Chapter 4
Processor Management

Understanding Operating Systems,
Fourth Edition

Objectives

You will be able to describe:

- The difference between job scheduling and process scheduling, and how they relate
- The advantages and disadvantages of process scheduling algorithms that are preemptive versus those that are nonpreemptive
- The goals of process scheduling policies
- Up to six different process scheduling algorithms
- The role of internal interrupts and the tasks performed by the interrupt handler

Overview

- **Program (Job):**
  - A unit of work that has been submitted by user to an operating system
  - An inactive unit, such as a file stored on a disk
- **Process (Task):**
  - An active entity, which requires a set of resources, including a processor and special registers, to perform its function
  - A single instance of an executable program

Overview (continued)

- **Processor (CPU):** performs calculations and executes programs
  - **In single-user systems:**
    - Processor is busy only when user is executing a job, at all other times it is idle
    - Processor management is simple
  - **In a multiprogramming environment:**
    - Processor must be allocated to each job in a fair and efficient manner
    - Requires scheduling policy and a scheduling algorithm

Overview (continued)

- **Interrupt:** A hardware signal that suspends execution of a program and activates the execution of interrupt handler
- **Context Switch:** Saving a job's processing information in its PCB when interrupted
  - Context switching occurs in all preemptive policies

Job Scheduling Versus Process Scheduling

- Processor Manager has two submanagers:
  - **Job Scheduler:**
    - In charge of job scheduling
    - Initiates the job based on certain criteria
  - **Process Scheduler:**
    - In charge of process scheduling
    - Assigns the CPU to execute processes of those jobs placed on READY queue by Job Scheduler

Job Scheduling Versus Process Scheduling (continued)

Job Scheduler (High level scheduler):

- Initiates the job based on certain criteria
- Puts jobs in a sequence that uses all system's resources as fully as possible
- Strives for balanced mix of jobs with large I/O interaction and jobs with lots of computation
- Tries to keep most system components busy most of time

Job Scheduling Versus Process Scheduling (continued)

Process Scheduler (Low level scheduler):

- Determines which jobs will get the CPU, when, and for how long
- Decides when processing should be interrupted
- Determines which queues the job should be moved to during its execution
- Recognizes when a job has concluded and should be terminated

Job Scheduling Versus Process Scheduling (continued)

- **I/O-bound jobs** have many brief CPU cycles and long I/O cycles, e.g., printing a series of documents
- **CPU-bound jobs** have long CPU cycles and shorter I/O cycles, e.g., finding the first 300 prime numbers
- Total effect of all CPU cycles, from both I/O-bound and CPU-bound jobs, approximates a **Poisson distribution curve**

Job Scheduling Versus Process Scheduling (continued)

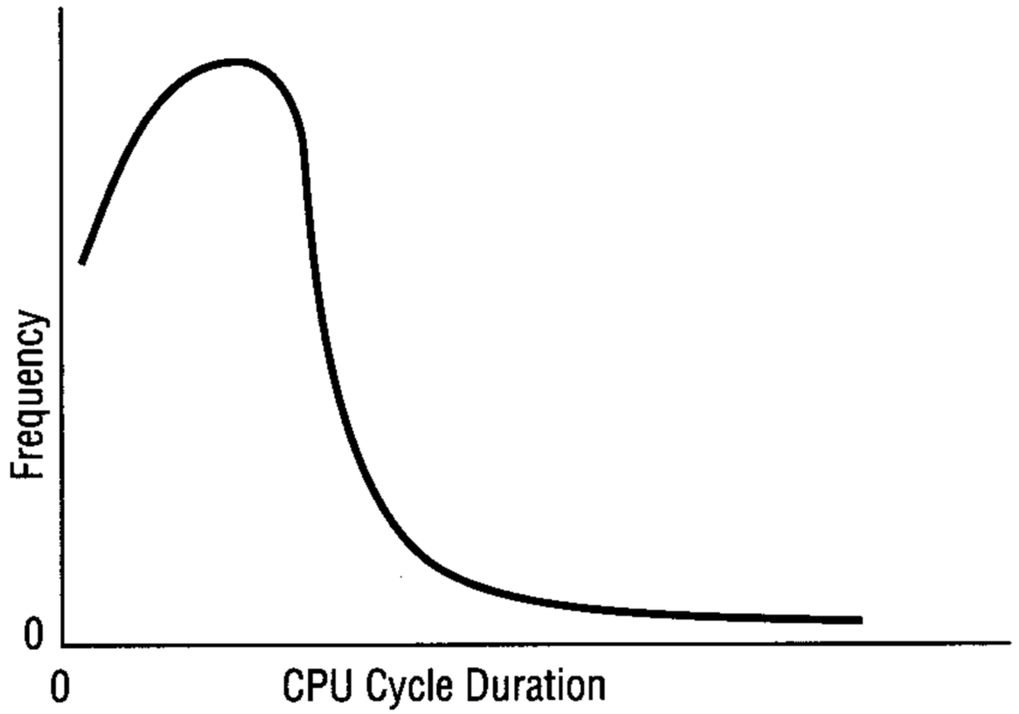


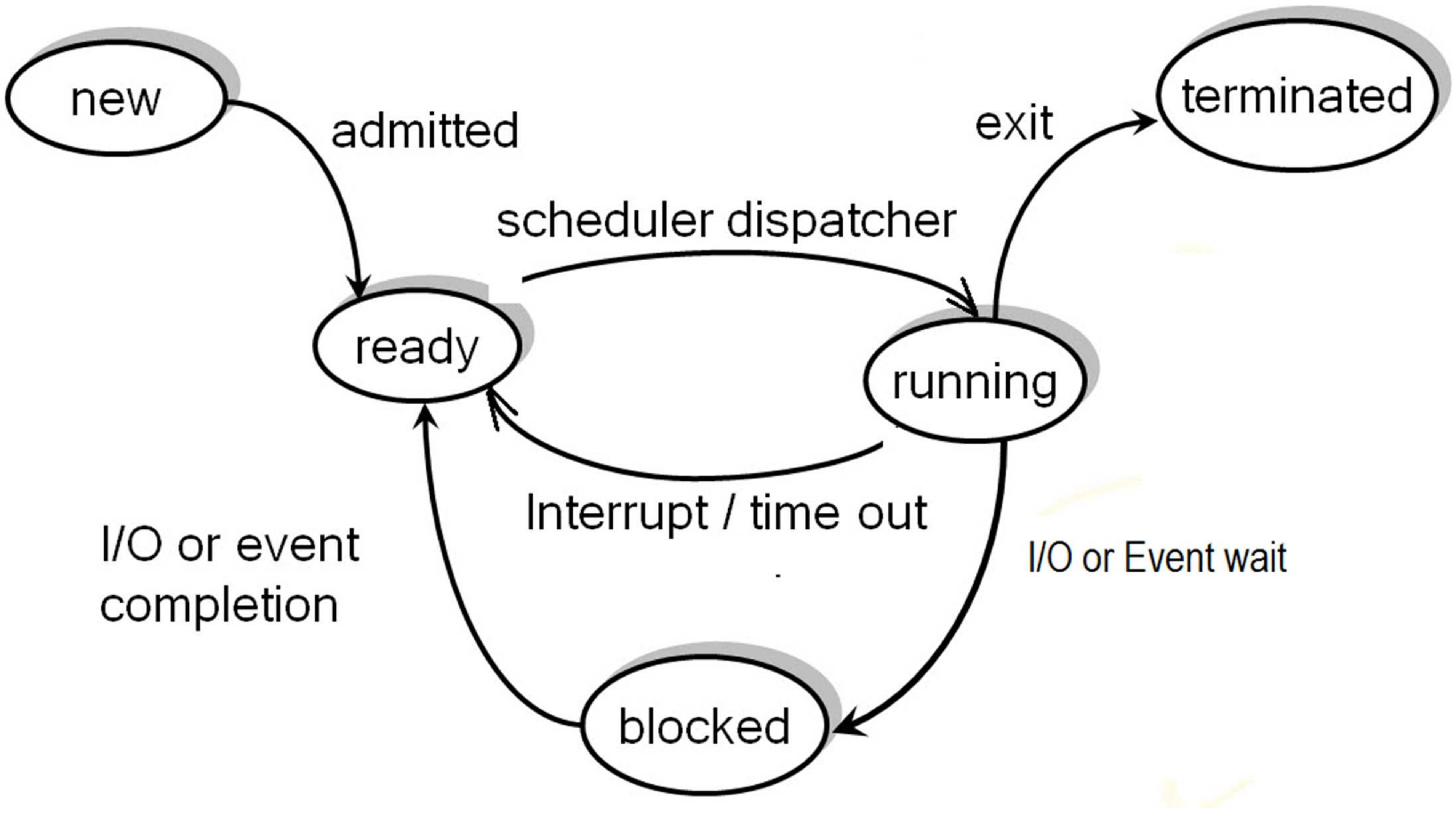
Figure 4.1: Distribution of CPU cycle times

Job Scheduling Versus Process Scheduling (continued)

- **Middle level scheduler** (third layer): Used in a highly interactive environment
  - Removes active jobs from memory to reduce degree of multiprogramming
  - Allows jobs to be completed faster

Job and Process Status

- **Job status:** A job takes one of the following states as it moves through the system
  - HOLD
  - READY
  - WAITING
  - RUNNING
  - FINISHED


new
admitted
terminated
exit
scheduler dispatcher
ready
running
Interrupt / time out
I/O or event completion
I/O or Event wait
blocked

Job and Process Status (continued)

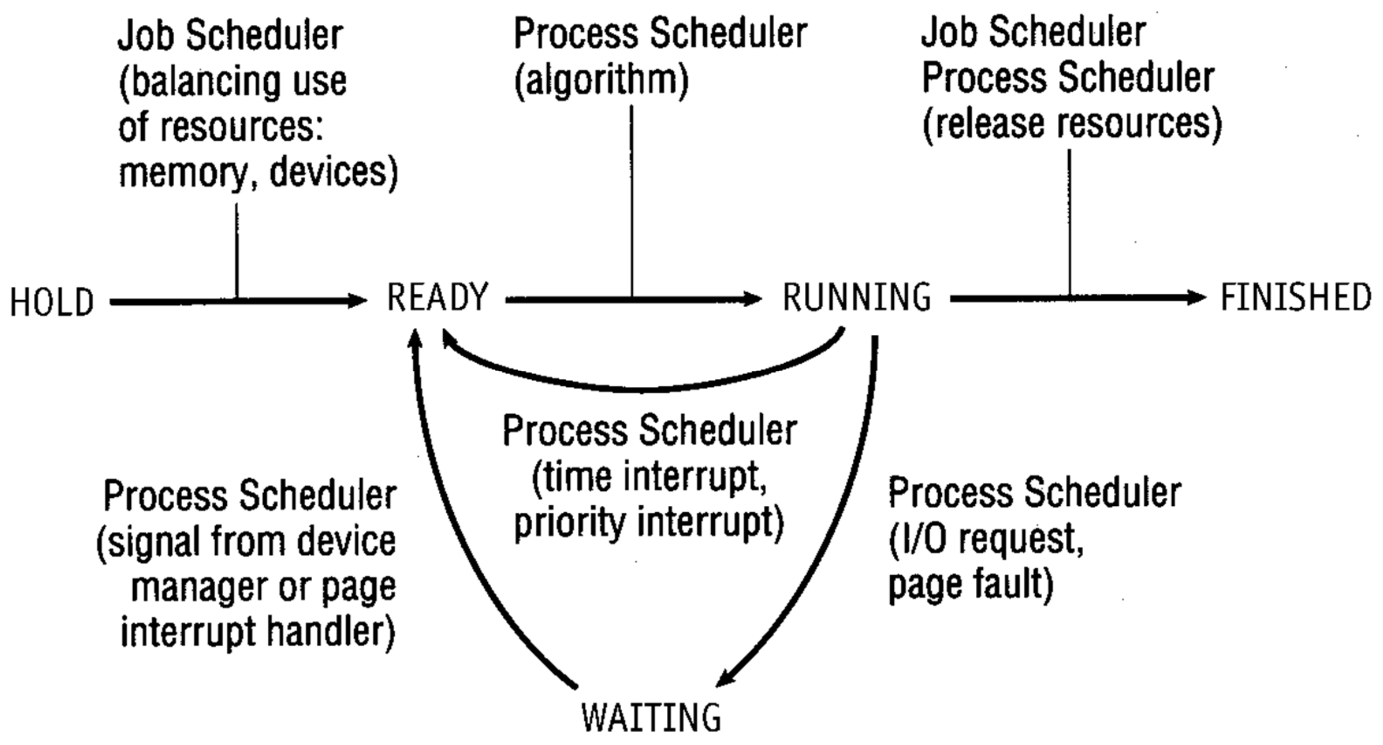


Figure 4.2: A typical job (or process) changes status as it moves through the system from HOLD to FINISHED

Job and Process Status (continued)

- **Transition** from one status to another is initiated by either the Job Scheduler **(JS)** or the Process Scheduler **(PS)**:
  - HOLD to READY: JS, using a predefined policy
  - READY to RUNNING: PS, using some predefined algorithm
  - RUNNING back to READY: PS, according to some predefined time limit or other criterion
  - RUNNING to WAITING: PS, and is initiated by an instruction in the job

Job and Process Status (continued)

- **Transition** (continued):
  - WAITING to READY: PS, and is initiated by signal from I/O device manager that I/O request has been satisfied and job can continue.
  - RUNNING to FINISHED: PS or JS, if job is finished or error has occurred

Process Control Blocks

- **Process Control Block (PCB):** Data structure that contains basic info about the job including
  - What it is
  - Where it’s going
  - How much of its processing has been completed
  - Where it’s stored
  - How much it has spent in using resources

Process Control Blocks (continued)

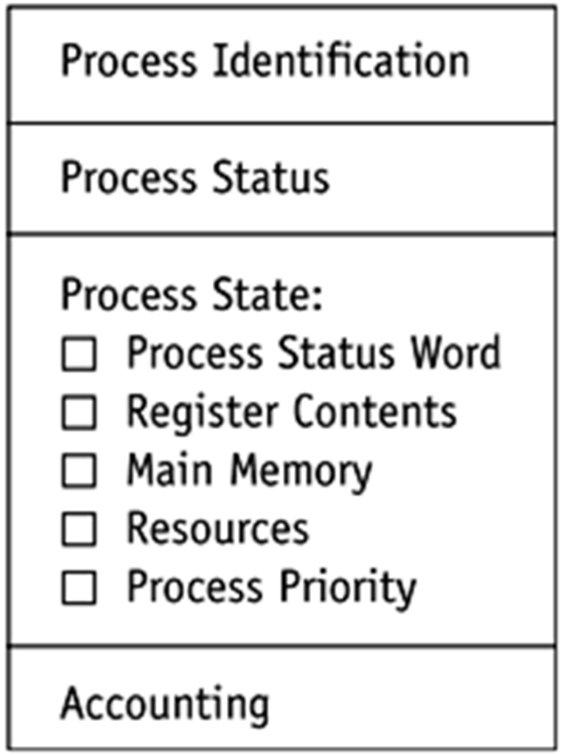


Figure 4.3: Contents of each job’s Process Control Block

Process Control Blocks (continued)

- **Contents of Process Control Block (PCB):**
  - **Process identification**
  - **Process status** (HOLD, READY, RUNNING, WAITING)
  - **Process state** (process status word, register contents, main memory info, resources, process priority)
  - **Accounting** (CPU time, total time, memory occupancy, I/O operations, number of input records read, etc.)

PCBs and Queuing

- **PCB of a job:** Contains all of the data about the job needed by the operating system to manage the processing of the job
  - Created when job scheduler accepts the job
  - Updated as job goes from beginning to end of its execution
- **Queues** use PCBs to track jobs
  - PCBs, not jobs, are linked to form queues
  - Queues must be managed by process scheduling policies and algorithms

PCBs and Queuing (continued)

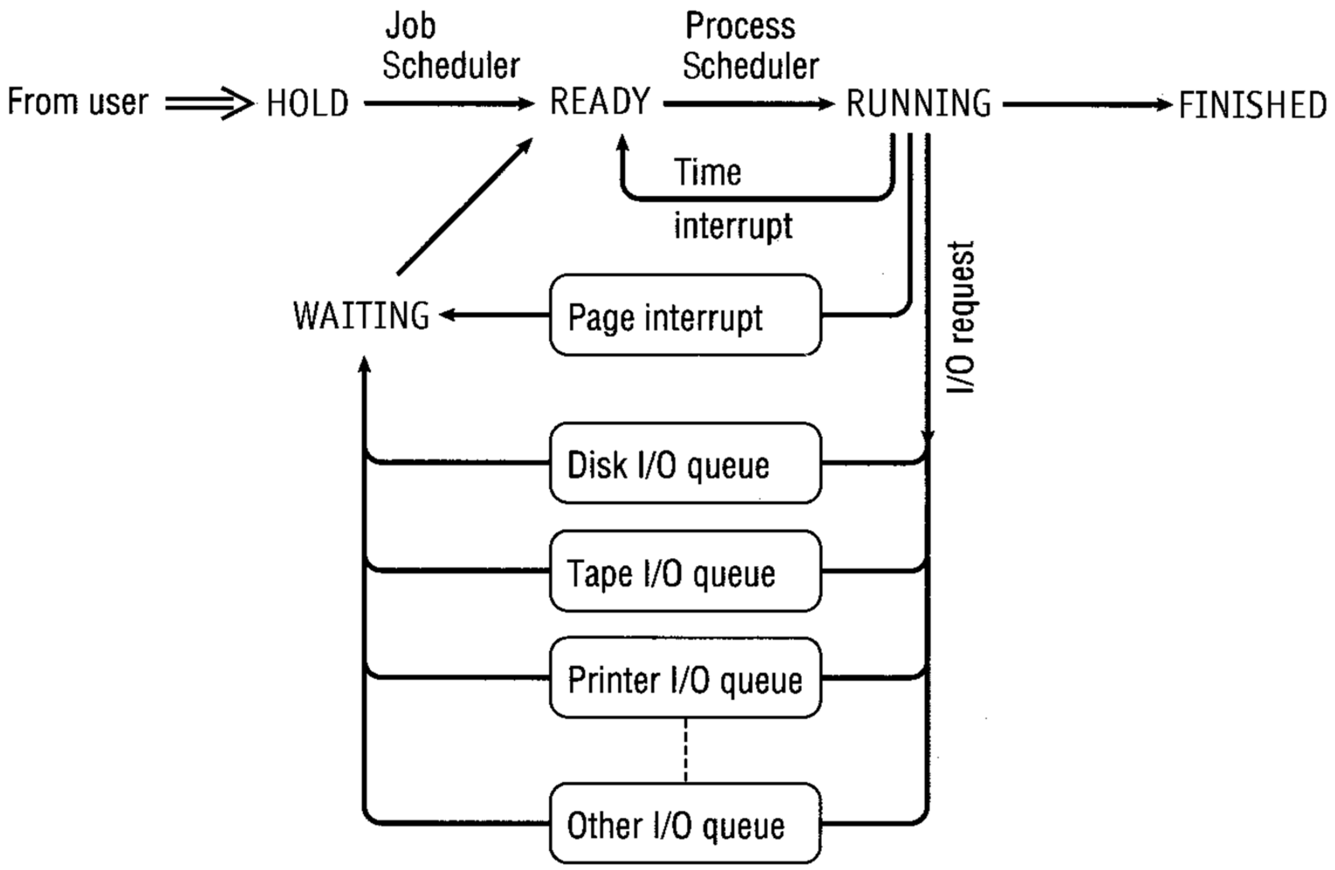


Figure 4.4: Queuing paths from HOLD to FINISHED

Process Scheduling Policies

- Operating system must resolve three limitations of a system before scheduling all jobs in a multi-programming environment:
  - Finite number of resources (e.g., disk drives, printers, and tape drives)
  - Some resources can’t be shared once they’re allocated (e.g., printers)
  - Some resources require operator intervention (e.g., tape drives)

Process Scheduling Policies (continued)

- A **good process scheduling policy** should
- **Maximize throughput** by running as many jobs as possible in a given amount of time
  - **Minimize response time** by quickly turning around interactive requests
  - **Minimize turnaround time** by moving entire jobs in and out of system quickly
  - **Minimize waiting time** by moving jobs out of READY queue as quickly as possible

Process Scheduling Policies (continued)

- (continued):
  - **Maximize CPU efficiency** by keeping CPU busy 100 percent of time
  - **Ensure fairness** for all jobs by giving every one an equal amount of CPU and I/O time

Process Scheduling Policies (continued)

- **Need for Interrupts:** When a job claims CPU for a very long time before issuing an I/O request
  - Builds up READY queue & empties I/O queues
  - Creates an unacceptable imbalance in the system
- Process Scheduler uses interrupts when a predetermined slice of time has expired
  - Suspends all activity on the currently running job
  - Reschedules it into the READY queue

Process Scheduling Policies (continued)

- **Types of Scheduling Policies:**
  - **Preemptive scheduling policy:**
    - Interrupts processing of a job and transfers the CPU to another job
  - **Nonpreemptive scheduling policy:**
    - Functions without external interrupts

Process Scheduling Algorithms

- **Types of Process Scheduling Algorithms:**
  - First Come, First Served (FCFS)
  - Shortest Job Next (SJN)
  - Priority Scheduling
  - Shortest Remaining Time (SRT)
  - Round Robin
  - Multiple Level Queues

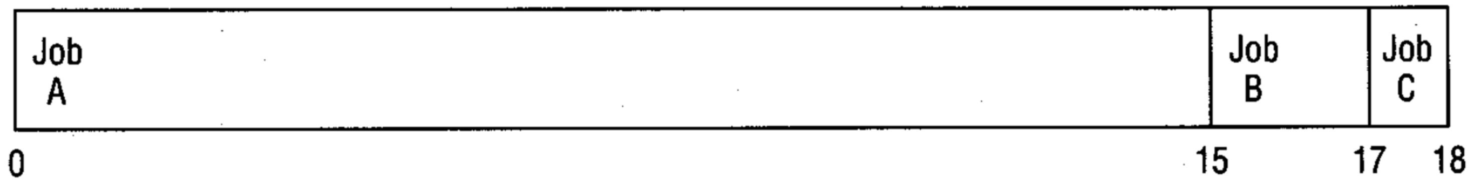
First-Come, First-Served

- Nonpreemptive
- Handles jobs according to their arrival time: the earlier they arrive, the sooner they're served
- Simple algorithm to implement: uses a FIFO queue
- Good for batch systems; unacceptable for interactive systems
- Turnaround time is unpredictable

First-Come, First-Served (continued)

Jobs arrival sequence: A, B, C

- Job A has a CPU cycle of 15 milliseconds
- Job B has a CPU cycle of 2 milliseconds
- Job C has a CPU cycle of 1 millisecond



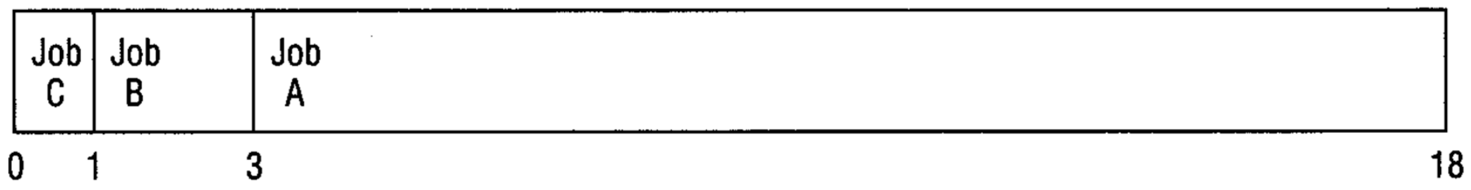
Average turnaround time: 16.67 s

Figure 4.5: Timeline for job sequence A, B, C using the FCFS algorithm

First-Come, First-Served (continued)

Jobs arrival sequence: C, B, A

- Job A has a CPU cycle of 15 milliseconds
- Job B has a CPU cycle of 2 milliseconds
- Job C has a CPU cycle of 1 millisecond



Average turnaround time: 7.3 s

Figure 4.6: Timeline for job sequence C, B, A using the FCFS algorithm

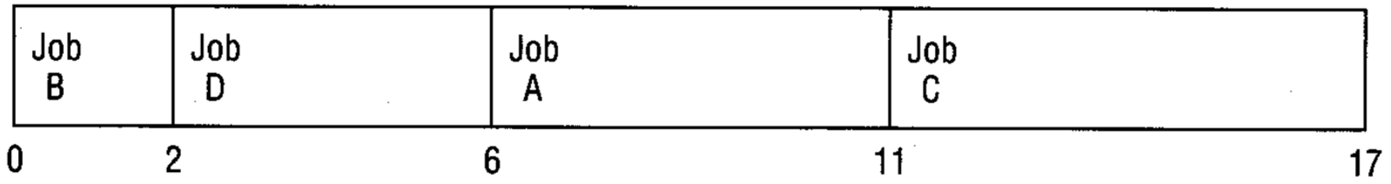
Shortest Job Next (SJN)

- Nonpreemptive
- Handles jobs based on length of their CPU cycle time
- Easiest to implement in batch environments
- Doesn’t work in interactive systems
- Optimal only when all jobs are available at same time and the CPU estimates are available and accurate

Shortest Job Next (continued)

Four batch jobs A, B, C, D, all in the READY queue

Job:	A	B	C	D
CPU cycle:	5	2	6	4



Average turnaround time: 9 s

Figure 4.7: Timeline for job sequence B, D, A, C using the SJN algorithm

Priority Scheduling

- Nonpreemptive
- Gives preferential treatment to important jobs
  - Programs with highest priority are processed first
  - Not interrupted until CPU cycles are completed or a natural wait occurs
- FCFS policy is used if two or more jobs with equal priority in READY queue
- System administrator or Processor Manager use different methods of assigning priorities

Shortest Remaining Time

- Preemptive version of the SJN algorithm
- Processor allocated to job closest to completion
  - Current job can be preempted if newer job in READY queue has shorter time to completion
- Cannot be implemented in interactive system
  - Requires advance knowledge of the CPU time required to finish each job
- SRT involves more overhead than SJN
  - OS monitors CPU time for all jobs in READY queue and performs context switching

Shortest Remaining Time (continued)

Arrival time:	0	1	2	3
Job:	A	B	C	D
CPU cycle:	6	3	1	4

Job:	A	B	C	D
Turnaround:	14	4	1	6

Average Turnaround: 6.25s

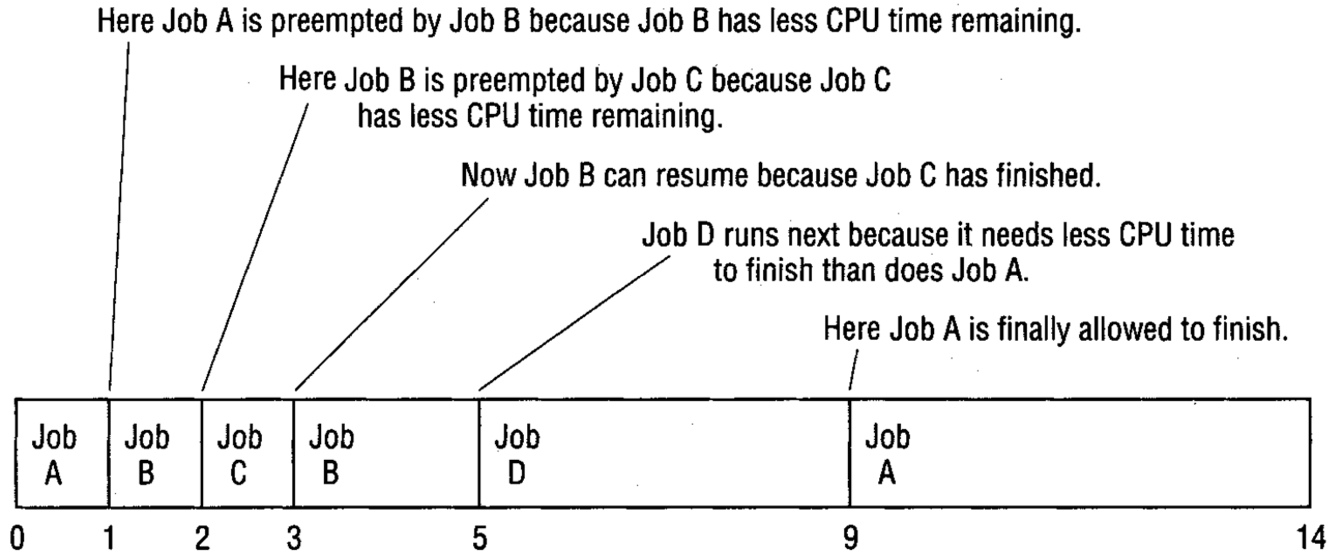

Here Job A is preempted by Job B because Job B has less CPU time remaining.
Here Job B is preempted by Job C because Job C has less CPU time remaining.
Now Job B can resume because Job C has finished.
Job D runs next because it needs less CPU time to finish than does Job A.
Here Job A is finally allowed to finish.
Job A
Job B
Job C
Job B
Job D
Job A
0
1
2
3
5
9
14

Figure 4.8: Timeline for job sequence A, B, C, D using the preemptive SRT algorithm

Shortest Remaining Time (continued)

Arrival time:	0	1	2	3
Job:	A	B	C	D
CPU cycle:	6	3	1	4

Job:	A	B	C	D
Turnaround:	6	9	5	11

Average Turnaround: 7.75s

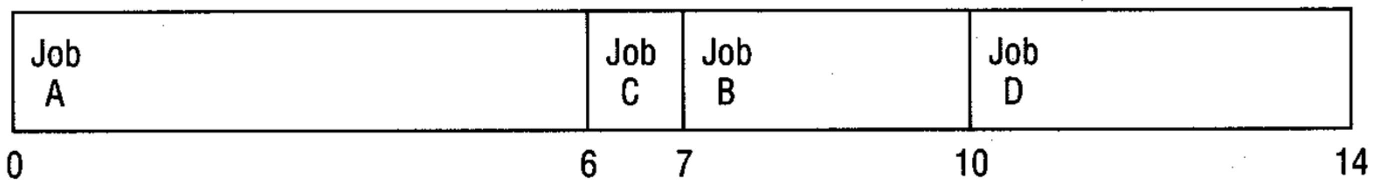


Figure 4.9: Timeline for job sequence A, B, C, D using the nonpreemptive SJN algorithm

Round Robin

- Preemptive
- Used extensively in interactive systems
- Based on a predetermined slice of time (time quantum) that’s given to each job
- Size of time quantum crucial to system performance
  - Usually varies from 100 ms to 1-2 s
- Ensures CPU is equally shared among all active processes and is not monopolized by any one job

Round Robin (continued)

Arrival time:	0	1	2	3
Job:	A	B	C	D
CPU cycle:	8	4	9	5

Job:	A	B	C	D
Turnaround:	20	7	24	22

Average Turnaround: 18.25 s

Time slice: 4ms

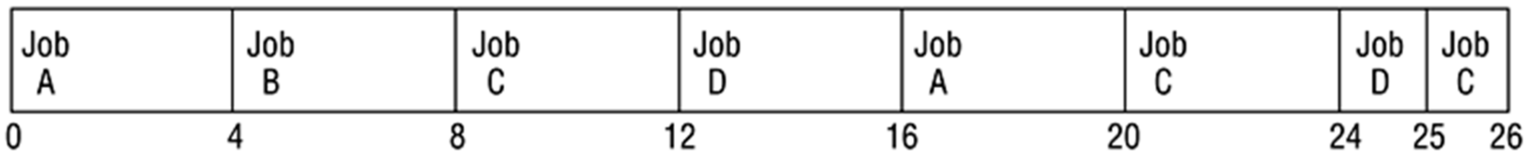
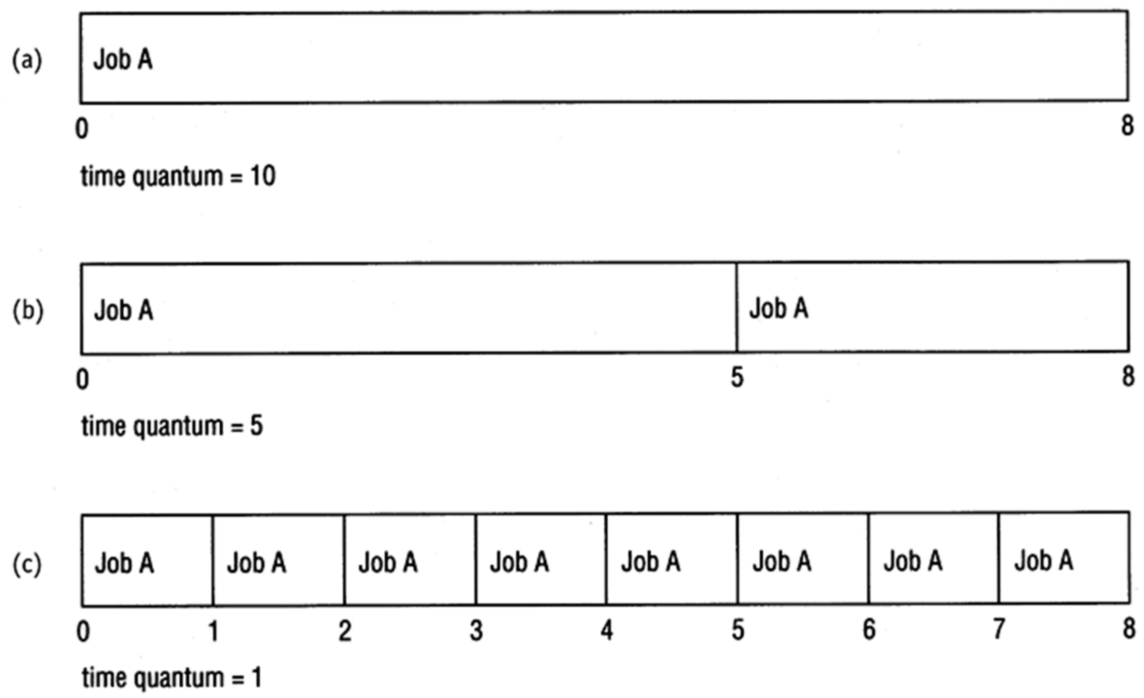


Figure 4.10: Timeline for job sequence A, B, C, D using the preemptive round robin algorithm

Round Robin (continued)

- **If Job’s CPU cycle > time quantum**
  - Job is preempted and put at the end of the READY queue and its information is saved in its PCB
- **If Job’s CPU cycle < time quantum**
  - If job is finished, all resources allocated to it are released & completed job is returned to user
  - If interrupted by I/O request, then info is saved in PCB & it is linked at end of the appropriate I/O queue
  - Once I/O request is satisfied, job returns to end of READY queue to await allocation of CPU

Round Robin (continued)



CPU cycle of job A: 8 ms

Figure 4.11: Context switches for job A with three different time quantums. In (a) the job finishes before the time quantum expires. In (b) and (c), the time quantum expires first, interrupting the job.

Round Robin (continued)

- **Efficiency** depends on the size of time quantum in relation to the average CPU cycle
- **If the quantum is too large** - larger than most CPU cycles
  - Algorithm reduces to the FCFS scheme
- **If the quantum is too small**
  - Amount of context switching slows down the execution of the jobs
  - Amount of overhead is dramatically increased

Round Robin (continued)

- General rules of thumb for selecting the proper time quantum:
  - Should be long enough to allow 80% of CPU cycles to run to completion
  - Should be at least 100 times longer than the time required to perform one context switch
- These rules are flexible and depend on the system

Multiple-Level Queues

- Work in conjunction with several other schemes
- Found in systems with jobs that can be grouped according to a common characteristic
- Examples:
  - Priority-based system with different queues for each priority level
  - System with all CPU-bound jobs in one queue and all I/O-bound jobs in another
  - Hybrid system with batch jobs in background queue and interactive jobs in a foreground queue

Multiple-Level Queues (continued)

- Four primary methods to the movement of jobs:
  - No Movement Between Queues
  - Movement Between Queues
  - Variable Time Quantum Per Queue
  - Aging

Multiple-Level Queues (continued)

- **No Movement Between Queues:**
  - The processor is allocated to the jobs in the high-priority queue in FCFS fashion
  - Allocated to jobs in lower priority queues only when the high priority queues are empty
- **Movement Between Queues**
  - Adjusts the priorities assigned to each job
  - A job may also have its priority increased
  - Good in interactive systems

Multiple-Level Queues (continued)

- **Variable Time Quantum Per Queue:**
  - Each of the queues is given a time quantum twice as long as the previous queue
  - CPU-bound job can execute for longer and longer periods of time, thus improving its chances of finishing faster
- **Aging:**
  - System moves the old job to the next highest queue, and so on until it reaches the top queue
  - Ensures that jobs in the lower-level queues will eventually complete their execution

A Word About Interrupts

- **Types of Interrupts:**
  - Page interrupts to accommodate job requests
  - Time quantum expiration interrupts
  - I/O interrupts when READ or WRITE command is issued
  - Internal interrupts (synchronous interrupts) result from arithmetic operation or job instruction
  - Illegal arithmetic operations (e.g., dividing by 0).
  - Illegal job instructions (e.g., attempts to access protected storage locations)

A Word About Interrupts (continued)

- **Interrupt handler:** Control program that handles the interruption sequence of events
- When operating system detects a nonrecoverable error, the interrupt handler follows this sequence:
  - The type of interrupt is described and stored
  - The state of the interrupted process is saved
  - The interrupt is processed
  - The processor resumes normal operation

Summary

- Process scheduler assigns the CPU to execute processes of those jobs placed on READY queue by Job Scheduler
- Total effect of all CPU cycles, from both I/O-bound and CPU-bound jobs, approximates a Poisson distribution curve
- Transition from one status to another is initiated by either Job Scheduler (JS) or Process Scheduler (PS)
- PCB of a job contains all data about the job needed by OS to manage the processing of the job

Summary (continued)

- A good process scheduling policy should maximize CPU efficiency by keeping CPU busy 100 percent of time
- FIFO has simple algorithm to implement but turnaround time is unpredictable
- SJN minimizes average waiting time but results in infinite postponement of some jobs
- Priority scheduling ensures fast completion of important jobs but results in infinite postponement of some jobs

Summary (continued)

- SRT ensures fast completion of short jobs but involves more overhead than SJN, incurred by context switching
- Efficiency in round robin policy depends on the size of time quantum in relation to the average CPU cycle
- Multiple-level queues counteract indefinite postponement with aging or other queue movement