

Homework 3 Solution

1. (a) Adding resources is always safe
- (b) Removing resources is safe iff the resources are not being used by any of the existing processes and all future needs of the existing running processes can be satisfied without the removed resources. In other words, iff the system remains safe (according to the Banker's algorithm) for the existing processes without the resources that are to be removed.
- (c) The *max* for a process can be increased iff the system will remain safe. That is, we must recompute that there will still be a deadlock-free sequence of resource assignments for the existing processes in the system.
- (d) Decreasing *max* for a process is always safe.
- (e) Increasing the number of processes is safe from the perspective of the Banker's algorithm as long as every one of the added process is safe individually (i.e. it can run on the system in isolation; $Max \leq Total$). The newly admitted processes may, at worst, have to wait until all the current resources are completed before their first resource request are fulfilled, and then be run one-at-a-time.
- (f) Decreasing the number of processes is also safe from the perspective of the banker's algorithm, since there is strictly less opportunity for deadlock.

2. (a) *Need*

ABCD
 0 0 0 0
 0 7 5 0
 1 0 0 2
 0 0 2 0
 0 6 4 2

- (b) Yes. One safe sequence is P_0, P_2, P_3, P_1, P_4 .

- (c) Yes. One safe sequence is P_0, P_2, P_1, P_3, P_4 .

3. (a) (not drawn to scale)

FCFS:

P_1	P_2	P_3	P_4	P_5
-------	-------	-------	-------	-------

SJF:

P_2	P_4	P_3	P_5	P_1
-------	-------	-------	-------	-------

nonpreemptive priority:

P_2	P_5	P_1	P_3	P_4
-------	-------	-------	-------	-------

round-robin:

P_1	P_2	P_3	P_4	P_5	P_1	P_3	P_5	P_1	P_5	P_1	P_5	P_1	P_5	...
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-----

P_1	P_1	P_1	P_1	P_1
-------	-------	-------	-------	-------

- (b) FCFS: 10, 11, 13, 14, 19
 SJF: 19, 1, 4, 2, 9
 nonpreemptive priority: 16, 1, 18, 19, 6
 round-robin: 19, 2, 7, 4, 14
- (c) Since all processes start at the same time, for all scheduling algorithms and processes, we have: waiting time = turnaround time - burst time, giving:
 FCFS: 0, 10, 11, 13, 14
 SJF: 9, 0, 2, 1, 4
 nonpreemptive priority: 6, 0, 16, 18, 1
 round-robin: 9, 1, 5, 3, 9
- (d) SJF has the minimal average waiting time: FCFS 9.6, SJF 3.2, nonpreemptive priority 8.2, round-robin 5.4.
4. (a) The process with two references in the ready queue will receive two time-slices of execution in each round robin iteration.
- (b) This approach is advantageous in that it is an easy way to implement priority without modifying the basic scheduler. Unfortunately, it allows only coarse granularity of priorities, since the total number of references in the queue determines the total number of time slices. Moreover, processes entering and leaving the ready queue alter the proportions of time allotted to processes remaining in the queue. Processes entering or leaving the ready queue need to remove *all* their references from the ready queue, which may require a scan through the queue. Finally, there is no guarantee that time-slices allotted to the same process will be contiguous, thus incurring many unnecessary context switches.
- (c) One could simply insert an integer into the ready queue element that would indicate the number of time slices to allot to the referenced processes. This would mean that processes entering or leaving the queue do not have to insert or remove multiple references to their PCB from the queue. Also, the time-slices allocated to a process can be contiguous, thus saving unnecessary context switches.
5. (a) SJF is a priority scheme with the priority set to the burst length, with lower numbers corresponding to higher priorities.
- (b) A multi-level feedback queue with all processes in the lowest queue would be FCFS.
- (c) FCFS is a priority scheme with the time of arrival (into the ready queue) as priority.
- (d) Unrelated.