Process Scheduling

CS 342 – Operating Systems

Ibrahim Korpeoglu
Bilkent University
Computer Engineering Department

Outline

- Introduction to Scheduling
  - Process Behavior
  - When to Schedule
  - Categories of Scheduling Algorithms
  - Scheduling Algorithms Goals
- Scheduling in Batch Systems
- Scheduling in Interactive Systems
- Scheduling in Real-Time Systems
- Thread Scheduling

Need for scheduling

- Modern computers runs several jobs (processes) at the same time.
  - Unix workstations
  - Personal Computers
  - Servers
  - Mainframes
- Multiple processes will be competing for CPU.
  - A daemon that *sends emails* in the background
  - A *video client* that is showing a video on the screen

  require different treatment for scheduling.

Scheduler

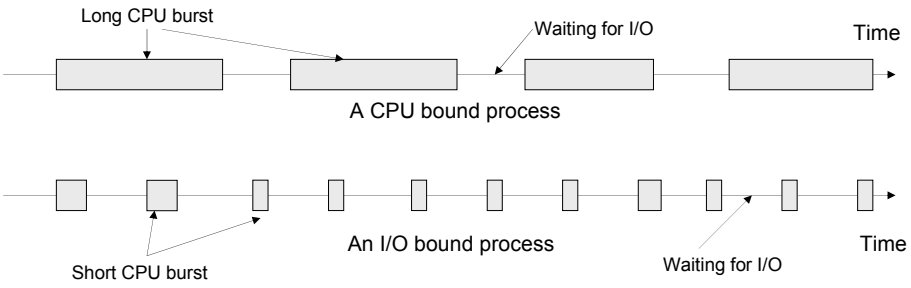
- Scheduler is responsible for
  - Picking up next process for CPU
  - Efficient CPU utilization
  - Making a balance in context switches
  - Achieving some scheduling goals.

Context Switches

- Context switch is taking one process out of CPU and giving the CPU to some other process
- Context switch involves
  - Switch from user mode to kernel mode
  - Save the state of current process
    - Values of registers
    - Memory mage (memory reference bits in the page table)
  - Select a new process
  - Load state of the new process
    - MMU must be loaded with memory map of the new process
    - Load CPU registers for the new process
  - Start the new process

Process Behavior

- Nearly all processes alternate between bursts of computing and I/O requests.
- I/O request involves blocking of process
  - Some I/O requests such as writing to video RAM involves CPU and therefore is considered as computing.



Process Behavior

- Compute-bound process
  - A process that spends most if its time in CPU activity.
- I/O-bound process
  - A process that spends most of its time in I/O activity.
- It is *the length of the CPU burst* that determines whether a process in compute-bound or I/O-bound.

When to schedule

- When a new process is created
  - Both parent and child are in ready state.
- When a process exits
  - CPU is no longer used
- When a process blocks on I/O or semaphore
- When an interrupt occurs from a device
  - A process might be blocking on that device and wil be waken up
- When hardware clock interrupts

Preemptive and non-preemptive scheduling

- A non-preemptive scheduling algorithms runs a process until it terminates or it blocks.
- A preemptive scheduling algorithms runs a process until
  - It terminates, or
  - It blocks, or
  - Its time slice expired by receiving a clock interrupt.

Categories of Scheduling Algorithms

- Different *application areas* and different *operating systems* have different *goals*
- Therefore they require *different scheduling algorithms*
- Three classes of environments
  - Batch systems
    - No user waiting for response. Non-preemptive scheduling can be used.
  - Interactive systems
    - Users are waiting for response. A process should not hog CPU for a long time. Preemption is necessary.
  - Real-time systems
    - Deadlines on operations need to be satisfied most of the time. Requires preemption.

Scheduling Algorithm Goals

- We need goals to judge how good an algorithm is.
- Some goals depend on the environment: batch, interactive, etc, …
- Some goals is valid for most environments.

Scheduling Algorithm Goals

- Fairness
  - Comparable processes should get comparable service.
- Enforcing System Policies
  - Some processes can be more important than the other ones.
- Balance
  - Keeping all parts of the system busy
  - Both CPU and I/O devices should be kept running
  - This will cause more work to be done.

Scheduling Algorithm Goals

- Throughput
  - Number of jobs per hour that system completes.
- Turnaround time
  - Average time from the moment a *batch* job is submitted until the moment it is completed.
- CPU utilization
  - Keeping CPU as busy as possible.

Scheduling Algorithm Goals

- Response Time
  - For interactive systems, the time between issuing a command and getting a result.
- Proportionality
  - For different operations, spending time proportional to the users expectations.
    - Modem connection time
    - Modem disconnection time.
- Meeting deadlines most of the time.
- Predictability
  - In completing operations
  - Important for multimedia: audio and video.

Systems and Goals

- All systems
  - Fairness, policy enforcement, balance
- Batch systems
  - Throughput, Turnaround time, CPU utilization
- Interactive systems
  - Response time, proportionality
- Real-time systems
  - Meeting deadlines
  - Predictability

Parameters of processes that affedt scheduling

- Arrival time
- Estimated duration until termination
- CPU or I/O bound?
- CPU usage so far
- Priority

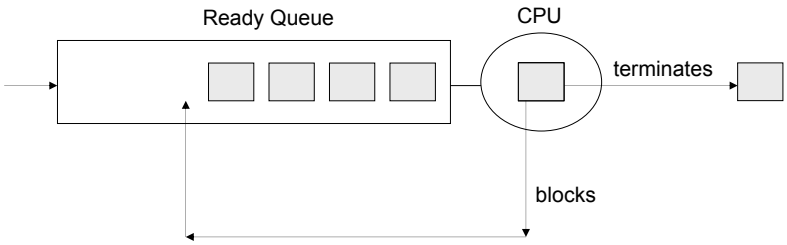
Scheduling in Batch Systems

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next
- Three-level scheduling

First Come First Served (FCFS)

- Non-preemptive
- Processes are assigned the CPU in the order they request it.
- Single queue of ready processes
- When a process blocks, the first process in the queue is run
- When a blocked process becomes ready, it is put to the end of the queue

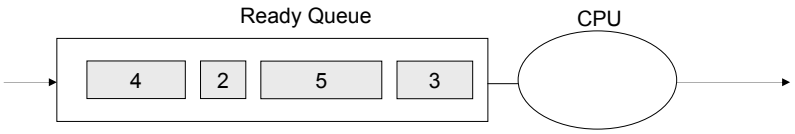
First Come First Served (FCFS)

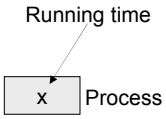


Shortest Job First

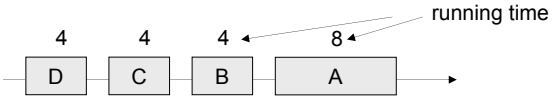
- Assumes run times of processes are known in advance
  - By using past experience.
- When several equally important jobs are available in the ready queue, the scheduler picks up the *shortest job first*.
- Provides optimal average turnaround time.
- It is optional when all the jobs are available simultaneously.

Shortest Job First





Shortest Job First



Arrival sequence of 4 jobs: A arrives first, then B arrives immediately

If we run them in the original order: D C B A

Turnaround time or each process: 20 16 12 8

Avg. Turnaround time = (20+16+12+8) / 4 = 13 units

If we run them using shortest first: A D C B

Turnaround time or each process: 20 12 8 4

Avg. Turnaround time = (20+12+8+4) / 4 = 11 units

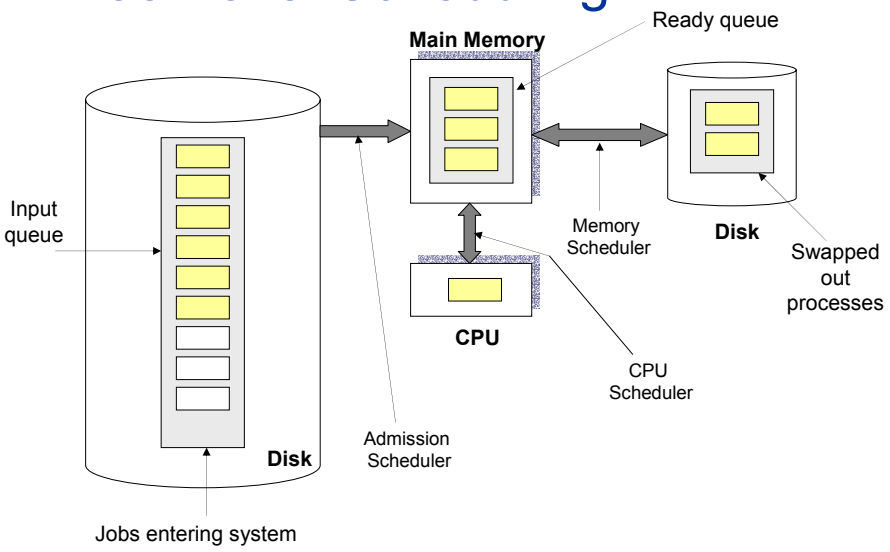
Shortest Remaining Time Next

- This is a preemptive version of Shortest-Time First Algorithm
- Scheduler always chooses the process whose remaining time is the shortest.
  - Running time has to be known in advance.
- When a new job arrives, the running of the *new job* is compared with the remaining running time of the *running job*.
  - If new job has shorter *running time* than the *remaining running time* of the running job, then the new is assigned the CPU.
- Allows new short jobs to get good service.

Three-Level Scheduling

- Batch systems allow scheduling at three levels
  - When jobs arrive system they are first put into an input queue stored on the hard disk.
  - The admission scheduler decides which of the jobs in input queue to run (assume M jobs can run).
  - The memory scheduler limits the number jobs (say N) that can simultaneously exist on the memory and swaps back and forth the processes between memory and disk.
    - N is called degree of multiprogramming
  - M is greater or equal to N.
  - CPU scheduler decides which job to run among the jobs that reside in memory.

Three-Level Scheduling



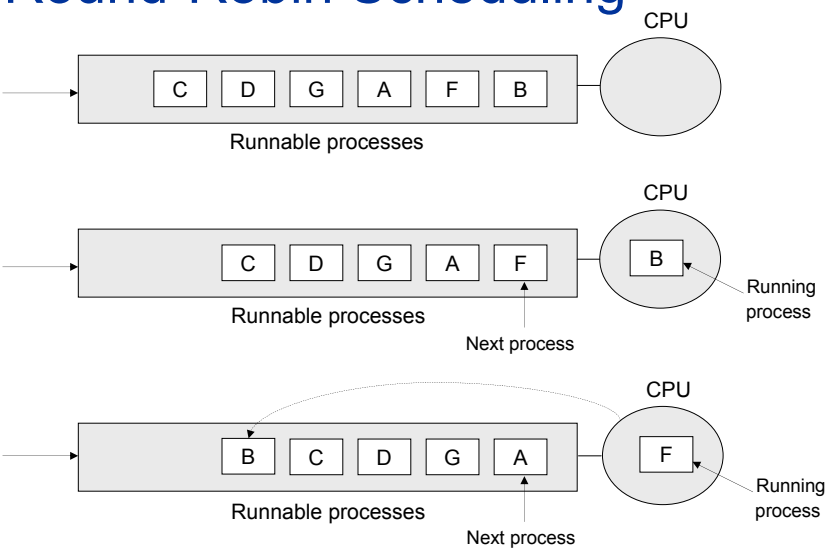
Scheduling in Interactive Systems

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-share Scheduling

Round-Robin Scheduling

- Simple, fair, widely used algorithm
  - Each process is assigned a time interval, called its quantum, which it is allowed to run.
  - If the process is still running at the of quantum, the CPU is preempted and given to another process.
  - If the process is blocked before it uses its **quanta**, it is again taken out off CPU.
- Easy to implement.

Round-Robin Scheduling



Round-Robin Scheduling

- The issue is determining the length of the quantum.
- If quantum is set to *short*, context switch overhead will cause waste of CPU cycles
- If quantum is set too *long*, average response time for processes will increase
  - May not be acceptable beyond a threshold.
- Rule of thumb: A quantum around 20-50ms is reasonable

Priority Scheduling

- Round-Robin makes implicit assumption that all processes are equally important.
  - This is not the case in real-life.
  - A daemon process that send e-mails should be assigned a lower-priority than a process displaying a video film on the screen in real-time.
- A process will be assigned priorities.

Priority Scheduling

- Two ways to implement
- First method is:
  - Scheduler picks up the highest priority process and runs it.
  - At each clock tick (depending on hardware clock)
    - The priority of the running process is decreased.
    - A check is made if some other process in the ready queue now has higher priority
      - If such a process with higher priority exists in the ready queue, the currently running process is taken out off CPU and that process is scheduled to run.

Priority Scheduling

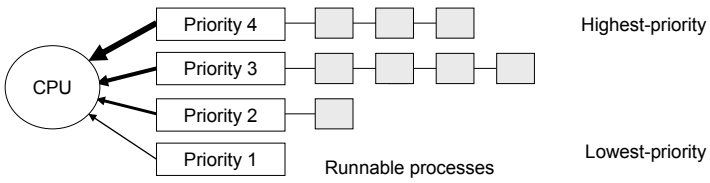
- Second method is:
  - Each process is assigned a maximum quantum.
  - Scheduler picks up the highest priority process and runs it.
  - If the quantum of the running process is used up, next highest priority process is scheduled to run.

Priority Scheduling

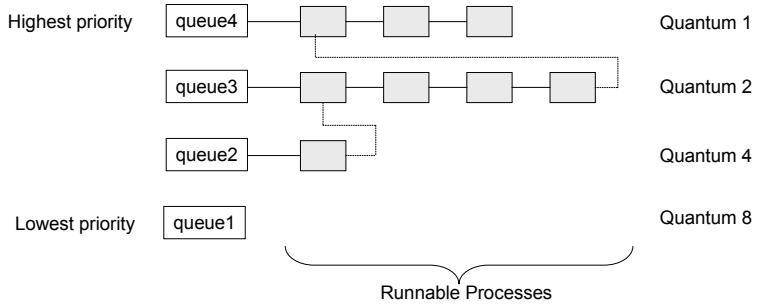
- Priorities can be assigned statically or dynamically.
- Dynamic assignment
  - Some processes are highly I/O bound
    - CPU can be given to such process immediately when it wants CPU:
      - so that it can start its next I/O request.
      - This will cause parallelism with some other compute-bound process.
    - Priority for I/O process can be adjusted depending on the fraction of last quantum (f) used by that process.
      - Priority = 1/f

Priority Scheduling

- Grouping processes into priority classes.
  - *Priority scheduling* among classes.
  - *Round-robin scheduling* within each class.
- Priorities need to adjusted occasionally so that lower priority classes do not starve always.



Multiple Queues



A process will be executed: 1 quantum in priority 4
2 quantums in priority 3
4 quantums in priority 2
8 quantums in priority 1

Shortest Process Next

- An interactive process cycle:
  - 1. Wait for next command from user
  - 2. Execute the command
  - 3. Goto Step 1
- The time to execute a command is the response time.
- We can estimate the running time of commands.
- Select the process that has the minimum command running time.

Shortest Process Next

- Estimation of running time of commands in a process
  - T_0 : estimate of running time so far
  - T_1 : running time measurement of the last command exeutes.
  - New estimate = a • T_0 + (1-a) • T_1 Aging
    - a is between 0 and 1.

$$T_0,\ \frac{T_0}{2}+\frac{T_1}{2},\ \frac{T_0}{2}+\frac{T_1}{4}+\frac{T_2}{4},\ \frac{T_0}{8}+\frac{T_1}{8}+\frac{T_2}{4}+\frac{T_3}{2}$$

Guaranteed Scheduling

- If there are N process, then each process shuld get 1/n of CPU service
- For each process
  - Measure CPU usage: X
  - Comopute the fair CPU allocation for that process: time_from_process_create/N = Y
- For each process computer X/Y
- Select the process that has minimum X/Y at scheduling decision points

Lottery Scheduling

- Algorithm sketch
  - Give lottery tickets to each process (each ticket has a unique number)
  - Choose a ticket randomly (do a lottery)
  - Execute the process that holds that ticket
- Each process may have different number of tickets
- Cooperating processes may exchange tickets.

Fair Share Scheduling

- There are users and processes in a sustem.
- We can allocate COU to users and trhen enforce usage according to allocation.
- Example:
  - 1)
    - 2 users in a system. Each one should get %50 service
      - User 1 processes are: A B C D
      - User 2 processes are: E
    - Sequence of exeution of processes could be: A E B E C E D E A E...
  - 2)
    - User 1 should get 66%, user 2 should get 33% service
    - Sequence of execution could be in this case: A B E C D E A B E C D E A B E