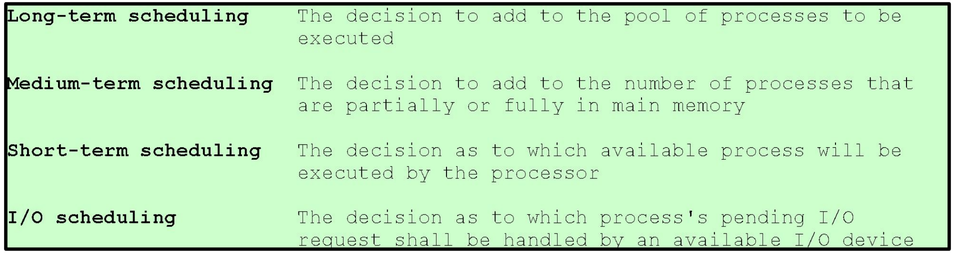
Operating Systems
CS-C3140, Lecture 9
Scheduling: Uniprocessor and Real-Time

Alexandru Paler

Processor Scheduling

- Assign processes to be executed by the processor in a way that meets system objectives:
  - response time
  - throughput
  - processor efficiency
- Broken down into three separate functions: long-, medium- and short-term scheduling



Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

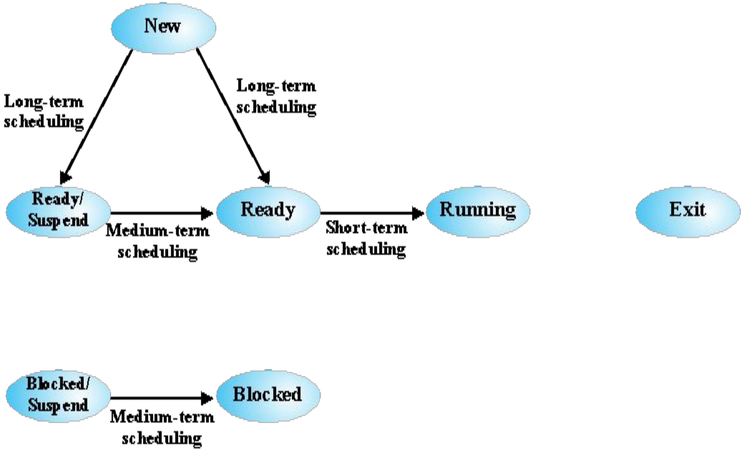
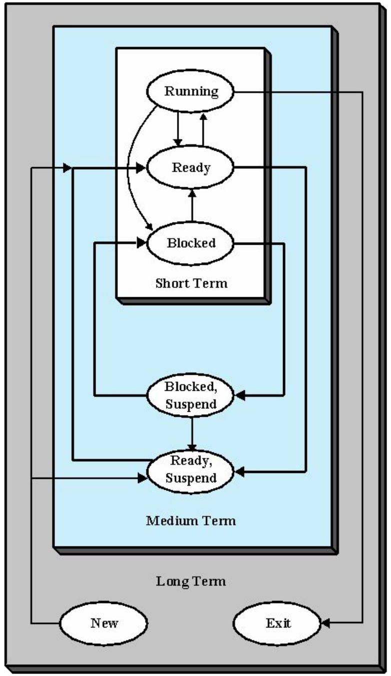
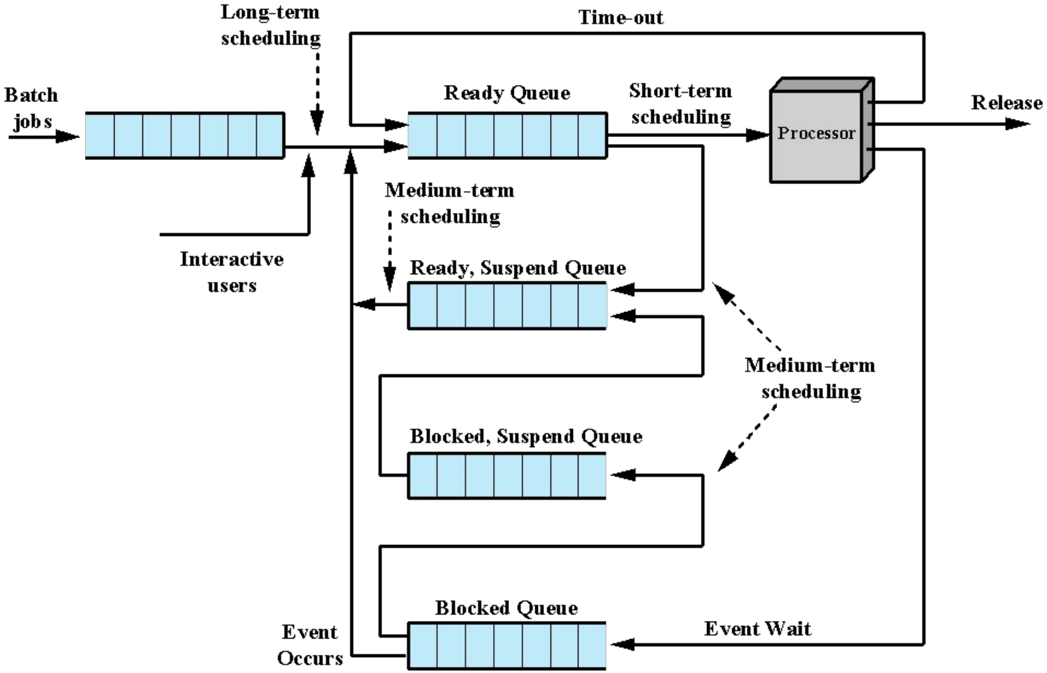


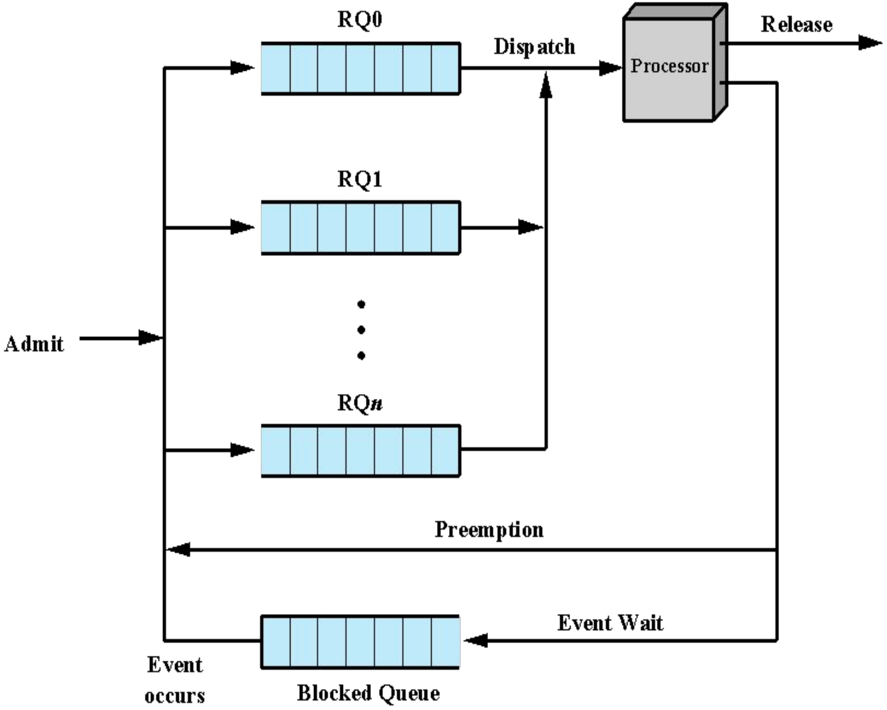
Figure 9.1 Scheduling and Process State Transitions

Scheduling : Managing queues to minimize queuing delay and to optimize performance in a queuing environment





Priority Queuing



For clarity, the queuing diagram is simplified, ignoring the existence of multiple blocked queues and of suspended states

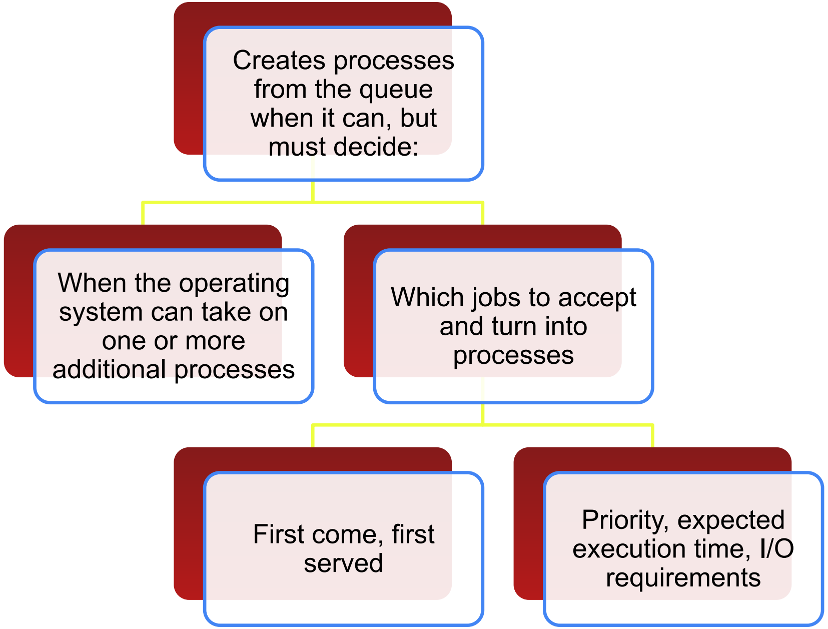
- Each process is assigned a priority
  - A set of queues, in descending order of priority
  - RQ0, RQ1, . . . , RQ n
  - priority[RQ i] > priority[RQ j] for i > j
- The scheduler will always choose a process of higher priority over one of lower priority
  - When a scheduling selection is to be made, the scheduler will start at the highest-priority ready queue (RQ0).
  - If there are one or more processes in the queue, a process is selected using some scheduling policy.
  - If RQ0 is empty, then RQ1 is examined, and so on.

Disadvantages:

- lower-priority processes may suffer starvation
- Happens if there is always a steady supply of higher-priority ready processes
- If this behavior is not desirable, the priority of a process can change with its age or execution history.

Long-Term Scheduler

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
  - The more processes that are created, the smaller the percentage of time that each process can be executed
  - May limit to provide satisfactory service to the current set of processes



Creates processes from the queue when it can, but must decide:

When the operating system can take on one or more additional processes

Which jobs to accept and turn into processes

First come, first served

Priority, expected execution time, I/O requirements

Medium- and Short- Term Scheduling

- Medium-Term - Part of the swapping function
  - Makes swapping-in decisions
  - Considers the memory requirements of the swapped-out processes
- Short-Term - a.k.a. the dispatcher
  - Executes most frequently
  - Allocate processor time to optimize certain aspects of system behavior
  - Makes the fine-grained decision of which process to execute next
  - Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another
    - Clock interrupts
    - I/O interrupts
    - Operating system calls
    - Signals (e.g., semaphores)

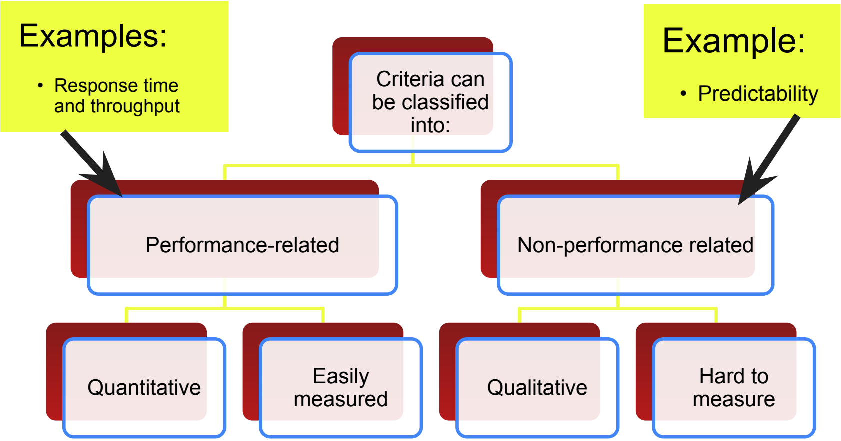
User-oriented criteria

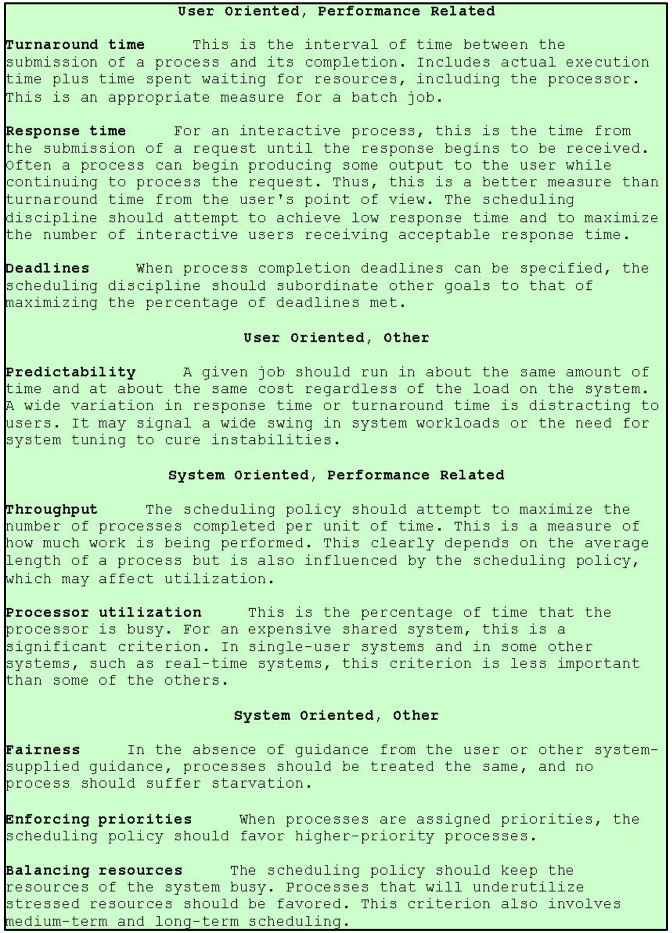
- Relate to the behavior of the system as perceived by the individual user or process (such as response time in an interactive system)
- Important on virtually all systems

System-oriented criteria

- Focus is on **effective and efficient utilization** of the processor (rate at which processes are completed)
- Generally of minor importance on single-user systems

Performance and Scheduling Criteria





User Oriented, Performance Related

Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

Deadlines When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

User Oriented, Other

Predictability A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.

System Oriented, Performance Related

Throughput The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

System Oriented, Other

Fairness In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

Enforcing priorities When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

Balancing resources The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.

Selection Function and Decision Modes

- Determines which process, among ready processes, is selected next for execution
- May be based on priority, resource requirements, or the execution characteristics of the process
- If based on execution characteristics, then important quantities are:
  - ***w*** = time spent in system so far, waiting
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e*; generally, this quantity must be estimated or supplied by the user

Mode: Specifies the instants in time at which the selection function is exercised

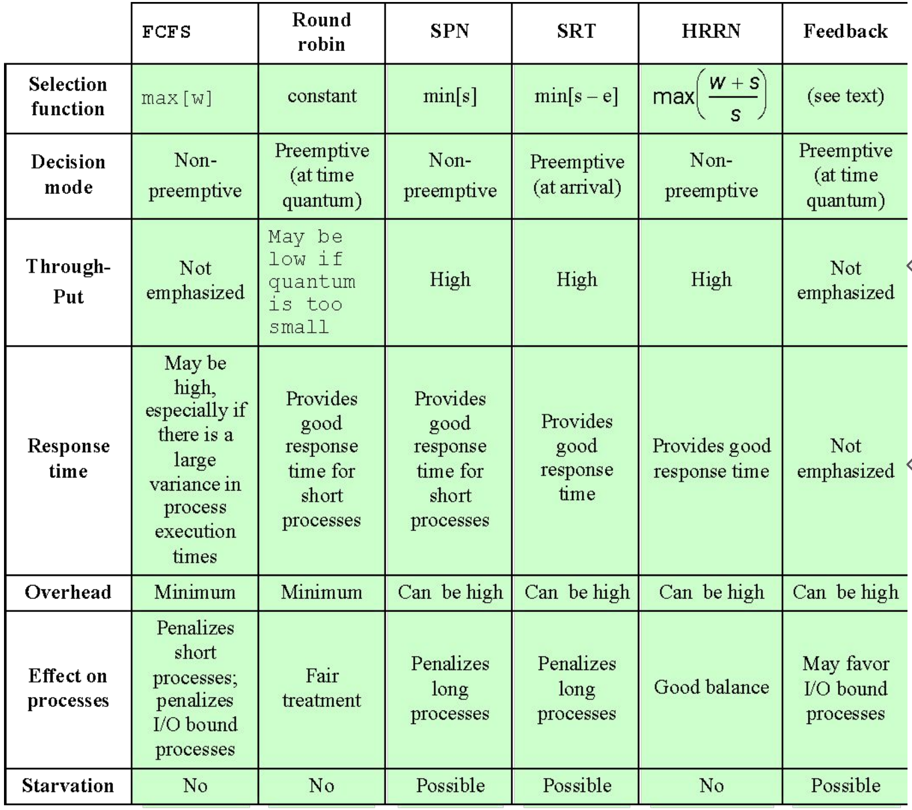
Nonpreemptive - once a process is in the Running state, it continues to execute until

- it terminates or
- it blocks itself to wait for I/O or to request some OS service.

Preemptive - Currently running process may be interrupted and moved to ready state by the OS

- Decision may be performed when
  - a new process arrives,
  - when an interrupt occurs that places a blocked process in the Ready state
  - periodically, based on a clock interrupt

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)
Through-Put	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

Number of processes completed per unit of time

Time from submission until first response

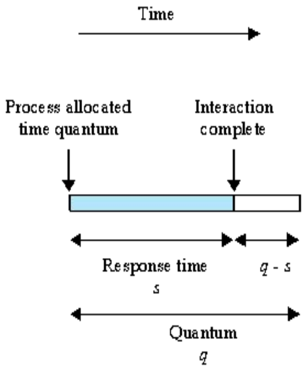
FCFS - First Come First Served; SPN - Shortest Process Next; SRT - Shortest Remaining Time; HRRN - Highest Response Ratio Next
w = time spent in system so far, waiting; e = time spent in execution so far; s = total service time required by the process, including e;

First-Come-First-Served (FCFS)

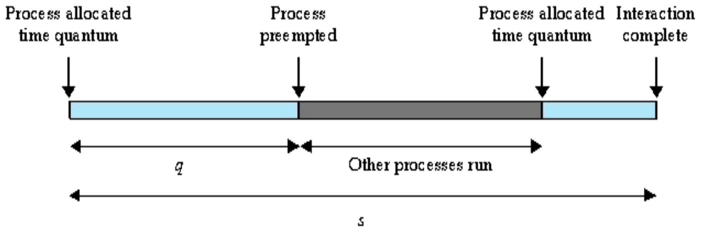
- Simplest scheduling policy
- Also known as first-in-first-out (FIFO) or a strict queuing scheme
- As each process becomes ready, it joins the ready queue
- When the currently running process ceases to execute, the process that has been in the ready queue the longest is selected for running
- Performs much better for long processes than short ones
- Tends to favor processor-bound processes over I/O-bound processes

Round Robin

- Time slicing
  - each process is given a slice of time before being preempted
  - Uses preemption based on a clock
  - Effective in a general-purpose time-sharing system or transaction processing system
- Principal design issue is the length of the time quantum, or slice
  - Time quantum should be slightly greater than the time required for a typical interaction or process function
  - if it is less, then most processes will require at least two time quanta
  - if it is longer than the longest running process, round robin degenerates to FCFS
- One drawback is its relative treatment of processor-bound and I/O-bound processes



(a) Time quantum greater than typical interaction


Process allocated time quantum
Process preempted
Process allocated time quantum
Interaction complete
q
Other processes run
s

(b) Time quantum less than typical interaction

Shortest Process Next (SPN)

- Nonpreemptive policy
- The process with the shortest expected processing time is selected next
- A short process will jump to the head of the queue
- Disadvantages
  - Possibility of starvation for longer processes
  - estimate the required processing time
  - if the estimate is substantially under the actual running time, the system may abort the job

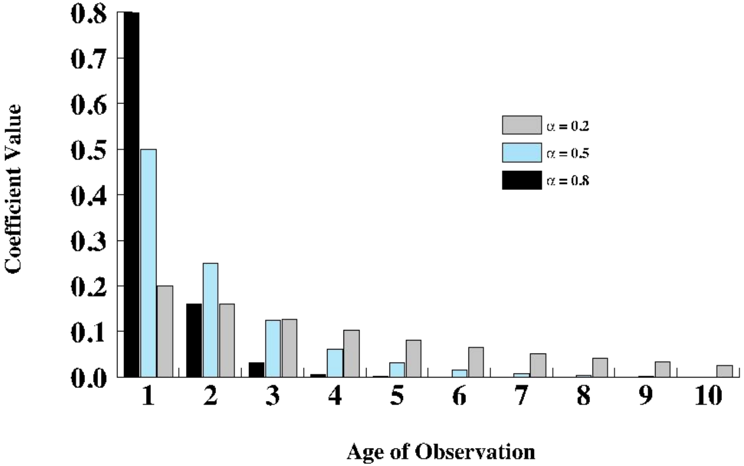
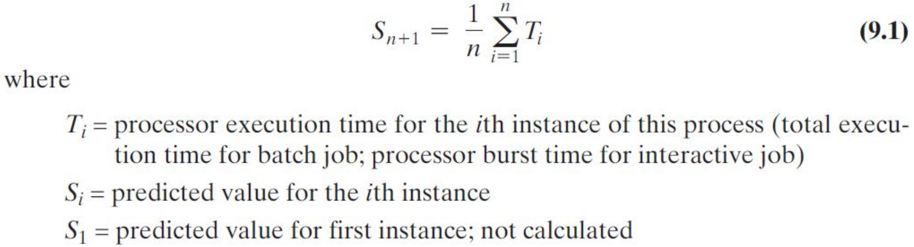


Figure 9.8 Exponential Smoothing Coefficients



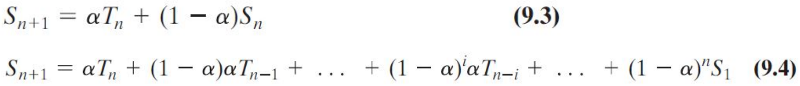
$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i \qquad \textbf{(9.1)}$$

where

T_i = processor execution time for the ith instance of this process (total execution time for batch job; processor burst time for interactive job)

S_i = predicted value for the ith instance

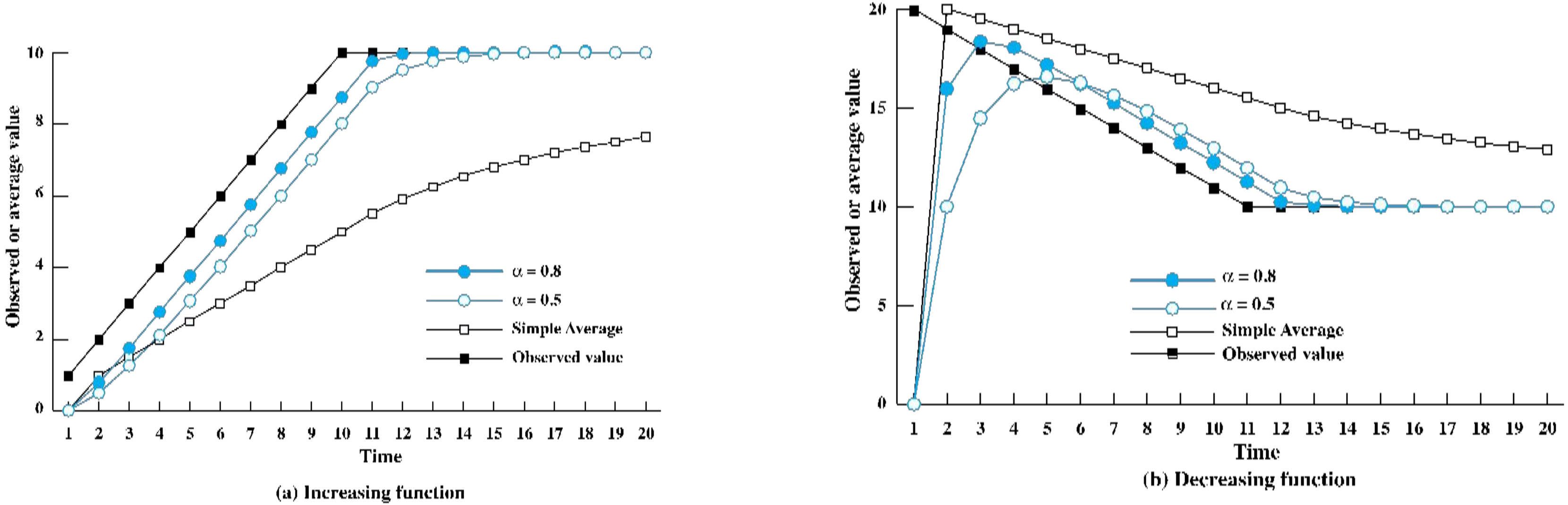
S_1 = predicted value for first instance; not calculated



$$S_{n+1} = \alpha T_n + (1 - \alpha)S_n \qquad \textbf{(9.3)}$$

$$S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \ldots + (1 - \alpha)^i \alpha T_{n-i} + \ldots + (1 - \alpha)^n S_1 \qquad \textbf{(9.4)}$$

Exponential averaging: Reaction to the change in the observed values



(a) Increasing function

(b) Decreasing function

Figure 9.9 Use of Exponential Averaging

Shortest Remaining Time (SRT) & Highest Response Ratio Next (HRRN)

- Preemptive version of SPN
- Scheduler always chooses the process that has the shortest expected remaining processing time
- Risk of starvation of longer processes
  - Should give superior turnaround time performance to SPN because a short job is given immediate preference to a running longer job

- Thus, our scheduling rule becomes the following: when the current process
- completes or is blocked, choose the ready process with the greatest value of *R* .
- This approach is attractive because it accounts for the age of the process. While
- shorter jobs are favored (a smaller denominator yields a larger ratio), aging without
- service increases the ratio so that a longer process will eventually get past competing
- shorter jobs.
- Chooses next process with the greatest ratio
- 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

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

Feedback Scheduling

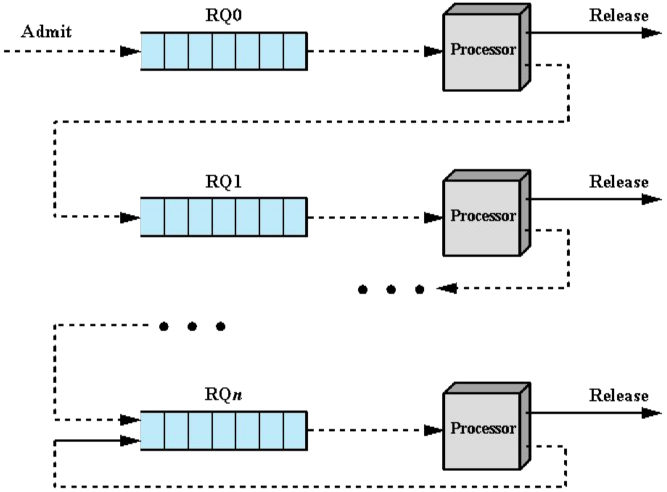


Figure 9.10 Feedback Scheduling

- Focus on the time spent in execution so far.
- This approach is known as multilevel feedback , meaning that the operating system allocates the processor to a process and, when the process blocks or is preempted, feeds it back into one of several priority queues.

Scheduling is done on a preemptive 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

Details:

- A short process will complete quickly, without migrating very far down the hierarchy of ready queues
- A longer process will gradually drift downward
- Newer, shorter processes are favored over older, longer processes
- Within each queue, except the lowest-priority queue, a simple FCFS mechanism is used.

Even with the allowance for greater time allocation at lower priority

- A longer process may still suffer starvation
- A possible remedy is to promote a process to a higher-priority queue after it spends a certain amount of time waiting for service in its current queue

Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

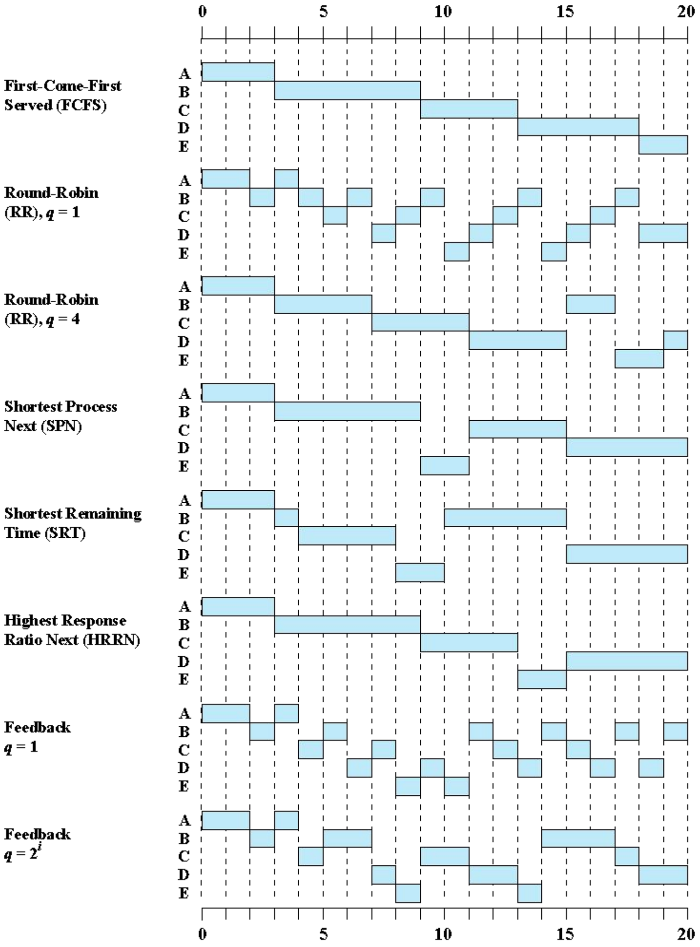


Figure 9.5 A Comparison of Scheduling Policies

Comparison of Scheduling Policies

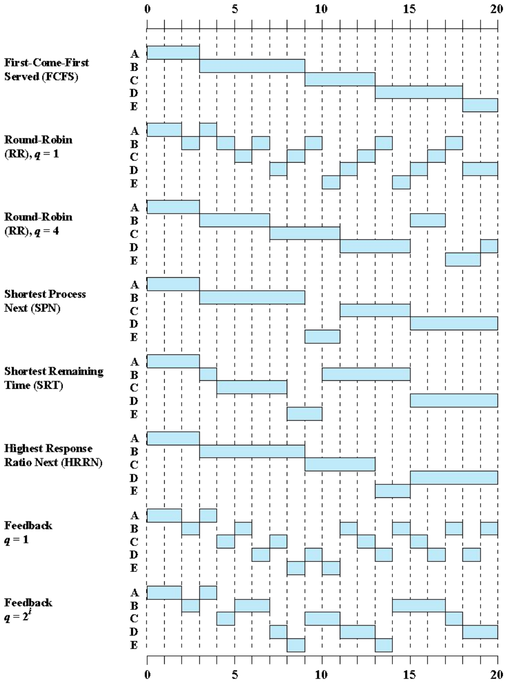
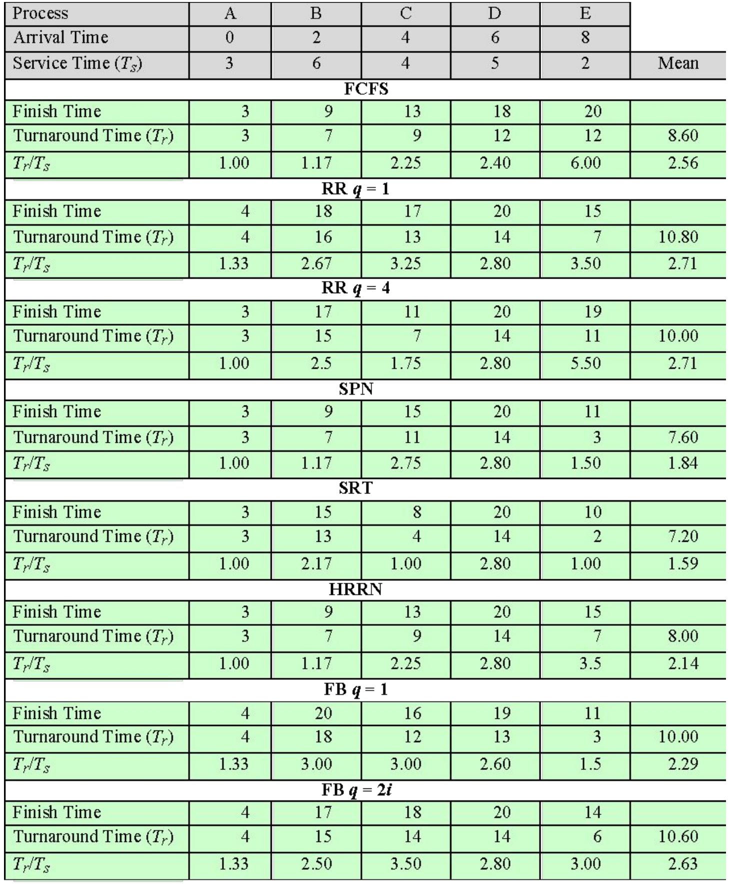


Figure 9.5 A Comparison of Scheduling Policies



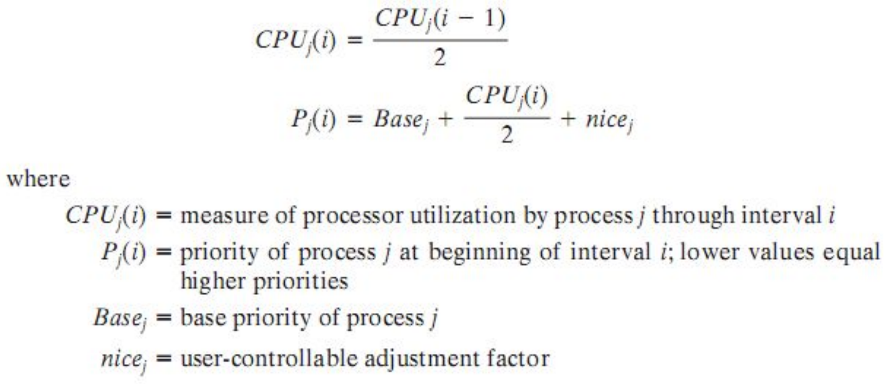
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

- Turnaround time (TAT) is the residence time T_r , or total time that the item spends in the system (waiting time plus service time).
- Normalized turnaround time, which is the **ratio of turnaround time to service time**. (increasing values -> decreasing level of service)

Traditional UNIX Scheduling

- 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
  - based on process type and execution history
  - recomputed once per second, at which time a new scheduling decision is made

- **base priority** divides all processes into fixed bands of priority levels.
- **CPU** and **nice** restrict a process from migrating out of its assigned band



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

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

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

Real-Time Systems

- The operating system, and in particular the scheduler, is perhaps the most important component
  - Control of laboratory experiments
  - Process control in industrial plants
  - Robotics
  - Air traffic control
  - Telecommunications
  - Military command and control systems
- Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
- Tasks or processes attempt to control or react to events that take place in the outside world
- These events occur in “real time” and tasks must be able to keep up with them

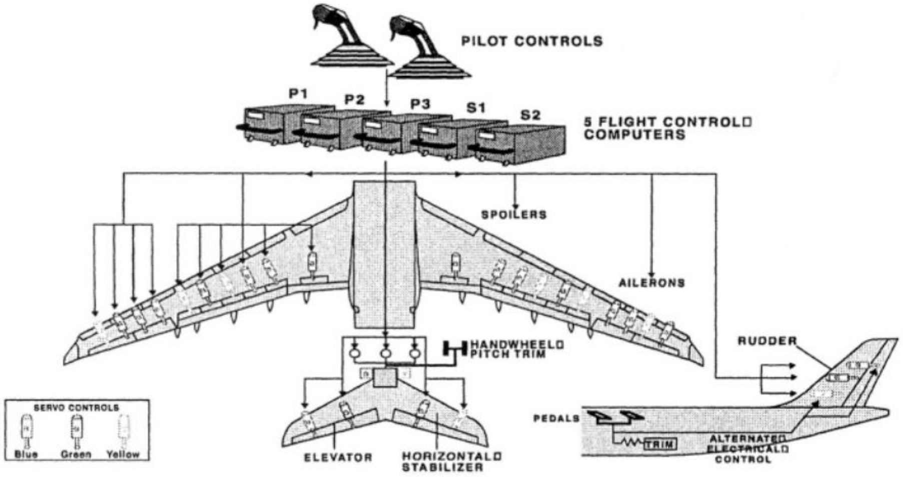


Figure 2: A340-600 system architecture

https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

Hard and Soft Real-Time Tasks

Hard real-time task

- One that must meet its deadline
- Otherwise it will cause unacceptable damage or a fatal error to the system

Soft real-time task

- Has an associated deadline that is desirable but not mandatory
- It still makes sense to schedule and complete the task even if it has passed its deadline

- Periodic tasks
  - Requirement may be stated as:
    - Once per period T
    - Exactly T units apart
- Aperiodic tasks
  - Has a deadline by which it must finish or start
  - May have a constraint on both start and finish time

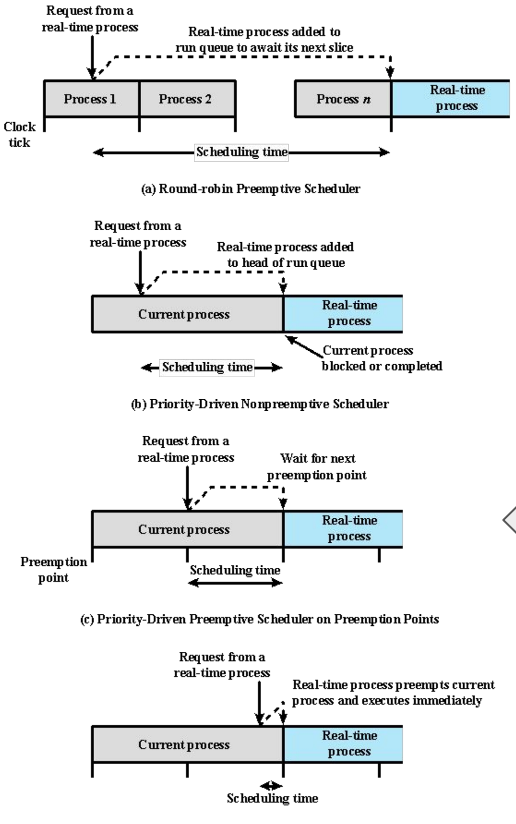
Characteristics of Real Time Systems (1)

- Determinism
  - Concerned with how long an operating system delays before acknowledging an interrupt
  - Operations are performed at fixed, predetermined times or within predetermined time intervals
  - When multiple processes are competing for resources and processor time, no system will be fully deterministic
- Responsiveness
  - Critical for real-time systems that must meet timing requirements imposed by individuals, devices, and data flows external to the system
  - Concerned with how long, after acknowledgment, it takes an operating system to service the interrupt
- User Control
  - It is essential to allow the user fine-grained control over task priority
  - User should be able to distinguish between hard and soft tasks and to specify relative priorities within each class
  - May allow user to specify such characteristics as:
    - Paging or process swapping
    - What processes must always be resident in main memory
    - What disk transfer algorithms are to be used
    - What rights the processes in various priority bands have

Characteristics of Real Time Systems (2)

- Reliability
  - Real-time systems respond to and control events in real time so loss or degradation of performance may have catastrophic consequences such as:
    - Financial loss
    - Major equipment damage
    - Loss of life
- Fail-Soft Operation
  - A characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible
  - Important aspect is stability
    - A real-time system is stable if the system will meet the deadlines of its most critical, highest-priority tasks even if some less critical task deadlines are not always met
- The following features are common to most real-time OSs
  - A stricter use of priorities than in an ordinary OS, with preemptive scheduling that is designed to meet real-time requirements
  - Interrupt latency is bounded and relatively short
  - More precise and predictable timing characteristics than general purpose OSs

RT Scheduling Examples


Request from a real-time process
Real-time process added to run queue to await its next slice
Process 1
Process 2
Process *n*
Real-time process
Clock tick
Scheduling time
Request from a real-time process
Real-time process added to head of run queue
Current process
Real-time process
Current process blocked or completed
Scheduling time
Request from a real-time process
Wait for next preemption point
Current process
Real-time process
Preemption point
Scheduling time
Request from a real-time process
Real-time process preempts current process and executes immediately
Current process
Real-time process
Scheduling time

(a) Round-robin Preemptive Scheduler

(b) Priority-Driven Nonpreemptive Scheduler

(c) Priority-Driven Preemptive Scheduler on Preemption Points

(d) Immediate Preemptive Scheduler

A **preemptive scheduler that uses simple round-robin scheduling**. A real-time task is added to the ready queue to await its next time slice. Scheduling time will generally be unacceptable for real-time applications.

A **nonpreemptive scheduler and priority scheduling**. A real-time task that is ready would be scheduled as soon as the current process blocks or runs to completion. This could lead to a delay of several seconds if a low, low-priority task were executing at a critical time. Again, this approach is not acceptable.

Priorities with clock-based interrupts. Preemption points occur at regular intervals. When a preemption point occurs, the currently running task is preempted if a higher-priority task is waiting. This would include the preemption of tasks that are part of the operating system kernel. Such a delay may be on the order of several milliseconds.

Immediate preemption. Operating system responds to an interrupt almost immediately, unless the system is in a critical-code lockout section. Scheduling delays for a real-time task are very low.

Classes of Real-Time Scheduling Algorithms

Static table-driven approaches

- Performs a static analysis of feasible schedules of dispatching
- Result is a schedule that determines, at run time, when a task must begin execution

Static priority-driven preemptive approaches

- A static analysis is performed but no schedule is drawn up
- Analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used

Dynamic planning-based approaches

- Feasibility is determined at run time rather than offline prior to the start of execution
- One result of the analysis is a schedule or plan that is used to decide when to dispatch this task

Dynamic best effort approaches

- No feasibility analysis is performed
- System tries to meet all deadlines and aborts any started process whose deadline is missed

Static Scheduling Algorithms

Static table-driven scheduling

- Applicable to tasks that are periodic
- Input to the analysis consists of the periodic arrival time, execution time, periodic ending deadline, and relative priority of each task
- This is a predictable approach but one that is inflexible because any change to any task requirements requires that the schedule be redone
- Earliest-deadline-first (EDF) or other periodic deadline techniques are typical of this category

Static priority-driven preemptive scheduling

- Makes use of the priority-driven preemptive scheduling mechanism common to most non-real-time multiprogramming systems
- Priority assignment is related to the time constraints associated with each task
- Example: rate monotonic algorithm (RMS) which assigns static priorities to tasks based on the length of their periods

Dynamic Scheduling Algorithms

Dynamic planning-based scheduling

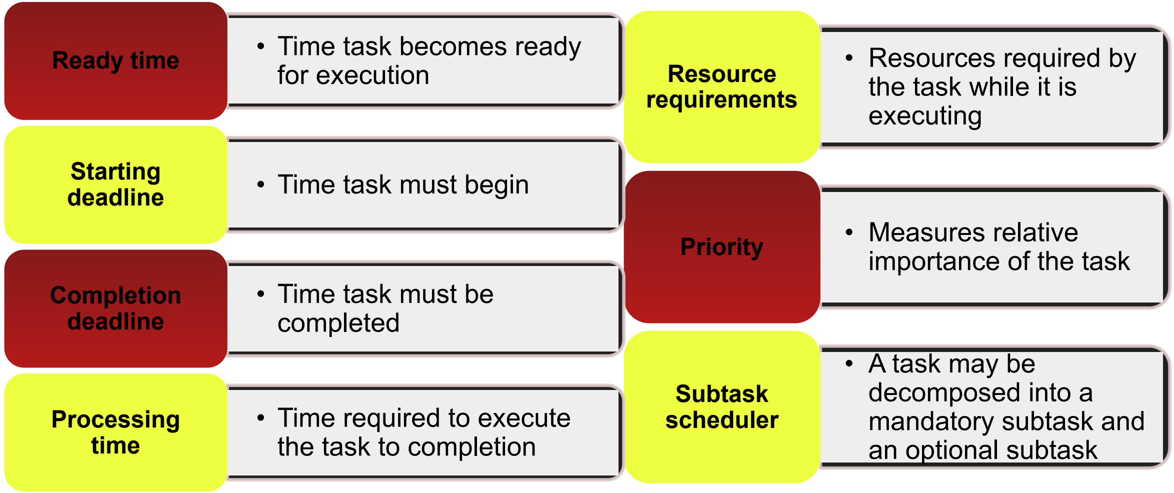
- After a task arrives, but before its execution begins, an attempt is made to create a schedule that contains the previously scheduled tasks as well as the new arrival
- If the new arrival can be scheduled in such a way that its deadlines are satisfied and that no currently scheduled task misses a deadline, then the schedule is revised to accommodate the new task

Dynamic best effort scheduling

- The approach used by many real-time systems that are currently commercially available
- **When a task arrives, the system assigns a priority based on the characteristics of the task**
- **Some form of deadline scheduling is typically used**
- Typically the tasks are aperiodic so no static scheduling analysis is possible
- The major disadvantage of this form of scheduling is, that until a deadline arrives or until the task completes, we do not know whether a timing constraint will be met
- Its advantage is that it is easy to implement

Deadline Scheduling

- Real-time operating systems
  - designed with the objective of starting real-time tasks as rapidly as possible and emphasize rapid interrupt handling and task dispatching
  - Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times
- Priorities provide a crude tool and do not capture the requirement of completion (or initiation) at the most valuable time



Ready time	• Time task becomes ready for execution
Starting deadline	• Time task must begin
Completion deadline	• Time task must be completed
Processing time	• Time required to execute the task to completion
Resource requirements	• Resources required by the task while it is executing
Priority	• Measures relative importance of the task
Subtask scheduler	• A task may be decomposed into a mandatory subtask and an optional subtask

Schedule Examples

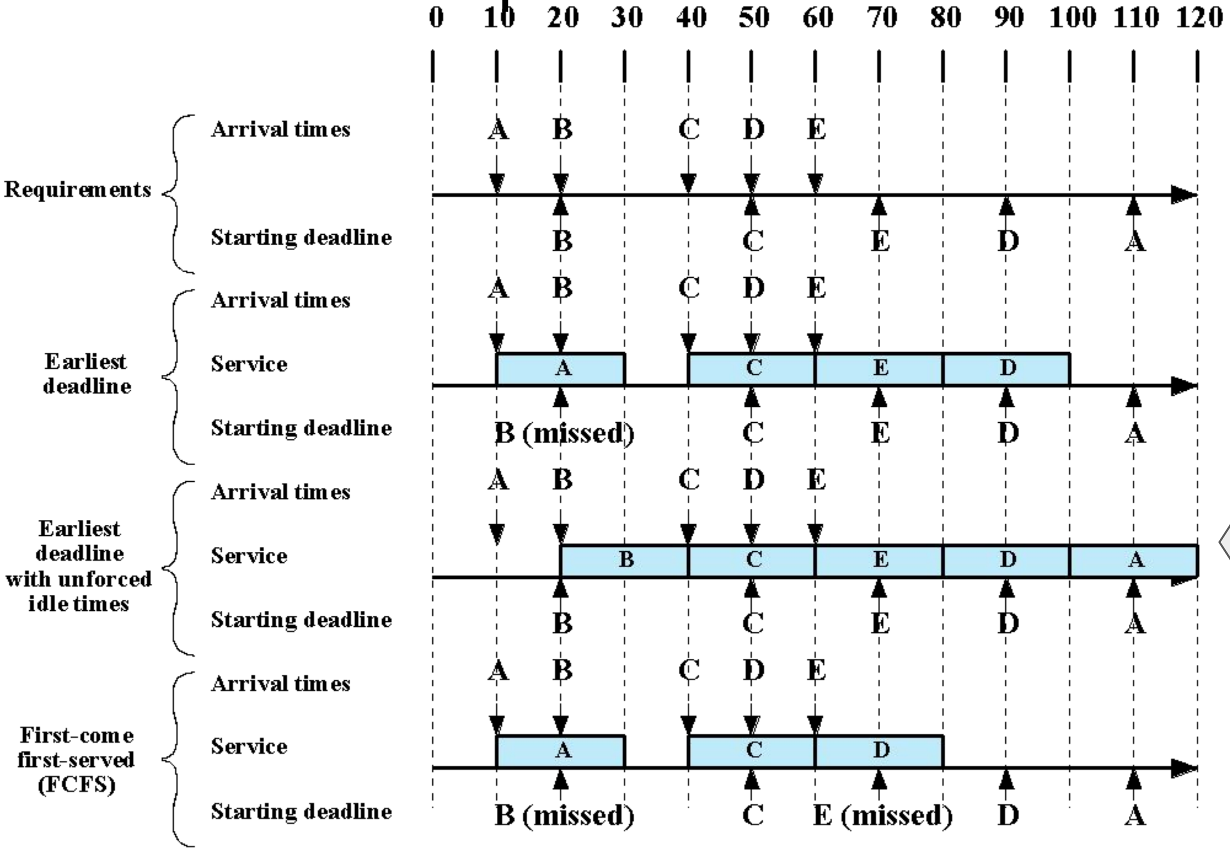
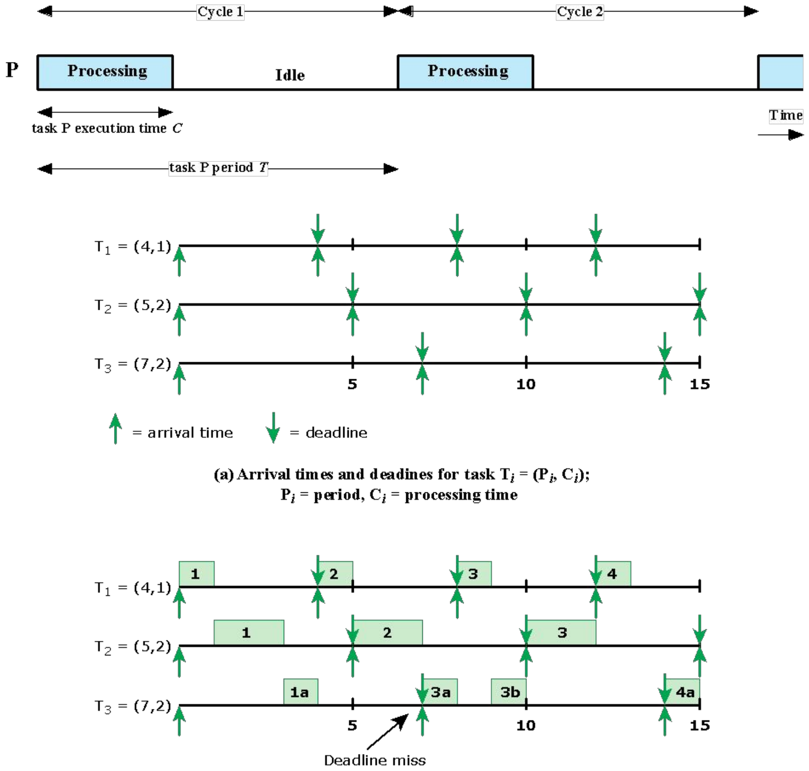


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

- Always schedule the eligible task with the **earliest deadline** and let that task run to completion.
- An eligible task may not be ready, and this may result in the processor remaining idle

Rate Monotonic Scheduling



(a) Arrival times and deadlines for task $T_i = (P_i, C_i)$;
P_i = period, C_i = processing time

(b) Scheduling results

Assigns priorities to tasks on the basis of their periods

- the highest-priority task is the one with the shortest period
- the second highest-priority task is the one with the second shortest period, and so on.
- When more than one task is available for execution, the one with the shortest period is serviced first.

Plot the priority of tasks as a function of their rate, the result is a monotonically increasing function, hence the name, “rate monotonic scheduling.”

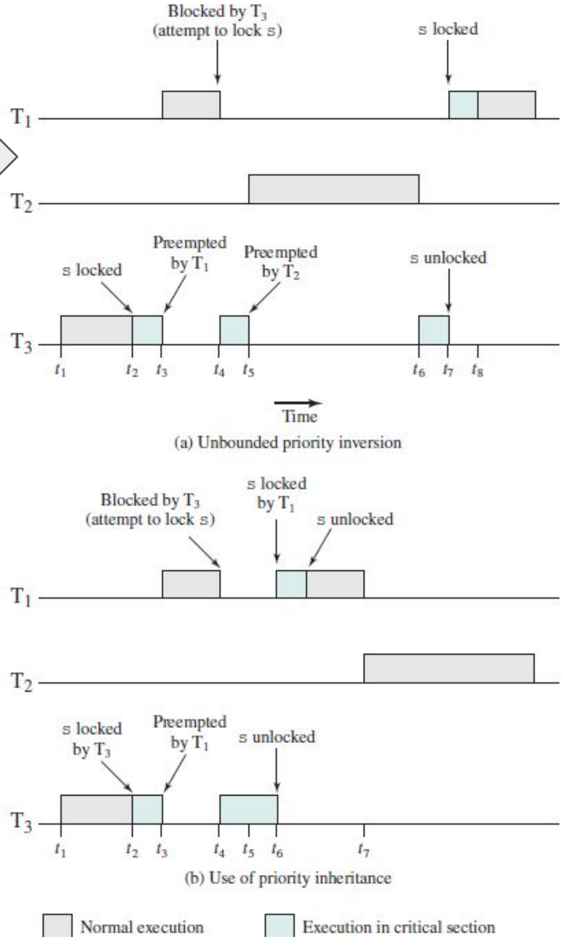
Example:

- Task instances are numbered sequentially with in tasks.
- For task 3, the second instance is not executed because the deadline is missed. The third instance of task 3 experiences a preemption but is still able to complete before the deadline

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**: The duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks
- **Priority inheritance**
  - a lower-priority task inherits the priority of any higher-priority task pending on a resource they share
  - change takes place as soon as the higher-priority task blocks on the resource; it should end when the resource is released by the lower-priority task.

T1 must wait for both T3 and T2 to complete and fails to meet the deadline (e.g. reset a timer before it expires)



Blocked by T_3 (attempt to lock s)
s locked
T_1
T_2
s locked
Preempted by T_1
Preempted by T_2
s unlocked
T_3
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
Time
Blocked by T_3 (attempt to lock s)
s locked by T_1
s unlocked
T_1
T_2
s locked by T_3
Preempted by T_1
s unlocked
T_3
t_1 t_2 t_3 t_4 t_5 t_6 t_7
Normal execution
Execution in critical section

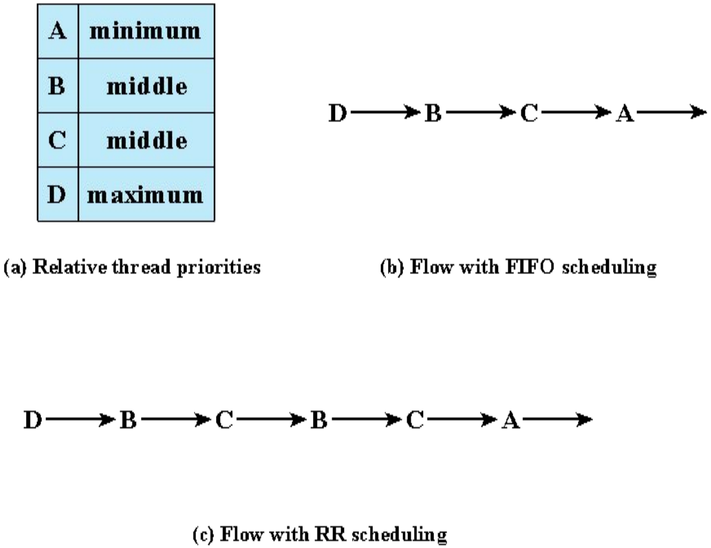
(a) Unbounded priority inversion

(b) Use of priority inheritance

Linux Scheduling

- The three primary Linux scheduling classes are:
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_NORMAL: Other, non-real-time threads

- Within each class multiple priorities may be used, with priorities in the real-time classes higher than the priorities for the SCHED_NORMAL class



A	minimum
B	middle
C	middle
D	maximum

(a) Relative thread priorities

D → B → C → A →

(b) Flow with FIFO scheduling

D → B → C → B → C → A →

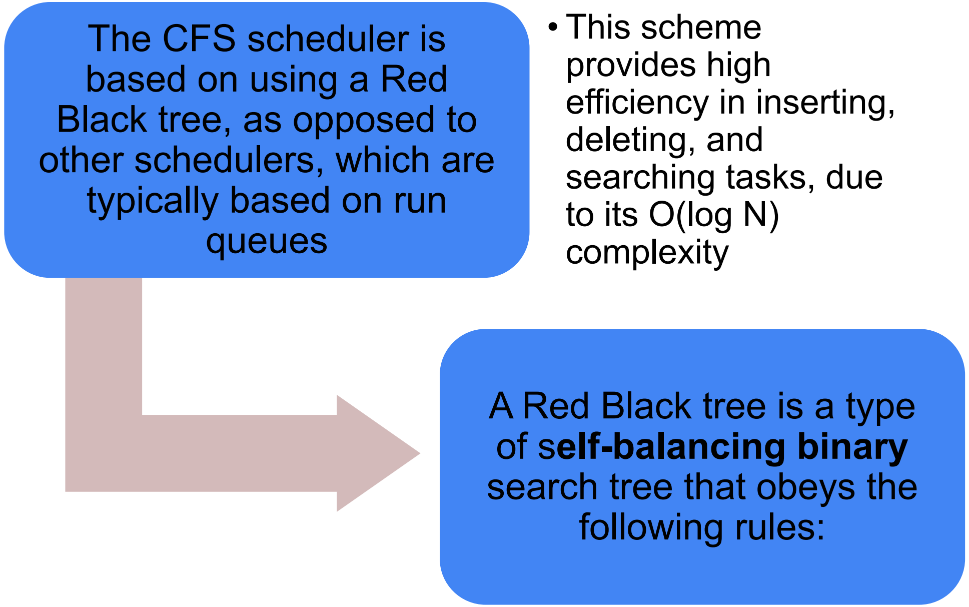
(c) Flow with RR scheduling

Figure 10.10 Example of Linux Real-Time Scheduling

Non-Real-Time Scheduling

- The Linux 2.4 scheduler for the SCHED_OTHER class did not scale well with increasing number of processors and increasing number of processes
- Linux 2.6 uses a completely new priority scheduler known as the O(1) scheduler
  - designed so the time to select the appropriate process and assign it to a processor is constant regardless of the load on the system or number of processors
  - The O(1) scheduler proved to be unwieldy in the kernel because the amount of code is large and the algorithms are complex
- Completely Fair Scheduler (CFS)
  - Used as a result of the drawbacks of the O(1) scheduler
  - Models an ideal multitasking CPU on real hardware that provides fair access to all tasks
  - In order to achieve this goal, the CFS maintains a virtual runtime for each task
    - The virtual runtime is the amount of time spent executing so far, normalized by the number of runnable processes
    - The smaller a task’s virtual runtime is, the higher is its need for the processor
  - Includes the concept of sleeper fairness to ensure that tasks that are not currently runnable receive a comparable share of the processor when they eventually need it

Red Black Tree



The CFS scheduler is based on using a Red Black tree, as opposed to other schedulers, which are typically based on run queues

- This scheme provides high efficiency in inserting, deleting, and searching tasks, due to its O(log N) complexity

A Red Black tree is a type of s**elf-balancing binary** search tree that obeys the following rules:

- A node is either red or black
- The root is black
- All leaves (NIL) are black
- If a node is red, then both its children are black
- Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes

CFS Scheduler Red Black Tree

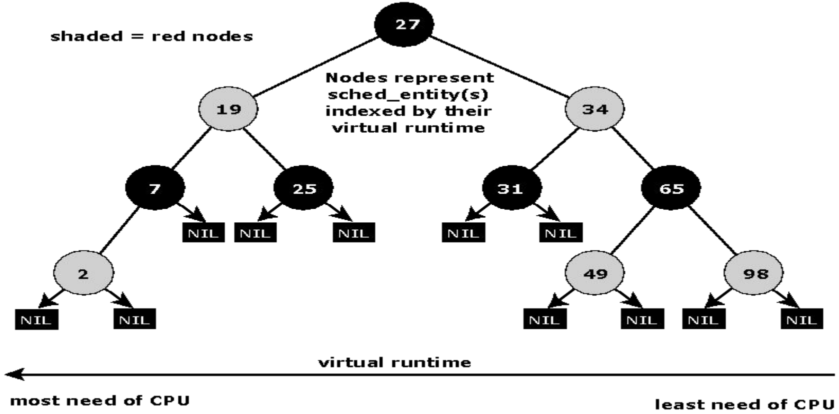


Figure 10.11 Example of Red Black Tree for CFS

The rb_node elements of the tree are embedded in *sched_entity* object.

The Red Black tree is ordered by *vruntime*

- the leftmost node of the tree represents the process that has the lowest , and that has the highest need for CPU
- this leftmost is the first one to be picked by the CPU
- when it runs, its *vruntime* is updated according to the time it consumed;
- when it is inserted back into the tree, very likely it will no longer be the one with the lowest *vruntime* ,
- the tree will be rebalanced to reflect the current state

While an insert operation is being performed in RB tree, the leftmost child value is cached in sched_entity for fast lookup.