|  |  |  |
| --- | --- | --- |
|  | SRM INSTITUTE OF SCIENCE AND TECHNOLOGY  SCHOOL OF COMPUTING  DEPARTMENT OF DATASCIENCE AND BUSINESS SYSTEMS  21CSC202J OPERATING SYSTEMS |  |
| MINI PROJECT REPORT  **Pre-emptive and Non pre-emptive Priority Scheduling Algorithms** | | |
| **Name: MEGHAVI RATHOD**  **Register Number: RA2112704010012**  **Mail ID: mp0487@srmist.edu.in**  **Department: M.Tech Integrated CSE**  **Specialization: DSBS**  **Semester: 3** | | |
| **Team Members :**  **Name : BADIGI SATHWIKA Registration Number: RA2112704010032** | | |

**Abstract**

There are priority pre-emptive scheduling algorithm and priority non-pre-emptive scheduling algorithm in operating system. When the newly arrived process is compared its priority to the process that is running and if its priority is higher it will executed by the CPU. This is known as priority pre-emptive scheduling algorithm where when the recently arrived process is positioned at the head of the queue and cannot be interrupted, this is known as priority non-pre-emptive scheduling algorithm. This priority algorithm is based on execution of process over a priority value. The higher the value, the higher the priority. Every process has its own priority. The first process to be executed is the process with higher priority, then continue until all the processes finish. In this project we compared the priority pre-emptive scheduling algorithm. With the priority number from 0 until 10, where the 0 the lowest priority and 10 is the highest priority. Two case studies are discussed here.

**Chapter 1: Introduction and Motivation [Purpose of the problem statement (societal benefit)**

Operating System (OS) is software which its role is as an interface between a user and the computer hardware. OS is known as a resource manager because its main duty is to manage the resources of computer system. Scheduling is one fundamental and most important design. Scheduling refers to set of rules, policies and mechanism that govern the order in which resources is allocated to various processes and the work is to be done. The scheduling is a methodology of managing many queues of processes in order to make delay minimum and to make performance optimal of the system. A scheduler is a module in operating system that implements the scheduling policies. Its main objective is to make system’s performance’s optimal that match the criteria set by the system designer.

Scheduling is a prime concept in multiprocessing and multitasking of OS design and in real-time operating system design by arranging switching in the CPU among process. Priority Scheduling Algorithm is a well-known algorithm in CPU processing.

For system of single processor, when multiple process comes, then one process can be execute at a time and other process remain in waiting state until the CPU becomes idle or can be scheduled again. To expand the CPU usage, the goal of multiprogramming is to have some procedure running at all times. CPU scheduling manages the issue of choosing which of the procedures in the ready queue is to be assigned the CPU. Operating system usually performs scheduling of processes which is major task of a system.

Priority pre-emptive scheduling algorithm is usually use among various other algorithms for scheduling CPU, however it can make the problem of starvation which happens when processes with priority which is lower are not given any chance of CPU utilization due to prolong CPU usage by processes with higher priorities.

Priority non-pre-emptive scheduling is one of hard real-time periodic tasks. In many real-time scheduling problems such as I/O scheduling, pre-emption is often impossible or prohibitively expensive due to properties of hardware device and software handling the device. For example, showed that non-pre-emptive fixed priority scheduling can be used for real-time signal processing applications because the amount of processor state including the stack to be stored can be reduced. Also non-pre-emptive scheduling on unit-processor systems eliminates the synchronization overhead of resource protecting mechanism. Since we can expect much lower overhead and smaller memory requirement at run-time with non-pre-emptive scheduling, non-pre-emptive scheduling is widely used in embedded systems, especially devices such as cell phones whose behaviour largely depends on network I/O.

**Chapter 2: Review of Existing methods and their Limitations**

For Priority Scheduling, the basic idea is straightforward, each process is assigned a priority. Priority processes which are equal to FCFS process. SJF is one example. SJF has the same idea as Priority Scheduling. Namely, the longer the burst of CPU, then it makes the lower the priority and the smaller the burst time, the higher the priority. Priority can be defined internally or externally. Internally defined priorities use some measurable quantities or qualities to computer priority of a process. Priority scheduling algorithm manages processes in its queue based on its priority. Something else that gives priority on running state is pre-emption.

**Limitations:**

We can lose all the low-priority processes if the system crashes. This process can cause starvation if high-priority processes take too much CPU time. The lower priority process can also be postponed for an indefinite time.

A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

**Chapter 3: Proposed Method with System Architecture / Flow Diagram**

Every process has a priority and the CPU is allocated based on the highest priority. The important of any process is based on a range of values. Any process entering the ready queue will be allocated its importance based on priority. This priority value defined the decision of allocating the CPU to a process in a way that the process with high value of priority will occupy the CPU first or next. There are two version, one is pre-emptive, and another one is non-pre-emptive.

**Pre-emptive Priority Scheduling:**

In pre-emptive algorithm, the higher the value of the priority, the higher the priority. Let’s say the priority is 3 compared with priority 1, so the priority 3 will get the CPU first, since its priority is higher. And in the pre-emptive system, the process can be interrupted by a new process in ready queue if the new process has higher priority than the running one. Starvation can occur in this system if the value of priority is always higher than the one in the waiting queue.

Processes with the same priority are executed based on FCFS. Priority can be defined based on time requirements, memory requirements of any other resources requirements. Priority are indicated by some fixed range of numbers, such 0-7, in this analysis we use priority number from 1 to 5.

**Non pre-emptive Priority Scheduling:**

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy, will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That’s because it doesn’t need special hardware (for example, a timer) like pre-emptive scheduling.

It’s similar to SJF scheduling. In SJF, burst time was the priority. Here, priority is explicitly provided. The process that has highest priority gets the CPU first. The processes gets serviced by the CPU in order of their priority in descending order. Higher number always doesn’t represents higher priority. In some cases, 0 may be the highest priority and 100 the lowest. It depends on systems. We have shown the Flowchart for Pre-emptive and Non pre-emptive Priority Scheduling respectively:

**p[i].pid = i+1**

**FINISH**

**p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time**

**p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time**

**burst\_remaining[idx] == 0**

**idx = i**

**p[i].arrival\_time < p[idx].arrival\_time**

**arrival\_time <= current\_time && is\_completed[i] == 0**

**i<n**

**Current\_time = 0**

**burst\_remaining = burst\_time**

**i<n**

**pid, arrival\_time, burst\_time, priority, start\_time,completion\_time, turnaround\_time, waiting\_time, response\_time**

**START**

**i<n**

**pid, arrival\_time, burst\_time, priority, start\_time,completion\_time, turnaround\_time, waiting\_time, response\_time**

**START**

**p[i].pid = i+1**

**FINISH**

**p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time**

**p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time**

**idx != -1**

**idx = i**

**p[i].arrival\_time < p[idx].arrival\_time)**

**idx = i**

**p[i].arrival\_time <= current\_time && is\_completed[i] == 0**

**i<n**

**completed != n**

**Chapter 4: Modules Description**

We need to find the Waiting and Average Turnaround Time very often in the case of CPU scheduling. We do so by making use of the Completion, Arrival, and Burst Time. In this article, we will discuss the difference between Turn Around Time (TAT) and Waiting Time (WT) in CPU Scheduling. But let us first understand a bit more about these concepts.

**Turn Around Time (TAT):**

TAT refers to the total time interval present between the time of process submission and the time of its completion. The difference between the time of completion and the time of arrival is known as the Turn Around Time of the process. The overall speed of the output device has an effect on the TATA. It gets limited. Different sets of CPU algorithms have the capability of producing different TATs for the very same process set.

CT (Completion Time) – It is the exact time when a process completes the execution part.

AT (Arrival Time) – It is the time of arrival of a process in its ready state (before its execution).

Thus, CT – AT = TAT

**Waiting Time (WT):**

WT refers to the total time that a process spends while waiting in a ready queue before reaching the CPU. The difference between (time) of the turn around and burst time is known as the waiting time of a process. The overall amount of time of process execution and I/O does not experience any alteration due to the CPU scheduling algorithm. It only affects the time that each process spends while waiting in a ready queue.

BT (Burst Time) – It is the total time that a process requires for its overall execution.

Thus, TAT – BT = WT

Now, we can also easily calculate the Turn Around Time using the Burst Time and the Waiting Time.

Here, BT + WT = TAT

**Response Time (RT):**

Response time is the time spent when the process is in the ready state and gets the CPU for the first time.

Response time – amount of time from when a request was submitted until the first response is produced. Associate with each process the length of its next CPU burst. Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds.

RT = ST – AT

Where, ST – Starting Time

AT – Arrival Time

**Chapter 5: Implementation requirements**

**Pre-emptive:**

**Input:**

1. Number of processes
2. Arrival time of each process. If all process arrives at the same time, this can be set to 0 for all processes.
3. Burst time of each process
4. Priority of each process

**Output:**

1. Start Time (ST), Completion Time (CT), Turnaround Time (TAT), Waiting Time (WT) and Response Time (RT) for each process.
2. Average turnaround Time, average waiting time and average response time.
3. Throughput and CPU utilization.

**Algorithm:**

completed = 0

current\_time = 0

while(completed != n) {

find process with maximum priority time among process that are in ready queue at current\_time

if(process found) {

if(process is getting CPU for the first time) {

start\_time = current\_time

}

burst\_time = burst\_time - 1

current\_time = current\_time + 1

if(burst\_time == 0) {

completion\_time = current\_time

turnaround\_time = completion\_time - arrival\_time

waiting\_time = turnaround\_time - burst\_time

response\_time = start\_time - arrival\_time

mark process as completed

completed++

}

}

else {

current\_time++

}

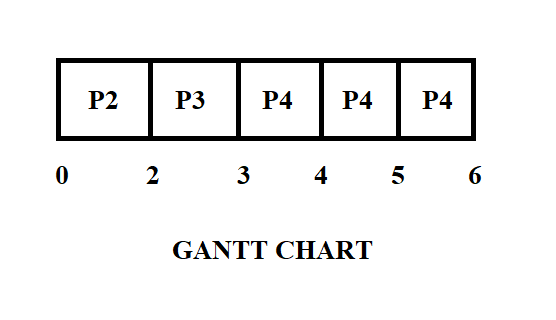
}

TAT = CT - AT

WT = TAT - BT

RT = ST – AT

**Gantt chart:**

****

**Non pre-emptive:**

**Input:**

1. Number of processes
2. Arrival time of each process. If all process arrives at the same time, this can be set to 0 for all processes.
3. Burst time of each process
4. Priority of each process

**Output:**

1. Start Time (ST), Completion Time (CT), Turnaround Time (TAT), Waiting Time (WT) and Response Time (RT) for each process.
2. Average turnaround Time, average waiting time and average response time.
3. Throughput and CPU utilization.

**Algorithm:**

completed = 0

current\_time = 0

while(completed != n) {

find process with maximum priority among process that are in ready queue at current\_time

if(process found) {

start\_time = current\_time

completion\_time = start\_time + burst\_time

turnaround\_time = completion\_time - arrival\_time

waiting\_time = turnaround\_time - burst\_time

response\_time = start\_time - arrival\_time

mark process as completed

completed++

current\_time = completion\_time

}

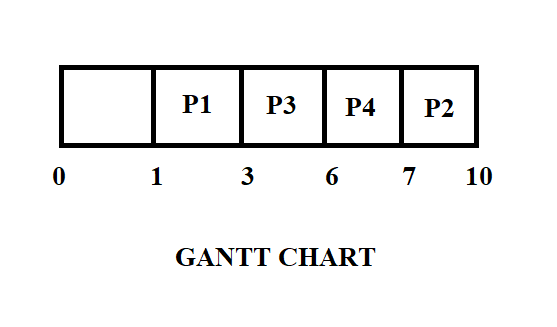
else {

current\_time++

}

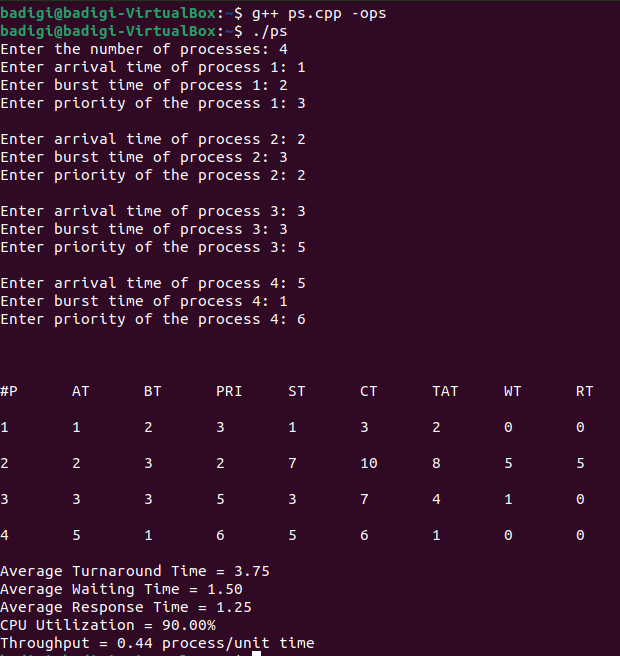
}

**Gantt chart:**

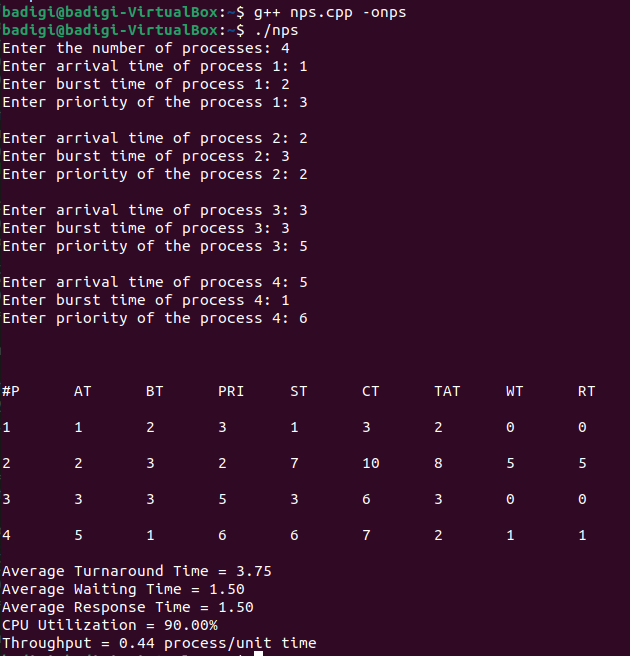
****

**Chapter 6: Output Screenshots**

Pre-emptive:



Non pre-emptive:



**Conclusion**

In this project, we compared Priority Pre-emptive Scheduling and Priority Non pre-emptive Scheduling Algorithms in CPU. The calculation of two case studies show that priority process scheduling has different average response time. For case study 1, we get average turnaround time 3.75, average waiting time 1.50 and average response time 1.25 milliseconds. While in case study 2, we get average turnaround time 3.75, average waiting time 1.50 and average response time 1.50 milliseconds. These two case studies give understanding about the priority process scheduling more thoroughly. The pre-emptive priority scheduling will interrupt the process if the running process has lower priority than the ready process.

**References**

* Operating System Concepts – Abraham Silberschatz . Peter B Galvin . Gerg Gagne(Ninth Edition)
* [www.geeksforgeeks](http://www.geeksforgeeks)
* [www.javatpoint.com](http://www.javatpoint.com)

**Appendix A – Source Code**

**Pre-emptive:**

*#include <iostream>*

*#include <algorithm>*

*#include <iomanip>*

*#include <string.h>*

*using namespace std;*

*struct process {*

*int pid;*

*int arrival\_time;*

*int burst\_time;*

*int priority;*

*int start\_time;*

*int completion\_time;*

*int turnaround\_time;*

*int waiting\_time;*

*int response\_time;*

*};*

*int main() {*

*int n;*

*struct process p[100];*

*float avg\_turnaround\_time;*

*float avg\_waiting\_time;*

*float avg\_response\_time;*

*float cpu\_utilisation;*

*int total\_turnaround\_time = 0;*

*int total\_waiting\_time = 0;*

*int total\_response\_time = 0;*

*int total\_idle\_time = 0;*

*float throughput;*

*int burst\_remaining[100];*

*int is\_completed[100];*

*memset(is\_completed,0,sizeof(is\_completed));*

*cout << setprecision(2) << fixed;*

*cout<<"Enter the number of processes: ";*

*cin>>n;*

*for(int i = 0; i < n; i++) {*

*cout<<"Enter arrival time of process "<<i+1<<": ";*

*cin>>p[i].arrival\_time;*

*cout<<"Enter burst time of process "<<i+1<<": ";*

*cin>>p[i].burst\_time;*

*cout<<"Enter priority of the process "<<i+1<<": ";*

*cin>>p[i].priority;*

*p[i].pid = i+1;*

*burst\_remaining[i] = p[i].burst\_time;*

*cout<<endl;*

*}*

*int current\_time = 0;*

*int completed = 0;*

*int prev = 0;*

*while(completed != n) {*

*int idx = -1;*

*int mx = -1;*

*for(int i = 0; i < n; i++) {*

*if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {*

*if(p[i].priority > mx) {*

*mx = p[i].priority;*

*idx = i;*

*}*

*if(p[i].priority == mx) {*

*if(p[i].arrival\_time < p[idx].arrival\_time) {*

*mx = p[i].priority;*

*idx = i;*

*}*

*}*

*}*

*}*

*if(idx != -1) {*

*if(burst\_remaining[idx] == p[idx].burst\_time) {*

*p[idx].start\_time = current\_time;*

*total\_idle\_time += p[idx].start\_time - prev;*

*}*

*burst\_remaining[idx] -= 1;*

*current\_time++;*

*prev = current\_time;*

*if(burst\_remaining[idx] == 0) {*

*p[idx].completion\_time = current\_time;*

*p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;*

*p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;*

*p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;*

*total\_turnaround\_time += p[idx].turnaround\_time;*

*total\_waiting\_time += p[idx].waiting\_time;*

*total\_response\_time += p[idx].response\_time;*

*is\_completed[idx] = 1;*

*completed++;*

*}*

*}*

*else {*

*current\_time++;*

*}*

*}*

*int min\_arrival\_time = 10000000;*

*int max\_completion\_time = -1;*

*for(int i = 0; i < n; i++) {*

*min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);*

*max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);*

*}*

*avg\_turnaround\_time = (float) total\_turnaround\_time / n;*

*avg\_waiting\_time = (float) total\_waiting\_time / n;*

*avg\_response\_time = (float) total\_response\_time / n;*

*cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;*

*throughput = float(n) / (max\_completion\_time - min\_arrival\_time);*

*cout<<endl<<endl;*

*cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;*

*for(int i = 0; i < n; i++) {*

*cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;*

*}*

*cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;*

*cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;*

*cout<<"Average Response Time = "<<avg\_response\_time<<endl;*

*cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;*

*cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;*

*}*

*/\**

*AT - Arrival Time of the process*

*BT - Burst time of the process*

*ST - Start time of the process*

*CT - Completion time of the process*

*TAT - Turnaround time of the process*

*WT - Waiting time of the process*

*RT - Response time of the process*

*Formulas used:*

*TAT = CT - AT*

*WT = TAT - BT*

*RT = ST - AT*

*\*/}*

**Non pre-emptive:**

*#include <iostream>*

*#include <algorithm>*

*#include <iomanip>*

*#include <string.h>*

*using namespace std;*

*struct process {*

*int pid;*

*int arrival\_time;*

*int burst\_time;*

*int priority;*

*int start\_time;*

*int completion\_time;*

*int turnaround\_time;*

*int waiting\_time;*

*int response\_time;*

*};*

*int main() {*

*int n;*

*struct process p[100];*

*float avg\_turnaround\_time;*

*float avg\_waiting\_time;*

*float avg\_response\_time;*

*float cpu\_utilisation;*

*int total\_turnaround\_time = 0;*

*int total\_waiting\_time = 0;*

*int total\_response\_time = 0;*

*int total\_idle\_time = 0;*

*float throughput;*

*int is\_completed[100];*

*memset(is\_completed,0,sizeof(is\_completed));*

*cout << setprecision(2) << fixed;*

*cout<<"Enter the number of processes: ";*

*cin>>n;*

*for(int i = 0; i < n; i++) {*

*cout<<"Enter arrival time of process "<<i+1<<": ";*

*cin>>p[i].arrival\_time;*

*cout<<"Enter burst time of process "<<i+1<<": ";*

*cin>>p[i].burst\_time;*

*cout<<"Enter priority of the process "<<i+1<<": ";*

*cin>>p[i].priority;*

*p[i].pid = i+1;*

*cout<<endl;*

*}*

*int current\_time = 0;*

*int completed = 0;*

*int prev = 0;*

*while(completed != n) {*

*int idx = -1;*

*int mx = -1;*

*for(int i = 0; i < n; i++) {*

*if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {*

*if(p[i].priority > mx) {*

*mx = p[i].priority;*

*idx = i;*

*}*

*if(p[i].priority == mx) {*

*if(p[i].arrival\_time < p[idx].arrival\_time) {*

*mx = p[i].priority;*

*idx = i;*

*}*

*}*

*}*

*}*

*if(idx != -1) {*

*p[idx].start\_time = current\_time;*

*p[idx].completion\_time = p[idx].start\_time + p[idx].burst\_time;*

*p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;*

*p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;*

*p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;*

*total\_turnaround\_time += p[idx].turnaround\_time;*

*total\_waiting\_time += p[idx].waiting\_time;*

*total\_response\_time += p[idx].response\_time;*

*total\_idle\_time += p[idx].start\_time - prev;*

*is\_completed[idx] = 1;*

*completed++;*

*current\_time = p[idx].completion\_time;*

*prev = current\_time; }*

*else {*

*current\_time++;*

*} }*

*int min\_arrival\_time = 10000000;*

*int max\_completion\_time = -1;*

*for(int i = 0; i < n; i++) {*

*min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);*

*max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);*

*}*

*avg\_turnaround\_time = (float) total\_turnaround\_time / n;*

*avg\_waiting\_time = (float) total\_waiting\_time / n;*

*avg\_response\_time = (float) total\_response\_time / n;*

*cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;*

*throughput = float(n) / (max\_completion\_time - min\_arrival\_time);*

*cout<<endl<<endl; cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;*

*for(int i = 0; i < n; i++) {*

*cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl; }*

*cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;*

*cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;*

*cout<<"Average Response Time = "<<avg\_response\_time<<endl;*

*cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;*

*cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;*

*}*

**Appendix B – GitHub Profile and Link for the Project**

* https://github.com/RA2112704010012