Uniprocessor Scheduling

Chapter 3

1

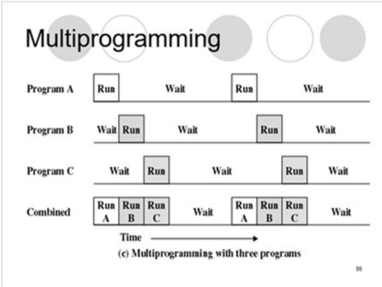
Chapter Objectives

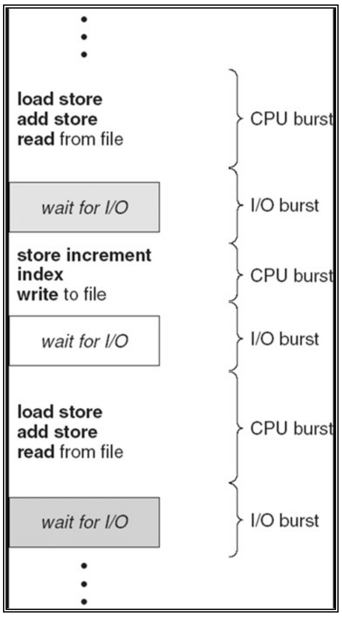
1. To introduce CPU scheduling, which is the basis for multiprogrammed OS
2. To describe various CPU-scheduling algorithms
3. To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system.

2

2

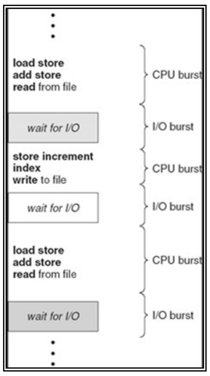
Alternating Sequence of CPU And I/O Bursts





3

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 blocking state
  2. Switches from running to ready state
  3. Switches from blocking to ready
  4. Terminates

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

4

Decision Mode

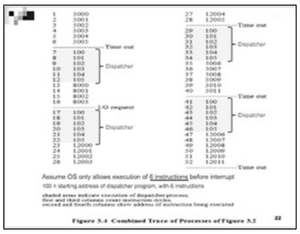
- Nonpreemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

5

5

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

6

6

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – number 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)

7

7

Optimization Criteria

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

8

8

Aim of Scheduling

- Assign processes to be executed by the processor(s)
- Response time
- Throughput
- Processor efficiency

Table 9.1 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

9

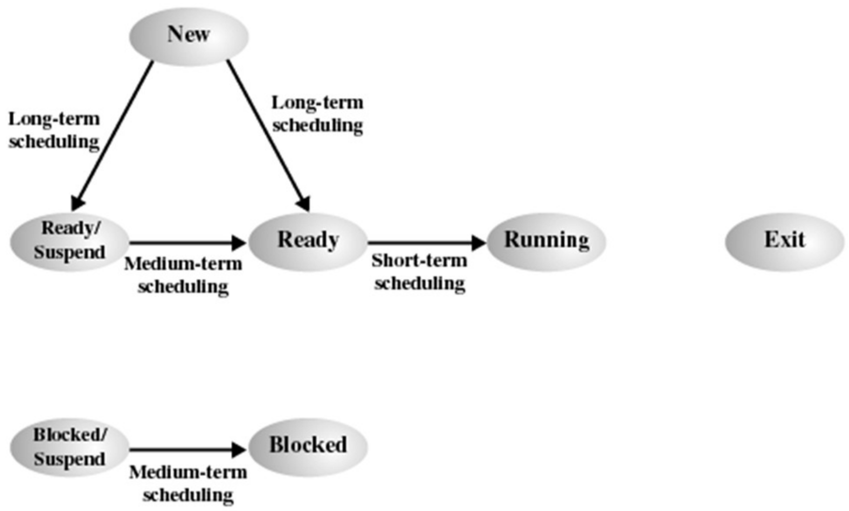


Figure 9.1 Scheduling and Process State Transitions

10

10

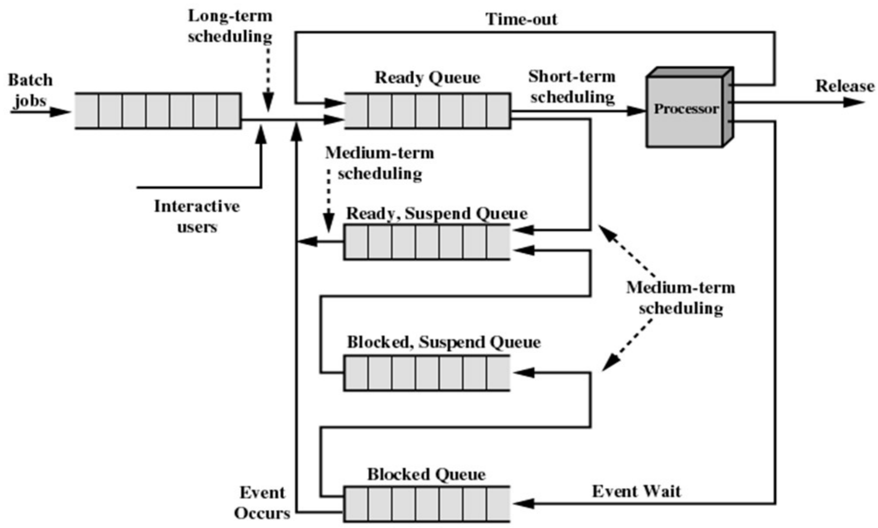


Figure 9.3 Queuing Diagram for Scheduling

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

13

13

Short-Tem Scheduling Criteria

- User-oriented
  - Response Time
    - Elapsed time between the submission of a request until there is output.
- System-oriented
  - Effective and efficient utilization of the processor
- Performance-related
  - Quantitative
  - Measurable such as response time and throughput

14

14

Table 9.2 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.

15

15

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.

16

Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
  - non-preemptive
  - preemptive
- Round Robin
- Priority
  - Non-preemptive
  - Preemptive

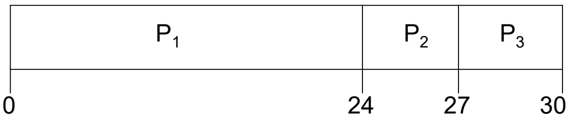
17

17

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- All processes arrive at time = 0.
- 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

18

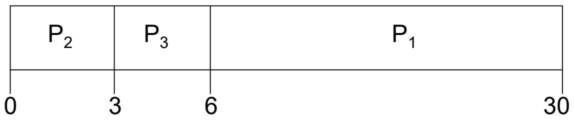
18

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- All processes arrive at time = 0.
- 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

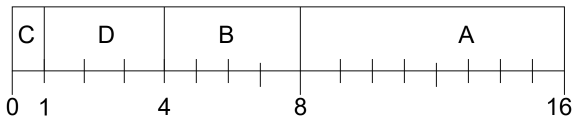
Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - Non-preemptive – 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 also known as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of SJF

Process	Arrival Time	Burst Time
A	0.0	8
B	0.0	4
C	0.0	1
D	0.0	3

- SJF



- Average waiting time = (8 + 4 + 0 + 1)/4 = 3.25

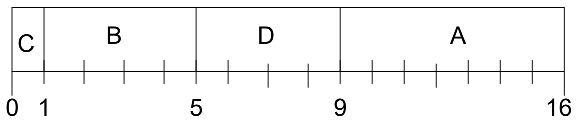
21

21

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
A	0.0	7
B	0.0	4
C	0.0	1
D	0.0	4

- SJF (non-preemptive)



- Average waiting time = (9 + 1 + 0 + 5)/4 = 3.75

22

22

Example of Non-Preemptive SJF

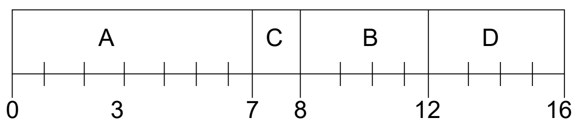
Process	Arrival Time	Burst Time
A	0.0	7
B	0.0	4
C	0.0	1
D	0.0	4

	0					5					10					15
A																
B																
C																
D																

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

- SJF (non-preemptive)



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

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

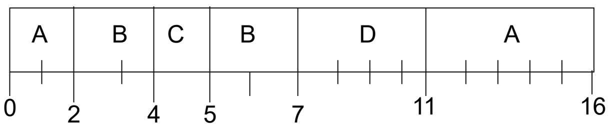
	0					5					10					15
A																
B																
C																
D																

25

Example of Preemptive SJF

Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

- SJF (preemptive)



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

26

Example of Preemptive SJF

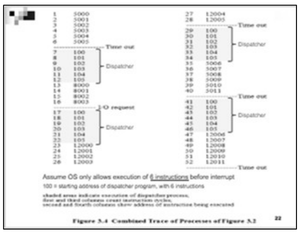
Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

	0					5					10					15
A																
B																
C																
D																

27

27

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

28

28

Example of Round Robin (RR)

Process	Arrival Time	Burst Time
A	0.0	7
B	0.0	4
C	0.0	1
D	0.0	4

- Time quantum, q = 3
- Average waiting time?

29

29

Example of Round Robin (RR)

Process	Arrival Time	Burst Time
A	0.0	7
B	0.0	4
C	0.0	1
D	0.0	4

q = 3

	0					5					10					15
A																
B																
C																
D																

30

30

RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- All processes arrive at time = 0.
- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

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

31

31

Example of Round Robin (RR)

Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

- q = 3
- Average waiting time?

32

32

Example of Round Robin (RR)

Process	Arrival Time	Burst Time
A	0.0	7
B	2.0	4
C	4.0	1
D	5.0	4

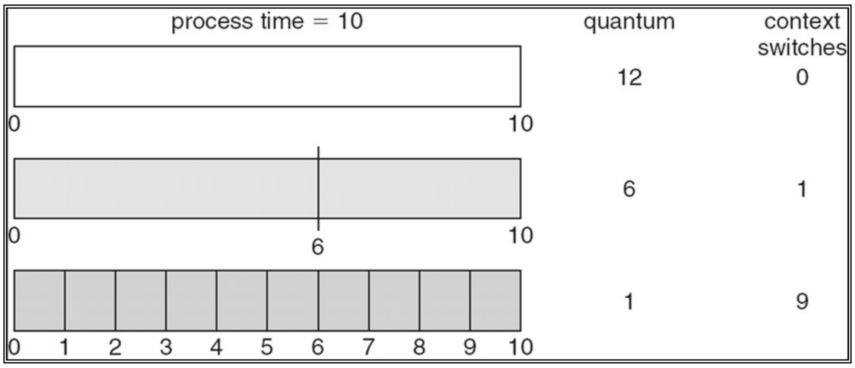
q = 3

	0					5					10					15
A																
B																
C																
D																

33

33

Time Quantum and Context Switch Time



34

34

Process Scheduling Example

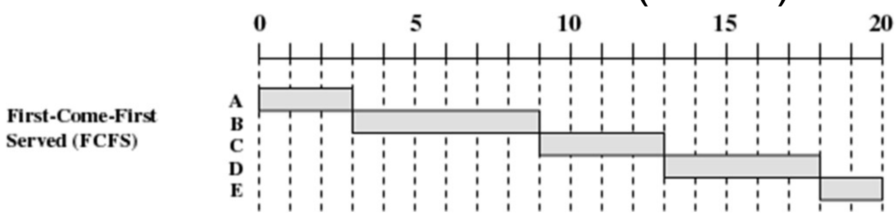
Process Name	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Algorithms:
FCFS
SJF nonpreemptive
SJF preemptive
RR (q=2)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

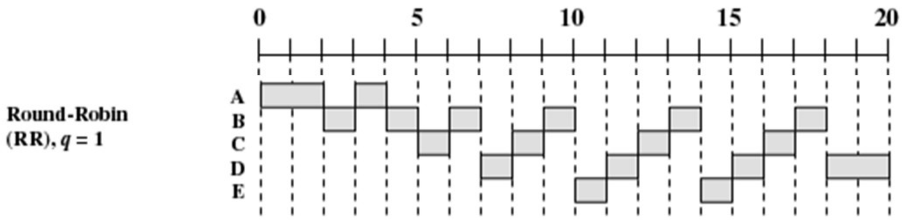
A																				
B																				
C																				
D																				
E																				

First-Come-First-Served (FCFS)



- Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected
- A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes

Round-Robin



- Uses preemption based on a clock
- An amount of time is determined that allows each process to use the processor for that length of time
- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the read queue
  - Next ready job is selected
- Known as time slicing

37

37

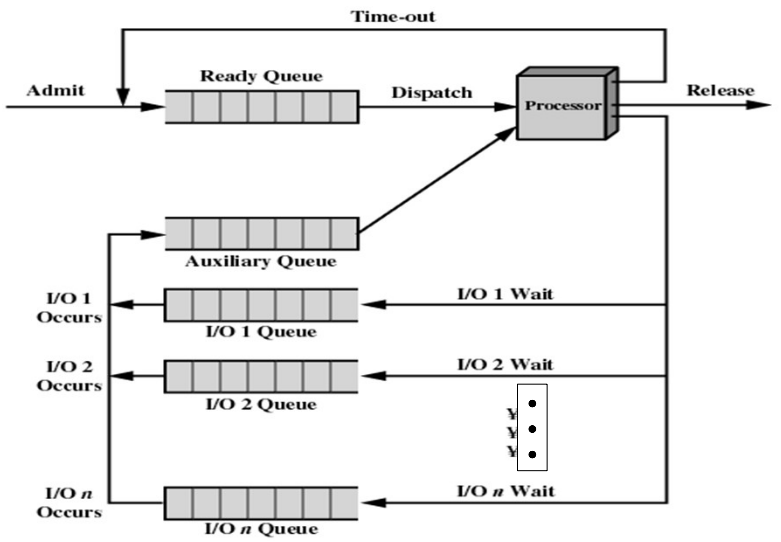
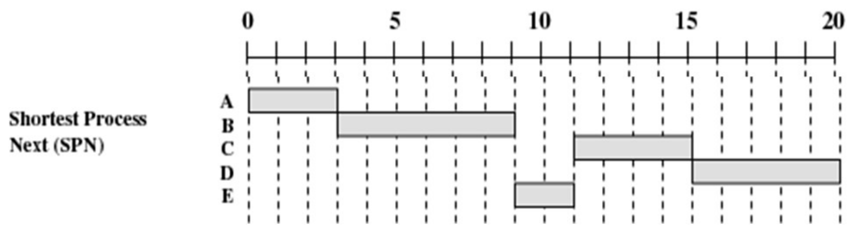


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

38

38

SJF Non-preemptive

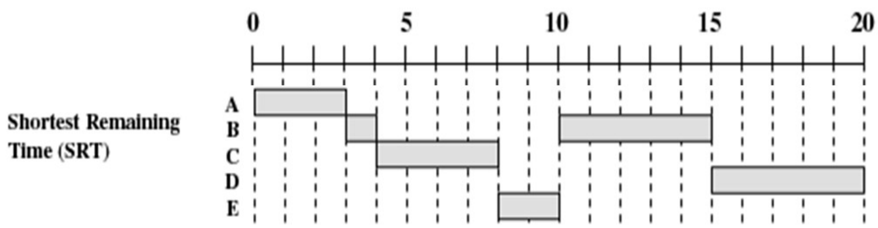


- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes
- Predictability of longer processes is reduced
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes

39

39

SJF Preemptive



- A.k.a shortest remaining time
- Preemptive version of Non-preemptive SJF (a.k.a shortest process next) policy
- Must estimate processing time

40

40

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 waiting
e = time spent in execution so far
s = total service time required by the process, including e

41

41

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 starvation
  - Allow a process to change its priority based on its age or execution history

42

42

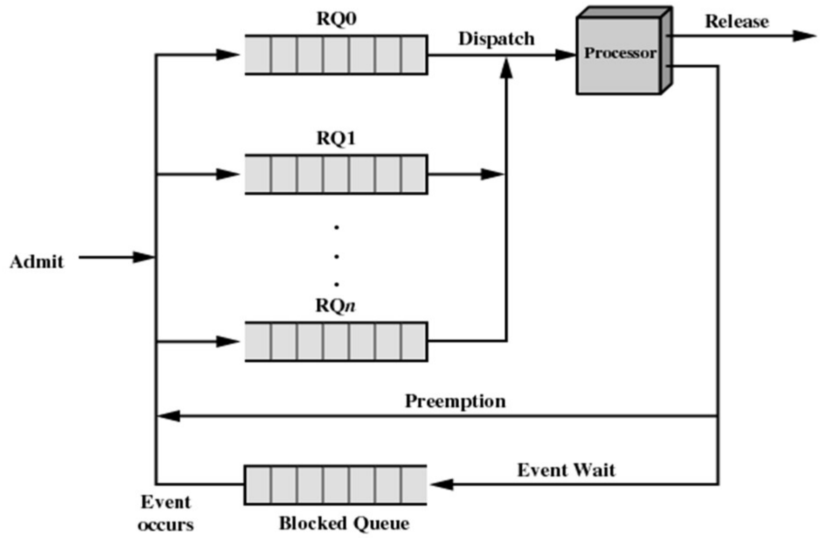


Figure 9.4 Priority Queuing

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

Priority Scheduling

Process	Arrival Time	Processing Time	Priority
A	0	8	3
B	0	3	4 (lowest)
C	0	5	2
D	0	2	1 (highest)

	0					5					10					15		
A																		
B																		
C																		
D																		

Priority Scheduling

Process	Arrival Time	Processing Time	Priority
A	0	8	3
B	0	3	4 (lowest)
C	0	5	2
D	0	2	1 (highest)

	0					5					10					15		
A																		
B																		
C																		
D																		

- Processes arrive at the same time → Non-preemptive

Priority Scheduling (Non-preemptive)

Process	Arrival Time	Processing Time	Priority
A	0	8	3
B	1	3	4 (lowest)
C	3	5	2
D	5	2	1 (highest)

	0					5					10					15			
A																			
B																			
C																			
D																			

47

Priority Scheduling (Preemptive)

Process	Arrival Time	Processing Time	Priority
A	0	8	3
B	1	3	4 (lowest)
C	3	5	2
D	5	2	1 (highest)

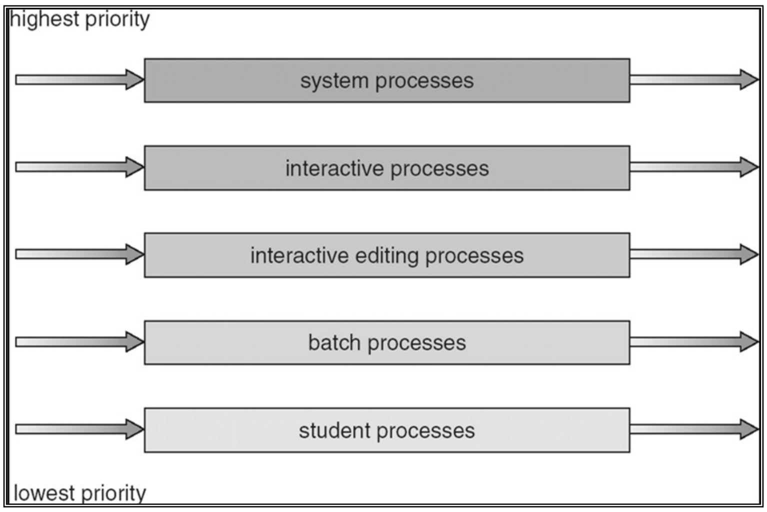
	0					5					10					15			
A																			
B																			
C																			
D																			

48

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

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

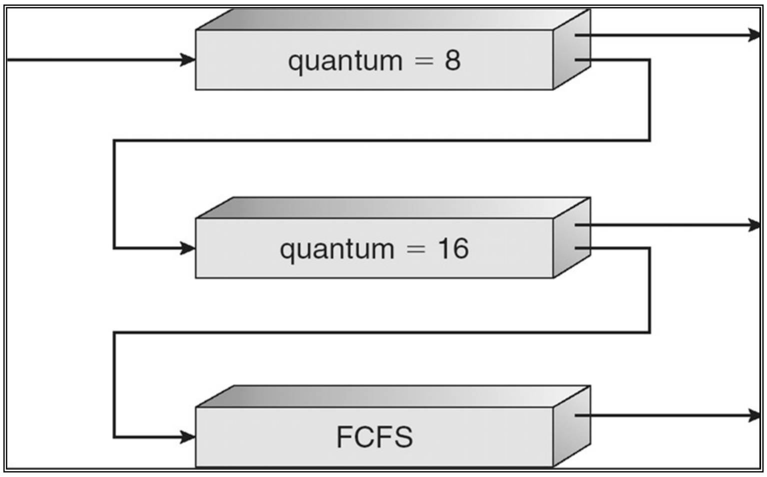
51

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR 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.

52

Multilevel Feedback Queues



53

53

Algorithm Evaluation

- How do we select a CPU scheduling algorithm for a particular system?
- One way – evaluate several choices, see which one best meets system goal(s). E.g., if the goal is minimum turnaround time, try to come up with an average turnaround time for each proposed choice.
- First, define the criteria, e.g.
  - Maximize the CPU utilization under the constraint that the maximum response time is I second
  - Maximize throughput such that turnaround time is on average linearly proportional to total execution time
  - minimize average waiting time
- Next, evaluate the algorithms under consideration

54

54

Algorithm Evaluation

- Methods of evaluation:
  1. Deterministic modeling / analytic evaluation
  2. Queuing models
  3. Simulation
  4. Implementation

Algorithm Evaluation

1. Deterministic modeling / analytic evaluation
   - takes a particular predetermined workload and defines the performance of each algorithm for that workload
   - Given a predetermined workload is known
   - Guestimate the performance of each algorithm (e.g. SJF, FCFS, RR)
   - Useful if behavior repeats or the same workload repeats
   - Pros:
     - Simple and fast
     - Gives an exact number
     - Makes it easy to compare algorithms
   - Cons:
     - Too specific because requires exact numbers for input, and only applies to them
     - Requires too much exact knowledge to be useful

Algorithm Evaluation

2. Queuing models
   - Deterministic model is not realistic
   - CPU has its ready queue. Knowing the arrival rates & service rates, we can compute utilization, average queue length, average wait time, etc.
   - The system is described as a network of servers
   - Each server has a queue of waiting processes:
     - CPU is a server with its ready queue
     - I/O system with its device queues
   - Collect data from real system on CPU bursts, I/O bursts, and process arrival times
     - Queuing analysis can be used to compute utilization, average queue length, average wait time, etc.

57

57

Algorithm Evaluation

2. Queuing models - Little's formula:
   - If the system at steady state, the number of processes leaving the queue must equal to the number of processes that arrive.
   - In this case, Little's formula applies:

$$n = \lambda W$$

   - where: n = the average queue length

     W = the average wait time

     λ = the average arrival rate of processes
   - By knowing 2of the above variables, we can compute the 3rd
     - For example, if we know that 8 processes arrive every second and there are normally 16 processes in the queue, we can compute that the average waiting time per process is 2 seconds
   - Powerful methods, but real systems are often too complex to model neatly, and need to approximate/make assumptions may be a problem

58

58

Algorithm Evaluation

3. Simulations
   - involves programming a model of the computer system.
   - The simulator modifies the system state to reflect the activities of the devices, scheduler and processes
   - The simulator keeps track of statistics about system performance
   - Input data (arrival times and burst times) can be generated by either:
     - A random number generator
     - A trace tape (a record of arrival and burst times on an actual system)
   - Advantage: potentially very accurate
   - Drawbacks:
     - Simulation is expensive. It can take hours of computing time.
     - More detail yields more accurate results, but takes more time to compute.

Algorithm Evaluation

4. Implementation
   - the only completely accurate way to evaluate a scheduling algorithm – code it and run it (put it in the OS)
   - Difficulty:
     - High cost – coding the algorithm and modifying the OS to support it can take many hours of programming
     - The environment in which the algorithm is used may change, so that the initial results no longer apply
   - Flexible scheduling algorithms can be altered by the system manager or users to accommodate changes in the environment
     - So it is good to separate mechanism from policy – build into the algo some parameters that can be changed at run time, by users and/or sysadmin

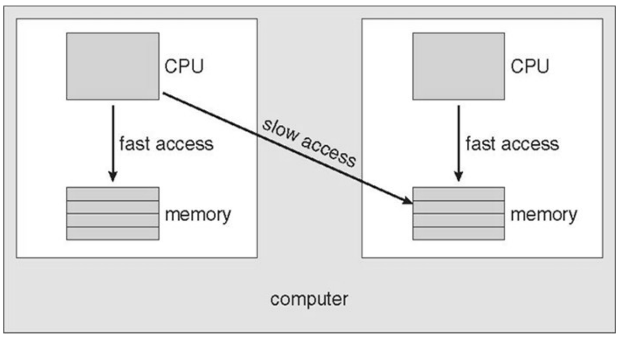
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Load sharing**
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- **Processor affinity –** process has affinity for processor on which it is currently running
  - soft affinity
  - hard affinity

61

61

NUMA and CPU Scheduling



62

62

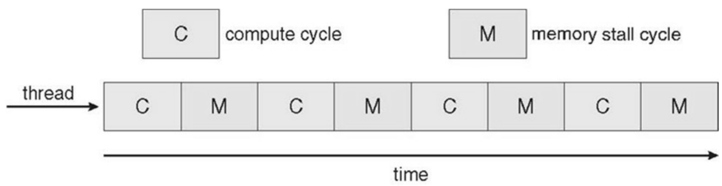
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consume less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

63

63

Multithreaded Multicore System



64

64

Real-Time Scheduling

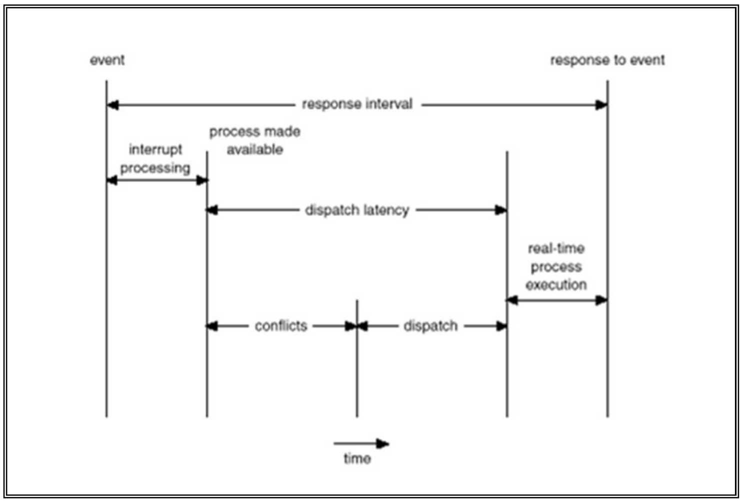
- *Hard real-time* systems
  – required to complete a critical task within a guaranteed amount of time

- *Soft real-time* computing
  – requires that critical processes receive priority over less fortunate ones

65

65

Dispatch Latency



66

66

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

67

67

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM THREADS 5
int main(int argc, char *argv[])
{
     int i;
     pthread t tid[NUM THREADS];
     pthread attr t attr;
     /* get the default attributes */
     pthread attr init(&attr);
     /* set the scheduling algorithm to PROCESS or SYSTEM */
     pthread attr setscope(&attr, PTHREAD SCOPE SYSTEM);
     /* set the scheduling policy - FIFO, RT, or OTHER */
     pthread attr setschedpolicy(&attr, SCHED OTHER);
     /* create the threads */
     for (i = 0; i < NUM THREADS; i++)
             pthread create(&tid[i],&attr,runner,NULL);
     /* now join on each thread */
     for (i = 0; i < NUM THREADS; i++)
             pthread join(tid[i], NULL);
}
 /* Each thread will begin control in this function */
void *runner(void *param)
{
     printf("I am a thread\n");
     pthread exit(0);
}
```

68

68

Operating System Examples

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

69

69

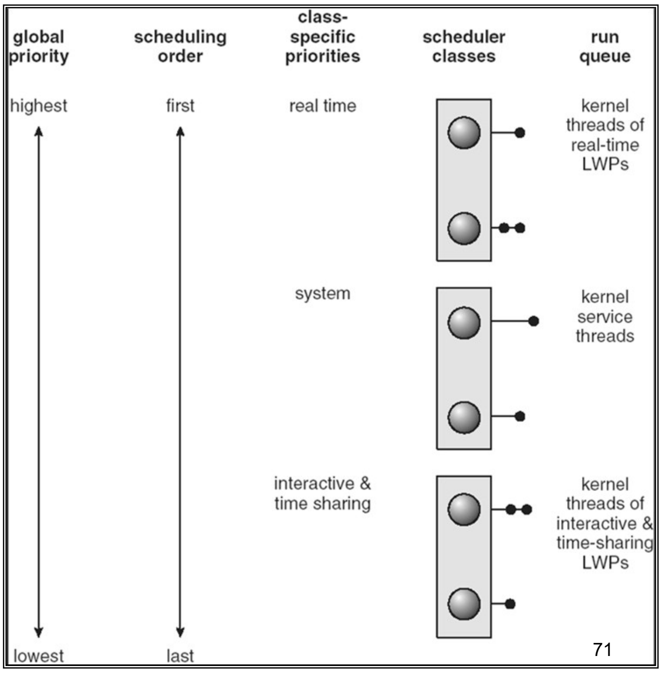
Solaris scheduling (Priority based)

- Default: time sharing
  - Dynamically alter priorities
  - Assign time slices based on a multilevel feedback queue
- Solaris 9 introduces
  - Fixed priority
    - No dynamic adjustment
  - Fair share
    - Look at CPU shares

70

70

Solaris 2 Scheduling



Solaris Dispatch Table

priority	time quantum	time quantum expired	return from sleep
0	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	20	49	59

Windows XP Priorities

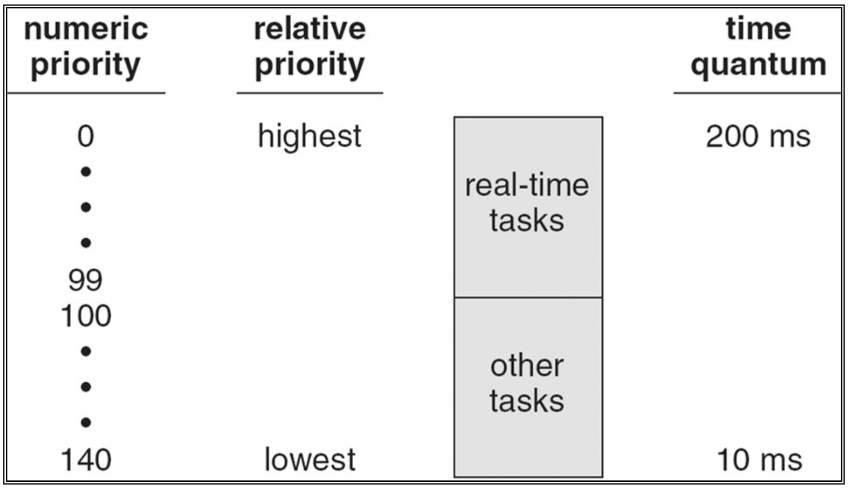
- priority-based, preemptive scheduling

	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

Linux Scheduling

- Two algorithms: time-sharing and real-time
- Time-sharing
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all processes have credit = 0, recrediting occurs
    - Based on factors including priority and history
- Real-time
  - Soft real-time
  - Posix.1b compliant – two classes
    - FCFS and RR
    - Highest priority process always runs first

The Relationship Between Priorities and Time-slice length



75