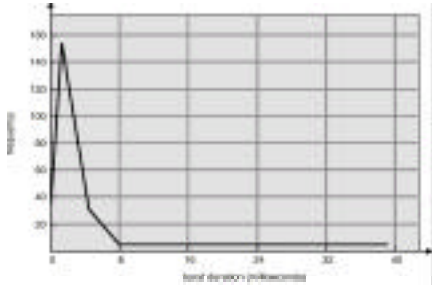


Histogram of CPU-burst Times



Applied Operating System Concepts 6.4 Silberschatz, Galvin, and Gagne 1999

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.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive*.

Applied Operating System Concepts 6.5 Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts 6.6 Silberschatz, Galvin, and Gagne 1999

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, **not** output (for time-sharing environment)

Applied Operating System Concepts

6.7

Silberschatz, Galvin, and Gagne 1999

Optimization Criteria

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

Applied Operating System Concepts

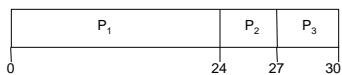
6.8

Silberschatz, Galvin, and Gagne 1999

First-Come, First-Served (FCFS) Scheduling

- Example:

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
The Gantt Chart for the schedule is:



- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

Applied Operating System Concepts

6.9

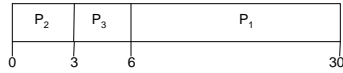
Silberschatz, Galvin, and Gagne 1999

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

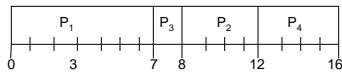
Shortest-Job-First (SJR) Scheduling

- 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 known as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

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

- SJF (non-preemptive)



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

Example of Preemptive SJF

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

- SJF (preemptive)

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

Applied Operating System Concepts 6.13 Silberschatz, Galvin, and Gagne 1999

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 length of n^{th} CPU burst
 2. τ_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 < \alpha < 1$
 4. Define:

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

Applied Operating System Concepts 6.14 Silberschatz, Galvin, and Gagne 1999

Examples of Exponential Averaging

- $\alpha = 0$
 - $\tau_{n+1} = t_n$
 - 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:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots + (1 - \alpha)^{n-1} \alpha t_1 + (1 - \alpha)^n \tau_0$$
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

Applied Operating System Concepts 6.15 Silberschatz, Galvin, and Gagne 1999

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
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem Starvation – low priority processes may never execute.
- Solution Aging – as time progresses increase the priority of the process.

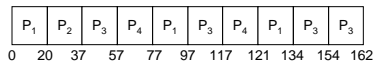
Round Robin (RR)

- 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: RR with Time Quantum = 20

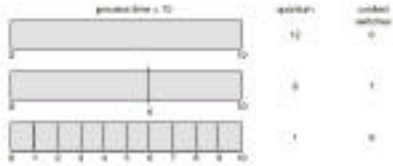
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



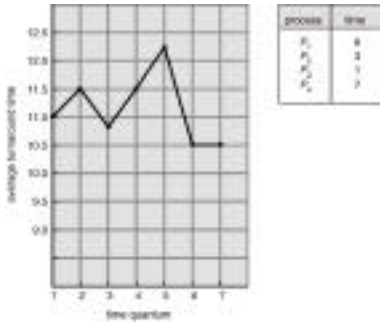
- Typically, higher average turnaround than SJF, but better response.

How a Smaller Time Quantum Increases Context Switches



Applied Operating System Concepts 6.19 Silberschatz, Galvin, and Gagne 1999

Turnaround Time Varies With The Time Quantum

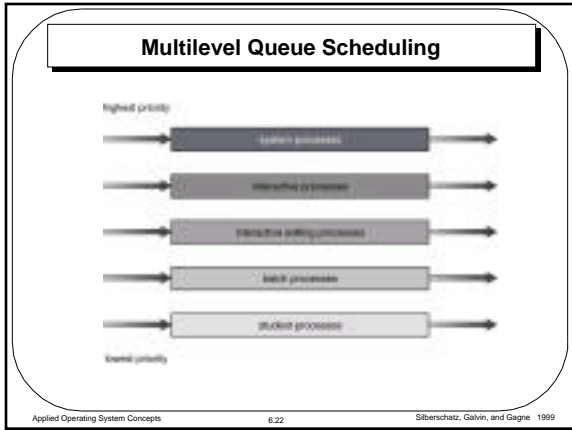


Applied Operating System Concepts 6.20 Silberschatz, Galvin, and Gagne 1999

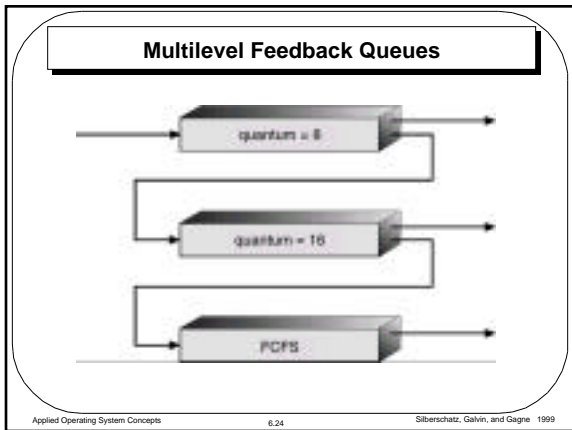
Multilevel Queue

- 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 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

Applied Operating System Concepts 6.21 Silberschatz, Galvin, and Gagne 1999



- ### 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
- Applied Operating System Concepts 6.23 Silberschatz, Galvin, and Gagne 1999



Example of 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 .

Applied Operating System Concepts 6.25 Silberschatz, Galvin, and Gagne 1999

Multiple-Processor Scheduling

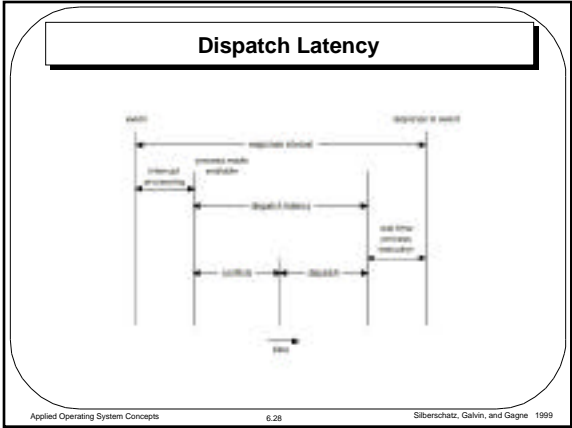
- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Symmetric Multiprocessing (SMP)* – each processor makes its own scheduling decisions.
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.

Applied Operating System Concepts 6.26 Silberschatz, Galvin, and Gagne 1999

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.

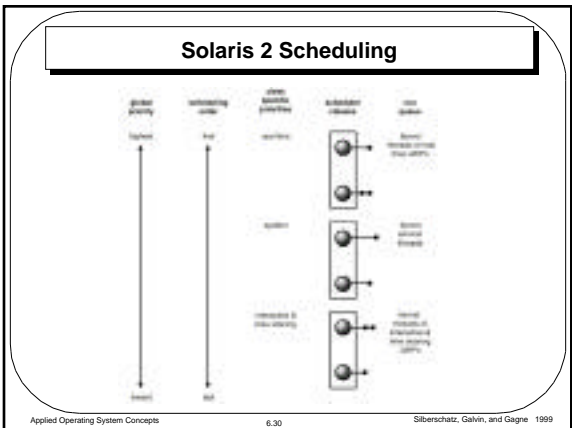
Applied Operating System Concepts 6.27 Silberschatz, Galvin, and Gagne 1999



Thread Scheduling

- Local Scheduling – How the threads library decides which thread to put onto an available LWP.
- Global Scheduling – How the kernel decides which kernel thread to run next.

Applied Operating System Concepts 6.29 Silberschatz, Galvin, and Gagne 1999



Java Thread Scheduling

- JVM Uses a Preemptive, Priority-Based Scheduling Algorithm.
- FIFO Queue is Used if There Are Multiple Threads With the Same Priority.

Applied Operating System Concepts 6.31 Silberschatz, Galvin, and Gagne 1999

Java Thread Scheduling (cont)

JVM Schedules a Thread to Run When:

- The Currently Running Thread Exits the Runnable State.
- A Higher Priority Thread Enters the Runnable State

* Note – the JVM Does Not Specify Whether Threads are Time-Sliced or Not.

Applied Operating System Concepts 6.32 Silberschatz, Galvin, and Gagne 1999

Time-Slicing

- Since the JVM Doesn't Ensure Time-Slicing, the `yield()` Method May Be Used:

```
while (true) {  
    // perform CPU-intensive task  
    ...  
    Thread.yield();  
}
```

This Yields Control to Another Thread of Equal Priority.

Applied Operating System Concepts 6.33 Silberschatz, Galvin, and Gagne 1999

Thread Priorities

- Thread Priorities:

Priority	Comment
Thread.MIN_PRIORITY	Minimum Thread Priority
Thread.MAX_PRIORITY	Maximum Thread Priority
Thread.NORM_PRIORITY	Default Thread Priority

Priorities May Be Set Using setPriority() method:
 setPriority(Thread.NORM_PRIORITY + 2);

Applied Operating System Concepts 6.34 Silberschatz, Galvin, and Gagne 1999

Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queuing models
- Implementation

Applied Operating System Concepts 6.35 Silberschatz, Galvin, and Gagne 1999

Evaluation of CPU Schedulers by Simulation

Applied Operating System Concepts 6.36 Silberschatz, Galvin, and Gagne 1999
