Jimmy Tran 028343131

Project 4 Report

Code Outputs

FCFS Algorithm:

```
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % make fcfs
gcc -Wall -c driver.c
gcc -Wall -c list.c
gcc -Wall -c CPU.c
gcc -Wall -c schedule_fcfs.c
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % ./fcfs schedule.txt
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T8] [10] [25] for 25 units.
```

Priority Algorithm:

```
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % make priority
gcc -Wall -c driver.c
gcc -Wall -c list.c
gcc -Wall -c CPU.c
gcc -Wall -c schedule_priority.c
schedule_priority.c:33:18: warning: unused variable 'i' [-Wunused-variable]
    int swapped, i;
                 ^
1 warning generated.
gcc -Wall -o priority driver.o schedule_priority.o list.o CPU.o
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % ./priority pri-schedule.txt
Running task = [T1] [1] [50] for 50 units.
Running task = [T4] [2] [50] for 50 units.
Running task = [T3] [3] [50] for 50 units.
Running task = [T2] [4] [50] for 50 units.
Running task = [T5] [5] [50] for 50 units.
Running task = [T6] [6] [50] for 50 units.
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 %
```

Round Robin Algorithm:

```
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % make rr
gcc -Wall -c driver.c
gcc -Wall -c list.c
gcc -Wall -c CPU.c
gcc -Wall -c schedule_rr.c
gcc -Wall -o rr driver.o schedule_rr.o list.o CPU.o
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 % ./rr rr-schedule.txt
Running task = [T1] [40] [50] for 10 units.
Running task = [T2] [40] [50] for 10 units.
Running task = [T3] [40] [50] for 10 units.
Running task = [T4] [40] [50] for 10 units.
Running task = [T5] [40] [50] for 10 units.
Running task = [T6] [40] [50] for 10 units.
Running task = [T1] [40] [40] for 10 units.
Running task = [T2] [40] [40] for 10 units.
Running task = [T3] [40] [40] for 10 units.
Running task = [T4] [40] [40] for 10 units.
Running task = [T5] [40] [40] for 10 units.
Running task = [T6] [40] [40] for 10 units.
Running task = [T1] [40] [30] for 10 units.
Running task = [T2] [40] [30] for 10 units.
Running task = [T3] [40] [30] for 10 units.
Running task = [T4] [40] [30] for 10 units.
Running task = [T5] [40] [30] for 10 units.
Running task = [T6] [40] [30] for 10 units.
Running task = [T1] [40] [20] for 10 units.
Running task = [T2] [40] [20] for 10 units.
Running task = [T3] [40] [20] for 10 units.
Running task = [T4] [40] [20] for 10 units.
Running task = [T5] [40] [20] for 10 units.
Running task = [T6] [40] [20] for 10 units.
Running task = [T1] [40] [10] for 10 units.
Running task = [T2] [40] [10] for 10 units.
Running task = [T3] [40] [10] for 10 units.
Running task = [T4] [40] [10] for 10 units.
Running task = [T5] [40] [10] for 10 units.
Running task = [T6] [40] [10] for 10 units.
(base) jimmytran@Jimmys-MacBook-Pro CECS-326Proj4 %
```

The main part of this project was understanding how makefiles work and to implement the three scheduling algorithms. For all three of my scheduling algorithms, I used the same code for the **add()** function because the functionality of that does not change throughout the schedulers. All schedulers use a linked list structure to store their processes, so the code remains the same. For my FCFS algorithm, it was fairly simple as all I needed to do was to reverse the linked list and then traverse through that list and execute the **run()** function provided. I reversed the list because the **insert()** function provided added new nodes to the beginning of the list, which was the opposite of FCFS. For my priority algorithm, I sorted the linked list according to their priority number, from least to greatest. I used bubblesort as it is one of the easiest

algorithms to implement and performance is not a focus point. Then I traversed the list and executed the **run()** function. For the round robin algorithm, I also had to reverse the list so the first process in will be the first process to go. It then loops through the list and executes each process for the amount of time specified by the quantum. The loop stops only if all processes are finished. Some difficulties I had were understanding how makefiles worked and the overall structure of the program because usually I create all my files from scratch and work from there. I understood all the code though after looking through the files for a good amount of time.

Video link: https://youtu.be/K_K-rYT1_1s