COMPSCI 3SH3, Winter 2021
Student Name: Jatin Chowdhary
Mac ID: Chowdhaj
Student #: 400033011
Date: April 2nd, 2021

Lab Assignment 5 – Scheduling Algorithms

Q1 – Shortest Job First (SJF) Scheduler

Module Description

The *schedule_sjf* program is an implementation of the "Shortest Job First" scheduling algorithm. This scheduling algorithm executes the task/process with the shortest CPU burst time first. Then, the task/process with the next smallest burst time is executed, and so on. The *schedule_sjf.c* program has three functions in it; *add()*, *schedule()*, and *pickNextTask()*. The *add()* function is quite trivial and adds a process/task to the queue, of tasks/processes. Note, this queue is actually a linked list. The *schedule()* function is also trivial; its job is to call the *pickNextTask()* function, run the task, and then delete it from the linked list. The *pickNextTask()* function is where most of the work happens. This function is responsible for finding the task/process with the shortest burst time, and returning it to the *schedule()* function. It finds the correct task/process by iterating through the linked list that contains the tasks/processes, and checks each task's CPU burst time. If it finds a task with a smaller CPU burst time, then it saves the reference to the task in a local variable. Once the end of the linked list has been reached, the local variable that stores the reference to the task with the smallest CPU burst time is returned to the calling function, *schedule()*.

Note: This is a semi-high level description of the *schedule_sjf* program. For more information (i.e. Data types, control flow, logic, etc.) please refer to the source file, *schedule_sjf.c*, and use the *Makefile* to compile it.

***Source Code:** *schedule_sjf.c*

Q2 – Priority With Round-Robin Scheduler

Module Description

The *schedule_priority_rr* program is an implementation of the "Priority Based Round-Robin" scheduling algorithm. This scheduling algorithm executes tasks/processes using Round Robin (RR) but based on priority; tasks with a higher priority are given precedence, and are executed first. Tasks with a higher priority are executed before tasks with a lower priority. Note, a low priority number means that the task has a high priority, and a big priority number means that the task has a low priority. The *schedule_priority_rr* program has three functions in it: *add()*, *schedule()*, and *pickNextTask()*. The *add()* function is quite trivial and adds a process/task to the queue, of tasks/processes. Note, this queue is actually a linked list. The *schedule()* function is also trivial; its job is to iterate through the linked list, that contains the tasks, until the list is empty, and upon each iteration, it calls the *pickNextTask()* function to decide which function should be executed based on priority. Once the task has been selected, if its burst time is greater than the time quantum, *q,* then *q* is subtracted from the task's burst time, and a counter is incremented. If the burst time is less than *q*, then the burst time is decreased to 0, and the entire task is removed from the linked list. The *pickNextTask()* function is where most of the work happens. This function is responsible for finding the task/process with the highest priority that wasn't previously executed, and returning it to the *schedule()* function. It does this by iterating through the linked list and finds a task/process with a priority that is identical to the previously executed task. Each time it iterates through the linked list, it continues from where it left off. For instance, if the function chooses a task at position 2, then the next time it runs, it will start at position 3. When the function iterates through the linked list, it checks to see if there is a task/process with an identical priority to the previous task. If it does not find one, the index counter is reset and the function finds the next task or set of tasks with a high priority and returns them to the *schedule()* function. This process continues until all tasks/processes have been executed in a "Priority Based Round-Robin" manner.

Note_1: This is a semi-high level description of the *schedule_priority_rr* program. For more information (i.e. Data types, control flow, logic, etc.) please refer to the source file, *schedule_priority_rr.c*, and use the *Makefile* to compile it.

Note_2: The time complexity for this specific implementation is relatively large. Thus it would be better if the list of task/processes was sorted before the scheduling algorithm is executed.

***Source Code:** schedule_priority_rr.c*

**Source Code is attached in a zip file and as individual files*