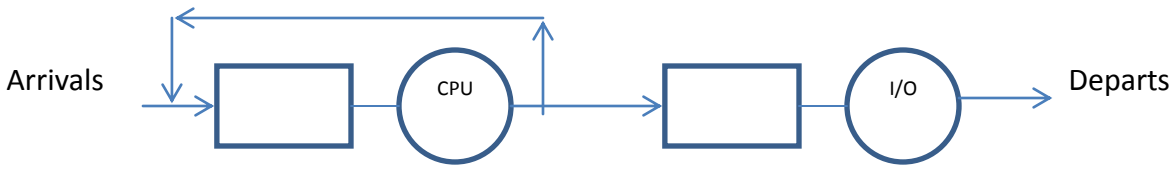
Program 3 {September 22, 2014} 42 Points

Event-Driven Simulation of a Simple Computer System Model

Due: Tuesday, September 23, 2014 at 11:59 p.m.

This assignment focuses on the use of data structures built in C with **structures.** One specific goal of this assignment is to give the students practice with C pointers.

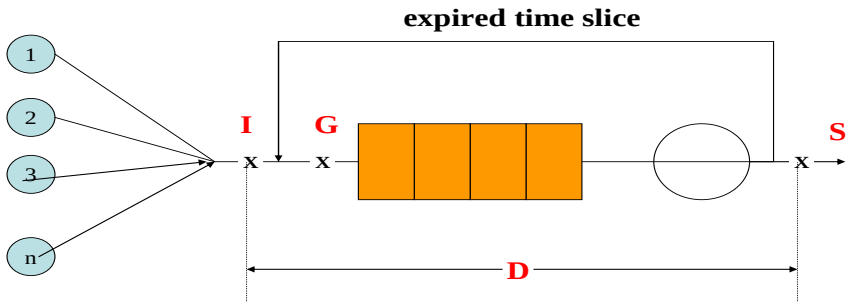
Additionally, completing this assignment requires understanding the basics of event-driven simulation which is used prominently in the performance analysis of computer systems and computer networks. The primary objective of this assignment is to use event-driven simulation together with a **FCFS (First Come First Served) scheduler** and a **RR (Round Robin) scheduler.** The simulation diagram for a Simple Computing Model of a computer below consists of **n** processes arriving at the CPU scheduler for service. The CPU server uses a RR scheduler. Once a job completes its total time required at the CPU, it goes to the FCFS I/O server.



Simple Computing Model

In **RR scheduling**, arriving customers are enqueued at the back of the scheduler queue. The process at the front of the queueing system gets up to one time slice of CPU service. If the process does not finish within its time slice, as indicated in the figure, the time slice expires and the process returns to the back of the scheduler queue to await its turn for another time slice.

Round Robin Queue



Assignment Inputs

Your program begins by reading in two command line arguments source and time-slice to indicate respectively, the number of source processes to simulate and the time slice to be used in the **RR scheduler**. For this simulator all time will be represented in 100 milliseconds units. Thus, a command line time-slice of 2 indicates a 200 millisecond time slice. Moreover, 100 milliseconds is the smallest granularity of time within the simulator.

The simulator program reads source lines of input from an ASCII script file where each line of input contains:

process-id arrival_time cpu_time io_time

where arrival_time, cpu_time and io_time are in simulation units (i.e. 100 millisecond units)

For example, the input line:

2166 30 24 65

indicates that process **2166** arrives at the scheduler at simulated time **3** seconds needing **2.4** seconds of CPU service and **6.5** seconds of I/O time (Note – I have converted milliseconds to seconds here!).

Main Assignment

In this assignment, you are REQUIRED is to use an **event list** mechanism to simulate the performance of the simple computing model. (Note – for Program 5 you will need an implementation of both RR and FCFS, but you can drop the event list mechanism in Program 5, if your team prefers an alternate simulator design.) Your final output consists of **two log files** that annotate the simulation of the computer system and track customers. You need to run your simulator twice (once for each of two different RR time slices). Each output file logs for each process simulated: its arrival time to the simulation, its arrival time at the CPU scheduler, its arrival time at the I/O device and its completion time. The log files output events in chronological order.

The Event List

Event-driven simulation is controlled by a linked list known as the **event list**. Future events are simulated by inserting them into the event list in **chronological order** with the next event in simulated time at the front of the list. In this assignment, the event list will hold three event types: process arrivals, completion time of the process currently running on the CPU and the completion time of the process currently running on the I/O device. The simulation runs in a continuous loop processing the next event at the front of the event list until the event list becomes empty.

Processing an arrival event involves taking the entry off the event list and sending the process to the CPU scheduler queue. Processing a CPU completion event involves first determining whether the process still requires more CPU time. If the process has completed, process statistics are calculated and recorded before the process moves to the I/O scheduler queue. If the process still needs more CPU time, the process is enqueued at the back of the CPU queue. In either case, taking the completion event off the event list triggers a call to the processor scheduler. If there is a process at the front of the scheduler, it is taken off the front of the scheduler queue and a new event with the appropriate completion time is inserted into the event list. If the CPU queue is empty when the current time slice completes, this process can get another time slice immediately and a new completion event is placed on the event list. Similarly for a process currently receiving I/O service, when the completion event for I/O comes off the event list, this triggers a call to the I/O scheduler.

With a **discrete** event driven simulator, it is common to have two or events placed on the event list with the EXACT same event time. To keep all student output consistent and facilitate grading, you MUST use process id as the 'tie breaker' for insertion in the event list such that in the case of identical event times, the process with the LOWER process ID number gets inserted BEFORE the process with the higher process ID number in the event list.

The FCFS and the RR Scheduler

The **FCFS** scheduler is implemented as a queue data structure with arriving processes enqueued at the back of the I/O queue and the next process to receive I/O service taken off the front of the FCFS queue and added to the event list.

The **RR** scheduler is also implemented as a queue with arriving processes enqueued at the back of the queue and the next process to receive service taken off the front of the RR queue and inserted in the event list. The difference for the RR scheduler is a process only receives a maximum of one time slice of service each time it receives CPU service. If after a time slice, the process still needs more service the processor sends this process to the back of the queue.

For both the **FCFS** and the **RR** scheduler, when a process arrives at a specific device (CPU or I/O), the device scheduler needs to check if another process is currently running. If the device is idle, the arriving process is sent to the event list with the appropriate completion time. If the device is currently busy, the process gets enqueued at the back of the scheduler queue.

Performance Metrics

For each process in the simulation, you need to compute the total time spent waiting (in the queue) at both devices and the overall time spent by the process to traverse the system. You need to compute the average process time enqueued at each device and the overall average time spent by processes in the system.

What to turn in for Program 3 (prog3)

An official test file will be made available a few days before the due date. Assume the simulation will simulate a maximum of 20 processes going through the simple compute system. Your assignments involves running the same test data using two distinct time slices (200 milliseconds and 500 milliseconds) and using the simulator to analyze which time slice works better for the test data. Turn in your assignment using the ***turnin*** program on the CCC machines. You should turn in a tarred file that includes your source code, **your two output log files (one for each time slice run),** a make file and a README file. The README file should provide any information to help the TA or SA test your program for grading purposes and provide a clear statement about the parts of the simulator that were not yet working when the tar file was submitted to ***turnin.***