Comp 204: Computer Systems and Their Implementation

Lecture 10: Process Scheduling

Today

- Deadlock
  - Wait-for graphs
  - Detection and recovery

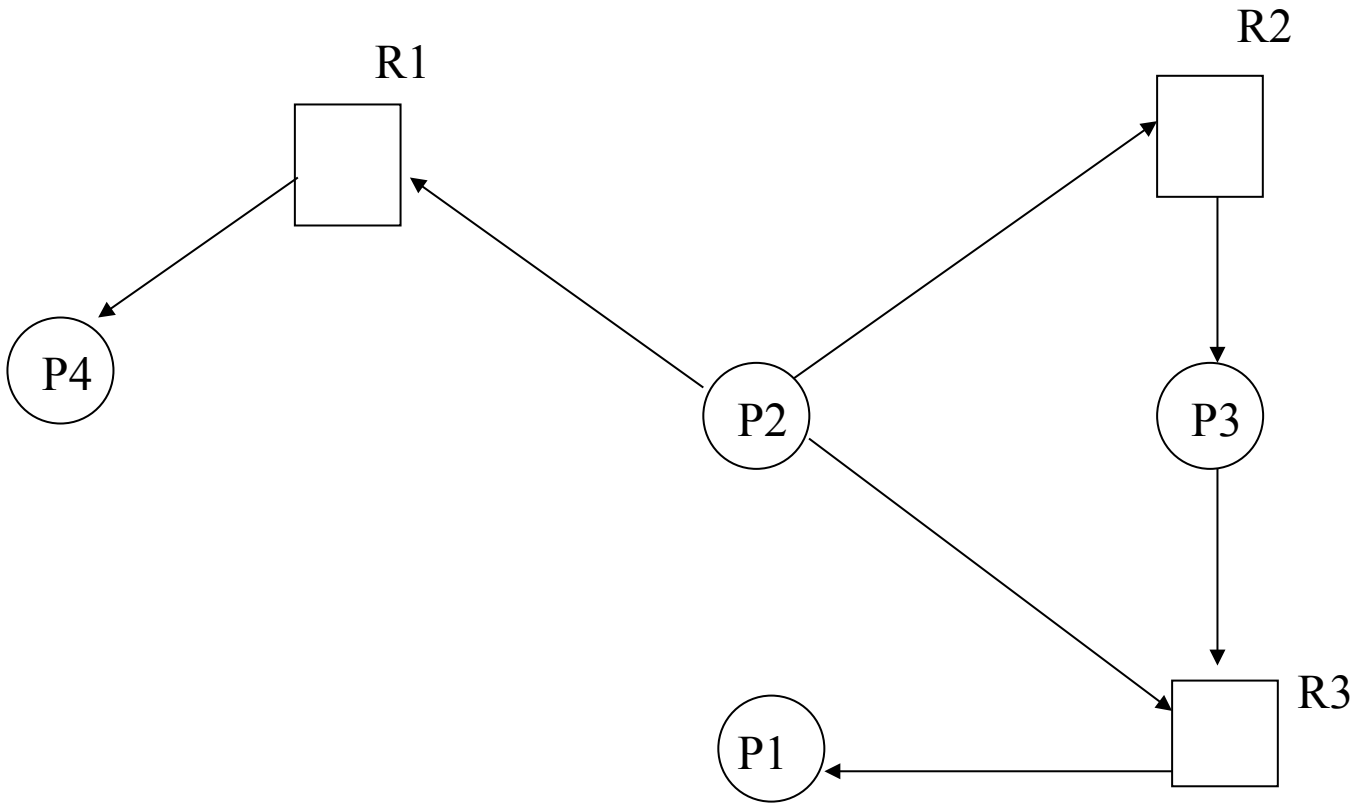
- Process scheduling
- Scheduling algorithms
  - First-come, first-served (FCFS)
  - Shortest Job First (SJF)

Wait-For Graph

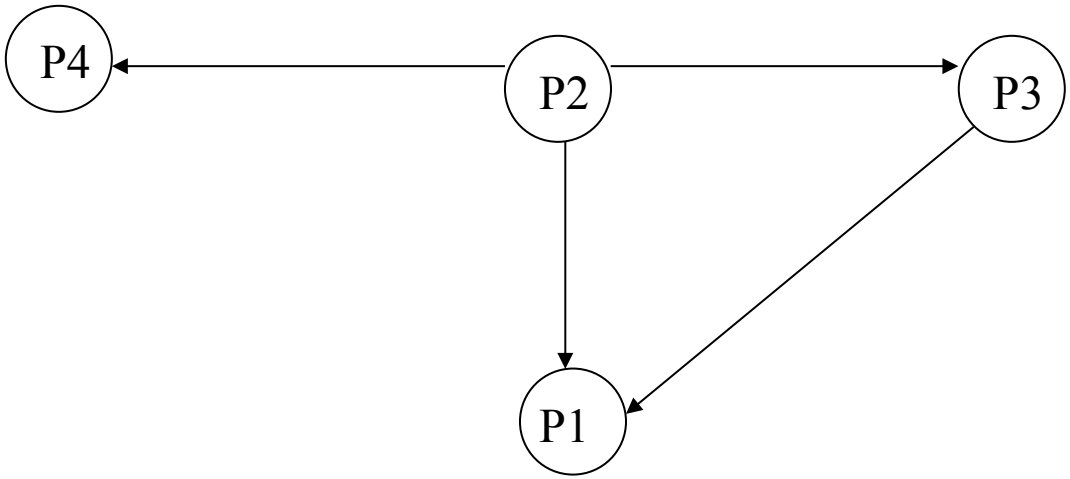
- Precise definition:
- An edge from P_i to P_j implies that process P_i is waiting for process P_j to release a resource that P_i needs
- An edge $P_i \rightarrow P_j$ exists in a wait-for graph if and only if the corresponding resource-allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q
- Deadlock is present if there is a cycle in the wait-for graph

Exercise

- Construct the wait-for graph that corresponds to the following resource allocation graph and say whether or not there is deadlock:



Corresponding Wait-For Graph



Wait-For Graph

- In order to be able to effectively detect deadlock, the system must maintain the wait-for graph and run an algorithm to search for cycles, at regular intervals

- Issues:
  - How often should the algorithm be invoked?
    - Costly to do for every request
    - May be better to do at regular intervals, or when CPU utilisation deteriorates too much
  - How many processes will be affected by the occurrence of deadlock?
    - One request may result in many cycles in the graph

Detection and Recovery

- Once the detection algorithm has identified a deadlock, a number of alternative strategies could be employed to recover from the situation

- Recovery could involve process termination
  - All involved
    - May be huge loss of computation
  - One at a time
    - Expensive: requires re-run of detection algorithm after each termination
- Recovery could involve preemption
  - Choice of victim
  - Rollback
  - Starvation

Scheduling

- In any multiprogramming situation, processes must be scheduled

- The long-term scheduler (job scheduler) decides which jobs to load into memory
  - must try to obtain a good job mix: compute-bound vs. I/O-bound

- The short-term scheduler (CPU/process scheduler) selects next job from ready queue
  - Determines: which process gets the CPU, when, and for how long; when processing should be interrupted
  - Various different algorithms can be used…

Scheduling

- The scheduling algorithms may be preemptive or non-preemptive
  - Non-preemptive scheduling: once CPU has been allocated to a process the process keeps it until it is released upon termination of the process or by switching to the ‘waiting’ state
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - Invoked during every switch: needs to be fast
- CPU–I/O Burst Cycle: process execution consists of a cycle of CPU execution and I/O wait
- So what makes a good process scheduling policy?

Process Scheduling Policies

- Several (sometimes conflicting) criteria could be considered:
- Maximise throughput: run as many processes as possible in a given amount of time
- Minimise response time: minimise amount of time it takes from when a request was submitted until the first response is produced
- Minimise turnaround time: move entire processes in and out of the system quickly
- Minimise waiting time: minimise amount of time a process spends waiting in the ready queue

Process Scheduling Policies

- Maximise CPU efficiency: keep the CPU constantly busy, e.g. run CPU-bound, not I/O bound processes
- Ensure fairness: give every process an equal amount of CPU and I/O time, e.g. by not favouring any one, regardless of its characteristics
- Examining the above list, we can see that if the system favours one particular class of processes, then it adversely affects another, or does not make efficient use of its resources
- The final decision on the policy to use is left to the system designer who will determine which criteria are most important for the particular system in question

Process Scheduling Algorithms

- The short-term scheduler relies on algorithms that are based on a specific policy to allocate the CPU

- Process scheduling algorithms that have been widely used are:
  - First-come, first-served (FCFS)
  - Shortest job first (SJF)
  - Shortest remaining time first (SRTF)
  - Priority scheduling
  - Round robin (RR)
  - Multilevel queues

First-Come, First Served

- Simplest of the algorithms to implement and understand
  - Uses a First-In-First-Out (FIFO) queue
- Non-preemptive algorithm that deals with jobs according to their arrival time
  - The sooner a process arrives, the sooner it gets the CPU
  - Once a process has been allocated the CPU it keeps until released either by termination or I/O request
- When a new process enters the system its PCB is linked to the end of the ‘ready’ queue
- Process is removed from the front of the queue when the CPU becomes available, i.e. after having dealt with all the processes before it in the queue

Example

- Suppose we have three processes arriving in the following order:
  - P_1 with CPU burst of 13 milliseconds
  - P_2 with CPU burst of 5 milliseconds
  - P_3 with CPU burst of 1 millisecond

- Using the FCFS algorithm we can view the result as a Gantt chart:

First-Come, First Served

- Given the CPU burst times of the processes, we know what their individual wait times will be:
  - 0 milliseconds for P_1
  - 13 milliseconds for P_2
  - 18 milliseconds for P_3

- Thus, the average wait time will be (0 + 13 + 18)/3 = 10.3 milliseconds

- However, note that the average wait time will change if the processes arrived in the order P_2, P_3, P_1

Exercise

- What will the average wait time change to if the processes arrived in the order P_2, P_3, P_1?

Answer

- P_2 with CPU burst of 5 milliseconds
- P_3 with CPU burst of 1 millisecond
- P_1 with CPU burst of 13 milliseconds

- Represented as the following Gantt chart:

- Thus, the average wait time will be (0 + 5 + 6)/3 = 3.7 milliseconds

First-Come, First Served

- Advantages:
  - Very easy policy to implement

- Disadvantages:
  - The average wait time using a FCFS policy is generally not minimal and can vary substantially
  - Unacceptable for use in time-sharing systems as each user requires a share of the CPU at regular intervals
    - Processes cannot be allowed to keep the CPU for an extended length of time as this dramatically reduces system performance

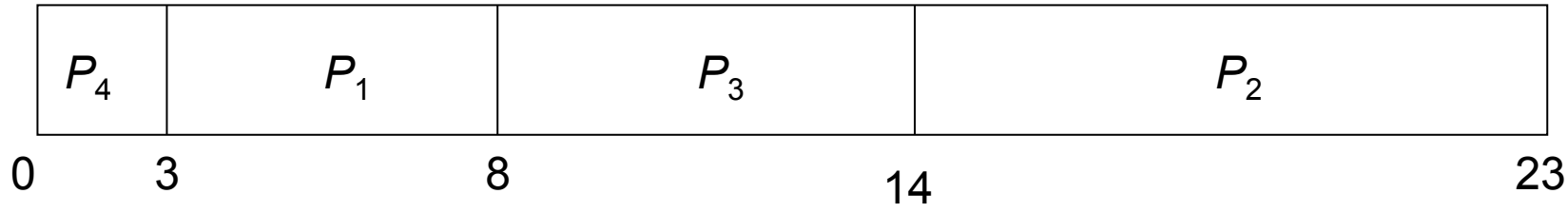
Shortest Job First

- Non-preemptive algorithm that deals with processes according to their CPU burst time
  - When the CPU becomes available it is assigned the next process that has the smallest burst time
  - If two processes have the same burst time, FCFS is used to determine which one gets the CPU

- Suppose we have four processes as follows:
  - P_1 with CPU burst of 5 milliseconds
  - P_2 with CPU burst of 9 milliseconds
  - P_3 with CPU burst of 6 milliseconds
  - P_4 with CPU burst of 3 milliseconds

- Using the SJF algorithm we can schedule the processes as viewed in the following Gantt chart…..

Example



- The wait times for each process are as follows:
  - 3 milliseconds for P_1
  - 14 milliseconds for P_2
  - 8 milliseconds for P_3
  - 0 milliseconds for P_4

- Thus, the average wait time is (3 + 14 + 8 + 0)/4 = 6.25 milliseconds

Exercise

- In the previous example, what would the average wait time be if we had been using the First Come, First Served algorithm?

Answer

- The Gantt chart above shows the wait times using FCFS
- The average wait time under FCFS is:

$$(0 + 5 + 14 + 20)/4 = 9.75 \text{ milliseconds}$$

- Thus, the Shortest Job First algorithm produces a shorter average wait time than FCFS

Shortest Job First

- Advantages:
  - SJF reduces the overall average waiting time
    - Thus SJF is provably optimal in that it gives the minimal average waiting time for a given set of processes
- Disadvantages:
  - Can lead to starvation
  - Difficult to know the burst time of the next process requesting the CPU
    - May be possible to predict, but not guaranteed
    - Unacceptable for use in interactive systems