Kyle Kolodziej

CS 5343 – Dr. Hakki Cankaya

Operating Systems Project – Part 2

July 22nd, 2021

Operating Systems Project Part 2

Algorithms

Shortest Job First Scheduler (Non-Preemptive)

For the Shortest Job First Scheduler, I started with reading in the input file. I make a Process object for each Process in the input file which contains its process ID, arrival time, burst time, and priority. Each Process object that was created was added into an array. After reaching the end of the input file, I then sorted the array of Processes by their arrival time such that the Process' with the earliest arrival time are at the front. Then, I pass this sorted "Process Array" to a function called runShortestJobFirst(processArray).

The runShortestJobFirst(processArray) takes in an array of Processes that are sorted by their arrival time. Each input file contains only one Process that has an arrival time of zero. Also, I decided to build the Shortest Job First Scheduler in a non-preemptive manner. Thus, I can always start with the Process at the front of the array. This Process is removed from the Process Array and executed.

Then, I need to determine which Process should be executed next. First, I get the priority and burst time of the Process that is next up in the array storing them in variables

“currPriority” and “lowBurst” along with an “index” variable that will keep track of the index of the next Process to be executed. Next, I need to determine if there is another Process that is available for execution that has a lower burst time. I accomplish this by iterating through the Process Array until I reach a Process that has an arrival time that is greater than the current time (meaning it would not be available for execution). For each available Process, I compare the new Process’ burst time to that of “lowBurst”. If a Process has a higher burst time than “lowBurst”, I do not need to do anything. If a Process has a lower burst time than “lowBurst”, then I must update “currPriority”, “lowBurst”, and “index” to that of this new Process. If a Process has the same burst time as “lowBurst”, I will then compare its priority value to “currPriority”. If the new Process has a priority that is higher than “currPriority” (meaning the priority is a smaller number), then I will update the variables to that of this new Process. After reaching the end of the Process Array or hitting a Process that has an arrival time greater than the current time, I will then execute the Process that is in the Process Array at the index stored in the variable “index”. I update the current time, total wait time, and total turnaround time. Then, I remove this Process from the Process Array. While the Process Array contains two or more Processes in it, I continue this logic for executing Processes. Finally, I will reach a point where only one Process remains in the Process Array. By default, this Process can be executed, and I just need to update the statistics regarding the algorithm with this Process.

Non-Preemptive Priority Scheduler

For the Non-Preemptive Priority Scheduler, I started with reading in the input file. I make a Process object for each Process in the input file which contains its process ID, arrival

time, burst time, and priority. Each Process object that was created was added into an array. After reaching the end of the input file, I then sorted the array of Processes by their priority such that the Process' with the highest priority (smallest number) are at the front. Then, I pass this sorted "Process Array" to a function called runHighestPriority(processArray).

The runHighestPriority (processArray) takes in an array of Processes that are sorted by their arrival time. Each input file contains only one Process that has an arrival time of zero. Also, this is a non-preemptive algorithm. Thus, I can always start with the Process at the front of the array. This Process is removed from the Process Array and executed.

Then, I need to determine which Process should be executed next. First, I get the priority and burst time of the Process that is next up in the array storing them in variables "bestPriority" and "currBurst" along with an "index" variable that will keep track of the index of the next Process to be executed. Next, I need to determine if there is another Process that is available for execution that has a lower burst time. I accomplish this by iterating through the Process Array until I reach a Process that has an arrival time that is greater than the current time (meaning it would not be available for execution). For each available Process, I compare the new Process' priority to that of "bestPriority". If a Process has a worse priority (bigger number) than "bestPriority", I do not need to do anything. If a Process has a better priority (smaller number) than "bestPriority", then I must update "bestPriority", "currBurst", and "index" to that of this new Process. If a Process has the same priority as "bestPriority", I will then compare its burst time to "currBurst". If the new Process has a burst time that is lower than "currBurst" (meaning that it has a faster execution time), then I will update the variables to that of this new Process. After reaching the end of the Process Array or hitting a Process that

has an arrival time greater than the current time, I will then execute the Process that is in the Process Array at the index stored in the variable “index”. I update the current time, total wait time, and total turnaround time. Then, I remove this Process from the Process Array. While the Process Array contains two or more Processes in it, I continue this logic for executing Processes. Finally, I will reach a point where only one Process remains in the Process Array. By default, this Process can be executed, and I just need to update the statistics regarding the algorithm with this Process.

Results

Below, are my results from running both algorithms on the sample input file that contains 5 Processes and custom random generated input files that contain 10, 25, 50, and 100 Processes. Both the Shortest Job First and Non-Preemptive Priority Scheduler complete the execution of all of the Processes in the same time frame as seen in Figure 1.

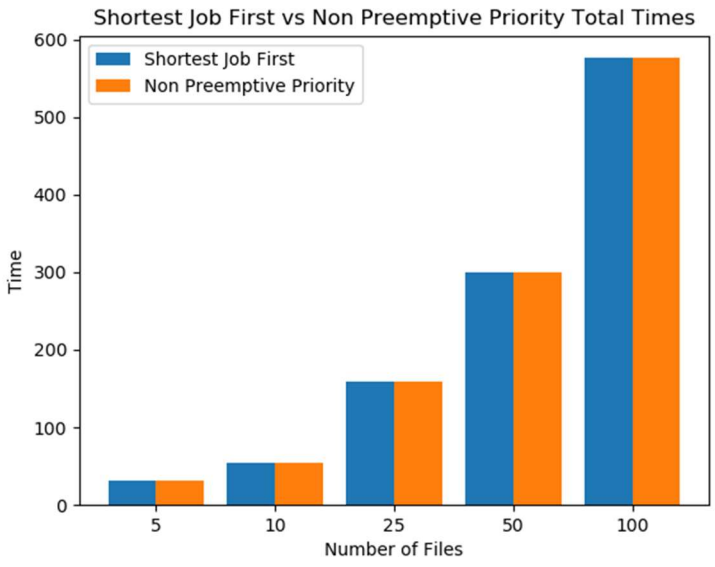

Figure 1: Shortest Job First vs Non Preemptive Priority - Total Time

However, the two algorithms differ in their average turnaround and wait times. As seen in Figure 2, the Shortest Job First Scheduler has a faster average turnaround time than that of the Non-Preemptive Priority Scheduler. Similarly, Figure 3 shows that the Shortest Job First Scheduler also has a better average wait time across all the input files. Both the faster average turnaround time and average wait time of the Shortest Job First Scheduler is due to the fact its selection criteria for executing Process' focuses on smaller burst times rather than higher priorities. For the Shortest Job First Scheduler, Processes with small burst times will get executed fairly soon as to when they are available for execution. This leads to many of these "smaller" Processes being executed right as they become available, thus decreasing the average wait time and average turnaround time. On the other hand, the Non-Preemptive Priority Scheduler uses priority values as its selection criteria for determining which Process to execute next. This results in "bigger" Processes with a better priority being selected over the "smaller" Processes with a worse priority. Thus, these "smaller" Processes are left waiting while the "bigger" Processes execute, resulting in the greater average wait time and average turnaround time. Overall, both the Shortest Job First Scheduler and the Non-Preemptive Priority Scheduler completed the Processes in the same total time. In terms of both average turnaround time and

average wait time, the Shortest Job First Scheduler outperforms the Non-Preemptive Priority Scheduler across all the input files.

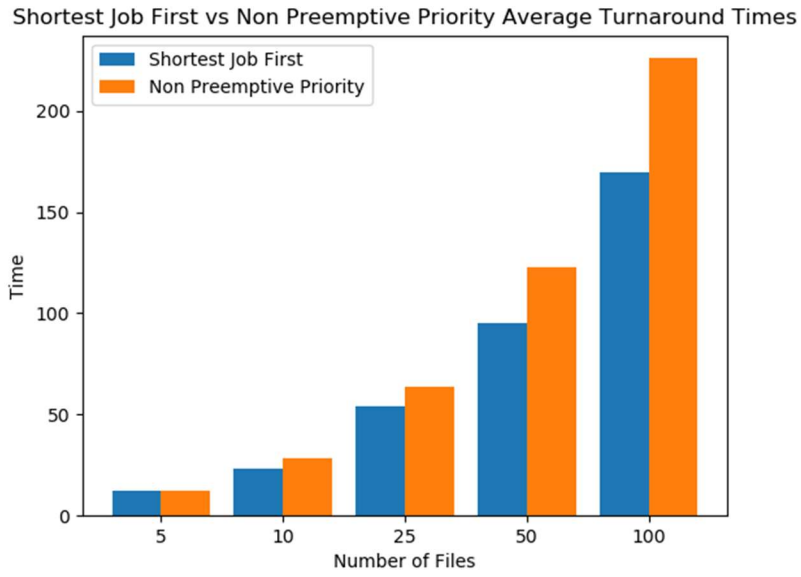

Figure 2: Shortest Job First vs Non Preemptive Priority - Average Turnaround Time

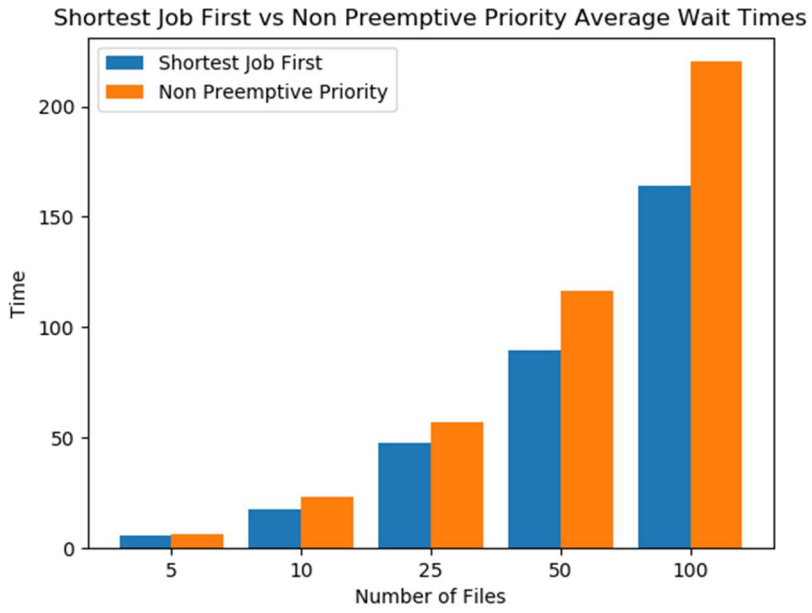

Figure 3: Shortest Job First vs Non Preemptive Priority - Average Wait Time

Programming Environment

I used my Windows HP Spectre x360 laptop for this project. This laptop has an Intel® Core™ i7-8550U CPU and 16.0 GB of RAM. It uses four cores and eight threads along with the one processor. This system utilizes a 64-bit operating system. I coded in Python using the application PyCharm.

Appendix – Project Part 1

Data Structures

For the Process Control Board Queue, I decided to use a Singly Linked List. I decided on using a Singly Linked List as this data structure does a great job of efficiently fulfilling the functionality of the Process Control Board queue. In comparison to a Doubly Linked List, the Singly Linked List takes up less memory. For deleting a process, we will need to locate the process with a specific process ID. Similarly, we will need to locate where a process' priority falls within the queue for inserting. Having to locate these processes mitigates the advantages a Doubly Linked List has with inserting and deleting over a Singly Linked List.

I created a class called Process. Objects of type Process are created for each Process and added to the Process Control Board Queue. Each Process contains a process ID and a priority value. The process ID is required for the Process while the priority is optional. If not given a priority value, the Process will set its priority value to -1 which is used to indicate that a Process does not contain a priority. Since I decided to use a Singly Linked List, each Process object only has a next pointer. This will be used for operations performed on the Process Control Block Queue such as inserting and deleting.

Additionally, I created a class called ProcessControlBlock. A Process Control Block can be initialized with or without a Process such that it could be an empty queue. Each Process Control Block object contains Process(es) that are connected by a Singly Linked List. The Process Control

Board also contains a pointer for the head and tail of the queue. These pointers are useful in the functions that need to be performed on the queue.

Algorithms Used

For adding processes to the process control board queue, I created a function called addProcess(self, processID, priority=None). Processes are required to have a process ID, but the priority is optional. I decided the best way to accommodate this with providing full functionality for inserting processes was to make a function that requires a process ID but the priority is optional. This way, I can always use this function regardless of if the process contains a priority or not. If a process being added does not have a priority, the default position for this process to be added is at the end of the process control board queue. Additionally, if multiple processes have the same priority value, the ordering for those processes with the same priority will function as a FIFO queue. For example, the processes that were in the process control board queue before will be farther up front than the process that is just added in with the same priority.

Upon inserting a process, I first check whether the queue is empty. If the queue is empty, I can just initialize the head and tail of the queue with this process. Otherwise, I will take one of two paths for inserting the process. First, I check if a priority was passed in. If no priority value was passed in, then I just need to add the process at the end of the queue. To do this, I just update the current tail's next pointer to point to the new process being added. Then, I update the tail to be pointing to the process that was just added at the end of the queue. If a

priority value was passed into the process and the queue is not empty, then I need to figure out where the new process falls in terms of priority. I start by checking if the head's priority value. If it does not have a priority or if the incoming process' priority value is lower (meaning it has a higher priority), the process being added becomes the new head of the queue. If this is not the case, then I iterate until either the incoming priority's value is lower than that "current" process' priority in the queue, the "current" process does not have a priority, or I reach the end of the queue. I insert the new process at this position, also updating the tail in the case that I reached the end of the queue.

For deleting a process, I first check if the Process Control Block Queue is empty. If it is empty, then I will print out a message to the user informing them of this. If this is not the case, then I determine whether I need to delete a specific process. When there is no process ID passed in, then I will just delete the process from the default position at the head. Otherwise, I need to search to find if this process exists in the queue. First, I check if the process ID matches with the head's process ID. Then, I will just delete the head, printing out this process' information to the user, and update the head to its next process that it is pointing to. For both the case that no process ID was passed in and the case that a process ID was passed in and matched with the head's process ID, I check to see if the head was the only process in the queue. Deleting this process would mean that the queue is now empty. Then, I would update the tail to reflect this. When passing in a process ID to be deleted that does not match with the head's process ID, I then check whether the head is the only process. If it is the only process in the queue, then I inform the user that they are unable to delete that process as it does not exist. On the other hand, if more process(es) exist, then I set a "curr" pointer to the head's next

process and a “prev” pointer to the head. While the process ID passed in to be deleted does not equal the process ID of “curr” and “curr” is not the last process in the queue, I iterate both “curr” and “prev” forward. When this while loop breaks, this either means that I have reached the process that needs to be deleted or the end of the queue. If I reached the process that should be deleted, I delete it by setting “prev’s” next pointer to “curr’s” next pointer and update the tail to be “prev” if “curr” was the last process in the queue. Otherwise, I output a message to the user that I was unable to delete a process with that process ID as there was no match.

For printing the process control board queue, I start by checking if the Process Control Board Queue is empty. If it is empty, I will output a message informing the user that the queue is empty. Otherwise, I will print the position (relative to in the queue), process ID, and priority of each process with an arrow pointed towards the next process in the queue. I iterate through all the processes by using a pointer to the head process called “curr”. I continue to move “curr” forward until it reaches the end of the queue, printing out the information of each process along the way.

Execution Trace

```
-----------------------------------------------------------------------------------------------------

Welcome to Kyle Kolodziej's Operating System's Project #1: Process Control Board queue Manipulation!

-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
Would you like to...
    1) Add Process(es) via an Input File
    2) Add Process(es) via a process ID and priority from your input
    3) Delete a Process
    4) Print the Process Control Board
    5) Exit
-----------------------------------------------------------------------------------------------------

Please input your option (1-5):
```

Figure 4: The welcome message and menu options shown at the start of running the program.

```
-----------------------------------------------------------------------------------------------------

Please input your option (1-5): 4

Printing the Process Control Board queue...

Process Control Board queue is empty!

-----------------------------------------------------------------------------------------------------
```

Figure 5: Attempting to print the Process Control Board queue when it is empty.

```
-----------------------------------------------------------------------------------------------------

Please input your option (1-5): 3
Would you like to delete a specific Process? If not, will default to the Process at the start of the queue
    1) Yes
    2) No
Please enter your choice (1 or 2): 2

Deleting Process from the default position (the head)...

Error! Not able to delete a Process...the Process Control Board queue is empty!


-----------------------------------------------------------------------------------------------------
```

Figure 6: Attempting to delete a default process when the Process Control Board queue is empty.

```
-----------------------------------------------------------------------------------------------------

Please input your option (1-5): 1
Please enter the input file's name: processInputFile

Successfully added Processes from:  processInputFile.txt
-----------------------------------------------------------------------------------------------------
```

Figure 7: Adding in processes from an input file.

```
-----------------------------------------------------------------------------------------------

Please input your option (1-5): 4

Printing the Process Control Board queue...

[Position: 1, Process ID = 7, Priority = 400] -----> [Position: 2, Process ID = 1, Priority = 403] -----> [Position: 3, Process ID = 16, Priority = 563]


-----------------------------------------------------------------------------------------------
```

Figure 8: Printing the Process Control Board queue after reading in the input file.

```
--------------------------------------------------------------------------------------------------

Please input your option (1-5): 2
Please enter the Process ID: INSERTING_THIS_MANUALLY
Would you like to enter a priority for this process?
    1) Yes
    2) No
Please enter your choice (1 or 2): 1

Please enter the priority (integer >= 1): 401
--------------------------------------------------------------------------------------------------
```

Figure 9: Inserting a process from a user's manual input of process ID and priority.

```
--------------------------------------------------------------------------------------------------

Please input your option (1-5): 4

Printing the Process Control Board queue...

[Position: 1, Process ID = 7, Priority = 400] -----> [Position: 2, Process ID = INSERTING_THIS_MANUALLY, Priority = 401] -----> [Position: 3, Process ID = 1, Priority = 403] --
```

Figure 10: Printing the queue after the manually entered process from above is added to the queue.

```
--------------------------------------------------------------------------------------------------

Please input your option (1-5): 2
Please enter the Process ID: INSERTING_MANUALLY_WITHOUT_PRIORITY
Would you like to enter a priority for this process?
    1) Yes
    2) No
Please enter your choice (1 or 2): 2
--------------------------------------------------------------------------------------------------
```

Figure 11: Inserting a process from a user's manual input of a process ID, but without a priority.

```
-----> [Position: 51, Process ID = 30, Priority = 9943] -----> [Position: 52, Process ID = INSERTING_MANUALLY_WITHOUT_PRIORITY, Priority = None]
```

Figure 12: Printing the queue and showing the tail after inserting the process without a priority.

```
-----------------------------------------------------------------------------------------------------

Please input your option (1-5): 3
Would you like to delete a specific Process? If not, will default to the Process at the start of the queue
    1) Yes
    2) No
Please enter your choice (1 or 2): 2

Deleting Process from the default position (the head)...

Process Deleted: [Position: 1, Process ID: 7, Priority: 400]


-----------------------------------------------------------------------------------------------------
```

Figure 13: Deleting a process from the default position (the head) by not passing in a process ID.

```
-----------------------------------------------------------------------------------------------------

Please input your option (1-5): 4

Printing the Process Control Board queue...

[Position: 1, Process ID = INSERTING_THIS_MANUALLY, Priority = 401] -----> [Position: 2, Process ID = 1, Priority = 403] -----> [Position: 3, Process ID = 16, Pri

-----------------------------------------------------------------------------------------------------
```

Figure 14: Printing the queue after deleting the default process at the head.

```
-----------------------------------------------------------------------------------------------------------

Please input your option (1-5): 3
Would you like to delete a specific Process? If not, will default to the Process at the start of the queue
    1) Yes
    2) No
Please enter your choice (1 or 2): 1
Please enter the Process ID for the Process to be deleted: 1

Deleting Process with process ID '1'...

Process Deleted: [Position: 2, Process ID: 1, Priority: 403]


-----------------------------------------------------------------------------------------------------------
```

Figure 15: Deleting a specific process from the queue.

```
-----------------------------------------------------------------------------------------------------------

Please input your option (1-5): 4

Printing the Process Control Board queue...

[Position: 1, Process ID = INSERTING_THIS_MANUALLY, Priority = 401] -----> [Position: 2, Process ID = 16, Priority = 563] -----> [Position: 3, Process ID = 11,

-----------------------------------------------------------------------------------------------------------
```

Figure 16: Printing the queue after deleting the specific process from above.

```
----------------------------------------------------------------------------------------------------

Please input your option (1-5): 3
Would you like to delete a specific Process? If not, will default to the Process at the start of the queue
    1) Yes
    2) No
Please enter your choice (1 or 2): 1
Please enter the Process ID for the Process to be deleted: THIS_PROCESS_DOES_NOT_EXIST

Deleting Process with process ID 'THIS_PROCESS_DOES_NOT_EXIST'...

Error! Process ID 'THIS_PROCESS_DOES_NOT_EXIST' does not exist in the Process Control Board queue!


----------------------------------------------------------------------------------------------------
```

Figure 17: Attempting to delete a specific process by a process ID that does not exist.

```
----------------------------------------------------------------------------------------------------
Would you like to...
    1) Add Process(es) via an Input File
    2) Add Process(es) via a process ID and priority from your input
    3) Delete a Process
    4) Print the Process Control Board
    5) Exit
----------------------------------------------------------------------------------------------------

Please input your option (1-5): 5

----------------------------------------------------------------------------------------------------

Thank you for using Kyle Kolodziej's Process Control Board Queue! Goodbye!

----------------------------------------------------------------------------------------------------


Process finished with exit code 0
```

Figure 18: Exiting the program.

Programming Environment

I used my Windows HP Spectre x360 laptop for this project. This laptop has an Intel® Core™ i7-8550U CPU and 16.0 GB of RAM. It uses four cores and eight threads along with the one processor. This system utilizes a 64-bit operating system. I coded in Python using the application PyCharm.