Understanding Operating Systems Sixth Edition

Chapter 4
Processor Management

Learning Objectives

After completing this chapter, you should 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

Learning Objectives (cont'd.)

- The goals of process scheduling policies in single-core CPUs
- Up to six different process scheduling algorithms
- The role of internal interrupts and the tasks performed by the interrupt handler

Overview

- Simple system
  - Single user
  - One processor: busy only when executing the user's job or system software
- Multiprogramming environment: multiple processes competing to be run by a single CPU
  - Requires fair and efficient CPU allocation for each job
- Single processor systems
  - Addressed in this chapter

Overview

- **Single-user systems** (without multiprogramming)
  - Busy state: executing a job
  - Idle state: all other times
  - Simple processor management
- **Program** (job)
  - Inactive unit
    - File stored on a disk
  - A unit of work submitted by a user
    - Not a process

Overview (cont'd.)

- **Process** (task)
  - Active entity
    - Requires resources to perform function
    - Processor and special registers
  - Executable program single instance
- **Thread**
  - Portion of a process
  - Runs independently
- **Processor**
  - Central processing unit (CPU)
  - Performs calculations and executes programs

Overview (cont'd.)

- **Multiprogramming environment**
  - Processor allocated for a time period
  - Deallocated at appropriate moment: delicate task
- **Interrupt**
  - Call for help
  - Activates higher-priority program
- **Context Switch**
  - Saving job processing information when interrupted
- Single processor
  - May be shared by several jobs (processes)
  - Requires scheduling policy and scheduling algorithm

About Multi-Core Technologies

- Processor (core)
  - Located on chip
- Multi-core CPU (more than one processor)
  - Dual-core, quad-core, or more
- Single chip may contain multiple cores
  - Multi-core engineering
    - Resolves leakage and heat problems
    - Multiple calculations may occur simultaneously
    - More complex than single core: discussed in Chapter 6

Job Scheduling Versus Process Scheduling

- Processor Manager
  - Composite of two submanagers
    - Hierarchy between them
- **Job Scheduler:** higher-level scheduler
  - Job scheduling responsibilities
  - Job initiation based on certain criteria
- **Process Scheduler:** lower-level scheduler
  - Process scheduling responsibilities
  - Determines execution steps
  - Process scheduling based on certain criteria

Job Scheduling Versus Process Scheduling (cont'd.)

- Job Scheduler functions
  - Selects incoming job from queue
  - Places in process queue
  - Decides on job initiation criteria
    - Process scheduling algorithm and priority
- **Goal**
  - Sequence jobs
    - Efficient system resource utilization
  - Balance I/O interaction and computation
  - Keep most system components busy most of time

Process Scheduler

- Process Scheduler functions
  - Determines job to get CPU resource
    - When and how long
  - Decides interrupt processing
  - Determines queues for job movement during execution
  - Recognizes job conclusion
    - Determines job termination
- Lower-level scheduler in the hierarchy
  - Assigns CPU to execute individual actions: jobs placed on READY queue by the Job Scheduler

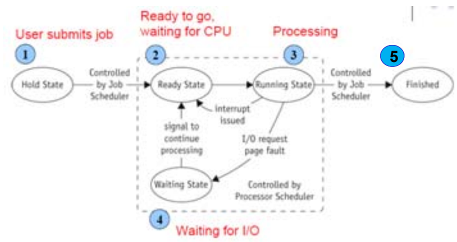
Process Scheduler (cont'd.)

- Exploits common computer program traits
  - Programs alternate between two cycles
    - CPU and I/O cycles
    - Frequency and CPU cycle duration vary
- General tendencies exists
  - **I/O-bound job**
    - Many brief CPU cycles and long I/O cycles (printing documents)
  - **CPU-bound job**
    - Many long CPU cycles and shorter I/O cycles (math calculation)

Job and Process Status

- Jobs move through the system
- Five states
  - HOLD
  - READY
  - WAITING
  - RUNNING
  - FINISHED
- Called **job status** or **process status**

Job and Process Status (cont'd.)



Job and Process Status (cont'd.)

- User submits job (batch/interactive)
  - Job accepted
    - Put on HOLD and placed in queue
  - Job state changes from HOLD to READY
    - Indicates job waiting for CPU
  - Job state changes from READY to RUNNING
    - When selected for CPU and processing
  - Job state changes from RUNNING to WAITING
    - Requires unavailable resources
  - Job state changes to FINISHED
    - Job completed (successfully or unsuccessfully)

Job and Process Status (cont'd.)

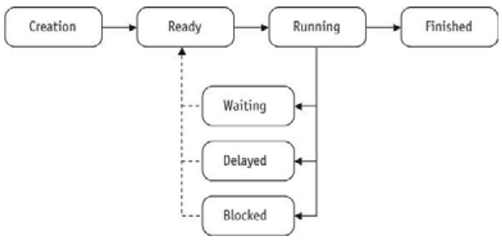
- Job Scheduler (JS) or Process Scheduler (PS) incurs state transition responsibility
  - HOLD to READY
    - JS initiates using predefined policy
  - READY to RUNNING
    - PS initiates using predefined algorithm
  - RUNNING back to READY
    - PS initiates according to predefined time limit or other criterion
  - RUNNING to WAITING
    - PS initiates by instruction in job

Job and Process Status (cont'd.)

- Job Scheduler (JS) or Process Scheduler (PS) incurs state transition responsibility (cont'd.)
  - WAITING to READY
    - PS initiates by signal from I/O device manager
    - Signal indicates I/O request satisfied; job continues
  - RUNNING to FINISHED
    - PS or JS initiates upon job completion
    - Satisfactorily or with error

Thread States

- Five states as a thread moves through the system
  - READY
  - RUNNING
  - WAITING
  - DELAYED
  - BLOCKED
- Thread transitions
  - Application creates a thread: placed in READY queue
  - READY to RUNNING: Process Scheduler assigns it to a processor



(figure 4.3)
A typical thread changes states from READY to FINISHED as it moves through the system.

Thread States (cont'd.)

- Thread transitions
  - Application creates a thread: placed in READY queue
  - READY to RUNNING: Process Scheduler assigns it to a processor
  - RUNNING to WAITING: when dependent on an outside event, e.g., mouse click, or waiting for another thread to finish
  - WAITING to READY: outside event occurs or previous thread finishes

Thread States (cont'd.)

- Thread transitions (cont'd.)
  - RUNNING to DELAYED: application that delays thread processing by specified amount of time
  - DELAYED to READY: prescribed time elapsed
  - RUNNING to BLOCKED: I/O request issued
  - BLOCKED to RUNNING: I/O completed
  - RUNNING to FINISHED: exit or termination
    - All resources released

Thread States (cont'd.)

- Operating systems must be able to:
  - Create new threads
  - Set up a thread so it is ready to execute
  - Delay, or put to sleep, threads for a specified amount of time
  - Block, or suspend, threads waiting for I/O to be completed
  - Set threads to a WAIT state until a specific event occurs
  - Schedule threads for execution

Thread States (cont'd.)

- Operating systems must be able to:
  - Creating new threads
  - Synchronize thread execution using semaphores, events, or conditional variables
  - Delaying, putting to sleep for a certain time
  - Blocking or Suspending when waiting for I/O
  - Setting to wait state until specified event occurs
  - Terminate a thread and release its resources

Control Blocks

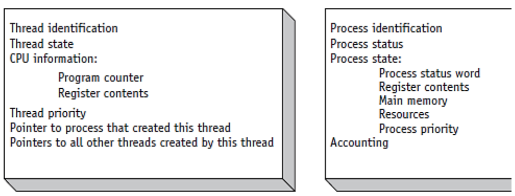
- Process Control Block (PCB): data structure for each process in the system
- Thread Control Block (TCB): data structure for each thread

Process Control Blocks

- Data structure
- Contains basic job information
  - What it is
  - Where it is going
  - How much processing completed
  - Where stored
  - How much time spent using resources

Process Control Blocks (cont'd.)

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

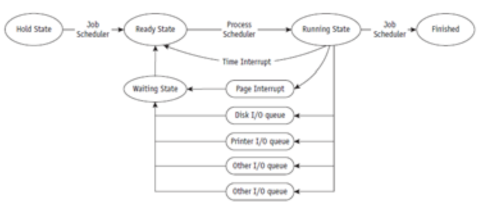


(figure 4.4)
Comparison of a typical Thread Control Block (TCB) vs. a Process Control Block (PCB).
© Cengage Learning 2014

PCBs and Queuing

- Job PCB
  - Created when Job Scheduler accepts job
  - Updated as job executes
  - **Queues** use PCBs to track jobs
  - Contains all necessary job management processing data
  - PCBs linked to form queues (jobs not linked)
- PCBs or TCBs: requires orderly management of queues
  - Determined by process scheduling policies and algorithms

PCBs and Queuing (cont'd.)



(figure 4.4)
Queuing paths from HOLD to FINISHED. The Job and Processor schedulers release the resources when the job leaves the RUNNING state.

Process Scheduling Policies

- Multiprogramming environment
  - More jobs than resources at any given time
- Operating system pre-scheduling task
  - Resolve three system limitations
    - Finite number of resources (disk drives, printers, tape drives)
    - Some resources cannot be shared once allocated (printers)
    - Some resources require operator intervention (tape drives)

Process Scheduling Policies (cont'd.)

- Good process scheduling policy criteria
  - Maximize throughput (favor short jobs)
    - Run as many jobs as possible in given amount of time
  - Minimize response time (favor interactive jobs)
    - Quickly turn around interactive requests
  - Minimize turnaround time (favor batch jobs)
    - Move entire job in and out of system quickly
  - Minimize waiting time (reduce # of users)
    - Move job out of READY queue quickly

Process Scheduling Policies (cont'd.)

- Good process scheduling policy criteria (cont'd.)
  - Maximize CPU efficiency (favor CPU bound jobs)
    - Keep CPU busy 100 percent of time
  - Ensure fairness for all jobs (no priority)
    - Give every job equal CPU and I/O time
- Final policy criteria decision in designer's hands

Process Scheduling Policies (cont'd.)

- Problem
  - Job claims CPU for very long time before I/O request issued
    - Builds up READY queue and empties I/O queues
    - Creates unacceptable system imbalance
- Corrective measure
  - **Interrupt**
    - Used by Process Scheduler upon predetermined expiration of time slice
    - Current job activity suspended
    - Reschedules job into READY queue

Process Scheduling Policies (cont'd.)

- Types of scheduling policies
  - **Preemptive**
    - Used in time-sharing environments
    - Interrupts job processing
    - Transfers CPU to another job
  - **Nonpreemptive**
    - Functions without external interrupts
  - Infinite loops interrupted in both cases

Process Scheduling Algorithms

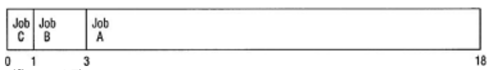
- Based on specific policy
  - Allocate CPU and move job through system
- Six algorithm types
  - First-come, first-served (FCFS)
  - Shortest job next (SJN)
  - Priority scheduling
  - Shortest remaining time (SRT)
  - Round robin
  - Multiple-level queues
- Current systems emphasize interactive use and response time (use preemptive policies)

First-Come, First-Served

- Nonpreemptive
- Job handled based on arrival time
  - Earlier job arrives, earlier served
- Simple algorithm implementation
  - Uses first-in, first-out (FIFO) queue
- Good for batch systems
- Unacceptable in interactive systems
  - Unpredictable turnaround time
- Disadvantages
  - Average turnaround time varies; seldom minimized

(figure 4.6)
Timeline for job sequence A, B, C using the FCFS algorithm.
© Cengage Learning 2014

Note: Average turnaround time: 16.67



(figure 4.7)
Timeline for job sequence C, B, A using the FCFS algorithm.
© Cengage Learning 2014

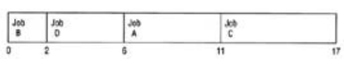
Note: Average turnaround time: 7.3

Shortest Job Next

- Nonpreemptive
- Also known as shortest job first (SJF)
- Job handled based on length of CPU cycle time
- Easy implementation in batch environment
  - CPU time requirement known in advance
- Does not work well in interactive systems
- Optimal algorithm when:
  - All jobs are available at same time
  - CPU estimates available and accurate

Shortest Job Next (cont'd.)

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



(figure 4.7)
Timeline for job sequence B, D, A, C using the SJN algorithm.

Average turnaround time = (2+6+11+17)/4 = 9.0

guarantees the shortest turnaround time

Priority Scheduling

- Nonpreemptive
- Preferential treatment for important jobs
  - Highest priority programs processed first
  - No interrupts until CPU cycles completed or natural wait occurs
- READY queue may contain two or more jobs with equal priority
  - Uses FCFS policy within priority
- System administrator or Processor Manager use different methods of assigning priorities

Priority Scheduling (cont'd.)

- Processor Manager priority assignment methods
  - Memory requirements
    - Jobs requiring large amounts of memory are allocated lower priorities (or vice versa)
  - Number and type of peripheral devices
    - Jobs requiring many peripheral devices are allocated lower priorities (or vice versa)

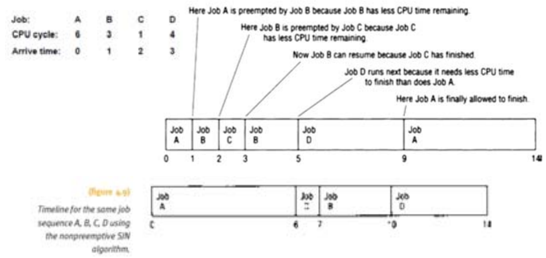
Priority Scheduling (cont'd.)

- Processor Manager priority assignment methods (cont'd.)
  - Total CPU time
    - Jobs having a long CPU cycle
    - Given lower priorities (vice versa)
  - Amount of time already spent in the system (aging)
    - Total time elapsed since job accepted for processing
    - Increase priority if job in system unusually long time

Shortest Remaining Time

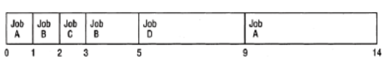
- Preemptive version of SJN
- Processor allocated to job closest to completion
  - Preemptive if newer job has shorter completion time
- Often used in batch environments
  - Short jobs given priority
- Cannot implement in interactive system
  - Requires advance CPU time knowledge
- Involves more overhead than SJN
  - System monitors CPU time for READY queue jobs
  - Performs context switching

Shortest Remaining Time (cont'd.)



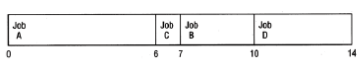
Job:	A	B	C	D
CPU cycle:	6	3	1	4
Arrive time:	0	1	2	3

SRT



Average turnaround time = (14 + 4 + 1 + 6)/4 = 6.25

SJN



Average turnaround time = (6 + 9 + 5 + 11)/4 = 7.75

Round Robin

- Preemptive
- Used extensively in interactive systems
- Based on predetermined slice of time
  - Each job assigned **time quantum**
- Time quantum size
  - Crucial to system performance
  - Varies from 100 ms to 1-2 seconds
- CPU equally shared among all active processes
  - Not monopolized by one job

Round Robin (cont'd.)

- Job placed on READY queue (FCFS scheme)
- Process Scheduler selects first job
  - Sets timer to time quantum
  - Allocates CPU
- Timer expires
- If job CPU cycle > time quantum
  - Job preempted and placed at end of READY queue
  - Information saved in PCB

Round Robin (cont'd.)

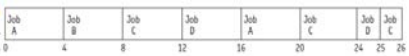
- If job CPU cycle < time quantum
  - Job finished: allocated resources released and job returned to user
  - Interrupted by I/O request: information saved in PCB and linked to I/O queue
- Once I/O request satisfied
  - Job returns to end of READY queue and awaits CPU

Round Robin (cont'd.)

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

Time quantum = 4

(figure 4.10)
Timeline for job sequence A, B, C, D using the preemptive round robin algorithm with time slices of 4 ms.



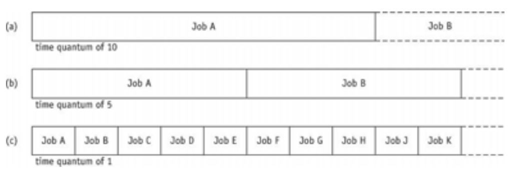
Job:	A	B	C	D
Turnaround:	20	7	24	22

Average turnaround time = (20+7+24+22)/4 = 18.25

Round Robin (cont'd.)

- Efficiency
  - Depends on time quantum size
    - In relation to average CPU cycle
- Quantum too large (larger than most CPU cycles)
  - Algorithm reduces to FCFS scheme
- Quantum too small
  - Context switching occurs
    - Job execution slows down
    - Overhead dramatically increased

Round Robin (cont'd.)



(figure 4.12)
Context switches for three different time quantums. In (a), Job A (which requires only 8 cycles to run to completion) finishes before the time quantum of 10 expires. In (b) and (c), the time quantum expires first, interrupting the jobs.

Round Robin (cont'd.)

- Best quantum time size
  - Depends on system
    - Interactive: response time key factor
    - Batch: turnaround time key factor
  - General rules of thumb
    - Long enough for 80% of CPU cycles to complete
    - At least 100 times longer than context switch time requirement

Multiple-Level Queues

- Works in conjunction with several other schemes
- Works well in systems with jobs grouped by common characteristic
  - Priority-based
    - Different queues for each priority level
  - CPU-bound jobs in one queue and I/O-bound jobs in another queue
  - Hybrid environment
    - Batch jobs in background queue
    - Interactive jobs in foreground queue
- Scheduling policy based on predetermined scheme
- Four primary methods

Case 1: No Movement Between Queues

- Simple
- Rewards high-priority jobs
  - Processor allocated using FCFS
- Processor allocated to lower-priority job only when high-priority queues empty
- Good environment for
  - Few users with high-priority jobs
  - When high priority jobs complete spend time with low-priority jobs

Case 2: Movement Between Queues

- Processor adjusts priorities assigned to each job
- High-priority jobs
  - Initial priority favorable
    - Treated like all other jobs afterwards
- Quantum interrupt
  - Job preempted
    - Moved to next lower queue
    - May have priority increased if issues I/O
- Good environment
  - Jobs handled by cycle characteristics (CPU or I/O)
  - Interactive systems

Case 3: Variable Time Quantum Per Queue

- Each queue given time quantum size
  - Size twice as long as previous queue
  - High: 1ms, medium: 2ms, low: 4 ms
- Fast turnaround for CPU-bound jobs
- CPU-bound jobs execute longer and given longer time periods
  - Improves chance of finishing faster

Case 4: Aging

- Ensures lower-level queue jobs eventually complete execution
- System keeps track of job wait time
- If too "old"
  - System moves job to next highest queue
  - Continues until old job reaches top queue
  - May drastically move old job to highest queue
- Advantage
  - Guards against indefinite unwieldy job postponement
    - Major problem: discussed further in Chapter 5

Earliest Deadline First

- Dynamic priority algorithm
- Preemptive
- Addresses critical processing requirements of real-time systems: deadlines
- Job priorities can be adjusted while moving through the system
- Primary goal:
  - Process all jobs in order most likely to allow each to run to completion before reaching their respective deadlines

Earliest Deadline First (cont'd.)

- Initial job priority: inversely proportional to its absolute deadline
  - Jobs with same deadlines: another scheme applied
- Priority can change as more important jobs enter the system
- Problems
  - Missed deadlines: total time required for all jobs greater than allocated time until final deadline
  - Impossible to predict job throughput: changing priority nature
  - High overhead: continual evaluation of deadlines

A Word About Interrupts

- Interrupt Types
  - Page interrupt (memory manager)
    - Accommodate job requests
  - Time quantum expiration interrupt
  - I/O interrupt (will be explained in chapter 7)
    - Result from READ or WRITE command issuance
  - Internal interrupt
    - Synchronous
    - Result from arithmetic operation or job instruction
  - Illegal arithmetic operation interrupt
    - Dividing by zero; bad floating-point operation

A Word About Interrupts (cont'd.)

- Interrupt Types (cont'd.)
  - Illegal job instruction interrupt
    - Protected storage access attempt
- Interrupt handler
  - Control program
    - Handles interruption event sequence

A Word About Interrupts (cont'd.)

- Nonrecoverable error detected by operating system
  - Interrupt handler sequence
    - Interrupt type described and stored
    - Interrupted process state saved
    - Interrupt processed
    - Processor resumes normal operation

Summary

- Processor Manager allocates CPU among all users
- Job Scheduler
  - Assigns job to READY queue
    - Based on characteristics
- Process Scheduler
  - Instant-by-instant allocation of CPU
- Scheduling algorithm is unique
  - Characteristics, objectives, and applications
- System designer selects best policy and algorithm
  - After careful strengths and weaknesses evaluation

Algorithm	Policy Type	Disadvantages	Advantages
First Come, First Served	Nonpreemptive	Unpredictable turnaround times; has an element of chance	Easy to implement
Shortest Job Next	Nonpreemptive	Indefinite postponement of some jobs; requires execution times in advance	Minimizes average waiting time
Priority Scheduling	Nonpreemptive	Indefinite postponement of some jobs	Ensures fast completion of important jobs
Shortest Remaining Time	Preemptive	Overhead incurred by context switching	Ensures fast completion of short jobs
Round Robin	Preemptive	Requires selection of good time quantum	Provides reasonable response times to interactive users; provides fair CPU allocation

(table 4.3)
Comparison of the scheduling algorithms discussed in this chapter.

Algorithm	Policy Type	Disadvantages	Advantages
Multiple-Level Queues	Preemptive/ Nonpreemptive	Overhead incurred by monitoring queues	Flexible scheme; allows aging or other queue movement to counteract indefinite postponement; is fair to CPU-bound jobs
Earliest Deadline First	Preemptive	Overhead required to monitor dynamic deadlines	Attempts timely completion of jobs

(table 4.3) (cont'd.)
Comparison of the scheduling algorithms discussed in this chapter.