Cleveland State University
Computer and Information Science

System Programming
CIS 340, Fall 2018

Term Project Description

The course will have a term project. You may work individually or in a team of up to two people. All members of a team may receive the same grade if they contribute to the project equally. You will be required to demo your project in a class and submit your source code with a document explaining different part of your code to the blackboard. Two mini projects should be completed as below.

1 CPU Scheduler

In modern computer systems, it is common for users to have many programs running at one time. One of the most basic functions of an operating system is to schedule these processes. For this project, we will assume a very simple round-robin scheduler. This simply means that each process gets to run for a set amount of time (e.g., 1 seconds time slot). If the process does not finish, it goes to the end of the line and the next process gets to run. This policy is very fair and very easy to implement. The best way to implement this policy is through the use of a single-linked circular list. A node structure is shown in Figure 1.

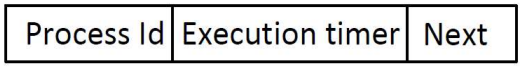

Figure 1: A process node struct.

A circular linked list should be firstly created from the input of text file. For instance, a linked list of the available processes are created by reading the *test1.txt* file with contents of P1:(1,3), P2:(2,1), P3:(3,2). The Figure 2 shows the circular linked list created from the available processes.

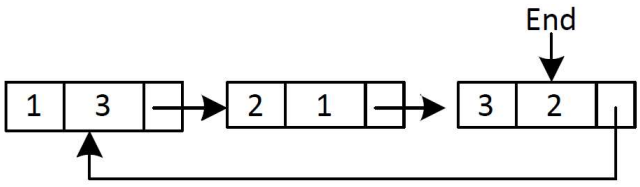

Figure 2: A new process is added to the end of the linked list.

Any new process is added to the "back" of the list (the spot shown by the pointer *End*). The program needs to define another pointer as *Cur* and scan the list every 1 second. The scan time can be triggered every 1 seconds using *sleep* function. At each scan time, it should decreases the execution timer data field and remove it from the linked list if it becomes zero. Figure 3 shows the processes list after the first scan.

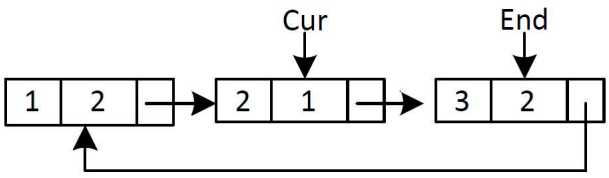

Figure 3: The list of active processes after the first scan using another pointer (e.g., *Cur*).

Figure 4 shows the linked list after the second scan. The second with execution time of one second is removed from the linked list and *Cur* moves one step forward pointing to the last node.

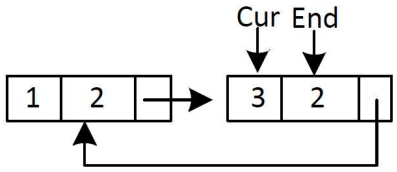

Figure 4: The list of active processes after the second scan.

Figure 5 shows the linked list after the third scan. The *Cur* moves one step forward pointing to the first node.

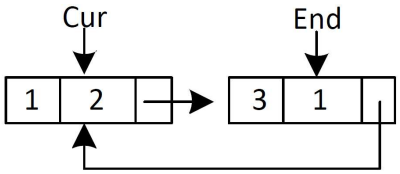

Figure 5: The list of active processes after the third scan.

The program should receive a test file (e.g., *test1.txt*) as an input argument. Each line of *test1.txt* file consists of two fields. The first is the *id* of the process, and the second is the time second the process needs to use the CPU to finish its execution. We assume the time slot of 1 seconds and the process should be removed when its execution is completed by the round robin fashion.
You will need to create a *node* struct and create circular linked list of them in this project. The *node* struct is used to represent a single node in your list. Its design and implementation are entirely up to you. It will only be referenced by the methods in your main method. It should contain at least the mentioned fields (and provide a means of accessing these fields). The following is a brief description of all the methods you must implement:

- node* CreateList(): Create a linked list from the data of the test1.txt file. The method returns the *End* pointer to the node.
- void Add(int ProcessID, int Exetime): Places a new Process at the end of the list. The end of the list is defined by the *End* pointer.
- node* ScanList(node* End): It scans the linked list every one second in order to find the process that needs to be removed from the list. It receives the *End* pointer to obtain the address of the first node and initiate the scanning process. It removes the address of the node that should be removed by making it null.
- node* RemoveCurrent((node*Current): Removes the process using the *Current* pointer when the process execution time is done. It can be used by *ScanList* that have found the node that

should be deleted and returns a reference to the recently removed node. If the list is empty, this should return null. If there is only one item in the list, it should be removed and current should refer to null.

- bool isEmpty(): It returns true if the list is empty and False otherwise.
- void PrintList(): Goes through every node in the list, retrieves the data stored in the node, and print the contents. If the list is empty, nothing should get printed.

2 File Management System

Write a basic file managment system with following capabilities. Suppose the name of the program is myFS. When it executes as below:

```
$ myFS ls -l /home
```

Figure 6: First command format

The program should provides the file listing of the given directory. The program should list the filename, file size, and time last modified for each file within the directory. The list is shown below:

```
-rwxr-xr-x  1 mamal None              62645 Jan 19  2017 a.exe
drwxr-xr-x+ 1 mamal None                  0 Jan 15  2017 bin
-rwxr-xr-x  1 mamal None                171 Feb 27  2017 bingo.c
-rwxr-xr-x  1 mamal None              62848 Feb 27  2017 bingo.exe
-rwxr-xr-x  1 mamal None                187 Feb 28  2017 buffer.c
-rwxr-xr-x  1 mamal None              62657 Feb 28  2017 buffer.exe
drwxr-xr-x+ 1 mamal None                  0 Jan 15  2017 build-gcc
-rwxr-xr-x  1 mamal None                110 Jan 19  2017 crash1.c
-rwxr-xr-x  1 mamal None              62645 Jan 24  2017 crash1.exe
-rwxr-xr-x  1 mamal None                891 Jan 24  2017 crash1.exe.stackdump
-rwxr-xr-x  1 mamal None                131 Jan 19  2017 crash2.c
-rwxr-xr-x  1 mamal None              63157 Jan 24  2017 crash2.exe
-rwxr-xr-x  1 mamal None                885 Jan 24  2017 crash2.exe.stackdump
```

Figure 7: Files list with their properties.

The program should provide the user with the option to sort the listing based on filename, on file size, or on time last modified. The program should run once and quit (it does not loop). The syntax (usage) for the program should be where -s indicates sort by file size and -t indicates sort by time last modified. For example:

```
$ myFS ls -l -t /home
```

Figure 8: Files list should be sorted by last time modified.

These flags are optional but mutually exclusive. The directory is also optional; if it is omitted, then a default directory of "." (the current directory) should be assumed. Every possible combination of command line arguments from the above syntax should be tested. The output should be displayed with fixed-width columns for the size and date. Column orders should be size, then date, then filename. The time last modified should be printed as a 12-character string (e.g., "Oct 10 11:37"). This can be done by manipulating the returned string from the ctime() function and printing only a portion of it.
The file system program should be able to makes a copy of the source file to the destination address. The syntax command could be as below:

```
$ myFS cp bing.c /home/bingocp.c
```

Figure 9: The source file (e.g., bingo1.c) is copied to destination(e.g., /home/bingocp.c).

The command (e.g., See Figure 9) copies bing1.c to the /home/bingocp.c.

3 Evaluation

You should document your implementation and provide a report explaining the functionality of each method with appropriate snapshots from your program inputs and outputs. It is possible that your implementation will be evaluated for documentation and design, as well as for correctness of results. The source code and report should be zipped with your family name of you and partner and submitted to blackboard by due date. One submission of either you or your partner will be sufficient.

4 Project Schedule

Source code and report submission to blackboard:	12/5/2018, 13:30
Project presentation:	12/5/2018, 13:30