CS 3413

Assignment 2

Due Date: January 20th, 2020 at 12:30 pm

ASSIGNMENT IS TO BE COMPLETED INDIVIDUALLY BY ALL STUDENTS!

Your solution is to be written in C and submitted via D2L.

Let's improve our scheduling from assignment 1 to be more realistic! First, CPUs today have multiple processing cores. Dual-core has 2 processors and Quad-core has 4 processors. Modify your program so that the output from Assignment #1 now reflects *n* processors operating at once. To make your program flexible, provide *n* as a command line argument to the program. The input format is the same, but the output is different. The output header now has a column for each CPU. At any given time you have to list what processes are operating on each CPU (tab delimited). For example with 3 CPUs:

Time	CPU1	CPU2	CPU3
2	A	B	C
3	A	B	-
…			

This example shows jobs A, B and C running on CPUs 1, 2 and 3 respectively at time 2. At time 3, A and B continue to run on CPUs 1 and 2 respectively, but CPU 3 is now idle. Note: Be careful! You cannot start a job until it has arrived. Also, CPUs are symmetric – meaning it does not matter which job is on which CPU at a given time! It is correct if at a given time you are executing the right "set" of jobs.

Second, let's improve the scheduling algorithm ☺ Computers today are being used in increasingly important applications where they need to act within a certain time frame in order to avoid an unpleasant outcome. A simple scheduling algorithm to achieve this is known as Priority. In Priority scheduling, each job gives a rank that indicates how important it is for that job to run. When considering what job to complete, you pick the job that has the highest rank (priority). To break ties, you pick the shortest job first. If you still have a tie then you pick the job that arrived first. For example, consider the following:

If we have 2 CPUs:

User	Process	Arrival	Duration	Priority
Jim	A	2	19	3
Mary	B	2	21	3
Sue	C	5	1	1

At time 2, A and B will start to run on CPU1 and CPU2. When job C arrives at time 5 it will get a CPU right away as it has the highest priority along with A (since it has the same priority as B, but wins the tie breaker because it is a shorter duration). After C is completed, then B gets a CPU again.

Modify your C program from question #1 that will read in job requests and print out the corresponding job schedule according to the algorithm above. The output format remains as two tables. First table is the running time and the job(s) currently executing (tab separated). The second table is a summary with the user name (in the order in which jobs arrive) and the time when their **last** job is completed.