CPE 435: OPERATING SYSTEMS LABORATORY.

Lab07

<u>Process Scheduling.</u>

Submitted by: Dan Otieno.

Date of Experiment: 02/24/23.

Report Deadline: 03/03/23.

Demonstration Deadline: 03/03/23.

Introduction:

An operating system can allow for multiple processes to be queued into executable memory at the same time, thereby permitting the sharing of CPU via time multiplexing. The goal for this lab was to understand how processes are scheduled and executed in the CPU by writing programs to simulate processing concepts.

Theory:

According to the lab manual, the operating system in itself, is implemented as a collection of processes, so, by that logic, there must be a way for the OS and the various application processes to share CPU resources. A scheduler allocates processor resources to each process. For this lab, we explored Round-robin and Priority scheduling, those are discussed a bit more below:

- Round Robin Scheduling:
  - Round robin is a process scheduling algorithm that works by allocating the CPU to each process for a time quantum. If the process does not relinquish the CPU before its time quantum expires, the process is preempted, and another process is scheduled to run for a time quantum. – (Source is CPE 434 Study material).

- Priority Based Scheduling:
  - In priority scheduling, each process is assigned a priority, and the CPU is allocated to the process with the highest priority. If multiple processes have the same priority, the order of processing would be in round robin of FCFS (First-Come-First-Serve) form. – (Source is CPE 434 Study material).

- Preemptive vs. Non-Preemptive Scheduling:
  - Preemptive Scheduling and Non-Preemptive Scheduling differ when it comes to control of the CPU. In Preemptive Scheduling, the CPU can be taken away from a process, whereas in Non-Preemptive Scheduling, the process must voluntarily relinquish control of the CPU. Majority of modern operating systems are preemptive. – (Source is CPE 434 Study material).

Results & Observation:

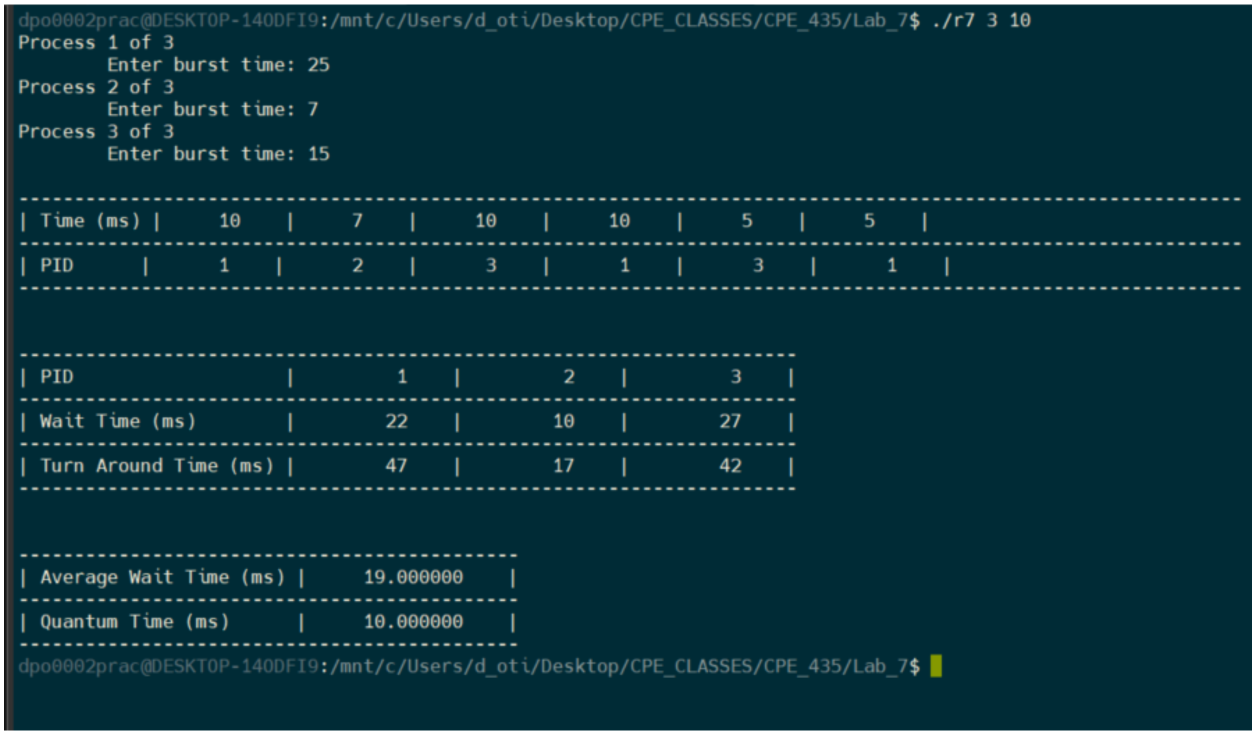
Assignment 1:

Description:

The goal for this assignment was to use C/C++ to write two programs to simulate Round Robin Scheduling, and Priority Scheduling. Both programs take user inputs for number of processes and Quantum time as command line arguments. Burst time and priority are read as user inputs. The output format is two tables that also show the average waiting time.

Program Outputs:

- Round Robin Scheduling:



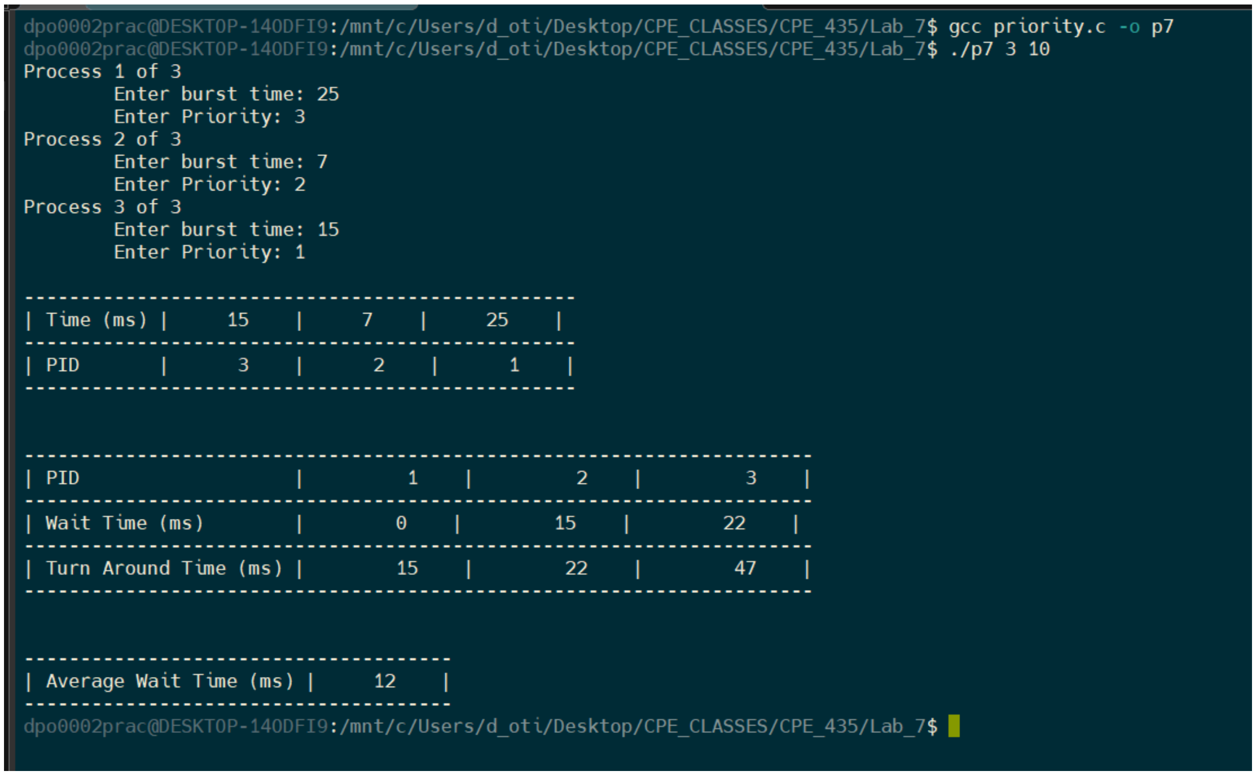
```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_7$ ./r7 3 10
Process 1 of 3
        Enter burst time: 25
Process 2 of 3
        Enter burst time: 7
Process 3 of 3
        Enter burst time: 15

--------------------------------------------------------------------------------------------------------------
| Time (ms) |     10     |     7     |     10     |     10     |     5     |     5     |
--------------------------------------------------------------------------------------------------------------
| PID       |      1     |     2     |      3     |      1     |     3     |     1     |
--------------------------------------------------------------------------------------------------------------


----------------------------------------------------------------------
| PID                 |          1     |          2     |          3     |
----------------------------------------------------------------------
| Wait Time (ms)      |         22     |         10     |         27     |
----------------------------------------------------------------------
| Turn Around Time (ms) |       47     |         17     |         42     |
----------------------------------------------------------------------


---------------------------------------------
| Average Wait Time (ms) |     19.000000     |
---------------------------------------------
| Quantum Time (ms)      |     10.000000     |
---------------------------------------------
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_7$
```

- Priority Scheduling (**DESIRED OUTPUT NOT ACHIEVED**):



```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_7$ gcc priority.c -o p7
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_7$ ./p7 3 10
Process 1 of 3
        Enter burst time: 25
        Enter Priority: 3
Process 2 of 3
        Enter burst time: 7
        Enter Priority: 2
Process 3 of 3
        Enter burst time: 15
        Enter Priority: 1

-------------------------------------------------
| Time (ms) |     15     |     7     |     25     |
-------------------------------------------------
| PID       |      3     |     2     |      1     |
-------------------------------------------------


----------------------------------------------------------------------
| PID                   |          1     |          2     |          3     |
----------------------------------------------------------------------
| Wait Time (ms)        |         0     |         15     |         22     |
----------------------------------------------------------------------
| Turn Around Time (ms) |         15     |         22     |         47     |
----------------------------------------------------------------------


--------------------------------------
| Average Wait Time (ms) |     12     |
--------------------------------------
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_7$
```

Conclusion:

Unfortunately, the Priority Scheduling program did not work as expected, this will be reviewed for a better demonstration in the next class. However, the Round robin output was in tandem with the sample outputs provided in the lab manual, therefore worked as expected.

Appendix:

Assignment 1 code – Round Robin.

```
/*
DAN OTIENO
CPE 435-01
LAB 7
Round Robin.
*/

#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <math.h>

struct process
{
        int pid;
        int bTime; // burst time.
        int trnTime; // turn around time
        int wtTime; // Wait Time = Turn around Time - Burst Time
        int remTime; //Remaining time.
        int wrkTime;//Working time.
        int priority;
        int arrTime;
};

void dashes(int d)
{
        printf("\n");
        for(int i = 0; i < d; i++)
        {
                printf("-");
        }
        printf("\n");
}

void swap(struct process *a, struct process *b)
{
        struct process tmp = *a;
        *a=*b;
        *b=tmp;
}

int main(int argc, char* argv[])
{
        int n = atoi(argv[1]); // number of processes
        int quantum = atoi(argv[2]); // quantum time unit
        int avgwt = 0;
        struct process *p;

        // Allocating memory for n number of processes
        p = malloc(sizeof(struct process) * n);


        if (p == 0)
        {
                exit(1);
        }

        for (int i = 0; i < n; i++)
        {
                printf("Process %d of %d\n", i+1, n);
                p[i].pid = i+1;
                printf("\tEnter burst time: ");
                scanf("%d", &p[i].bTime);
                p[i].remTime = p[i].bTime;
                p[i].wrkTime = p[i].trnTime = p[i].wtTime = p[i].arrTime =
0;

        }
        int count = 0, currTime = 0, nSteps = 0, rem = n;

        for(int i = 0; i < n; i++)
        {
                nSteps += ceil((double)p[i].bTime / quantum);
        }

        int pSteps[nSteps], tSteps[nSteps], count2 = 0;

   while (rem > 0)
   {
      for (int i = 0 ; i < n; i++)
      {
         if (p[i].remTime > 0)
         {
            if (p[i].remTime > quantum)
            {
               currTime += quantum;
               p[i].remTime -= quantum;
               pSteps[count2] = p[i].pid;
               tSteps[count2] = quantum;
               count2++;
            }
            else
            {
               pSteps[count2] = p[i].pid;
               tSteps[count2] = p[i].remTime;
               count2++;
               currTime += p[i].remTime;
               p[i].wrkTime = currTime;
               p[i].trnTime = p[i].wrkTime - p[i].arrTime;
               p[i].remTime = 0;
```

```
                p[i].wtTime = p[i].trnTime - p[i].bTime;
                rem--;
            }
        }
    }
}

for(int i = 0; i < n-1; i++)
{
     for(int j = i+1; j < n; j++)
     {
            if(p[i].priority < p[j].priority)
            {
                   swap(&p[i], &p[j]);
            }
     }
}

for (int i = 0; i < n ; i++)
{
            p[i].trnTime = p[i].bTime + p[i].wtTime;
}

     for (int i = 0; i < n; i++)
     {
            avgwt += p[i].wtTime;
     }
     avgwt = avgwt / n;

     /******************** Printing Outputs *********************/
     dashes(110);
     printf("| Time (ms) |");
     for (int i = 0; i < nSteps; i++)
     {
            printf("     %d    |", tSteps[i]);
     }

     dashes(110);
     printf("| PID      |");
     for (int i = 0; i < nSteps; i++)
     {
            printf("      %d    |", pSteps[i]);
     }
     dashes(110);

     printf("\n");
     dashes(70);
     printf("| PID                    |   ");
     for (int i = 0; i < n; i++)
     {
            printf("      %d    |   ", p[i].pid);
     }
     dashes(70);

     printf("| Wait Time (ms)         |   ");
     for (int i = 0; i < n; i++)
     {
```

```
                printf("     %d     |    ", p[i].wtTime);
        }
        dashes(70);

        printf("| Turn Around Time (ms) |    ");
        for (int i = 0; i < n; i++)
        {
                printf("     %d     |    ", p[i].trnTime);
        }
        dashes(70);

        printf("\n");
        dashes(45);
        printf("| Average Wait Time (ms) |      %f     |", (float)avgwt);
        dashes(45);
        printf("| Quantum Time (ms)       |      %f     |", (float)quantum);
        dashes(45);
/***********************************************************/
        void free(void*);
}
```

Assignment 1 code – Priority.

```
/*

DAN OTIENO

CPE 435-01

LAB 7

Priority.

*/

#include <stdbool.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>


struct process

{

        int pid; //Process ID

        int priority;

        int bTime; //CPU Burst Time

        int wTime; //Time this process executed, if wTime==bTime, process is
complete
```

```
        int trnTime; //turn around time, time needed for the process to
complete

        int wtTime; // Wait Time = Turn around Time - Burst Time

};


/***** Function to print hyphens *****/

void dashes(int d)

{

        printf("\n");

        for(int i = 0; i < d; i++)

        {

                printf("-");

        }

        printf("\n");

}

/**********************************/


void swap(struct process *a, struct process *b)

{

        struct process tmp = *a;

        *a=*b;

        *b=tmp;

}


int main(int argc, char* argv[])

{

        int n = atoi(argv[1]);

        int quantum = atoi(argv[2]);

        int avgwt;

        struct process *prc;


        prc = malloc(sizeof(struct process) * n); // Memory allocation for n
processes.
```

```
        if (prc == 0)
        {
                exit(1);
        }

        for (int i = 0; i < n; i++)
        {
                printf("Process %d of %d\n", i+1, n);
                prc[i].pid = i+1;
                printf("\tEnter burst time: ");
                scanf("%d", &prc[i].bTime);
                printf("\tEnter Priority: ");
                scanf("%d", &prc[i].priority);
        }

        for(int i = 0; i < n-1; i++)
        {
                for(int j = 0; j < n-i-1; j++)
                {
                        if(prc[j].priority > prc[j+1].priority)
                        {
                                swap(&prc[j], &prc[j+1]);
                        }
                }
        }

        int count = 0;
        prc[0].wtTime = 0;

        for(int i = 1; i < n; i++)
        {
                prc[i].wtTime = 0;
                for(int j = 0; j < i; j++)
                {
                        prc[i].wtTime += prc[j].bTime;
                }
                count += prc[i].wtTime;
        }

        avgwt = (float)count / n;

        int count2 = 0;
        for (int i = 0; i < n; i++)
        {
                prc[i].trnTime = prc[i].bTime + prc[i].wtTime;
                count2 += prc[i].trnTime;
        }

/******************** Printing Outputs ********************/
        dashes(49);
        printf("| Time (ms) |");
        for (int i = 0; i < n; i++)
        {
                printf("     %d    |", prc[i].bTime);
        }
        dashes(49);
        printf("| PID       |");
        for (int i = 0; i < n; i++)
        {
                printf("      %d    |", prc[i].priority);
        }
        dashes(49);

        printf("\n");
        dashes(70);
        printf("| PID                   |   ");
        for (int i = 0; i < n; i++)
        {
                printf("      %d    |   ", prc[i].pid);
        }
        dashes(70);

        printf("| Wait Time (ms)        |   ");
        for (int i = 0; i < n; i++)
        {
                printf("     %d    |   ", prc[i].wtTime);
        }
        dashes(70);

        printf("| Turn Around Time (ms) |   ");
        for (int i = 0; i < n; i++)
        {
                printf("     %d    |   ", prc[i].trnTime);
        }
        dashes(70);
        printf("\n");
        dashes(38);
        printf("| Average Wait Time (ms) |     %d    |", avgwt);
        dashes(38);
/*************************************************************/
        void free(void*); // Memory deallocation.
        return 0;
}
```