Operating Systems Fundamentals

CPU Scheduling

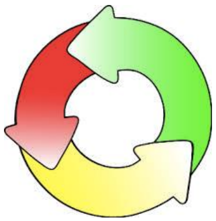
Scheduling

- Recall that the objective of multiprogramming is to **have some process running at all times**
- With multiprogramming, several processes are kept in memory at one time.
  - When one process has to wait, the OS takes the CPU away from that process and gives it to another – this pattern continues.
- Scheduling of this kind is a fundamental OS function.

Process Execution CPU – I/O Burst Cycle

Process execution consists of a cycle of

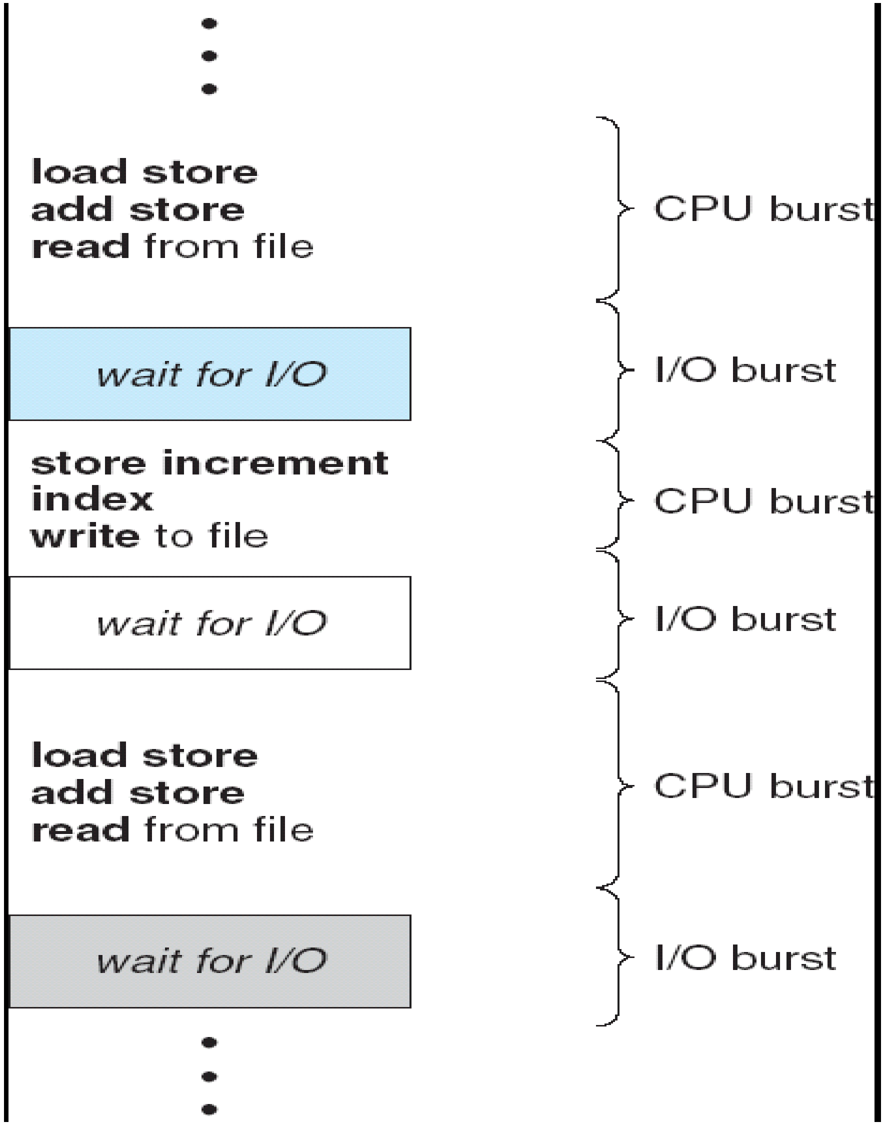
CPU execution and I/O wait.



DEFINITIONS

- **CPU Burst** is a time interval
  - when a process uses the CPU only
- **I/O burst** is a time interval
  - when a process uses I/O devices only

Typical Alternating Sequence of CPU and I/O Bursts



- Process execution <u>begins</u> with a CPU burst, followed by an I/O burst.....then CPU burst, I/O burst, etc...
- Finally, the CPU burst ends with a system request to terminate execution

CPU Scheduler

- The CPU scheduler / short-term scheduler selects from among the processes in memory that are ready to execute (i.e. in the ready queue), and allocates the CPU to one of them

- **Non-pre-emptive** scheduling:
  - A process can run either to **completion** or **waiting**

- **Pre-emptive** scheduling:
  - A process can be **temporarily** halted in order for another to run
    - The halted process goes to a ready state.

Nonpreemptive vs Preemptive

Nonpreemptive

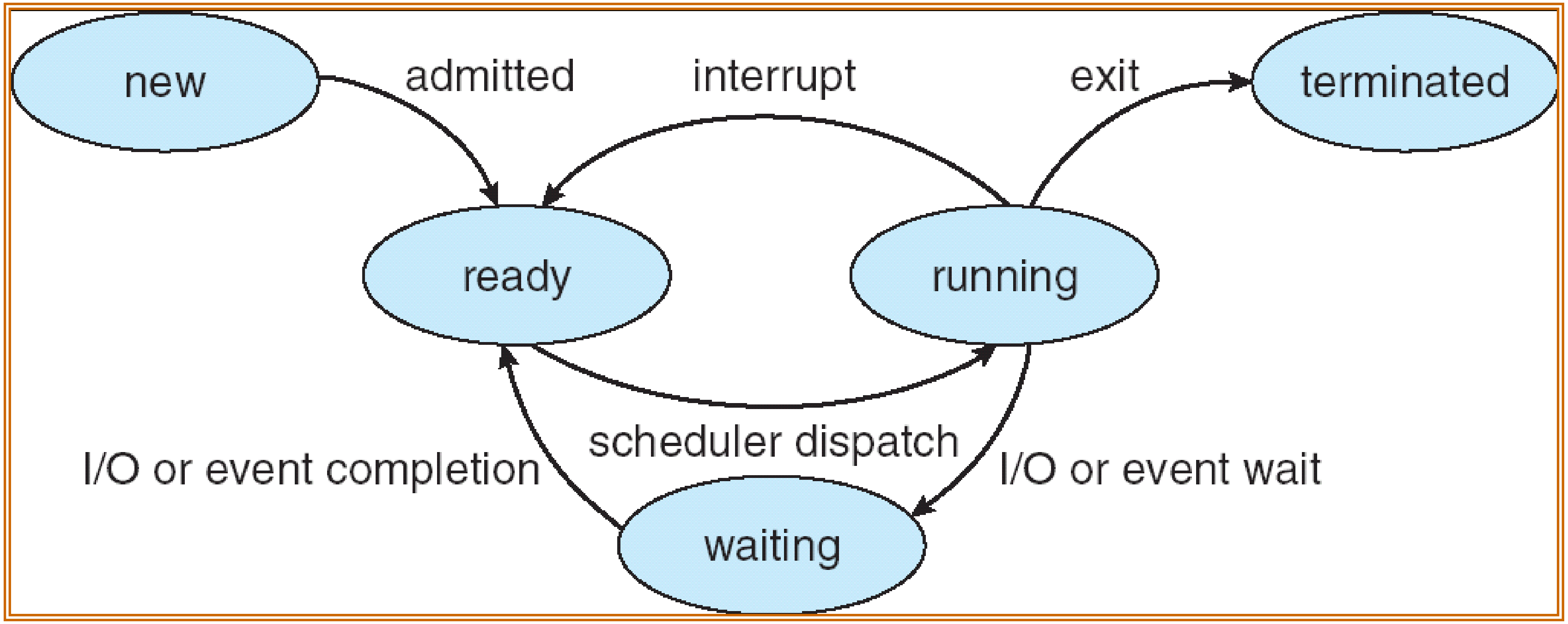
- Once a process is in the running state, it will continue until it terminates or blocks itself for I/O

Preemptive

- a currently running process may be interrupted and moved to ready state by the OS

- Preemption may occur when a new process arrives, on an interrupt, or periodically

Process State Transitions



Scheduling: Pre-emptive or Non Pre-emptive

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

Which above are pre-emptive?

Which are non-pre-emptive?

Scheduling:
Pre-emptive or Non Pre-emptive

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

Which above are pre-emptive?

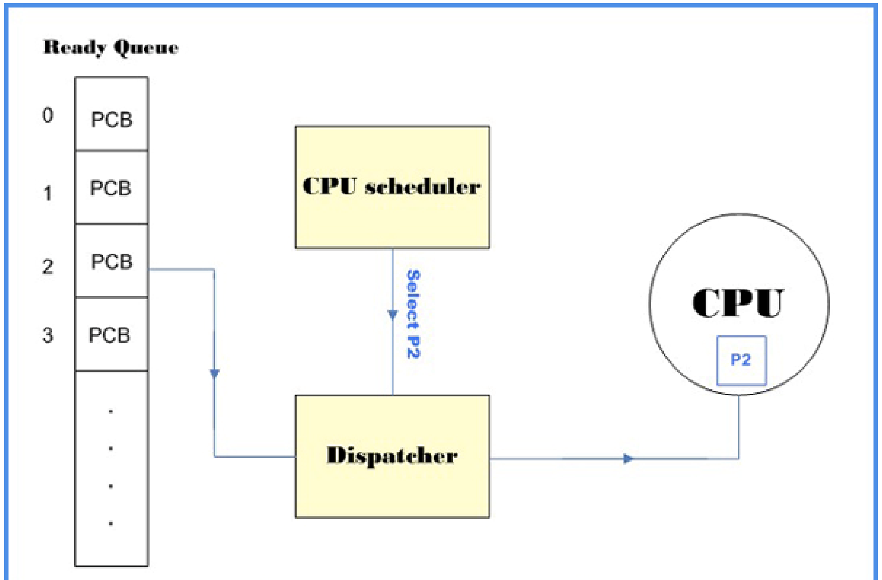
Which are non-pre-emptive?

- Scheduling under 1 and 4 is ***non-preemptive***
- All other scheduling (2 and 3) is ***pre-emptive***

Dispatcher

- **Dispatcher** module gives control of the CPU
  - to the process selected by the short-term (CPU) 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 one running



Scheduler Vs. Dispatcher

Scheduling Criteria

Scheduling Criteria

are factors that affect the choice of Scheduling **Algorithm**

- CPU Utilization
  - **CPU should be as busy as possible**
- Throughput
  - **Number of processes completed per unit time**
- Turnaround time
  - **Sum of time spent waiting, executing and doing I/O**
- Waiting time
  - **Sum of the time spent waiting in a ready queue**
- Response time
  - **A measure from time of submission to first response**

Optimisation Criteria

So Scheduler development should look to perform as best as possible on the following:

So which of below should be

maximised versus minimised

CPU utilization	Max/Min?
Throughput	Max/Min?
Turnaround time	Max/Min?
Waiting time	Max/Min?
Response time	Max/Min?

Optimisation Criteria

So Scheduler development should look to perform as best as possible on the following:

So which of below should be

maximised versus minimised

CPU utilization	Maximised
Throughput	Maximised
Turnaround time	Minimised
Waiting time	Minimised
Response time	Minimised

Scheduler Approaches

- Schedulers are not always able to meet the above requirements
- Different solutions exist to support the scheduler operations
- Each approach tries to address different concerns and optimisations

Scheduler Approaches, continued

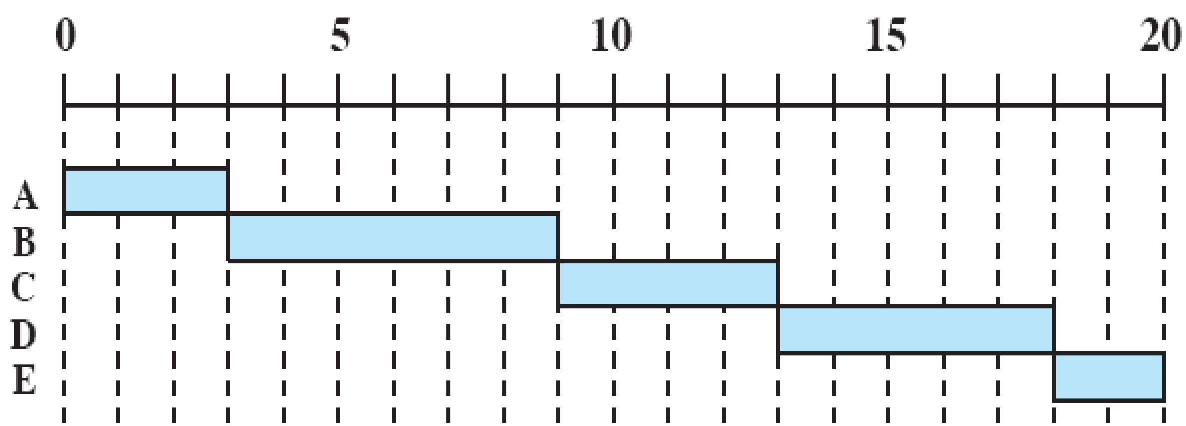
Different scheduler approaches include:

1. **First Come, First Served** **(FCFS)**
2. **Shortest Job First** **(SJF)**
3. **Shortest Remaining Job First** **(SRJF)**
4. **Round Robin** **(RR)**
5. **Priority Scheduling** **(PS)**

First-Come, First-Served (FCFS)

- Simplest scheduling policy
- Also known as first-in-first-out (FIFO) or a strict queuing scheme
- When the current process ceases to execute, the process that has been longest in the Ready queue is selected

First-Come-First Served (FCFS)



Example: First-Come, First-Served (FCFS)

Process	Burst Time
P1	24
P2	3
P3	3

- Suppose the processes are executed in the order P1 , P2 , P3
- Execution schedule will look like this (Gantt chart):

- Waiting time for P1 = 0; P2 = 24; P3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling, continued

Suppose the processes arrive in the order:

P2 , P3 , P1

- The Gantt chart for the schedule is:

- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Causes a long delay for short processes

    (Consider one CPU-bound (e.g. P1) and many I/O-bound processes)

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is Optimal/Idealistic
  - gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request. (We could ask the user!)
  - Time is set based on the expected running time

Shortest-Job-First (SJF) Scheduling, cont.

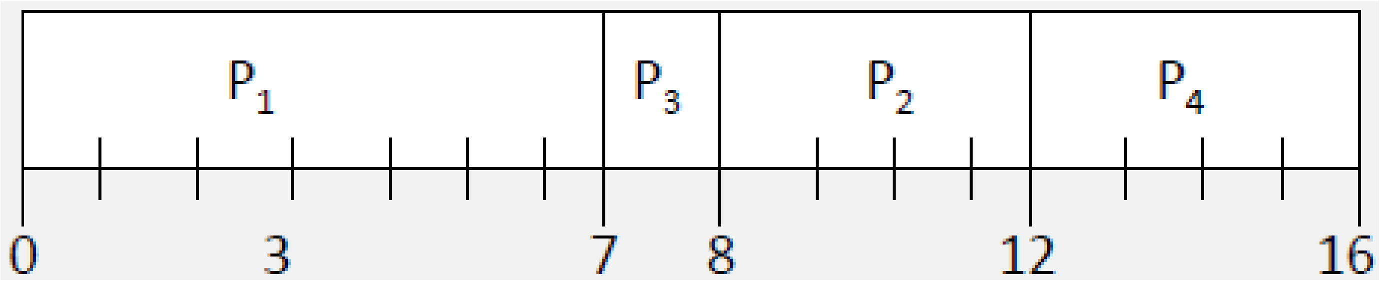
Shortest Job First scheduling may operate in 2 ways:

- **Non-preemptive SJF:**
  - Once CPU is given to the process
  - It cannot be pre-empted until it completes its CPU burst
- **Pre-emptive SRJF:**
  - If a new process arrives with a CPU burst length less than remaining time of current executing process, then pre-empt.
  - This scheme is know as the **Shortest-Remaining-Time-First** (**SRTF**) **Shortest-Remaining-Job-First** (**SRJF**)

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- SJF (non-premptive)



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4
- Q: Why did P3 run before P2?

Shortest Remaining Job First (SRJF)

- Preemptive version of SJF

- Scheduler always chooses the process that has the shortest expected remaining processing time

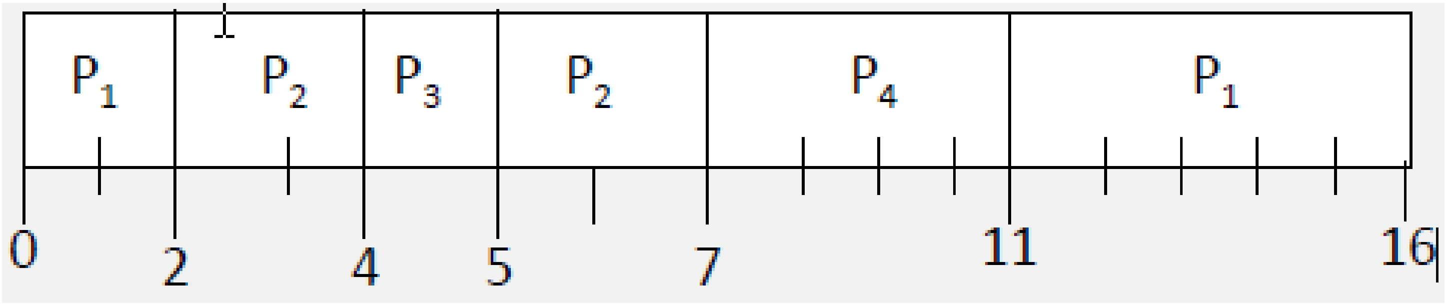
- The choice to schedule arises when a new process arrives at the ready queue while a previous process is executing.

- A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst first

Example: Shortest Remaining Job First (SRJF)

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

- The Gantt chart



- Average waiting time = (9 + 1 + 0 +2)/4 = 3
- Average Turnaround Times (16+5+1+6)/4 = 7
- **Q's**: Why does P2 and then P3 and then P4 all preempt P1
- Why does P3 preempt P2?

Exercise: SRJF

Process	Arrival Time	Burst Time
P1	0.0	8
P2	1.0	4
P3	2.0	9
P4	3.0	5

- Draw the Gantt chart
- Calculate wait times and average wait time
- Calculate turnaround time and average turnaround time.
- Compare the average wait time and average turnaround time with the nonpre-emptive SJF results.

There is a problem with SJF (and SRJF)

Round Robin (RR)

- SJF, while efficient, does not ensure all processes will run
  - If shorter processes continually arrive, longer and waiting processes may not get a chance to run if they are constantly put to the back of the ready queue
- The Round Robin solution is to evenly split the available time across all processes
  - Each process
    - gets a small unit of CPU time (time quantum),
    - usually 1 – 100 milliseconds.
  - After this time elapses,
    - the process is preempted
    - and added to the end of the ready queue (a **circular** queue),
    - or it finishes.

Round Robin (RR), continued

- 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 (as extreme example) – why is this?
  - q small => q must be large with respect to context switch, otherwise context switch overhead is too high

q usually 10ms to 100ms, context switch < 10 usec

Example of RR with Time Quantum = 4

Process	Burst Time
P1	24
P2	3
P3	3

- The Gantt chart looks like:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

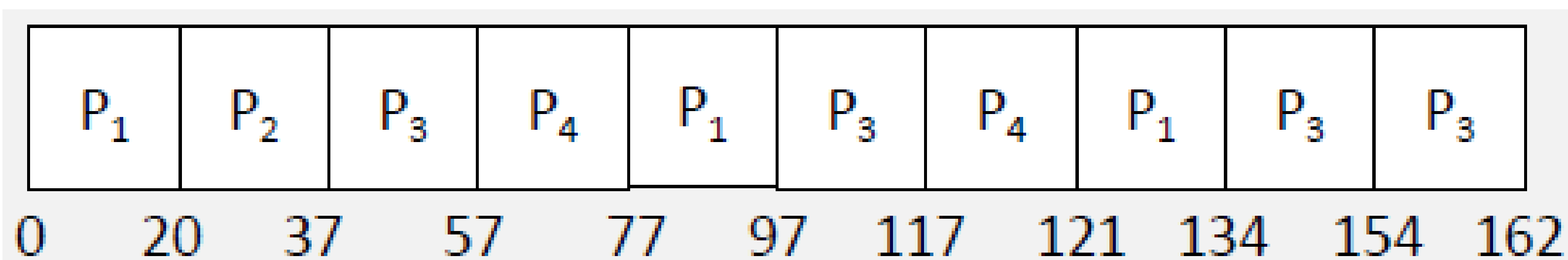
0 4 7 10 14 18 22 26 30

- Typically, RR has
  - Worse/higher average turnaround than SJF,
  - but better response time
- Average Turnaround is defined as
  - Sum of times of completion ÷ number of processes

Example of RR with Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

- The Gantt chart looks like:



Process Priorities

- Round Robin ensures fairness in relation to all processes

- But …. For correct OS behaviour
  - some processes have a need to run more efficiently than others
  - a process may have to perform critical OS functions

- Gives rise to the notion of **Process Priorities**
  - A number associated with a process
    - saying how important it is
  - Typically a small scale, 1 – 5, e.g. 1 is highest priority
  - Many processes will have the same priority
    - Remember, there are 1000's of processes

Priority Scheduling

- **The CPU is allocated to the process with the highest priority** (smallest integer = highest priority)

- When a process arrives at the ready queue,
  - It's priority is compared with the priority of the currently running process.

- Priority scheduling can be either
  - Preemptive
  - Non-preemptive

- We have already seen a priority scheduling approach ??

SJF

is a priority scheduling approach

where priority is the predicted next CPU burst time

Priority Scheduling
Preemptive and Non-preemptive

Let P_c = Priority of currently running process

P_n = Priority of newly arrived process

Preemptive Priority

If $P_n > P_c$ means priority is higher, not the number associated with priority

Then P_n preempts CPU

Non-preemptive Priority

If $P_n > P_c$

Then P_n is put at the head of the ready queue

i.e. best place to be when you want to run asap

Priority Scheduling, continued

Priority scheduling can cause issues for running processes:

- **Starvation**:
  - A low priority process does not get any time to run on the CPU
  - Processes with higher priority are always run,
    no time is given to lower priority processes

- How do we solve this problem?

Priority Scheduling, continued

Priority scheduling can cause issues for running processes:

- **Starvation**:
  - A low priority process does not get any time to run on the CPU
  - Processes with higher priority are always run,
    no time is given to lower priority processes

- How do we solve this problem?
- Process Aging is used to make sure
  - all processes get a chance to run on the CPU
  - Increase a processes priority
    if it has not been run for a certain amount of time