CMSC 312 Assignment 2 Theory Questions

Christopher Flippen

Professor Ghosh

2. Prove/disprove that SJF is optimal in terms of average turnaround times of the processes.

SJF is shortest-job-first scheduling. SJF gives minimum average waiting time for a given set of processes since it avoids the convoy effect that FCFS scheduling has. By running the shortest processes first, those processes have close to smallest possible turnaround time, and since the heavier processes are run after the shorter processes, their turnaround time is proportionally not impacted much by running the shorter processes first. Therefore, this process is optimal in terms of average turnaround times.

3. For what types of workloads does SJF deliver the same turnaround time as FIFO?

If the processes are received in shortest job order already, then running them as FIFO or SJF will run them in the same order. SJF and FIFO will also run the workloads in about the same time if all of the processes are approximately the same in terms of running time, since the order wouldn't matter for processes with similar length.

4. For what types of workloads and quantum lengths does SJF deliver the same response times as RR?

If all of the processes are approximately the same running time, giving each of the n process 1/n of the CPU time would make them run as efficiently as SJF.

5. What happens to response time with SJF as job length increases?

Response time with SJF is linearly proportional to job length which means if job length doubles, response time approximately doubles. SJF is also proportional to how many jobs there are since they are ordered in terms of length.

6. What happens to response time with RR as quantum lengths increase? Explain in words and write an equation that gives the worst-case response time, given N jobs.

The maximum wait time for a job in RR is q(N-1) (where q is the quantum length), so the response time with RR is proportional to q(N-1), meaning it increases as q increases.

7. "Preemptive schedulers are always less efficient than non-preemptive schedulers." Explain how this statement can be true if you define efficiency one way and how it is false if you define efficiency another way.

Non-preemptive schedulers are more efficient if you define efficiency as the ratio of CPU time spent on OS services vs time spent running userspace programs, because preemptive schedulers have to do context switches more frequently which uses up a large amount of overhead resources.

Preemptive schedulers are more efficient if you consider efficiency to be response time, since the most important jobs will be done first, the system will ensure those have an efficient response time.

8. What is the priority inversion problem? Does this problem occur if round-robin scheduling is used instead of priority scheduling?

The priority inversion problem is when a higher priority process is preempted by a lower priority process. This can cause errors with the high priority process not being able to obtain its required resources. Round robin scheduling could avoid this issue since the priority inversion problem only occurs when priority is being used and round robin scheduling doesn't use priorities. This definition of the priority inversion problem is from: https://www.tutorialspoint.com/priority-inversion. There are several other variations of the problem on other online sources, but they all deal with processes being preempted by processes of different priorities.

9. Does Peterson's solution to the mutual exclusion problem work when process scheduling is preemptive? How about when it is non-preemptive?

Peterson's solution is a solution for the mutual exclusion problem in which multiple processes try to use the same resource at the same time by using flags as interprocess communication to allow the processes to take turns with entering the critical section of their jobs.
Peterson's solution works with preemption since it uses the concept of stopping one process and waiting for the other one to finish which works cleanly in preemption. The algorithm might not work in a non-preemptive environment since if process 0 is chosen to use the critical section, but process 1 is already running, a kind of deadlock will occur since process 1 will wait forever since process 0 is supposed to run but must wait for process 1.

10. Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst-Time	Priority
P1	6	3
P2	1	1
P3	2	5
P4	3	4
P5	5	2

Processes arrive in order P1,P2,P3,P4,P5

FCFS:

```
        P1            P2            P3            P4            P5

0             6             7             9             12            17
```

SJF:

	P2	P3	P4	P5	P1
0	1	3	6	11	17

Non-preemptive priority:

	P2	P5	P1	P4	P3
0	1	6	12	15	17

RR with Quantum = 1:

```
0—1—2—3—4—5—6—7—8—9—10—11—12—13—14—15—16—17
 P1  P2  P3 P4  P5  P1  P3  P4  P5  P1  P4    P5    P1   P5   P1   P5    P1
```

Turnaround time for each process:

	FCFS	SJF	NPP	RR
P1	6	17	12	17
P2	7	1	1	2
P3	9	3	17	7
P4	12	6	15	11
P5	17	11	6	16
Avg	10.2	7.6	10.2	10.6

Waiting time for each process:

	FCFS	SJF	NPP	RR
P1	0	11	6	11
P2	6	0	0	1
P3	7	1	15	5

P4	9	3	12	8
P5	12	6	1	11
Avg	6.8	4.2	6.8	7.2

Which of the schedules in part a results in the minimal average waiting time?

SJF has the lowest average waiting time with a time of 4.2 ms

11. The aging algorithm with a = 1/2 is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 ms. What is the prediction of the next time?

T(0) = 40

T(1) = 20

T(2) = 40

T(3) = 15

T(4) = at_n+(1-a)at_(n-1)+(1-a)^2at_(n-2)+(1-a)^3t_(n-3)+(1-a)^4T(0)

= 1/2(15)+1/4(40)+1/8(20)+1/16(40)+1/16(40)

= 25 ms predicted time

12. Explain what a multi-level feedback scheduler is and why it approximates SRTF.

A multi-level feedback scheduler is a scheduler that uses two queues: a foreground, used for interactive processes, and a background queue, used for batch processes. Besides the foreground and background queues, the scheduler can have several other queues that correspond to different runtimes and specific process types. A multi-level feedback scheduler assigns priorities and time slices to each process. If a process takes longer than that time slice, it is preempted moved to another, lower priority queue to allow for the faster processes to finish first. Due to this mechanism of moving processes to different queues depending on their properties such as type of running time, a multi-level feedback scheduler does approximate SRTF.

13. Consider a system consisting of processes P1,P2,...,Pn, each of which has a unique priority number. Write the pseudocode of a monitor that allocates 3 identical line printers to these processes, using the priority numbers for deciding the order of allocation.

```
monitor printers
{
        int num_avail = 3;
        int waiting_processes[MAX PROCS];
        int num_waiting;
        condition c;                        //this is a condition variable

        void request_printer(int proc_number)
        {
                //two cases: either a printer is open and we can just allocate it or all printers are
busy, so we have to wait for a printer to open in order to allocate it
                if (num_avail > 0)
                {
                        num_avail--;
                        //give this process the printer
                        return;
                }
                else //add this process into the array of waiting processes
                {
                        waiting_processes[num_waiting] = proc_number;
                        num_waiting++;
                        //sort the list of waiting processes to ensure the highest priority one runs
next
                        sort(waiting_processes);
                }
                //wait for a printer to open
                while(num_avail == 0 && waiting_processes[0] != proc_number)
                {
                    //wait for c to signal that a printer is open
                    c.wait();
                    //update the array of waiting_processes
                    waiting_processes[0] = waiting_processes[num_waiting - 1];
                    num_waiting--;
                    sort(waiting_processes);
                    //give this process a printer
                    num_avail--;
          }
     }

     void release_printer()
     {
          //release the printer
          num_avail++;
          //signal that a new printer is open using our condition variable
          c.broadcast();
     }
}
```