System Programming

Final Project

May 03, 2018

Krishaun Bartlett

Ryan Reynolds

PART 1:

The round robin cpu scheduler was created using a circular linked list. The implementation of each method can be seen by the print outputs to the terminal in the output section for part 1.

Main() calls two methods CreateList() and ScanList() that perform all of the functionality of the cpu scheduler.

The CreateList() method opens the source file “test2.txt” that contains the initial list of processes. The entire file is sent to a buffer. Then each line is stored within the subs[][] array. The process ID and run time are extracted using their respective index in the char array. They are converted from ASCII to decimal by subtracting the ‘0’ char. The function then calls the add() function to add each node to the end of the linked list. After all the processes have been added the link tail→next=head is created to make the linked list circular.

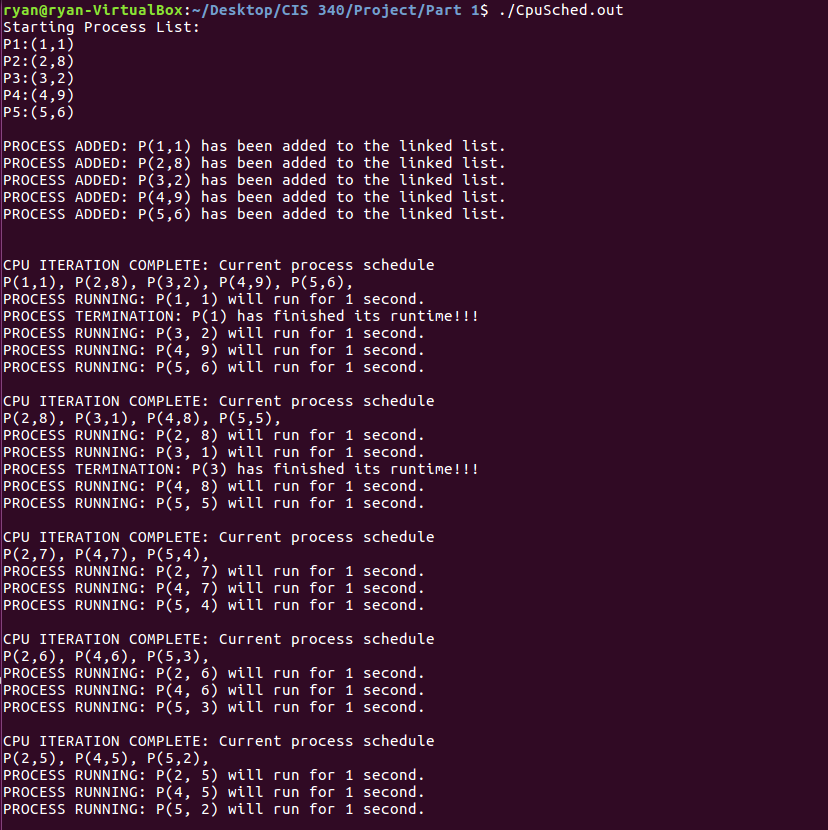
The ScanList() method simulates the round robin scheduler. It uses a node iterator temp to iterate through the linked list until there are no processes remaining, by checking the isEmpty() method each iteration. The process run time is decremented by one and the sleep(1) command is called to simulate the process running for 1 second. After decrementing the run time, if the run time is equal to 0 the RemoveCurrent() method is called to remove the current node from the linked list. The temp2 node is created as a potential improvement for the scheduler. RemoveCurrent returns a node the process ID of the node could be stored in a static array to keep track of the order in which the processes terminated.

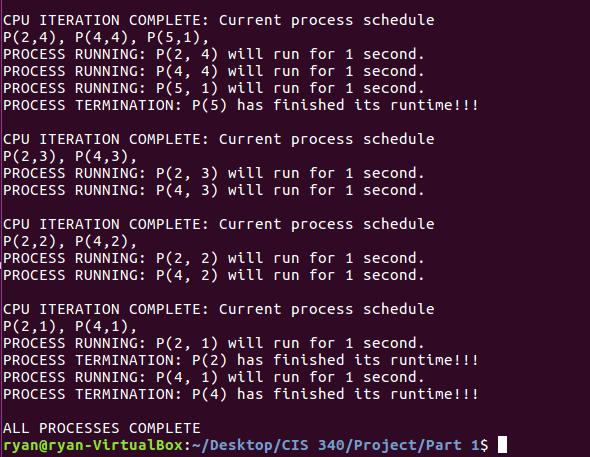
The RemoveCurrent() node removes the node passed to the method from the linked list. Currently it returns the current node which has been set to null. As an improvement It could be altered to return the node that was removed. RemoveCurrent also maintains the size of the list by decrementing the static int ListSize upon each call.

The add() method creates a new node given a process id and a run time. By default the node is added to the end of the linked list. Each time the add() method is call the listSize is incremented by 1.

Output:







PART TWO

The ls2.c code was used as the foundation for creating the necessary functions to do the sorting by size, time modified and filename. In order to sort by name, the do\_ls() was copied, renamed do\_lsbyname() and completely modified. The function still accepted an array as the parameter. The method was done by implementing two arrays to hold both the list with the sorted list. The method used a system of nested for loops and if statements to check each file found in the given directory and sorted accordingly inside an array. After which the array is the is passed through an for loop incrementing by one to the do stat() method, which then passes the filename and its information to the show\_file\_info() method which prints out the sorted list.

The do\_ls method was again copied and renamed to do\_ls1() accepts two parameters. Inside the original function’s else statement if statements were added to check how the directory would be sorted (either -s or -t). inside each if statement, the quick sort method was implement which then ran using the specific compare function in the method which based on on the list will be sorted, the files in the directory were compared based on their size or the last time it was modified.

The cp method was the same as giving in the lecture with just a minor addition which tells the user that a file already exists if the user tries to copy a file unto itself.

Output



