Scheduling Algorithm Comparison

I. Problem Statement

The assignment was to compare four different types of scheduling algorithms: First Come First Serve (FCFS), Shortest Process Next (SPN), Shorted Time Remaining (SRT), and Round Robin (RR). We also had to generate a 1000-element task stream, each with a size and time integer between 1-16. Then we were to run the task stream through each of the algorithms 1000 times and compare the results.

II. Initial Approach

To start I studied the four different algorithms mentioned in Chapter 9 of the textbook.

The normal distribution used to randomly generate the numbers was given to us in the assignment. I noticed that the given code didn’t seed random() and I thought about including this. In the end, however, I decided against it.

For the last assignment I used an array of structures with time and size integers within them to make up the task stream. This time I added a process ID within the structure so that I could easily “grab” a task from within the array.

I also made changes to the memory that I used in the last assignment. An array of integers wouldn’t be enough for what I wanted to do. So, I created a memory structure with time, a process ID, and arrival time variables.

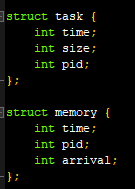
We were also asked to choose a placement algorithm from the last assignment. I went with First Fit. The first reason was because the results were similar between First Fit, Best Fit, Worst Fit, and Next Fit. This meant that I could have reasonably used any of them. However the second reason was simplicity to implement. For this, First Fit is by far the easiest to implement.

I started with FCFS. Since I just needed to increment the process ID that I was searching to process the task stream, this was a simple method to set up. In addition, I was able to write it in such a way that the other algorithms could be done with some simple edits.

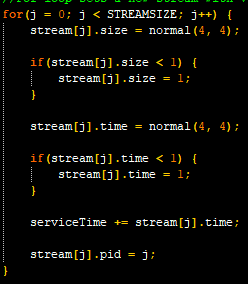
SPN and SRT were next. They were similar except for I looked for the process ID with the shortest time after completing a task with SPN and after every time decrement with SRT.

RR has a quantam of 1. I implanted this by incrementing the process ID that we are searching for comparing it to the process ID of the tasks in the memory for a match. If we reached max value, we’d start over from 0 again. If we made it back to our original process ID without finding a new one, then it was the last one left and we would keep the process ID.

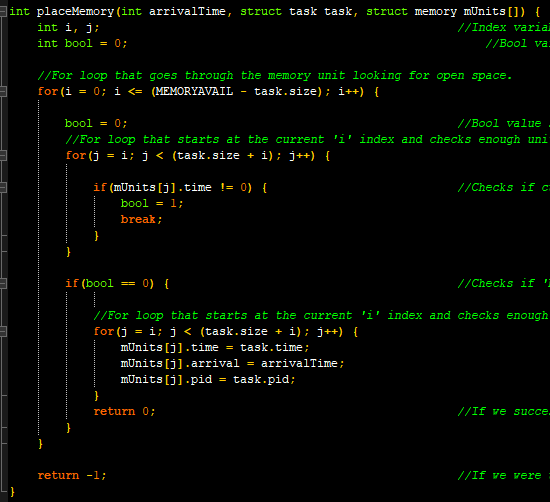
III. Process/Problems



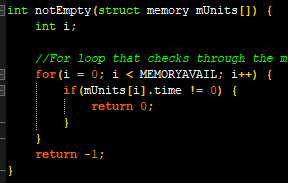
**Structures:** I added the process ID, “pid”, to the task structure I created for the last program. Memory has also been remade into a structure also with “time” and “pid”. Memory doesn’t need to know the size of the task, only the time. I gave it “arrival” to hold onto the value of the counter when it is entered into the memory so that we could calculate turnaround time.



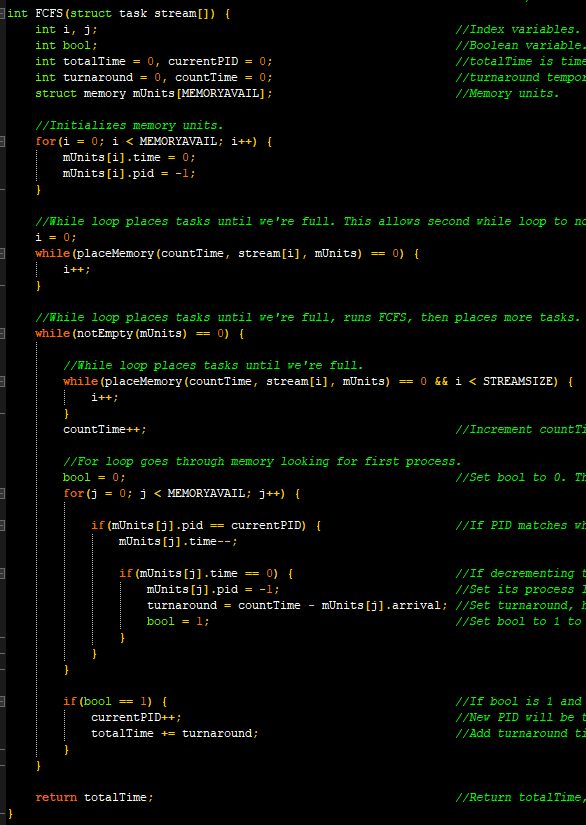
**Generate task stream:** This was given to us for the assignment. All size and time values will be between 1 and 16. The only addition I made was assigning a process ID based on the index value.



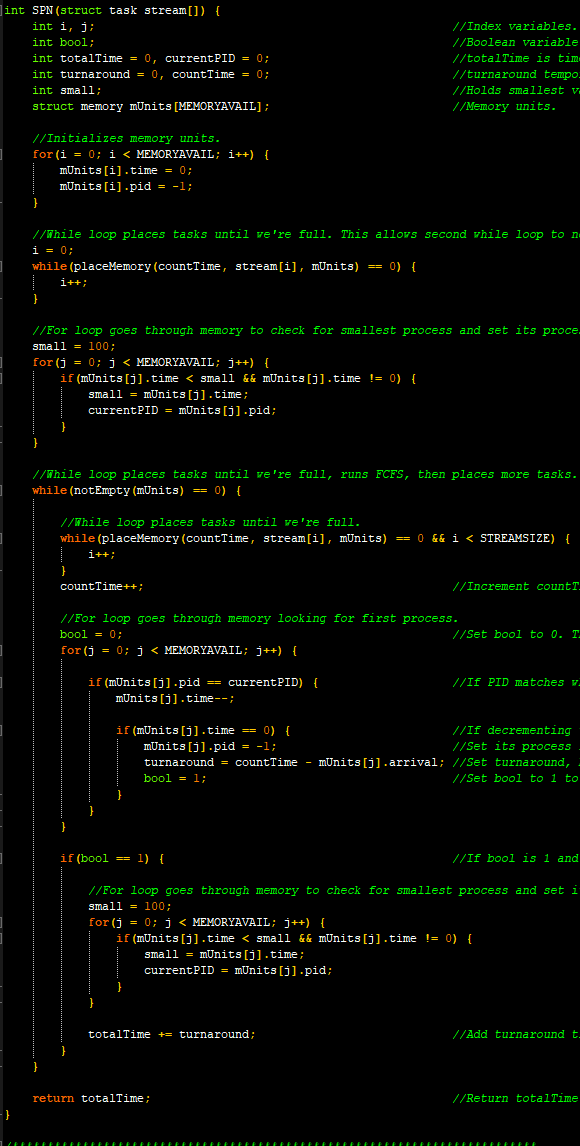
**Place Memory:** Using First Fit, we place the next task into the memory.



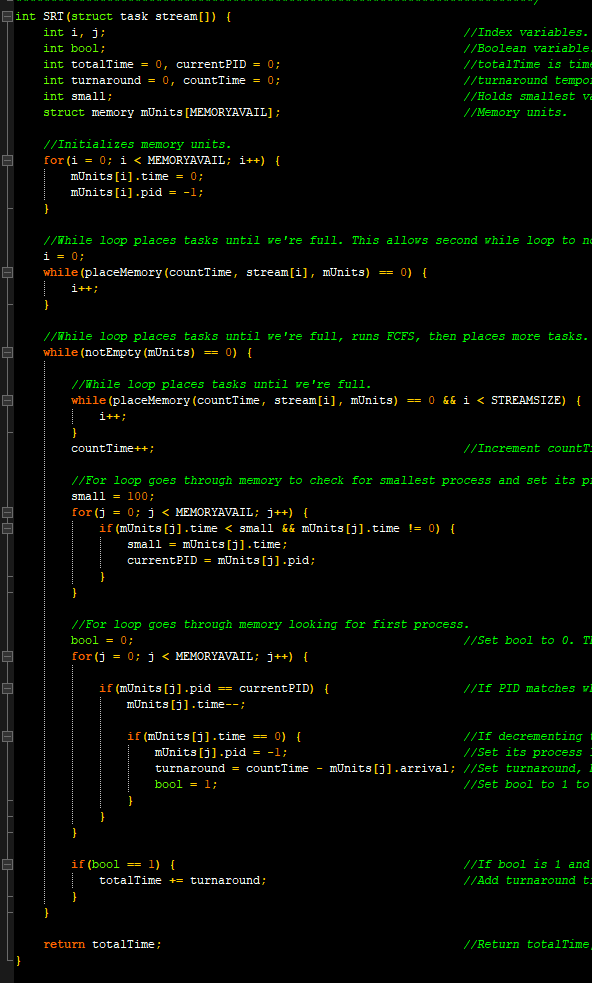
**Not Empty:** Checks through the memory to see if it is empty. This allows our while loop during the algorithms to keep running until we are done.



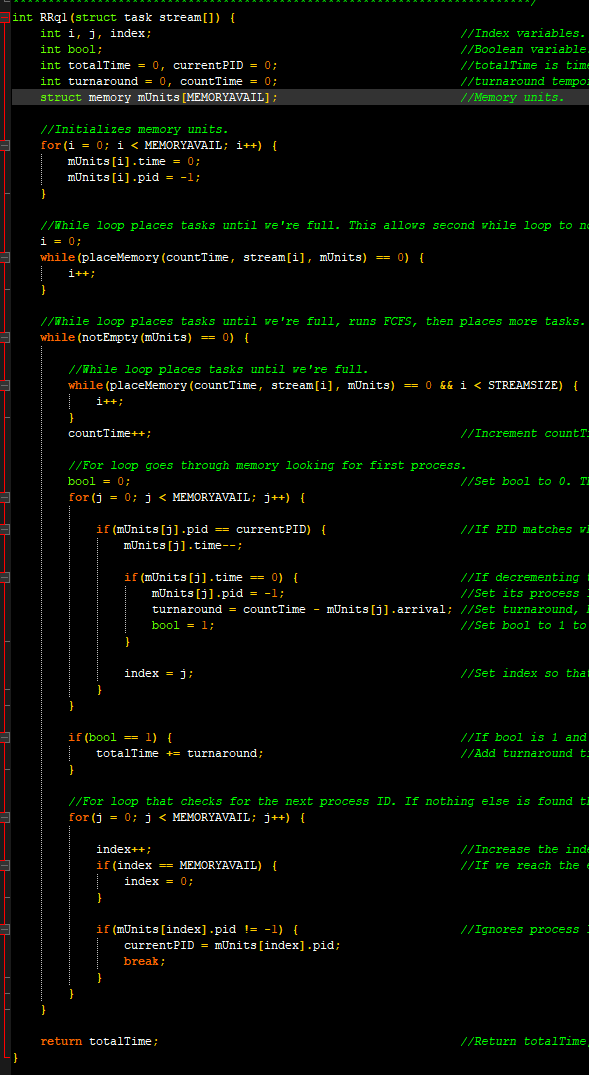
**First Come First Serve:** We increase currentPID every time we complete a task to move onto the next process.



**Shortest Process Next:** The only change made is instead of incrementing process ID we search for the process ID attached to the shortest time after the current task is complete. The picture above only includes the changed section.

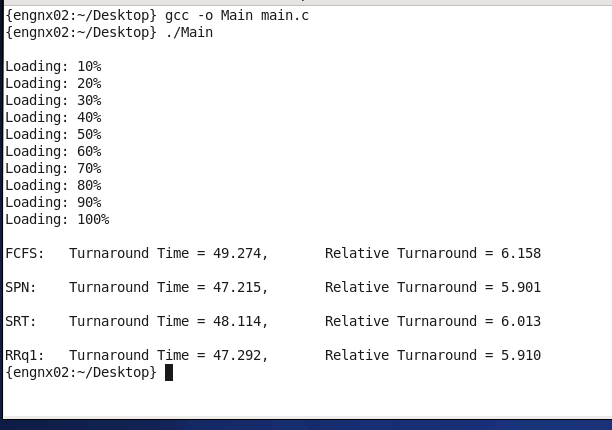


**Shortest Time Remaining:** Much like Shortest Process Next, but we search for a new process ID after every time we decrement a task. This means that if we have a shorter process appear it would switch over to it. Since this is uniprocessor and we fill up the memory every time this should be similar to SPN.



**Round Robin:** In this algorithm we increase the currentPID every time we decrement a task until we get a match for another process ID. With this we can make sure that no process waits for too long.

IV. Build/Final Thoughts



According to my results, SPN has the best result. This makes sense as it can “knock out” a lot of small processes for short tunaround times. The biggest downside would be the possibility of a big process being ignored for a long time.

Something that could be done to keep that from happening is setting it so that if a process is waiting for too long that it gets immediate priority.

In the end, the four algorithms have somewhat similar results. Any of them could be used effectively but I would probably go with SPN since it wasn’t difficult to implement and had the best results.