CS2303 Systems Programming Concepts A08

Lab 3 4 points

Command Line Arguments and Event List Operations

Both programs 3 and 5 involve event-driven simulation. While there other ways to run event-driven simulation, for program 3 you must use an event list. The ***key*** data structure in event-driven simulation is a linked list known as an event list. This lab is intended an introduction to event list operations, working with structs and pointers and how to input parameters to a C program using Linux/Unix command line arguments.

Your lab3 program begins by reading in one command line argument processes. Thus, the form is:

```
./lab3 processes
```

For example,

```
./lab3 10
```

Assume lab3 reads in processes lines of input from a script file where each line of input takes the form:

```
process-id  arrival-time  cpu-time
```

where arrival-time and cpu-time are in simulation units (i.e. 100 milliseconds).

For example, the input line:

```
20166  30  205
```

indicates that process 20166 arrives at the scheduler at simulated time 3 seconds needing 20.5 seconds of CPU service. For each line of input, your program needs to create a structure.

For lab3, you are to create at least three functions to operate on an event list: insert, next-event, and print-list.

insert (node, event-list)

Assume an event list built as a linked list where events are stored as structures in ***chronological order*** (based on process arrival time) such that the pointer to the event list points to the event that is the first event to occur in simulated time and the last node on the linked list chain (before the Null pointer) points to the event that will occur the farthest into the future of all the events on the event list.

The *insert* function receives a pointer to the node structure created in conjunction with an input line and a pointer to the event list and inserts this node into the event list in the proper place based on the node's arrival-time. If two sources are scheduled to arrive at the same time, use process-id as the tie breaker such that the event with the smaller process-id comes off the event list first.

node = next-event (event-list)

The *next-event* function uses a pointer to the event list to return a pointer to the next event to occur in simulated time (i.e., the first node in the event list). *next-event* removes this event from the event list.

print-list (event-list)

The *print-list* function uses a pointer to the event list to print out all the events on the event list in chronological order.

Depending on your program design, you may also want to write two other functions, *create-node* and *print-node*. *print-node* is a print function to facilitate debugging while working with structs and pointers.

Lab 3 main program

The lab3 program reads in processes input source lines. For each line of input, the program creates an event list node and inserts this node on the event list. Once the event list is completely built, it is printed out. The main program then takes the first event off the event list, prints out its contents, takes the second event off the event list, prints out its content and finally prints out the current remaining event list.

Lab 3 Assignment

0. Prior to coming to the lab prepare a preliminary solution to the program above.
1. Create a make file.
2. Test the program under your own test data input from the terminal.
3. Run the program on the provided test data file 'lab3.dat' redirecting the output to eventlist.out.
4. Create a README file that contains any useful information to assist in the grading of your lab program.
5. Create a tarred file that contains all the source and header files, the make file and the README file and your output file.
6. Use the Unix version of the 'turnin' to turn-in the tarred file. [The deadline for all lab turn-ins is 24 hours after the beginning of your assigned lab.]