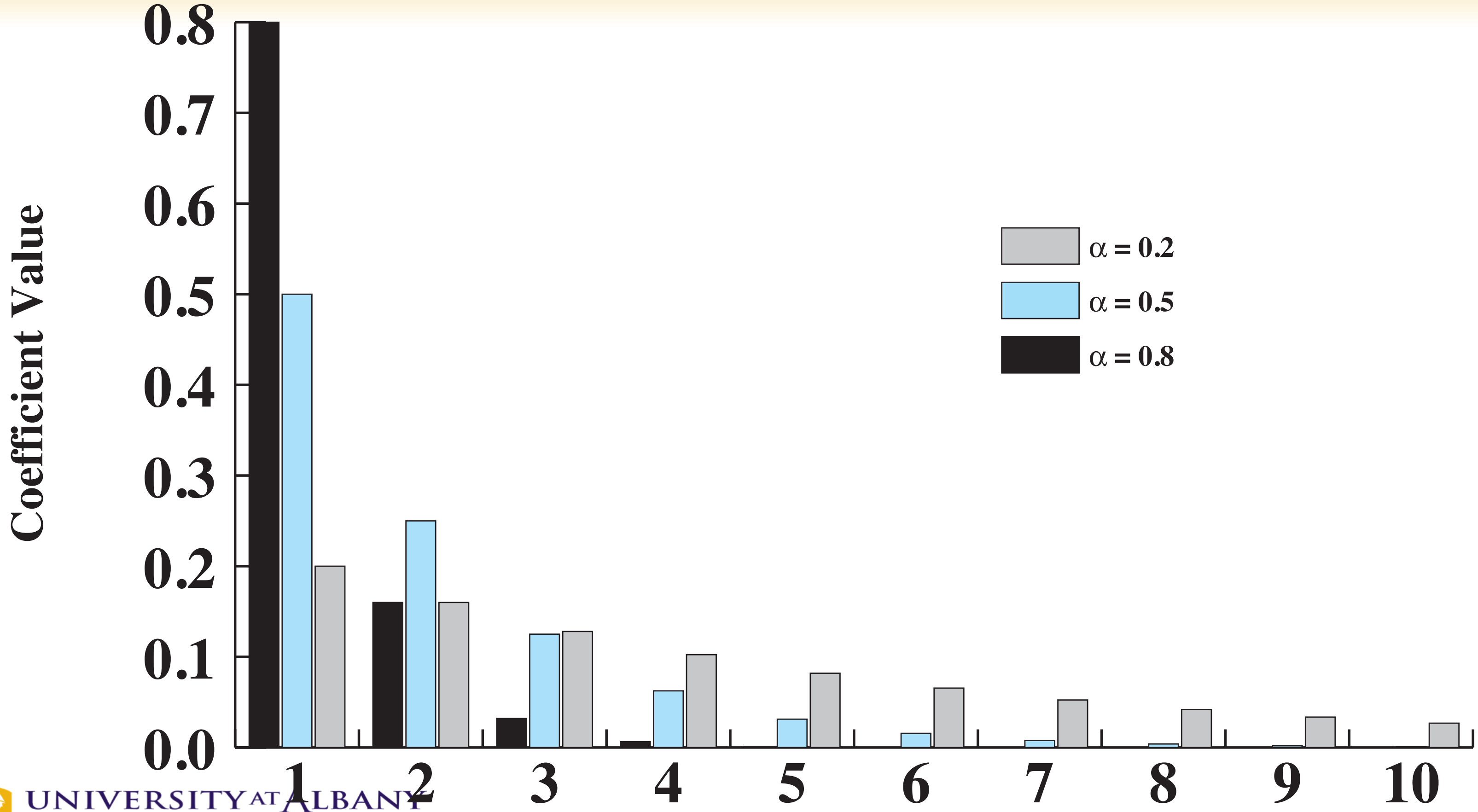
S_i =predicted value for the *i*th instance, and

S_1 =predicted value for first instance; not calculated.

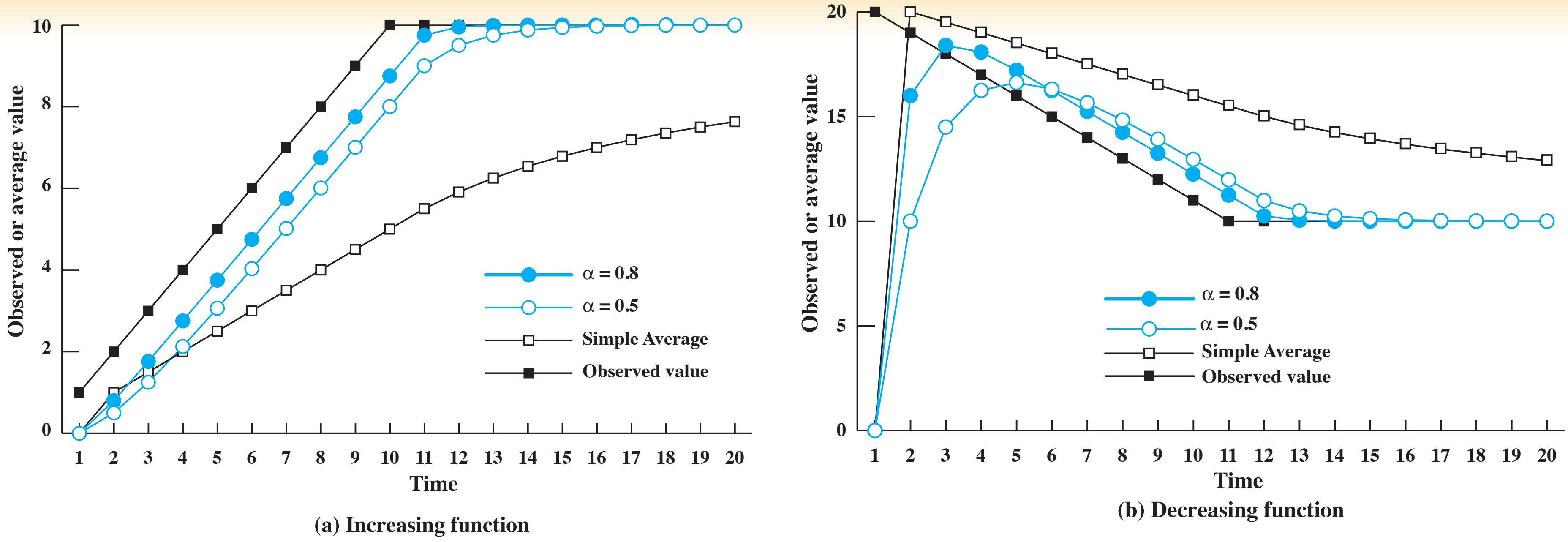
- Store the Sum $S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$
- Higher weight to recent instances $S_{n+1} = \alpha T_n + (1-\alpha) S_n$
- The older the observation, the less it is counted in to the average.

$$S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + \ldots + (1-\alpha)^i \alpha T_{n-i} + \ldots + (1-\alpha)^n S_1$$

Age of Observation



Exponential Averaging



Shortest Remaining Time (SRT) Scheduling

- Preemptive version of SPF
- Shorter arriving processes preempt a running process
- Very large variance of response times: long processes wait even longer than under SPF
- Not always optimal
  - Short incoming process can preempt a running process that is near completion
  - Context-switching overhead can become significant

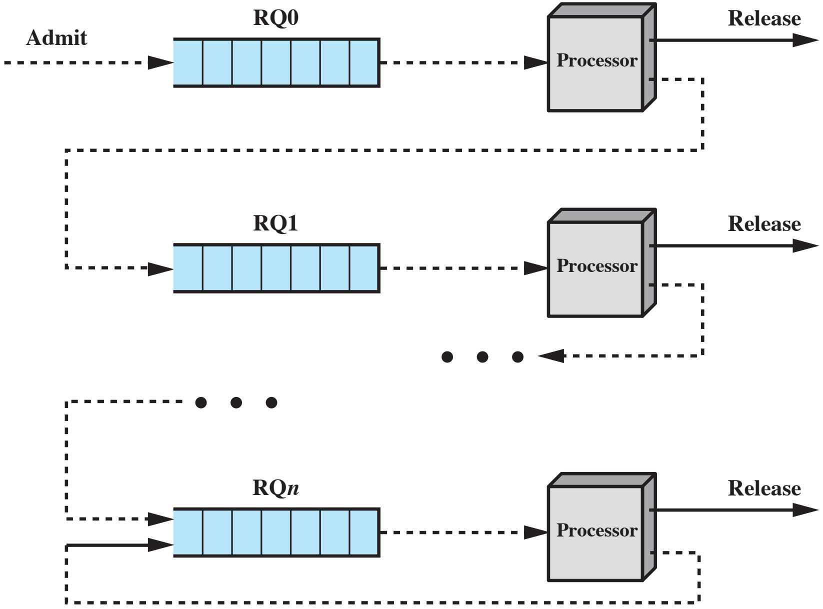
Highest Response Ratio Next (HRRN) Scheduling

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

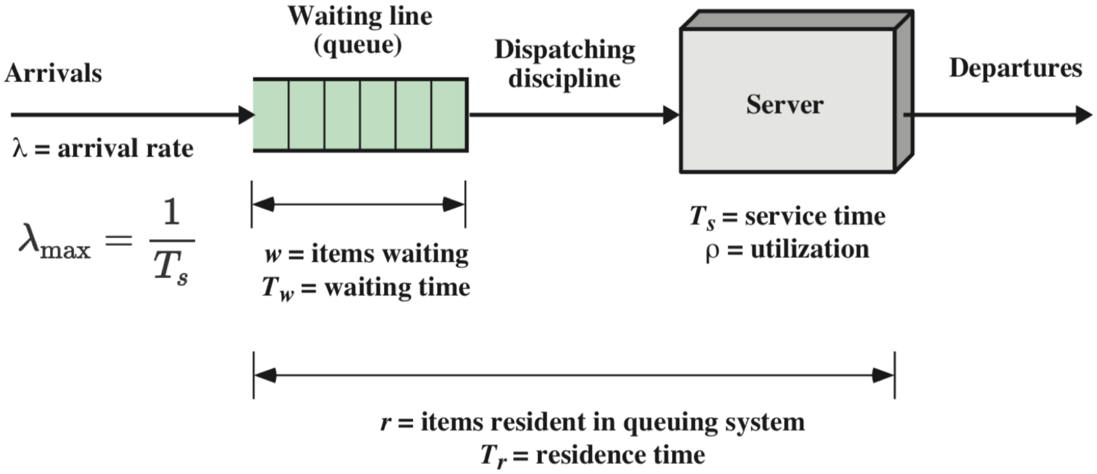
- Chooses next process with the greatest response ratio
- Min. value of R = 1 (when process is created)
- Attractive because it accounts for the age of the process
- While shorter jobs are favored, aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs

Feedback Scheduling

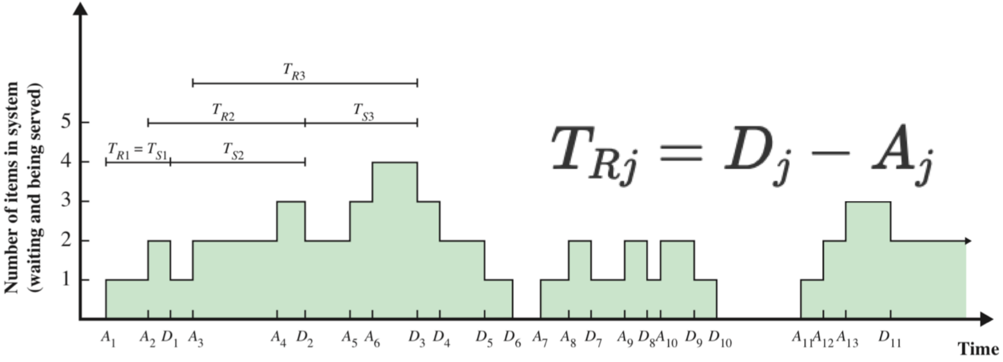
- Scheduling is done on a preemptive (at time quantum) basis, and a *dynamic priority* mechanism is used.
- When a process first enters the system, it is placed in RQ0.
- After its first preemption, when it returns to the Ready state, it is placed in RQ1.
- Each subsequent time that it is preempted, it is demoted to the next lower-priority queue.
- Once in the lowest-priority queue, it is returned to this queue repeatedly until it completes execution



Queuing Analysis



$$\lambda_{max} = \frac{1}{T_s}$$



$$T_{Rj} = D_j - A_j$$

$$T_{Rn+1} = T_{Sn+1} + \text{MAX}\,[0,\ D_n - A_{n+1}]$$

For item i:

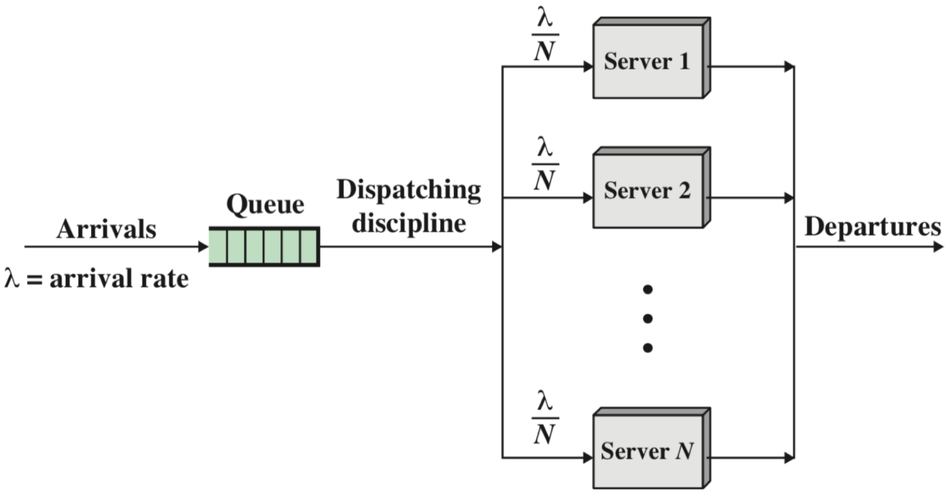
A_i = Arrival time

D_i = Departure time

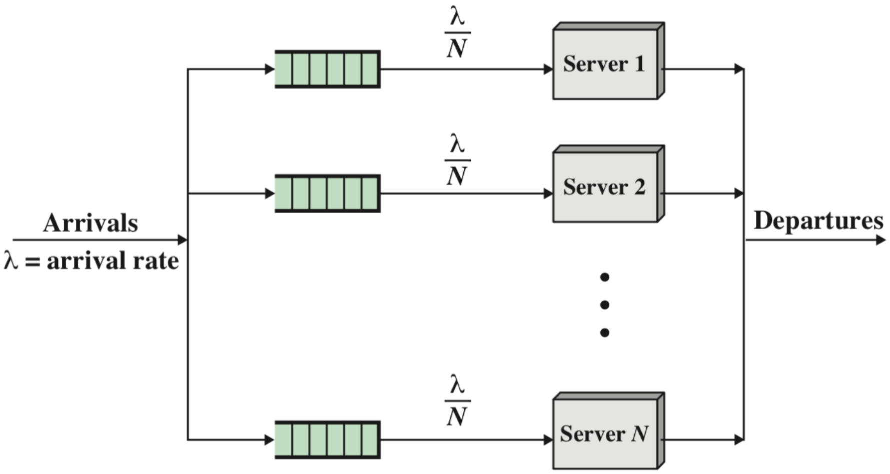
T_{Ri} = Residence time

T_{Si} = Service time

Multiple Server



(a) Multiserver queue



(b) Multiple Single-server queues

$N\rho$ utilization of the entire system

$$\lambda_{\max} = \frac{N}{T_s}$$

Queuing Relationship

General	Single Server	Multiserver
$r = \lambda T_r$ Little's formula $w = \lambda T_w$ Little's formula $T_r = T_w + T_s$	$\rho = \lambda T_s$ $r = w + \rho$	$\rho = \frac{\lambda T_s}{N}$ $u = \lambda T_s = \rho N$ $r = w + N\rho$

Performance

- Any scheduling policy that chooses the next item to be served independent of service time obeys the relationship:

$$\frac{T_r}{T_s} = \frac{1}{1 - \rho}$$

where

T_r = turnaround time or residence time; total time in system, waiting plus execution

T_s = average service time; average time spent in Running state

ρ = processor utilization

Single Server Queue with Two Priorities

Assumptions:
1. Poisson arrival rate.
2. Priority 1 items are serviced before priority 2 items.
3. First-come-first-served dispatching for items of equal priority.
4. No item is interrupted while being served.
5. No items leave the queue (lost calls delayed).

(a) General formulas

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 T_{s1}; \quad \rho_2 = \lambda_2 T_{s2}$$

$$\rho = \rho_1 + \rho_2$$

$$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$

$$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

Single Server Queue with Two Priorities

(b) No interrupts; exponential service times	(c) Preemptive-resume queuing discipline; exponential service times
$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 - \rho_1}$ $T_{r2} = T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}$	$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1}}{1 - \rho_1}$ $T_{r2} = T_{s2} + \frac{1}{1 - \rho_1}\left(\rho_1 T_{s2} + \frac{\rho T_s}{1 - \rho}\right)$

Example

- A data stream consisting of a mixture of long and short packets being transmitted by a packet-switching node and that the rate of arrival of the two types of packets is equal. Suppose both packets have lengths that are exponentially distributed, and the long packets have a mean packet length of 10 times the short packets. In particular, let us assume a 64-Kbps transmission link and the mean packet lengths are 80 and 800 octets. Then the two service times are 0.01 and 0.1 seconds. Also assume the arrival rate for each type is 8 packets per second. So the shorter packets are not held up by the longer packets, let us assign the shorter packets a higher priority.

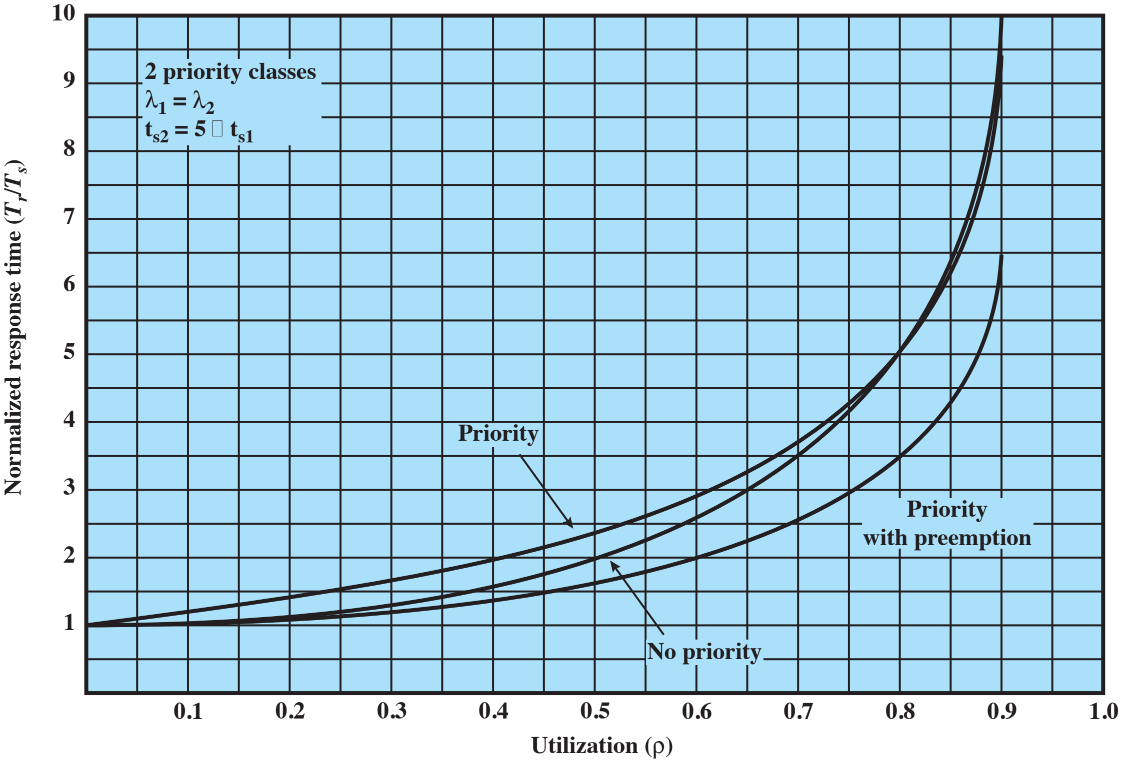
$$\rho_1 = 8 \times 0.01 = 0.08 \qquad \rho_2 = 8 \times 0.1 = 0.8 \qquad \rho = 0.88$$

$$T_{r1} = 0.01 + \frac{0.08 \times 0.01 + 0.8 \times 0.1}{1 - 0.08} = 0.098 \text{ seconds}$$

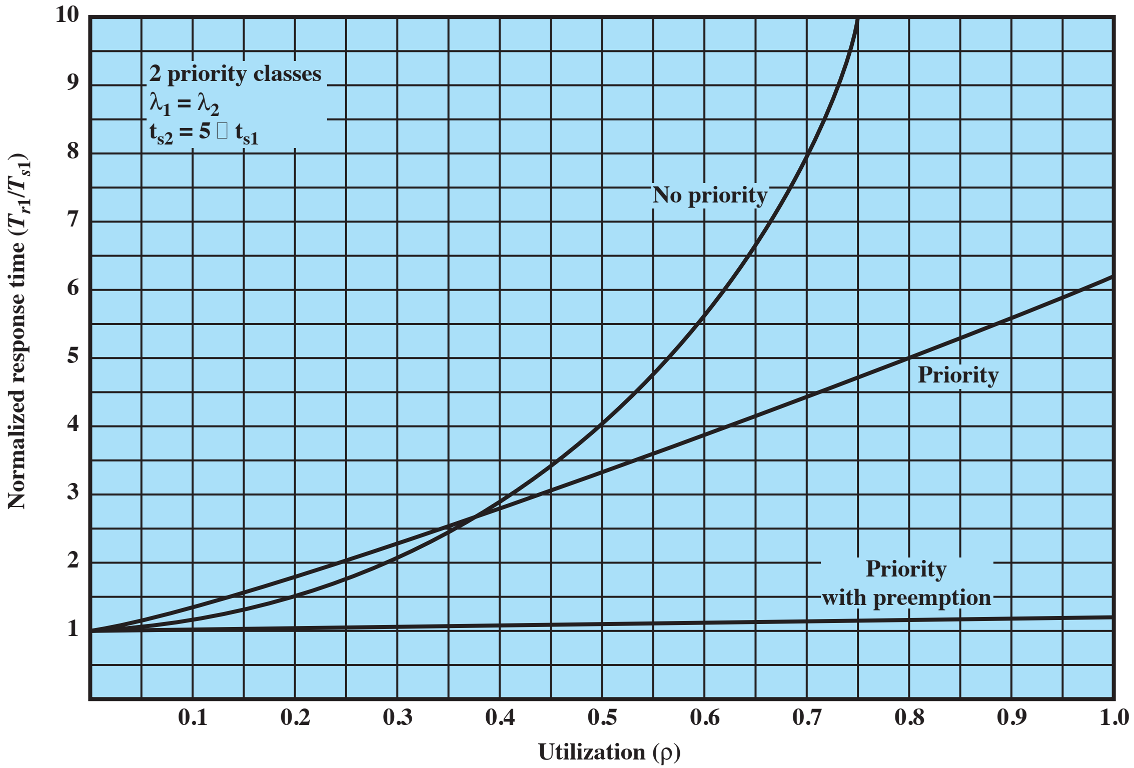
$$T_{r2} = 0.1 + \frac{0.098 - 0.01}{1 - 0.88} = 0.833 \text{ seconds}$$

$$T_r = 0.5 \times 0.098 + 0.5 \times 0.833 = 0.4655 \text{ seconds}$$

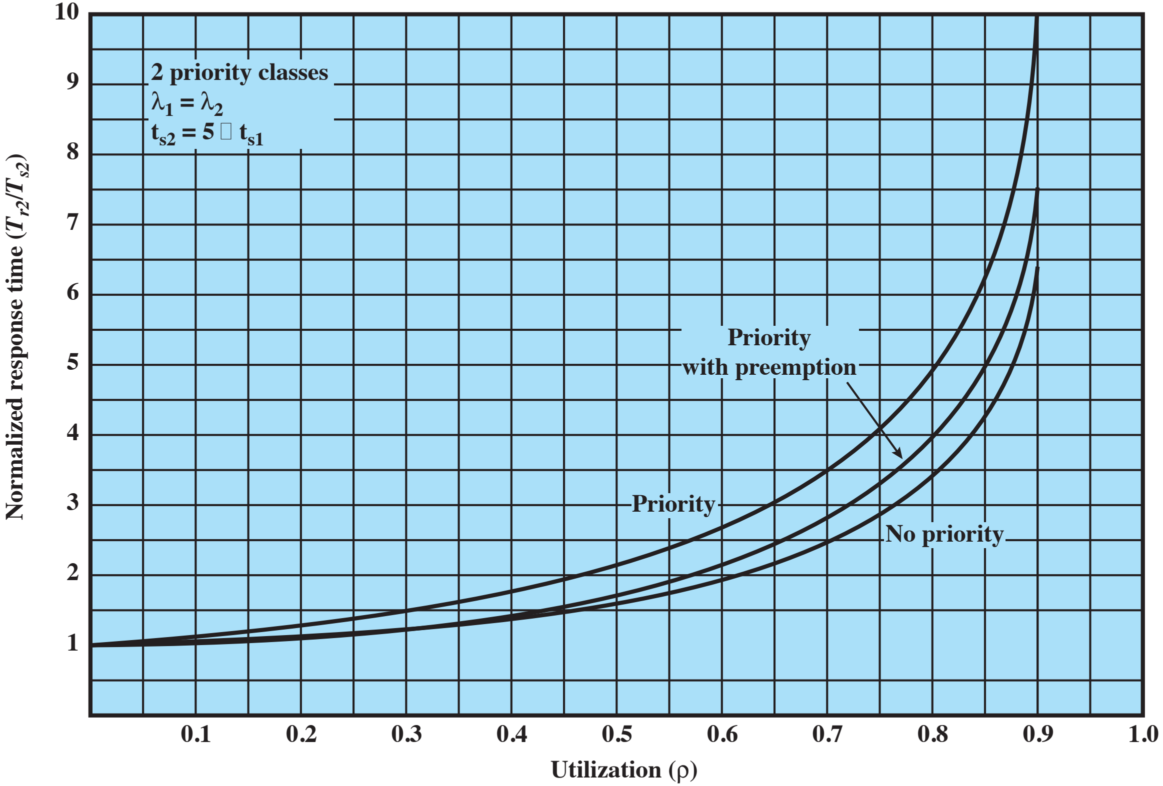
Normalized Response Time



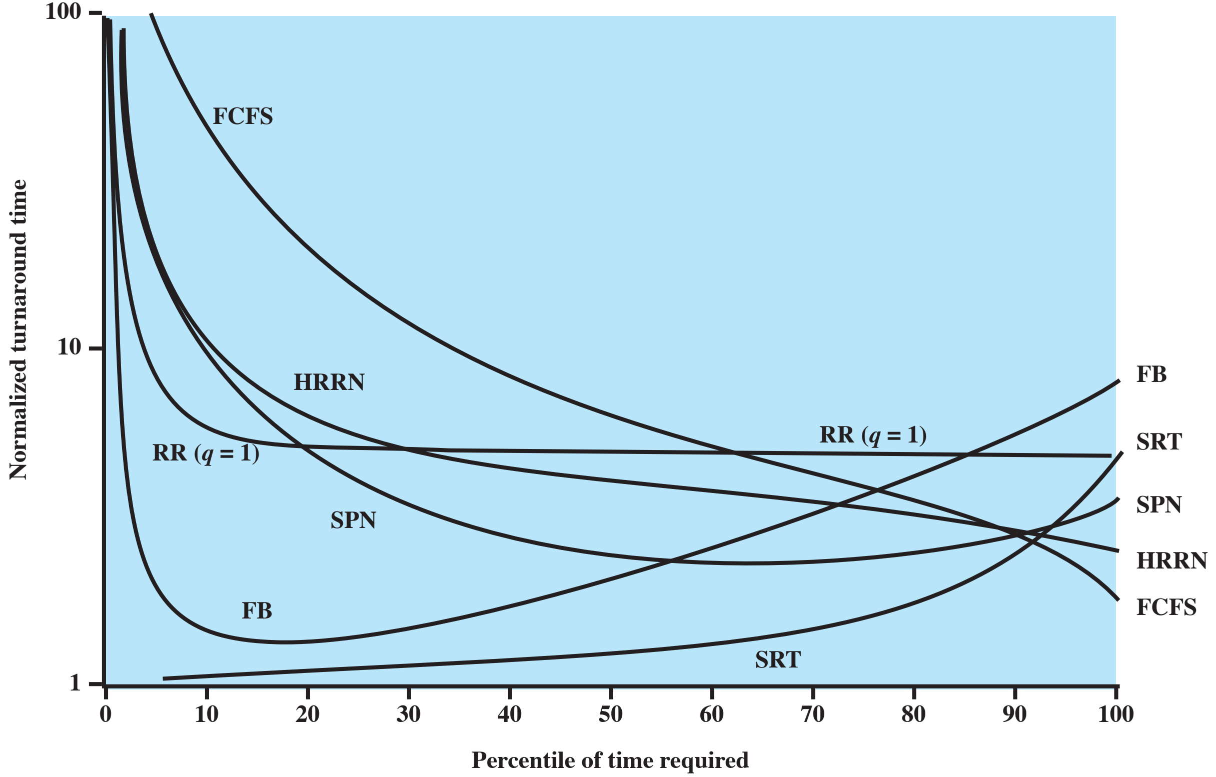
Normalized Response Time for Shorter Processes



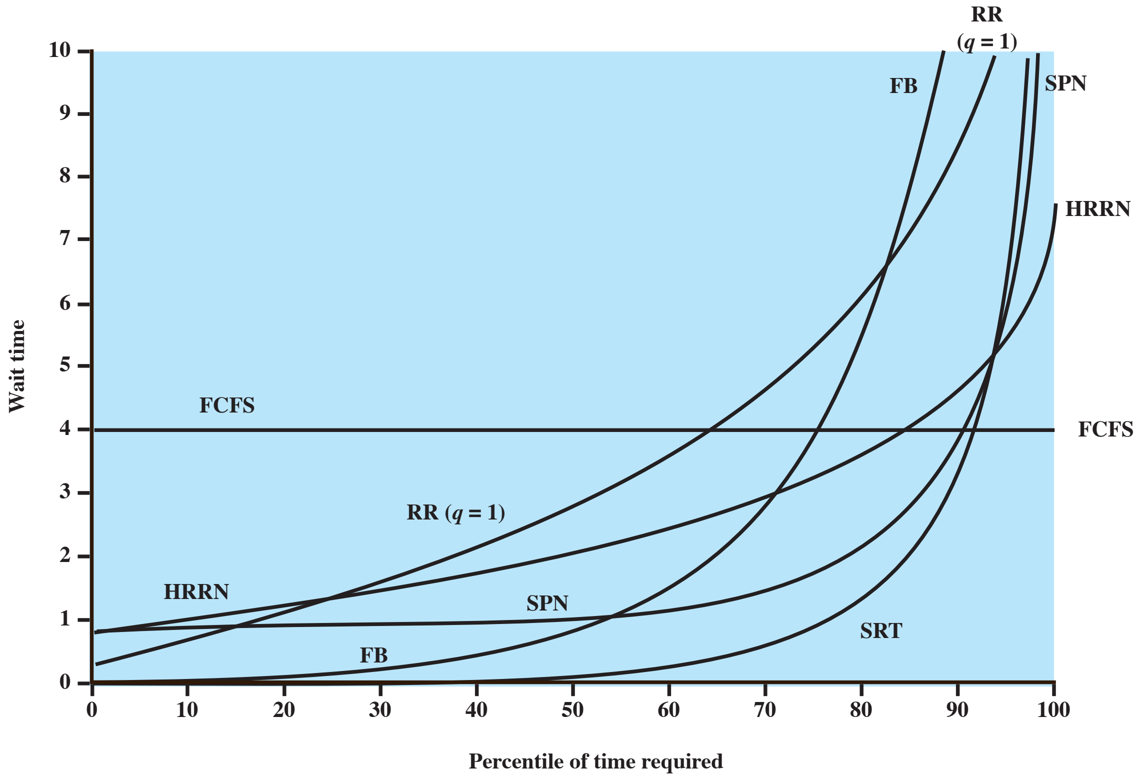
Normalized Response Time for Longer Processes



Normalized Turnaround Time



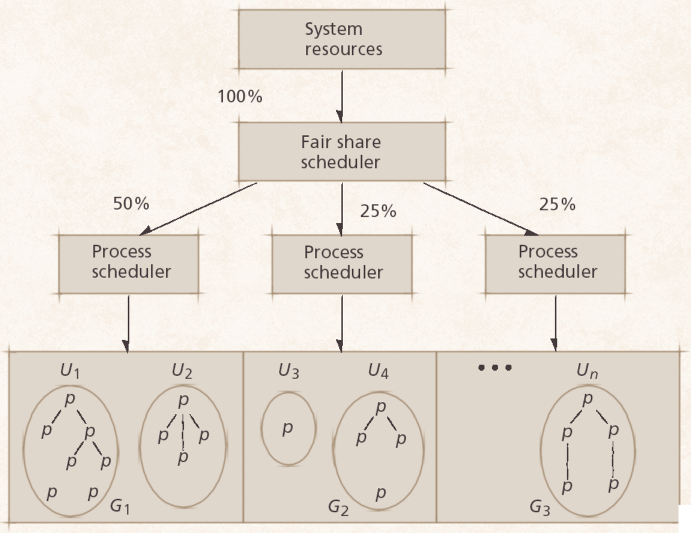
Waiting Time



Fair Share Scheduler

- Scheduling decisions based on the process sets
- Each user is assigned a share of the processor
- Objective is to monitor usage to give fewer resources to users who have had more than their fair share and more to those who have had less than their fair share
- Some user groups more important than others
- Ensures that less important groups cannot monopolize resources
- Unused resources distributed according to the proportion of resources each group has been allocated
- Groups not meeting resource-utilization goals get higher priority

Fair Share



$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

$CPU_j(i)$ = measure of processor utilization by process *j* through interval *i*,

$GCPU_k(i)$ = measure of processor utilization of group *k* through interval *i*,

$P_j(i)$ = priority of process *j* at beginning of interval *i*; lower values equal higher priorities,

$Base_j$ = base priority of process *j*, and

W_k = weighting assigned to group *k*, with the constraint that and $0 < W_k \leq 1$ and $\sum_k W_k = 1$.

Example

Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Group 1: Process A. Group 2: Process B, Process C.

UNIX Scheduler

- Designed to provide good response time for interactive users while ensuring that low-priority background jobs do not starve
- Employs multilevel feedback using round robin within each of the priority queues
- Makes use of one-second preemption
- Priority is based on process type and execution history

Scheduling Formula

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

where

$CPU_j(i)$ = measure of processor utilization by process j through interval i

$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities

$Base_j$ = base priority of process j

$nice_j$ = user-controllable adjustment factor

Characteristics of Various Scheduling Policies

	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
Selection Function	max[w]	constant	min[s]	$\min[s - e]$	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision Mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response Time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on Processes	Penalizes short processes; penalizes I/O-bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O-bound processes
Starvation	No	No	Possible	Possible	No	Possible