OS THEORY ASSIGNMENT – 3

Name-Nandini Yadav

Reg No-2020CA045

Q.1. Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

(a) **Round robin.**

(b) **Priority scheduling.**

(c) **First-come, first-served (run in order 10, 6, 2, 4, 8).**

(d) **Shortest job first.**

For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d) assume that only one job at a time runs, until it finishes. All jobs are completely

CPU bound.

answer-

Remember that the turnaround time is the amount of time that elapses between the job arriving and the job completing. Since we assume that all jobs arrive at time 0, the turnaround time will simply be the time that they complete.

(a) Round Robin: The table below gives a break down of which jobs will be processed during each time quantum. A * indicates that the job completes during that quantum

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

A B C D E A B C* D E A B D E A

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

B D* E A B E A B* E A E A E* A A A*

Average turnaround = (8 + 17 + 23 + 28 + 31)/5 = 107/5 = 21.4 minutes

(b) Priority Scheduling:

1-6 7-14 15-25 26-27 28-31

B E A C D

Avg. turnaround = (6 + 14 + 25 + 27 + 31)/5 = 103/5 = 20.6 minutes

(c) FCFS

1-11 12-17 18-19 20-23 24-31

A B C D E

Avg. turnaround = (11 + 17 + 19 + 23 + 31)/5 = 101/5 = 20.2 minutes

(d) SJF

1-2 3-6 7-12 13-20 21-31

C D B E A

Avg. turnaround = (2 + 6 + 12 + 20 + 31)/5 = 71/5 = 14.2 minutes

Finding the average process turnaround time can also be done by multiplying the number of unfinished processes by the time they remain unfinished dividing by number of processes. This makes finding solutions for parts b, c, and d particularly efficient. For example, the calculation for part C can be represented by (5*11 + 4*6 + 3*2 + 2*4 + 1*8)/5.

Grading guide

Each part of the four parts of question 3 were worth a total of 2.5 points. More than 2.5 points could not be lost for any part. They were graded as follows:

-1 Intermediate mistake in work such as incorrect process turnaround time or diagram, but correct final answer.

-2 Intermediate mistake in work, but correct final answer given that mistake. For example using process turnaround time of 15 minutes when it should be 25 minutes.

-1 Mistake in arithmetic.

-1 Using wrong time quantum in part a.

-1 Reversing priority in part b.

-1.25 Calculating wait time instead of turnaround time.

-2 Not using given values. For example making A take 12 minutes to complete instead of 11. (Points only taken off once for all of question 3)

-1 Determining order of round-robin by priority. As described on page 194 [Silberschatz], Round-Robin scheduling is the same as first come first serve, except with preemption. The assumption is that the processes are delivered in A,B,C,D,E order, and if a different assumption is made then they should be round-robin scheduled in the same order as the FCFS method.

-2.5 Incorrect final answer and incorrect or missing work.

Q.2. Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/Obound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 milliseconds and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when: (i) The time quantum is 1 millisecond

ans- (a) The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.

(ii) The time quantum is 10 milliseconds ans-(b)

The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1

+ 10.1 (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

Q.3. Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the SJF algorithm?

(i) α = 0 and τo = 100 milliseconds

(ii) α= 0.99 and τo= 10 milliseconds

ans-

a) Tn+1=α* tn+ (1-α) * τn

When α = 0 and τ0 = 100 milliseconds,

the equation will be:Tn+1= 0* tn+ (1- 0) * 100 = 100 milliseconds So, in this case the formula always makes a prediction of (100 milliseconds) forthe next CPU burst.

b)Tn+1=α* tn+ (1-α) * τn.

When α = 0.99 and τ0 = 10 milliseconds, the equation will be

:Tn+1= 0.99* tn+ (1- 0.99) * 10 = [0.99 tn+ 0.1] milliseconds So, in this case the most recent behavior (tn) of the process is given much higher weight (99%with comparison to 1%) than the past history associated with the process (τn=10).

Q.4. Consider a system implementing multilevel queue scheduling. What strategy can a computer use employ to maximize the amount of CPU time allocated to the user's process?

ans- The program could maximize the CPU time allocated to it by not fully utilizing its time quantums. It could use a large fraction of its assigned quantum, but relinquish the CPU before the end of the quantum, thereby increasing the priority associated with the process. For round-robin, it's okay to answer that there is no strategy exist. Or, another answer would be to break a process up into many smaller processes and dispatch each to be started and scheduled by the operating system.

Q.5. Consider a load-balancing algorithm that ensures that each queue has approximately the same number of threads, independent of priority. How effectively a priority-based scheduling algorithm

handle this situation if one run queue had all high-priority threads and a second queue had all low-priority threads?

Answer : Consider giving greater priority to both the queue contains the national priority comment section as well as, therefore, first method the thread throughout this queue. If the item from either the major priority queue becomes decoupled, if the balancing between some of the number of comments in some of these two queues becomes disrupted, instead decouple one thread from either the queue comprising the low-value thread as well as position that one in the queue with the highest - priority loop to match the number of connections in such 2 queues. Whenever a new thread arrives with some kind of proper description, a friendly interface could've been introduced to just the queue contains fewer threads to achieve stability. Therefore, the load-balancing requirements for keeping about the same number of threads would be retained and the meaningful matter of top priority thread would also be retained.