# Escalonamento de Processos

Chapter 9, Livro do William Stallings
Chapter 6, Livro do Silberschatz

Sistemas Operativos, 2004-2005

## Types of 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 |

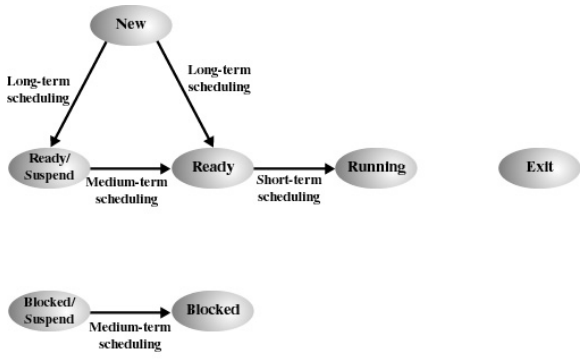



Figure 9.1 Scheduling and Process State Transitions

## Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

## Medium-Term Scheduling

- Part of the swapping function
- Based on the need to manage the degree of multiprogramming

## Short-Term Scheduling

- Known as the **dispatcher**
- Executes most frequently
- Invoked when an event occurs
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

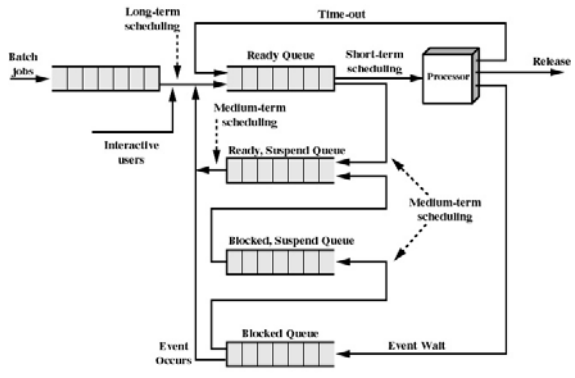



Figure 9.3 Queuing Diagram for Scheduling

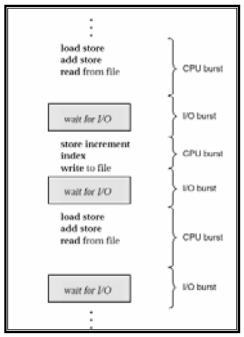
## Scheduling

- Maximize CPU utilization
- Process execution consists of a *cycle* of CPU execution and I/O wait:

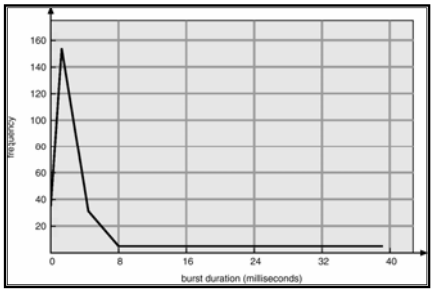
CPU Burst….. I/O Burst Cycle

- Adapt the Scheduling Algorithm to the type of the program.

## Alternating Sequence of CPU And I/O Bursts




## Histogram of CPU-burst Times




## CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.

## Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- ***Dispatch latency*** – time it takes for the dispatcher to stop one process and start another running.

## Preemptive vs Non-Preemptive

- **Non-Preemptive Scheduling**: once the CPU is allocated to a process, the process keeps the CPU until it terminates or it blocks in a I/O operations and switches to the waiting state (used by Windows 3.1).
- **Preemptive Scheduling**: when a new process arrives it can take the CPU out of the running process.

## Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible.
- **Throughput** – # of processes that complete their execution per time unit.
- **Turnaround time** – amount of time to execute a particular process.
- **Waiting time** – amount of time a process has been waiting in the ready queue.
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced.

## Optimization Criteria

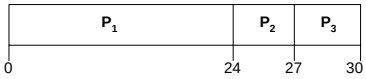
- **Max CPU utilization**
- **Max throughput**
- **Min turnaround time**
- **Min waiting time**
- **Min response time**

## Scheduling Algorithms

## First-Come, First-Served (FCFS)

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$
- *Arrival time=0,0.1,0.2*
- The Gantt Chart for the schedule is:



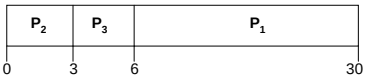

- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- This is a non-preemptive algorithm.

## FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$P_2$ , $P_3$ , $P_1$ .

- The Gantt chart for the schedule is:




- Waiting time for $P_1$ = 6; $P_2$ = 0, $P_3$ = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- *Convoy effect*: short process behind long process

## First-Come-First-Served

| Process | Arrival Time | Service Time (Ts) | Start Time | Finish Time | Turnaround Time (Tr) | Ts / Tr |
|---|---|---|---|---|---|---|
| W | 0 | 1 | 0 | 1 | 1 | 1 |
| X | 1 | 100 | 1 | 101 | 100 | 1 |
| Y | 2 | 1 | 101 | 102 | 100 | 100 |
| Z | 3 | 100 | 102 | 202 | 199 | 1.99 |
| mean | | | | | 100 | 26 |

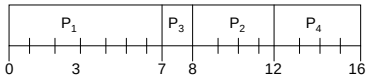
## Shortest-Process-Next (SPN)

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process **arrives** with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time (SRT).
- SPN is optimal – gives minimum average waiting time for a given set of processes.
- Difficulty: how to know the length of the next CPU request.

## Non-Preemptive SPN

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SPN (non-preemptive)

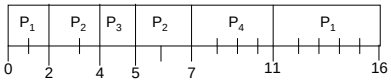



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

## Example of Preemptive SRT

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SRT (preemptive)



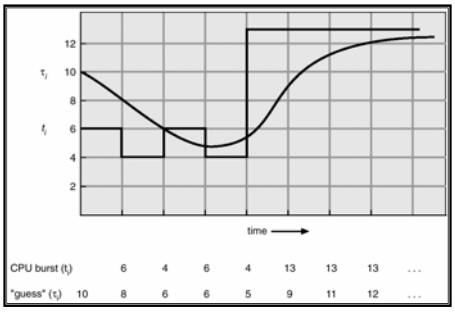

- Average waiting time = (9 + 1 + 0 +2)/4 = 3

## Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

1. $t_n$ = actual lenght of $n^{th}$ CPU burst
2. $\tau_{n+1}$ = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n.$

## Prediction of the Length of the Next CPU Burst



## Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_n - 1 + \ldots$$
$$+ (1 - \alpha)^j\, \alpha\, t_n - 1 + \ldots$$
$$+ (1 - \alpha)^{n=1}\, t_n\, \tau_0$$

- Since both $\alpha$ and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

## Priority Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
  - **Preemptive**
  - **Nonpreemptive**
- **Problem ≡** Starvation – low priority processes may never execute.
- **Solution ≡** Aging – as time progresses increase the priority of the process.

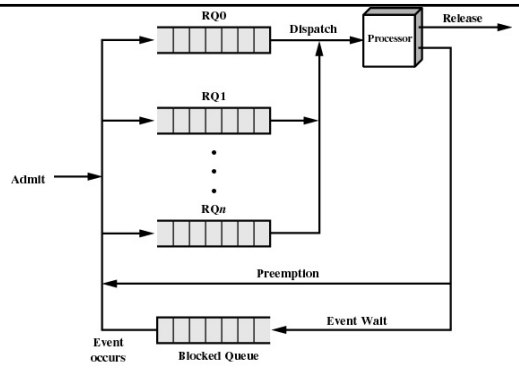



Figure 9.4 Priority Queuing

## Priorities

- Scheduler will always choose a process of higher priority over one of lower priority.
- Have multiple ready queues to represent each level of priority.
- Lower-priority may suffer from **starvation**
  - allow a process to change its priority based on its age or execution history

## Starvation: interesting note...

- When the system managers shutdown the IBM 7094 at MIT in 1973, they found a low-priority process that had been submitted in 1967 and had not yet run....

IBM 7094

## Starvation...

- Os adeptos destes dois clubes sabem bem o que é este fenómeno de starvation...


- Tudo por causa deste "processo" de alta-prioridade...

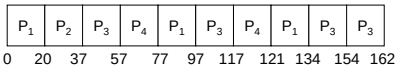

## Round Robin (RR)

- Each process gets a small unit of CPU time (***time quantum***), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high.

## Example of RR with Time Quantum = 20

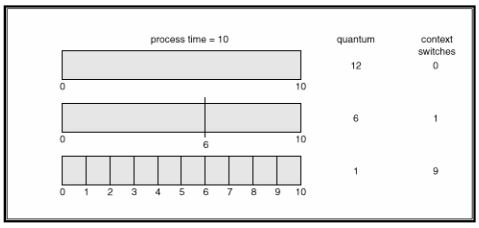
| Process | Burst Time |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

- The Gantt chart is:

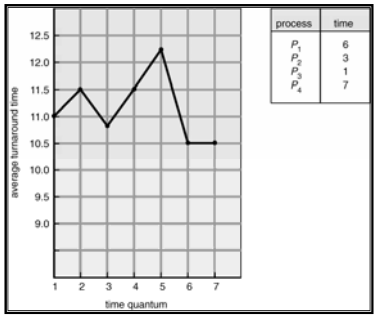



Typically, higher average turnaround than SJF, but better *response*.

## Time Quantum and Context Switch Time




## Turnaround Time Varies With The Time Quantum




## Go to quiz#2...

## FCFS vs SPN vs RR vs SRT

## Highest Response Ratio Next (HRRN)

- Choose next process with the highest ratio (R):

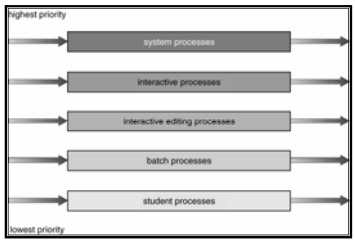
$$R = \frac{\text{time spent waiting} + \text{expected service time}}{\text{expected service time}}$$

- This algorithm accounts for the age of the process.
- Shorted jobs are favored (smaller denominator)
- But aging without service (waiting time) increases R.

## Multilevel Queue

- Ready queue is partitioned into separate queues:
  - *foreground (interactive)*
  - *background (batch)*
- Each queue has its own scheduling algorithm,
  foreground – RR
  background – FCFS
- Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all processes 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; i.e., 80% to foreground in RR; 20% to background in FCFS

## Multilevel Queue Scheduling




## Multilevel Feedback Queue

- A process can move between the various queues; **aging** can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

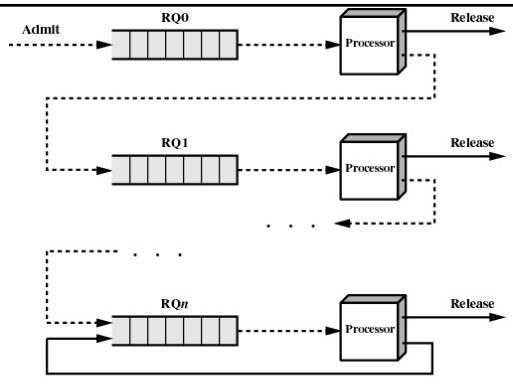
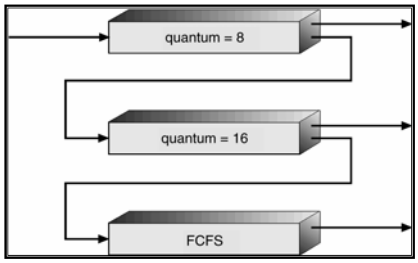



Figure 9.10 Feedback Scheduling

## Multilevel Feedback Queues




## Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – time quantum 8 milliseconds
  - $Q_1$ – time quantum 16 milliseconds
  - $Q_2$ – FCFS
- Scheduling
  - A new job enters queue $Q_0$ which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue $Q_1$.
  - At $Q_1$ job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$.

## Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies

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

$w$ = time spent in system so far, waiting and executing
$e$ = time spent in execution so far
$s$ = total service time required by the process, including $e$

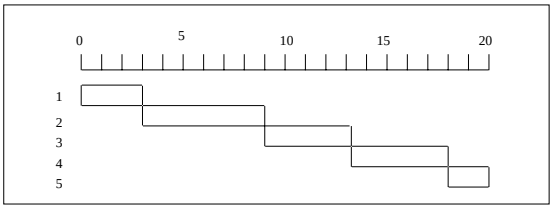
## Process Scheduling Example

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

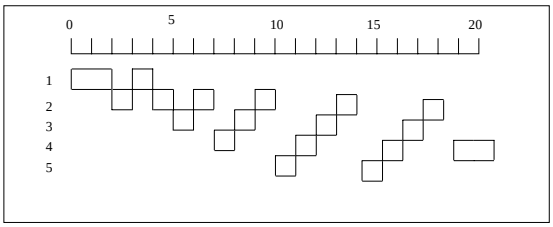
## Escalonamento

- Desenhe o histograma de execução dos 5 processos, usando os seguintes algoritmos:
  - **FCFS**
  - **Round-Robin (q=1)**
  - **Round-Robin (q=4)**
  - **Shortest Process Next (SPN)**
  - **Shortest Remaining Time (SRT)**
  - **Highest Response Ratio (HRRN)**
  - **Feedback (q=1)**
- Calcule o Finish-Time, Waiting-Time, Turnaround Time para cada processo.
- Apresente os valores médios do Waiting Time e do Turnaround Time.

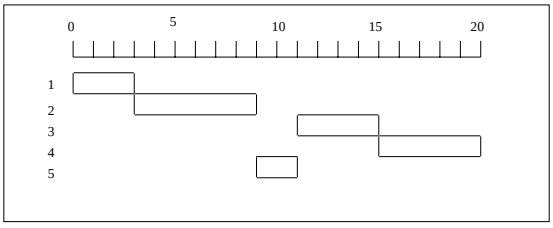
## First-Come-First-Served (FCFS)




## Round-Robin (q=1)

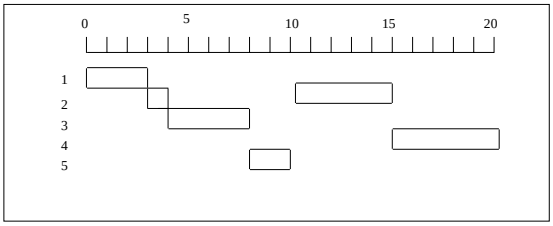



## Shortest Process Next (SPN)



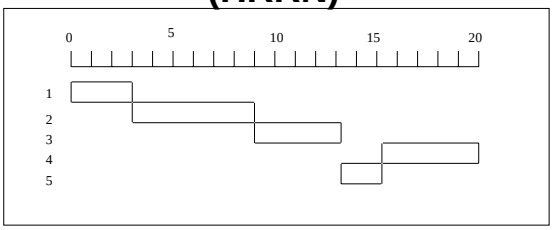

- Non-preemptive policy.
- Process with shortest expected processing time is selected next.
- Short process jumps ahead of longer processes.

## Shortest Remaining Time (SRT)




- Preemptive version of shortest process next policy.
- Must estimate processing time.

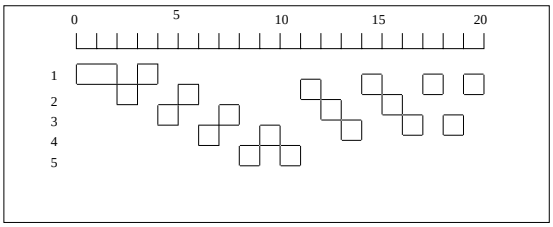
## Highest Response Ratio Next (HRRN)




- Choose next process with the highest ratio:

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

## Multilevel Feedback




- Penalize jobs that have been running longer.

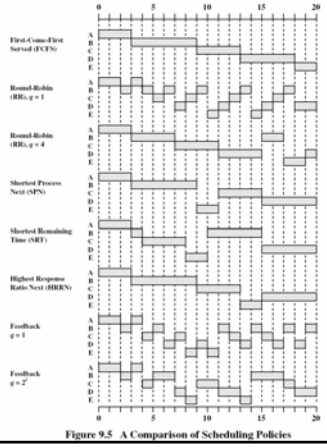



Figure 9.5 A Comparison of Scheduling Policies

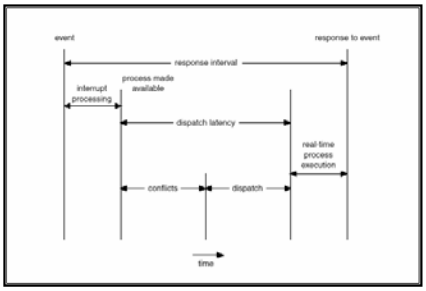
**Table 9.5 A Comparison of Scheduling Policies**

| | Process | A | B | C | D | E | Mean |
|---|---|---|---|---|---|---|---|
| | Arrival Time | 0 | 2 | 4 | 6 | 8 | |
| | Service Time ($T_s$) | 3 | 6 | 4 | 5 | 2 | |
| 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 $q = 2^i$ | 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 |

## 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.

## Dispatch Latency




## Overview of Scheduling:

**- Traditional Unix**
**- Solaris**
**- Windows 2000**
**- Linux**

# Traditional UNIX Scheduling

- Multilevel feedback using Round-Robin within each of the priority queues.
- Priorities are recomputed once per second.
- Base priority divides all processes into fixed bands of priority levels.
- Adjustment factor used to keep process in its assigned band.

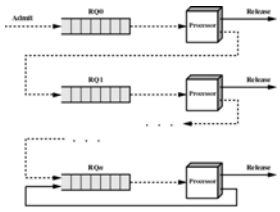



Figure 9.10 Feedback Scheduling

# Bands

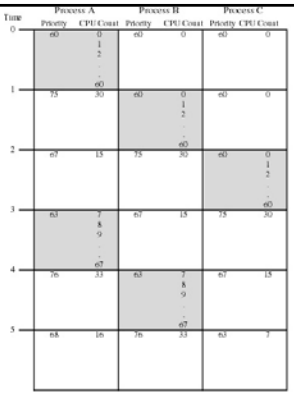
- Decreasing order of priority:
  - Swapper
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes

# Priority Calculation

- Process $P_x$
- $CPU_x(i) = CPU_x (i-1) / 2$
- $P_x(i) = Base_x + (CPU_x(i) / 2) + nice_x$
- The priority of each process is recomputed once per second.

Base priority=60

CPU count: incremented very second




Shaded rectangle represents executing process

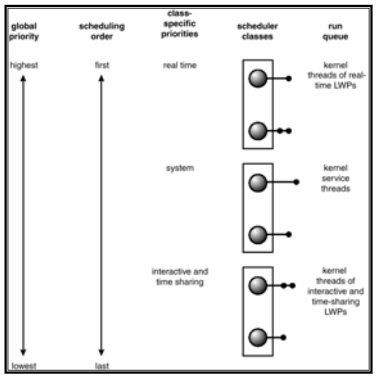
Figure 9.17 Example of Traditional UNIX Process Scheduling

## Solaris 2 Scheduling

Priority Scheduling:

4 classes:

- real-time
- system
- interactive
- time-sharing




## Windows 2000 Priorities

Priority classes

| | 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 |

Relative priority

**Priority-based, preemptive scheduling**

## Linux

- Linux provides two separate process-scheduling algorithms:
  - Time-Sharing Algorithm (fair preemptive)
  - Soft Real-Time Scheduling (priority-based)
- Linux allows only processes running in user-mode to be preempted.
- If a process is running in kernel-mode it cannot be preempted.

## Linux: Scheduling

- Linux uses a prioritized credit-based algorithm.
- When a new task must be chosen to run, the process with the most credits is selected.
- Every time that a timer interrupt occurs, the running process loses one credit. When its credits reaches zero, it is suspended and another process is chosen.
- If no runnable process has any credit, Linux performs a recrediting operation, adding credits to every process in the system:

  Credits = Credits/2 + Priority;