CSC 110 – Lab 9
Operating Systems Algorithms

Names: ____

Introduction
The purpose of this lab is to understand some basic operating systems algorithms and practice implementing parts of these algorithms in Python.

Exercises

1) Shortest Job First is a CPU scheduling algorithm that removes the shortest job from the process queue and executes it first. It keeps doing this until all of the processes are executed.

   What might be an advantage of this algorithm?

   What might be a disadvantage of this algorithm?

   In this exercise you will be modifying the CPU time slicing program that we discussed in class so that it implements a shortest-job-first scheduling algorithm instead.

   Here is the CPU time slicing code to start with:

   `http://www.cs.uri.edu/~cingiser/csc110/handouts/python/cpu_timeslice.py`

   Be sure to include enough print statements to indicate which process is executing its instructions.

   NOTE: The Queue data structure used in the time slice program is designed for FIFO access – it only allows you to get the process at the end of the queue. To implement SJF we will use a list to store the processes in this program so that we can get the shortest job to process. The main changes you will have to make to the program will be to replace the queue data structure with a list, and you will have to write a function to get the shortest job (process) from the list and remove it.

<u>Get the data from the file:</u>

To get you started, we provide the standard function for getting the two lists from the file:

```
def getProcs():
    fname = input("Enter the name of the data file")
    infile = open(fname, 'r')
    procList = []
    execList = []
# Loop through the file inserting processes into the list
    for line in infile:
        line = line.strip()
        proc, eTime = line.split(',')
        procList = procList + [proc]
        execList = execList + [int(eTime)]
    infile.close()
    return procList, execList
```

Modify the `cpu_timeslice.py` program so that it uses this function to get the data instead of the one that inserts the data into a queue.

Now modify the main so that all it does is call the `getProcs()` function and then print the two lists.

Get the short_processes.txt file:

http://www.cs.uri.edu/~cingiser/csc110/handouts/python/short_processes.txt

You will have to edit this file to take out the timeslice value because we are not using a timeslice in this scheduling algorithm. When you have this much done, test your program to make sure it works so far.

Instructor Initials: ______________________

Find the shortest process:

The next step is to write a function that finds the shortest process to execute. The function will take as a parameter the list of execution times, and return the POSITION of the shortest time (sound familiar??).

The signature for the function should look like this:

```
def findShortestProcess(execList):
     # Given a list of process execution times
     # (execList), find the position of the shortest
     # time in the list and return it

     return shortestPos
```

When you have this function working, add to your main function a call to the `findShortestProcess` function and print the process ID of the shortest process. Test the program again to make sure this much works so far.

Instructor Initials: ______________________

Schedule the processes:

Finally, we will modify the function that schedules the processes. In the original code for the time slice algorithm, the `scheduleProcs` function takes the processes out of a queue and runs them one at a time. In this program, we will be taking processes out of the two lists (`execList, procList`), one at a time, shortest first.

To make removal of the process from a list easy, you can use the Python built-in function `remove()`. So, if you had a list created like this:

```
     myList = [‘a’, ‘b’, ‘c’, ‘d’]
```

and you used the command:

```
     myList.remove(‘b’)
```

the result would be:

```
[‘a’, ‘c’, ‘d’]
```

Using this function and the `findShortestProcess` function, modify the function `scheduleProcs` so that it takes as parameters the two lists and executes the processes using the *Shortest Job First* algorithm. The signature for the function should be:

```
def scheduleProcs(procList, execList):
     # The code should look similar to the timeslice
     # program except now you will loop through until
     # the procList is empty and each time through the
     # loop, find the shortest process, execute the
     # shortest process, and then remove the shortest
     # process from both lists.

     return
```

Finish the program by making sure you call the functions correctly from the main function.

Instructor Initials: ____________________

2) Experiment with the Dining Philosophers program that we discussed in class. Download it from here:

http://www.cs.uri.edu/~cingiser/csc110/handouts/python/simple_dining_phils.py

Recall that this program cannot run from IDLE because it uses multiple processes. So let’s run it from the command line. The instructions here are for a Mac, but if you use a Windows machine, just open the command window, and the rest should be the same

a) Download the file and save it to the desktop.
b) From the Applications folder, open the Utilities folder and run Terminal.
c) You will need to change directories to the desktop. Do this by typing:

```
cd desktop
```

d) Make sure your dining philosophers program is in this directory by typing:

```
ls
```

to list the contents of the directory (that is the letter 'l' not the number one). You should see the file simple_dining_phils.py listed.

e) Run the program using Python version 3.2 by typing:

```
python3 simple_dining_phils.py
```

You should see the output of the program which is information about the philosophers and what they are doing.

Did deadlock occur when you ran the program?

Explain why.

If the program did deadlock, make a change to the code to fix it. If the code did not deadlock, make a change to the code to make deadlock occur.

Instructor Initials: ______________________

f) Now add a philosopher to the program. So you will have three philosophers and three chopsticks that they share. Modify the code so that it runs three processes, one for each philosopher. Can you make it deadlock with three philosophers? Can you avoid deadlock with three philosophers?

Instructor Initials: __________________________

Challenge Problems

1) Write a Python program that implements the Least Recently Used page replacement algorithm that we discussed in class.

   You can start with the following skeleton code:

   http://www.cs.uri.edu/~cingiser/csc110/labs/python/lru_skeleton.py

   This code assumes that you have a data file with two items on each line: data that is stored in memory, and the time that it was last used. For simplicity, the time is stored as an integer representing the number of seconds after the start of a particular event. In real operating systems, time is often stored as the amount of time (seconds) since the start of an "epoch". In Unix, this epoch began on January 1, 1970. So absolute time in Unix is represented as the number of seconds since January 1, 1970. For this program, let's assume that the epoch began when the system first started using the memory and storing data in it. So a low number in the data file represents an item that has not been used in quite a while. A higher number means that the data in the page was used more recently. You can find the data file here:

   http://www.cs.uri.edu/~cingiser/csc110/labs/python/pages.txt

Instructor Initials: __________________________