CS 4328.001

Operating Systems

Fall 2021

Dylan W. Ray(dwr48), Brett M. Owen(bmo23)

Programming Assignment #2: Simulation of CPU Scheduling Algorithms

Due: *Thursday, November 11 @ 11:55 PM*

Division of Work:

Dylan Ray: FCFS algorithm, SRTF algorithm, Metrics for SRTF

Brett Owen: RR algorithm, Metric Implementation, Program Report

Design and Implementation:

For our program, each simulator utilizes a linked list of events. Each algorithm starts with the first 10,000 arrival times, added in sequential order, with a queue of processes. Then, each algorithm enters a while loop that processes an event, calling a handler function for departure, arrival, or preemption, depending on the algorithm. When this loop is finished, the metrics (which are kept track of throughout the different event handlers) are calculated and the global variables are reset.

For FCFS, the arrival handler leads directly to a departure handler, which takes the first process in the process queue and calculates departure time. Any arrival times that happen are accepted and added in the proper place in the event queue by finding the place where the event time is smaller than the next one but greater than the event before it. Arrival events are not processed by the CPU until the process before it is finished. Once the arrival event is complete, the departure event starts, adding the burst time and the current clock time to calculate the departure time.

In SRTF, the implementation is intended to work thusly. Like in our implementations of RR and FCFS, we pre-load the events linked-list with randomly generated arrival events for 10,000 processes. An extra 2,500 extra arrival events are added to the events linked-list in order to account for processes that might arrive between the arrival of the 10,000th process and the 10,000th processes departure. The program then traverses the linked-list looking for the next event with a higher time then the current clock time. This found event will be the next event handled. When this event is an arrival the arrivalHandler() determines if either a departure event or preemption event needs to be added into the linked-list at a future time index. A third option is neither those events are chosen and the readyQueueSize variable in incremented, indicating a new process is waiting to be serviced. These choices are determined by the current state of the CPU, mainly if the CPU is idle, or if the arriving process will finish sooner then the currently running process. Now that a preemption (Note: importantly, if a preempt event is created a special function is called to roll back the next departure event in the linked list, because it is now incorrect) or departure event is added to the linked list let us examine what should occur when either of those events are grabbed next from the events list. If a departure event is found next in the linked list the departureHandler() will first remove the top process from the process priority queue, removing it from the simulation, and increments the departure counter while decrementing the readyQueueSize variable. It will then check to see if the readyQueueSize is 0 or not. If so, the CPU state changes to idle, and the function terminate. If the readyQueueSize is not 0 a new departure event is created using the process now sitting at the top of the process priority queue (note the queue is sorted by least remaining burst time left). This new event is then placed at a future point in the linked list. If the next event gotten from the linked list is a preemption event, these steps will be taken. The current running process is taken out of the CPU, its remaining burst time is updated, its corresponding member in the priority queue is popped, and the new updated process is added to the queue. While this occurs a new departure event is created using the state of the CPU and the incoming new process. At the end of the simulation all metrics are calculated using variables which tracked during execution

In RR, the algorithm works similarly to FCFS, but includes preemption based on the amount of work needed to complete the burst. If the burst time is less than the quanta, the arrival event adds a departure event. If the quantum is greater than the burst time, the arrival event adds a preempt event, subtracting the quantum from the remaining burst time. The preempt event does the same thing as arrival time, but takes the process in the top of the queue and adds it to the end of the queue. The departure time acts as it does in FCFS. Be warned, this algorithm takes a long time to run.

Compilation/Runtime Instructions:

All python files must be in the same directory. To run this program, type python3 program2.py, passing in the command line arguments. These arguments are 1) algorithm, 2) avg burst time, and 3), quantum. The processes per second is handled by program2.py, which calls one of the three algorithms main function and runs it 30 times, passing the number of the run as lambda (the processes per second). Each function returns a tuple of the four metrics, and the main function compiles these into an array of 30 values per each different metric, which it then prints out. We used these arrays to graph the next section.

Execution Data and Interpretation: