1. **Prove/disprove that SJF is optimal in terms of average turnaround times of the processes. (2 points)**

SJF is an optimal algorithm because the wait time for short processes much more than it increases the wait time for long process.

1. **For what types of workloads does SJF deliver the same turnaround times as FIFO? (2 points)**

SJF has the best turnaround time in the most cases compared to FIFO and RR. The only time SJF has the same turnaround time as FIFO is when all the job has the same number of burst times. Also, when process P1 come in at 1 with 1 burst time, P2 comes in at 2 with 2 Burt time then the average turnaround time for FIFO and SJF will be the same.

1. **For what types of workloads and quantum lengths does SJF deliver the same response times as RR? (2 points)**

SJF will have same response times as RR when the quantum length of RR is larger than the largest burst time to be serviced and when the process arrives in order of increasing size.

1. **What happens to response time with SJF as job lengths increase? (2 points)**

In SJF if the job length increases, then the response time with SJF will increase too. The larger the increase, the greater the average response time will be for SJF.

1. **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. (2 points)**

When quantum lengths increase in RR, then response time increases because each process gets a small unit of CPU time and will be taken out of CPU then added to the end of the ready queue once the time has elapsed. Increasing the quantum length will just increase the waiting time for each process to be preempted and added. If there are n processes in the ready queue and the quantum length is q, then each process gets 1/n of the CPU time. A process has to wait at worst q(n-1) meaning the longer the quantum length, the longer the process waits.

1. **"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. (2 points)**

Preemptive schedulers are usually more efficient than non-preemptive schedulers because in non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated. In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. In terms of efficiency, a preemptive scheduler would be considered less efficient than a non-preemptive scheduler because it has a higher overhead. It must switch out processes and change their states much more often. In the other hand, a preemptive scheduler will always have a faster response time compared to a non-preemptive scheduler that has to wait till one process completes before starting another while a preemptive scheduler can just switch to the processes necessary.

1. **What is the priority inversion problem? Does this problem occur if round-robin scheduling is used instead of priority scheduling? Discuss. (2 points)**

Priority inversion problem will not occur in Round-Robin. Due to the fact that RR schedules processes in a way that one process can only use a certain amount of time before it is forced to pop out and wait for the next process.

1. **Does Peterson's solution to the mutual exclusion problem work when process scheduling is preemptive? How about when it is non-preemptive? (2 points)**

Peterson’s solution was designed to work when the scheduling is preemptive. When the scheduling is non-preemptive, it may not work. It will loop through forever and never gets release.

1. **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**

**The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.**

1. **Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.**

A screenshot of a cell phone

Description automatically generated

1. **What is the turnaround time of each process for each of the scheduling algorithms in part (a)?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PROCESS | FIFO | SJF | NONPREEMPTIVE | 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 |
| TOTAL | 51 | 38 | 51 | 53 |

1. **What is the waiting time of each process for each of the scheduling algorithm s in part (a)?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PROCESS | FIFO | SJF | NONPREEMPTIVE | 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 |
| TOTAL | 34 | 21 | 34 | 36 |
| AVERAGE | 6.8 | 4.2 | 6.8 | 7.2 |

1. **Which of the schedules in part a result in the minimal average waiting time (over all processes)? (5 points)**

Shortest Job First has the minimal average waiting time.

1. **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 msec. What is the prediction of the next time?**

= (((40 + 20) / 2 + 40) / 2 + 15) / 2

= ((30 + 40) / 2 + 15) /2

= (35 + 15) / 2

= 25

1. **12) Explain what a multi-level feedback scheduler is and why it approximates SRTF.  
   True or False (also give an explanation for your choice): If a user knows the length of a CPU time-slice and can determine precisely how long his process has been running, then he can cheat a multi-level feedback scheduler. (2 points)**

Multilevel feedback queue-scheduling algorithm allows a process to move between queues. It uses many ready queues and associate a different priority with each queue. I think it is **True** that if a user knows the length of a CPU time-slice and can determine precisely how long his process has been running, then he can cheat a multi-level feedback scheduler. User can just execute an I/O request that would allow for the process to get more time that it is supposed to.

1. **Consider a system consisting of processes *P*1, *P*2, ..., *Pn*, each of which has a unique priority number. Write the pseudo-code of a monitor that allocates three identical line printers to these processes, using the priority numbers for deciding the order of allocation. (5 points) Start with the following and populate the two functions request\_printer() and release\_printer():**

**monitor printers *{*int num\_avail = 3;  
int waiting\_processes[MAX PROCS]; int num\_waiting;  
condition c;**

**void request\_printer(int proc\_number) *{***

***}***

**void release\_printer() *{***

***} }***

Solution:

monitor printers *{*

int num\_avail = 3;  
int waiting\_processes[MAX PROCS];

int num\_waiting;  
condition c;

void request\_printer(int proc\_number) *{*

*if (num\_avail less than o) {*

*num\_avail - -;*

*return;*

*}*

waiting\_processes[num waiting] = proc\_number

num\_waiting ++;

while(num\_avail == 0){

condition c.wait();

wait process[0] = wait processes[num waiting – 1]

num waiting--;

sort waiting processes;

num available --;

}

*}*

void release\_printer() *{*

*num avail ++;*

*condition c.broadcast();*

*} }*