High Performance Computing
Lecture 17

Matthew Jacob

Indian Institute of Science

Some Possible Process States

- A process could be Waiting for something to happen
  - Example: A parent process that made the wait() system call is waiting (sleeping) for its child process to terminate
- A process could be Ready for the OS to cause it to run
- Running, Waiting, Ready

Process Management

- What should the OS do when a process does something that will involve a long time?
  - e.g., Anything that involves a hard disk access (file read/write operation, page fault, …)
- If it does nothing, the processor will be idle for billions of cycles
  - The processor could have executed billions of instructions instead during that time

Process Management

- OS should try to maximize processor utilization
  - utilization: fraction of time that the processor is busy
- OS could change status of that process to `Waiting’ and make another process `Running’
- Question: Which other process?
  - Determined by the process scheduler

Process Scheduler

- The part of the OS that manages the sharing of CPU time among processes
- Possible considerations that the scheduler could use in making scheduling decisions
  - Minimize average program execution time
  - Fairness to all the programs in execution

Process Scheduling Policies

- Idea 1: Let the currently Running process continue to do so until it does something that involves a long time
  - Then switch to one of the Ready processes
  - But what if the currently Running process is executing an infinite loop

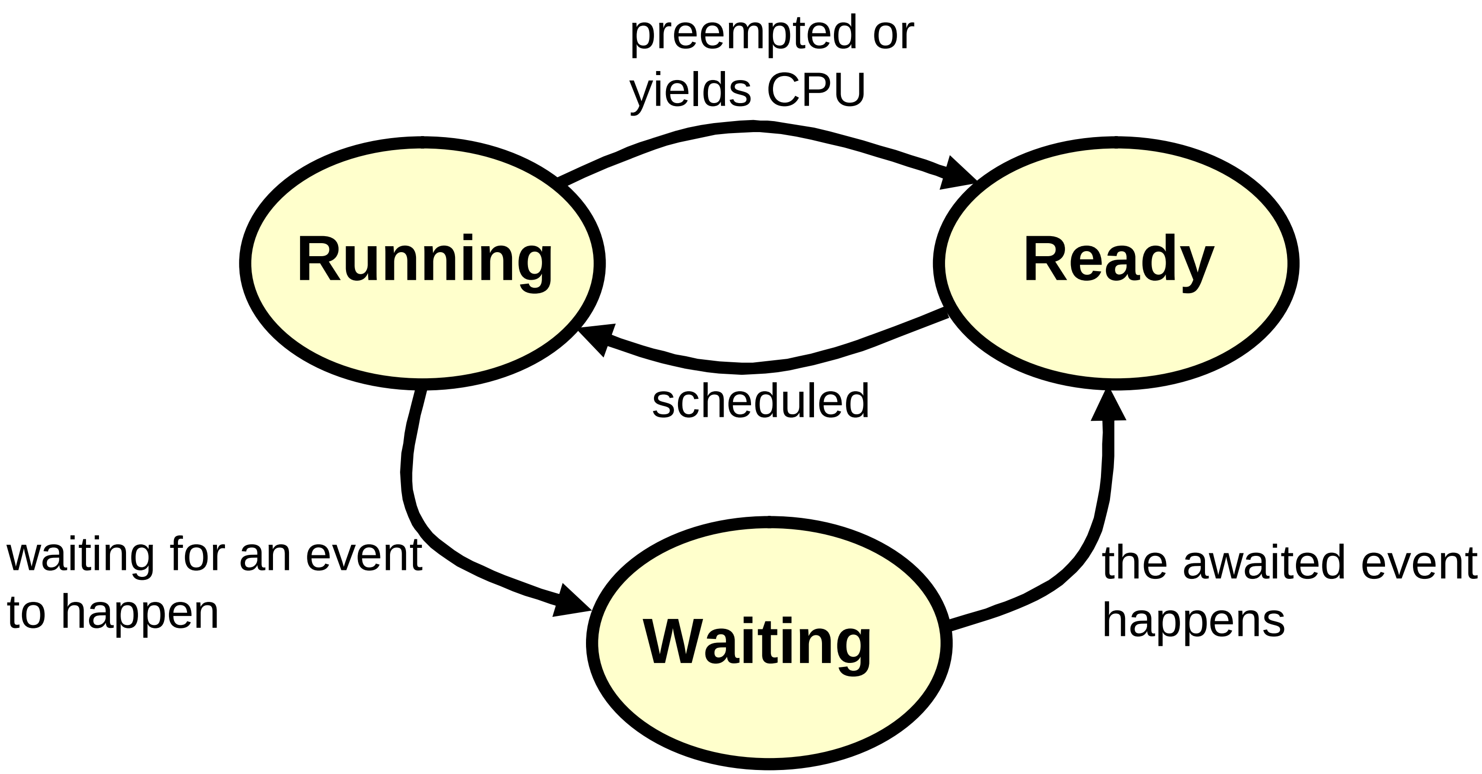
    ```
    while (1) ;                 /* a simple infinite loop */
    ```

  - No other process would ever get to run if the OS uses Idea 1
  - This is an example of a Non-preemptive policy
  - It does not seem to be very fair to other processes

Process Scheduling Policies

- Preemptive vs Non-preemptive
  - Preemptive policy: one where the OS `preempts' the running process from the CPU even though it is not waiting for something
  - Idea: give a process some maximum amount of CPU time before preempting it, for the benefit of the other processes
  - CPU time slice: maximum amount of CPU time allotted to a process before preempting it from the CPU

Process State Transition Diagram



Context Switch

- When the OS changes which process is currently running on the CPU
- The switch takes some time, as it involves replacing the hardware state of the previously running process with that of the newly scheduled process
  - Saving HW state of previously running process
  - Restoring HW state of newly scheduled process
- Amount of time would help in deciding what a reasonable CPU timeslice value would be

Non-Preemptive Scheduling Policies

1. First Come First Served (FCFS)
   - Idea: Maintain a queue of ready processes
   - Queue: a data structure with 2 operations
     1) Insert: Add a new process to the back of the queue
     2) Delete: Remove the process from the front of the queue

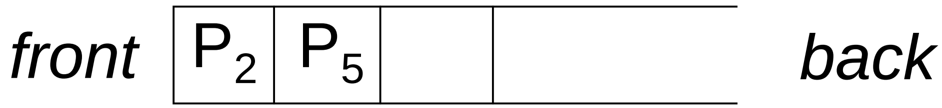
front	P_1	P_2	P_5		*back*

   - Schedule next the process from the front of the ReadyQ

Non-Preemptive Scheduling Policies

1. First Come First Served (FCFS)
   - Idea: Maintain a queue of ready processes
   - Queue: a data structure with 2 operations
     1) Insert: Add a new process to the back of the queue
     2) Delete: Remove the process from the front of the queue

after P_1 has been scheduled to run



   - Schedule next the process from the front of the ReadyQ

Non-Preemptive Scheduling Policies

1. First Come First Served (FCFS)
2. Shortest Process Next
   - The policy which results in the lowest possible average program execution time
   - Schedule next that ready process which requires the least CPU time in order to finish execution
   - Problem: How do you estimate how much more CPU time each process will require?

Preemptive Scheduling Policies

1. Round robin
   - Maintain a FCFS ReadyQ
   - When the currently running process is preempted, schedule the process from the front of the ReadyQ
   - Insert the previously running process at the end of the ReadyQ
   - This is much fairer than any of the non-preemptive scheduling policies