BRAC UNIVERSITY

User Priority Based Efficient CPU Scheduler Algorithm For Real Time Systems

Supervisor Name:
Ms Suraiya Tairin

Author

Meshkat Mahfooz Siam

Student ID: 11201041

This thesis is submitted in partial fulfillment of the requirements for the degree of Bachelor of Science at the Department of Computer Science and Engineering

April, 2017

Declaration of Authorship

I, MeshkatMahfooz Siam, declare that the thesis entitled ***'User Priority Based Efficient CPU Scheduler Algorithm For Real Time Systems'*** and the work presented in it is my own.

I confirm that-

- This work was done wholly or mainly while in candidature for Bachelor of Science degree in Computer Science and Engineering at BRAC University.
- I have clearly attributed the original authors wherever I consulted the published work of others.
- I have declared the source wherever I quoted from the work of others. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources which helped me to complete this thesis.

_____

MeshkatMahfooz Siam

Date: _____

Certificate of Approval

This undergraduate thesis report entitled ‘‘***Designing a User Priority Based CPU Scheduler”*** submitted to the department ofComputer Science and Engineering in partial fulfillment of the requirements for the Bachelor of Science degree, has been approved by the panel of examiners.

Signature of Supervisor

_____

Miss SuraiyaTairin

Assistant Professor

Dept. of CSE, BRAC University

Date: _____

Abstract

Operating systems at present face a large workload and are restricted by limited processing power. This may lead to lower than expected performance in practice. To counter this the operating system can apply an organizing algorithm to execute the required processes in strategically selected order. This activity is known as scheduling.

Scheduling allows the computer to operate in an efficient manner and achieve the targets set for it. On this way schedulers drastically improve CPU performance. Due to its importance, quite a few scheduler algorithms have popped up over the years and research into scheduling remains a hot topic in computing.

In this thesis attempts have been made to take the currently available scheduling algorithms and mold them in a planned manner into a procedure where the whole is greater than the sum of the components. As such a hybrid scheduler, named the '***User Priority Based Efficient CPU Scheduler Algorithm For Real Time Systems***' is proposed. The suggested scheduler is subsequently designed and implemented in a simulation environment. The performance metrics of this complex algorithm are then measured. These values are at that point compared with corresponding values found for the traditional algorithms to establish standards of performance of the novel scheduler.

Dedications

To my family and friends.

Acknowledgements

I am thankful to my respected supervisor Ms. SuraiyaTairin for her guidance and support in this thesis ever since its inception. Her mentorship has not just been limited to the confines of academic pursuits and I owe a lot to her for encouraging me in difficult times.

I also owe a debt of gratitude to my beloved seniors and juniors who have consistently helped me in my endeavors.

At this point I would like to acknowledge the aid of my friends who have gone out of their way many a time to perhaps teach me a new way to code or to keep me motivated.

Finally I would like to thank mywonderful family without whose support I would never be where I am now.

CONTENTS

List of Figures

Abbreviations

CPU	**Central Processing Unit**
OS	**Operating System**
FCFS	**First Come First Serve**
SJF	**Shortest Job First**
PS	**Priority Scheduling**
RR	**Round Robin**

Chapter 1

Introduction

1.1 Overview

Operating system (OS) is software that carries out the essential processes which set up a system and keep it running. The applications run by the OS essentially support the actual computing that the user wants to carry out.

As a result, the operating system is generally loaded into memory at the beginning of computer operation and a computer cannot function efficiently without it.

Initially, users required OSs as a way to handle complex input/output operations. The main objective of these operations was to handle different programs running in tandem. The usage of memory off of read only and random access memory was considerably complex and required a specifically written program to manage it.

Thus the Disk Operating System (DOS) was born. The name clearly indicates that Oss were originally designed to deal with communication between various types of disk drives.

Today we use our powerful computing devices to run tens of applications or programs at a time. We may have become rather conveniently used to this luxury but this is not a luxury our predecessors could afford. If we go back to the early days of computing, we observe that the first few generations of computers could run only one program at a time. For example, even in the late 1980's one would not be able to listen to music while leisurely going through the day's news on the internet. In fact, they would be very hard pressed to even imagine themselves doing that. As such, early computers did not have the capacity to cater to our voracious present day demands.

Now for a CPU to achieve this amazing feat of running multiple processes at a time, it must 'juggle' its application very delicately. That is, it must have a way of 'knowing' and 'planning' exactly how it will accomplish its duties. Only by efficiently allocating its limited resources can a computer meet the extravagant demands of modern day computing.

As such an operating system has to assess the requirements of a process and then assign its resources to make the most efficient possible itinerary of sorts to carry out the operation. This scheduling has to adhere to a number of specific goals.

Firstly is has to be fair; secondly, it cannot starve i.e. deliberately delay any single process from processing time; thirdly, it must make the most efficient use of the processors' time and finally it must have low overhead. Additionally, it may have to account for different levels of priority and various deadlines for different programs.

Throughout the history of computing, scheduling has been the focus of intensive research and consequently many different algorithms have been proposed and implemented for this purpose.

Scheduling has in fact been a key player in the utilization of multiprocessor systems, more prominently in multithreaded applications. It has also been integrally associated with real-time allocation of tasks. This is quite intuitive, as the main purpose of scheduling is to handle multiple actions efficiently.

In such situations, the processor has a few tasks it needs to handle at any one time. These tasks or processes exist simultaneously in its memory. Now, the general expected outcome is to have the most efficient usage of resources. To that end, each process alternates between using the processor and waiting for its turn to be processed. The processor can only do one task at a time and so it needs a proper schedule to effectively carry out its operation and meet operational demands.

1.2 Research Motivation

In the early days of computing, when the number and complexity of tasks had been smaller, simple scheduling algorithms were enough to carry out computations with acceptable efficiency. However, the computational power and demands of computer systems have both increased exponentially over the years in correlated fashion. Now if the electrical power required to execute each process increased even linearly, the power consumed would severely limit our computational capacity. Therefore various means of achieving supreme computational efficiency have been devised to counteract this predicament. Scheduling is perhaps the most important of these techniques. This has made possible the exponential increase in computational efficiency of computers over the years.

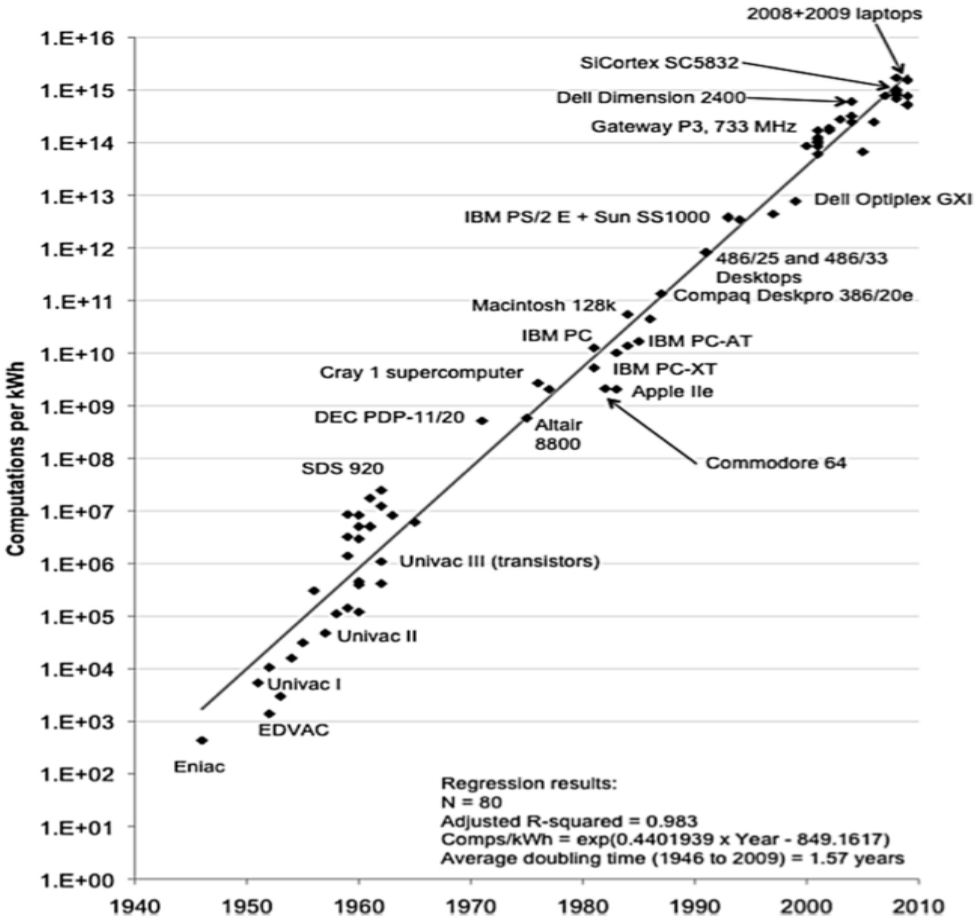


Fig 1.1: Exponentially increasing computational capacity over time (computations per second) [1]

This does not mean we can stop trying improve the computational efficiency of our devices. In fact we have to put continuous effort into making our computers work more smoothly.

That is why I have chosen to do my thesis on a new hybrid algorithm for scheduling.

1.3 Objective

The objective of my research is to analyze existing scheduling algorithms and use this analysis to come up with a new algorithm which provides generally favorable performance.

In order to test the efficiency of the algorithm, I have designed and implemented a system which is able to evaluate the performance of the novel algorithm and compare this with existing mainstream scheduling algorithms.

For the purpose of examination and analysis the user initially specifies each process along with its information, such as arrival times and burst time; then the time for execution of the processes can be calculated and relative performances compared.

Thus the proposal and implementation of an algorithm to compute and compare scheduling of processes and thereby incremental improvement of the existing algorithms is the main purpose of this thesis.

1.4 Organization of the Thesis

In chapter 2, I present the background of my thesis along with description of terminology. Along the way I discuss the various formulations that are necessary to my work.

Afterwards in chapter 3, I discuss the simulation setup of my thesis and analyze the computational efficiency associated performance. Later in the chapter I establish the comparative improvement of the algorithm compared to the existing algorithms.

Then in chapter 4 I present results and discussion on operating system scheduling under varying conditions. I proceed to compare the results for different process types and construct a performance based hierarchy.

Finally in chapter 5, I revisit the goals, motivations and results of our thesis. I scrutinize these factors to evaluate the effectiveness of my efforts. Going further, I consider the future prospects of proposed algorithm and finish with an overview of additional opportunities of study on the topic.

1.5 Chapter Summary

Optimization of processor task scheduling is an important aspect of computational efficiency and CPU performance. New algorithms for this job have to be initiated and improved to further augment the current situation. I have aimed to execute that in my thesis. This chapter denotes the necessity, motivations and organization of

my thesis work. The details of my work will be discussed thoroughly in the upcoming chapters.

Chapter 2

Background

2.1 CPU Scheduling

CPU scheduling is an importantprocedurein computing. It allows one process to use the CPU while the implementation of one or more processes is on holdi.ein waiting state.This is in lieu of the unavailability of computational resources like Input/Output.In this way scheduling helps to utilize the complete potential of the CPU. Thus it can be said that theultimate upshot of CPU scheduling is a more efficient system.

In computing, scheduling is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as threads, processes or data flows.These variety of elements may be then scheduled into hardware. Hardware itself can be of various types. For example- network links, processors etc.

Unsurprisingly, the thing that carries out the scheduling activity is called the scheduler. Schedulers are generallyemployedwith the following implications in mind:

- To keep all computer resources busy (i.e. load balancing)
- To let numerous users share system resources effectually
- To accomplish a specific quality of service

As one can observe, the processor would not be able to handle its job very efficiently without proper scheduling. Thus scheduling is, unequivocally, essential to computation itself, and as such it is an inherentportion of the implementationprototype of a computer system; the notion of scheduling ultimately leads to computers being able to do more than one task with a single central processing unit (CPU). This phenomenon is more popularly known as multitasking.

A scheduler may aim at one of many goals, for instance,

- maximizing *throughput* (the entirequantity of work finished per unit of time)
- minimizing *response time* (periodsince work becoming facilitated until the first point it instigates execution on resources)
- minimizing *latency* (the interval between work becoming enabled and its subsequent conclusion)
- Maximizing *fairness* (identical CPU commitment to each process, or more generally appropriate times conferring to the priority and load of each process).

However, these goals are such that, trying to achieve one generally makes another difficult. That is they are generally in conflict. In practice, these goals often are mutually dependent and oppose each other (e.g. throughput versus latency), making it rather difficult to achieve all at the same time. Thus a scheduler normally implements anappropriateconciliation between two opposing goals. Preference is specified to any one of the concerns declaredoverhead, depending upon the user's requirements and purposes.

In spontaneous situations, such as embedded systems for automatic control in manufacturing (for example a conveyor line, production pipeline etc.), the scheduler correspondingly must guarantee that processes can meet deadlines; this is critical for assuring that the system remains stable. Scheduled jobs can also be dispersed to distant devices through a network and controlled through an administrative back end.

2.2 Scheduling Criteria

As we have discussed earlier, scheduling algorithms can have many different goals. Moreover, these goals may be (and in practical situations generally are) in conflict with each other. Therefore the scheduling performance may be characterized on the basis of many different criteria. Most of the widely used criteria are described below:

- **CPU utilization**

  To ensure maximum performance or utilization of the processor, a CPU should be working most of the time(in the ideal case 100% of the time). Making an

allowance for a real system, CPU usage should vary from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput**

  Throughput is the total number of processes completed per unit time.It can be also thought of as the total amount of work done in a certain interval of time. This canbe from 10/second to 1/hour depending on the specific processes.

- **Turnaround time**

  This is the time taken to perform a specific process, that is the time period from the time of submission of the procedure to the time of completion of the application (i.e. Wall clock time).

- **Waiting time**

  The total time spent ahead of being processed in the ready queue.That is, amount of time a process has been waiting in the ready queue to procure control on the CPU.

- **Load average**

  Load Average is the average quantity of processes existing in the ready queue in the offing for their opportunity to get into the CPU. This variable gives a general idea as to how loaded the device is.

- **Response time**

  Response time is the amount of time it takes from the instant a request was acquiescedtill the first response is produced. It is important to remember that it is the time before the first response and not the completion of process execution(concluding response).

- **CPU Bursts**

CPU Burst is a very important concept in the study and analysis of scheduling. It is theamount of time the process has control of the processor before it is no longer prepared. CPU Bursts are of two kinds:

(i) long bursts –cases where the process is CPU bound (i.e. array work);
(ii) short bursts – cases where the process I/O bound

Although any of the aforementioned criteria can be maximized, in general CPU utilization and Throughput are capitalized on and other aspects are reduced for suitable optimization.

2.3 Types of Schedulers

Schedulers can be typed into three different categories in terms of the length of action time. They are described below.

(1) Long-term scheduler: The long-term scheduler is the initial scheduler

- selects process and loads it into memory for execution
- decides which process to start based on order and priority
- not used in timesharing systems

(2) Medium-term scheduler:

- schedule processes based on resources they require (memory, I/O)
- suspend processes for which adequate resources are not currently available
- commonly, main memory is the limiting resource and the memory manager acts as the medium term scheduler

(3) Short-term scheduler (CPU scheduler):

- shares the processor among the ready (run able) processes
- crucial the short-term scheduler be **very** fast -- a fast decision is more important than an excellent decision
- if a process requires a resource (or input) that it does not have, it is removed from the ready list (and enters the WAITING state)
- uses a data structure called a **ready list** to identify ready processes
  Started in response to a clock interrupt or when a process is suspended or exit.

2.4 Scheduling Algorithms

There are five major scheduling algorithms:

(1) First Come First Serve(FCFS) Scheduling
(2) Shortest-Job-First(SJF) Scheduling
(3) Priority Scheduling
(4) Round Robin(RR) Scheduling
(5) Multilevel Queue Scheduling

2.4.1 First Come First Serve (FCFS) Scheduling

The other names of this algorithm are:

1. First-In-First-Out: FIFO
2. Run to Completion
3. Run- Until- Done

First Come First Serve is a working framework for prepare, planning calculation and a system directing administration instrument that consequently executes lined demand and procedures by the request of their entry .With first start things out served, thefirst things out are taken care of first; the following solicitation line will be executed once the one preceding it is finished .

The main come, initially served (ordinarily alluded as FIFO: First in, First Out) Process Scheduling calculation is the least complex Process Scheduling calculation. It is one of the bunch frameworks. It is once in a while utilized as a part of current working frameworks, however it is some of the time in the other planning frameworks. To begin with Come First Serve likewise gives an effective, basic and mistake free process planning calculation that spares significant CPU assets. It utilizes no preemptive planning for which a procedure is naturally lined and preparing happens as indicated by an approaching solicitation or process arrange. FCFS gets its idea from genuine client benefit.

The FCFS carries on or acts like a typical line, For example; a line in the silver screen, the principal individual to arrive is the main individual to be managed. In the event that one individual in the line goes and concludes that they have overlooked something then they need to backpedal through. This additionally

happens to the working framework, on the off chance that one procedure touches base at the OS (First In) that is the one that will be managed (First Out).

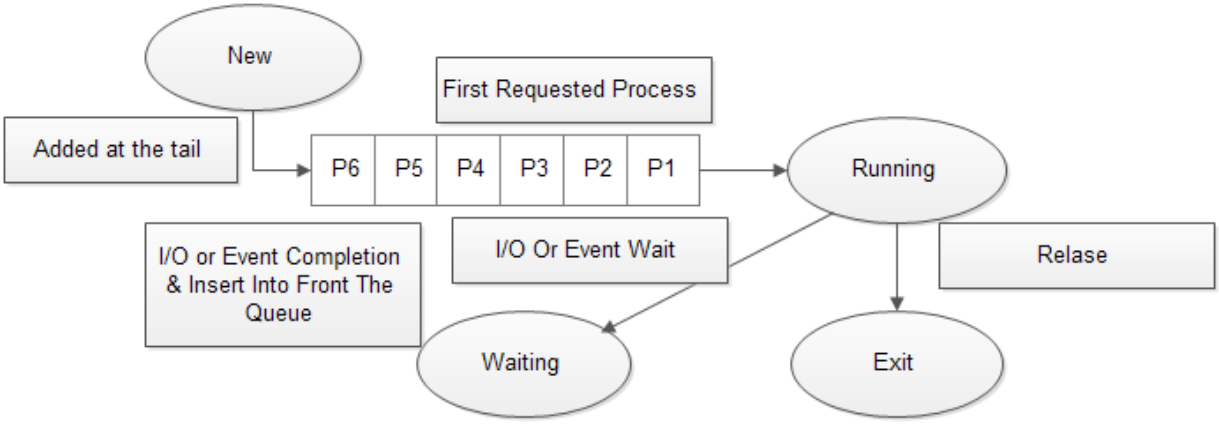


Fig 2.1: FIFO Scheduling

Here is an example of the FCFS process. Suppose there are three processes in a queue: D1, D2 and D3. D1 is placed in the processing register with a waiting time of zero seconds and 10 seconds for complete processing. The next process, D2, must wait 10 seconds and is placed in the processing cycle until D1 is processed. Assuming that D2 will take 15 seconds to complete, the final process, D3, must wait 25 seconds to be processed. FCFS may not be the fastest process scheduling algorithm, as it does not check for priorities associated with processes. These priorities may depend on the processes' individual execution times.

First-Come-First-Served algorithm processes are dispatched according to their arrival time on the ready queue. Being a non-preemptive discipline, once a process has a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait.

FCFS is more predictable than most of the other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write

and to understand. One of the major draw-back of this scheme is that the average time is often quite long.

The First-Come-First-Served algorithm is seldom utilized as a mainarrangement in contemporary operating systems but it is frequentlyimplanted within other structures.

Diagrammatical expression of the FCFS

This illustration is a graphical depiction of the first come first serve procedure in the batch system. Here there are three processes waiting to be implemented in a consecutive order.

Process	Duration	Order	Arrival Time
D1	24	1	0
D1	3	2	0
D3	4	3	0

Fig 2.2: FIFO Scheduling Diagram

P1 waiting time 0
P2 waiting time 24
P3 waiting time 27
= The average waiting time is;
(0+24+27)/3= 17

ADVANTAGES AND DISADVANTAGES OF THE FCFS

Advantages:

- FCFS is simple
- The FCFS is easy to understand
- FCFS is auto explanatoryi.e.it literally means first come, first served
- FCFS is reasonable/ not unfair

Disadvantages:

- Thistechnique is non preemptive, which means that the process will not executetill it ends.
- Because of this non preemptive scheduling, short processes which are at the back of the queue have to wait for the long process at the front to finish.
- One major disadvantage of FCFS is the convoy effect.This refers to the phenomenon where short processesget stuck waiting for lengthy processes.
- Also, in FCFS, waiting time disadvantageously depends on the order of arrival.

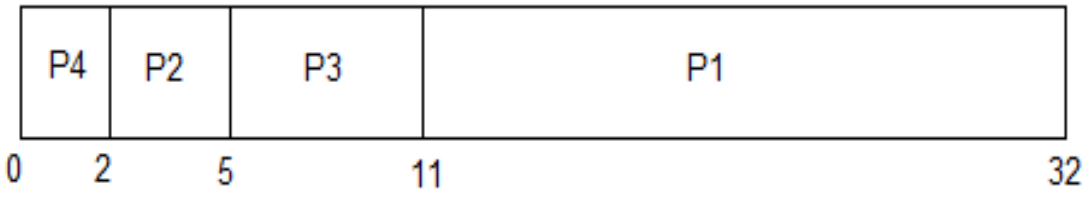
2.4.2 Shortest-Job-First(SJF) Scheduling

- SJF is the best approach to minimize waiting time.
- Definiteperiod taken by the process needs to bepreviouslyacknowledged to the processor.
- However this is an idealization, although it can be implemented in an approximate manner, the actual SJF according to the definition cannot be implemented.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



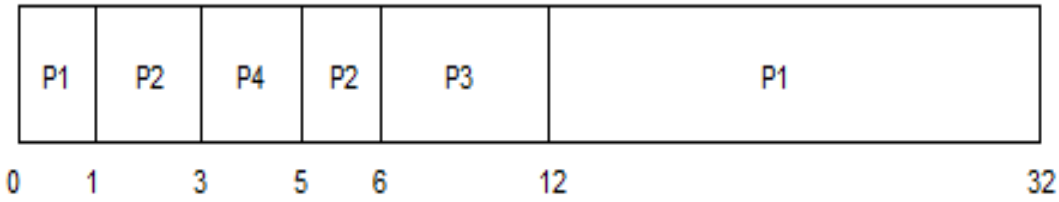
Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

Fig 2.3: SJF Scheduling

In the Preemptive SJF Scheduling, works are taken within a ready queue as they reach, but as soon a process with short burst time arrives, the current process is preempted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

Fig 2.4: SJF Scheduling Diagram

2.4.3Priority Scheduling

- In priority scheduling, a priority, i.e. a specific amount or weightage of importance is allocated for every process.
- The weightage or priority dictates that the process with the highest priority is executed first and so forth.
- To break ties, processes with equal priority are completed on FCFS basis.
- Priority can be decided based on memory necessities, time specifications or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

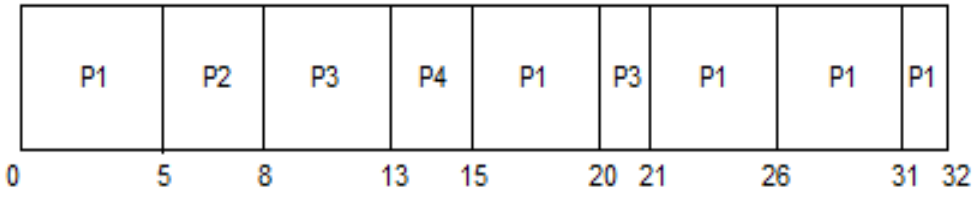
Fig 2.5: Priority Scheduling Diagram

2.4.4 Round Robin(RR) Scheduling

- In the round robin method, a fixed amount of time is allocated to each process and each gets one's turn to use the processor. Thetime allotted to each process for execution is called quantum.
- Round Robin Scheduling is unique in the sense that in this scheme, once a process is performed for given time period that process is preempted and the next process executes for the given time period, i.e. quantum.
- The process called context switching is used to save conditions of preempted processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Fig 2.6: RR Scheduling Diagram

2.4.5 Multilevel Queue Scheduling

- In this method, numerous queues are sustained for processes.
- And every queue may be given its very own scheduling algorithm. As such multiple algorithms for scheduling can be implemented.
- Afterwards priorities are allocated to all the queues so that each can be executed in a certain order. In this way different scheduling techniques can be implemented for different queues and overall efficiency and speed of scheduling can be achieved.

2.5 Chapter Summary

In this chapter we have discussed the basis of the thesis, namely the process of scheduling.

Scheduling is the process by which multiple jobs are handled by the processor. It is of critical importance when it comes to handling many tasks at once.

Scheduling is contingent upon various factors. In different circumstances, different systems parameters need to me maximized and hence different types of scheduling are required for each.

The main goals of scheduling are to maximize throughput and fairness, while minimizing response time and latency.

However these goals cannot be implemented all at once as they are in conflict with each other. Therefore it is up to the user to decide which scheduling algorithm to use to achieve his/her ends.

Scheduling can be of various types in terms of tie duration. They are long-term, medium-term, and short-term scheduling.

According to real-life practical situations, to achieve maximum efficiency, factors like CPU burst time, wait time, response time, turnaround time etc. have to be taken into account to employ proper useful scheduling. Various algorithms have been developed to make the best use of scheduling resources.

The different algorithms are First Come First Serve (FCFS), Shortest Job First (SJF), Priority, Round Robin (RR) and Multilevel Queue (MLQ) scheduling.

FCFS is self-explanatory and takes the processes based on arrival time. It is non-preemptive and keeps doing any task it gets. It is simple, but it increases the wait time of the system. It may also disrupt CPU performance by withholding the execution of important tasks.

Not unlike FCFS, SJF is also self-explanatory. It simply takes the shortest job first and executes it. This way it decreases wait time for all the tasks at hand. However, wait time is not the only performance metric here.

If there is an important task which has a relatively long CPU burst time, it automatically gets dropped into the low priority part of the queue and its execution s delayed. This can have a negative effect on CPU performance. If there are two or more jobs with equal burst times, then the shortest job first algorithm switches to FCFS mode.

The Priority based scheduling algorithm is another evolution of scheduling algorithms. In this system, processes are also assigned a priority along with their process ID and burst time. This way the processor knows which task to finish first. So even when a large task comes along the processor may pay attention to it first if it has a high priority assigned to it. This way it becomes easier for the processor to do the important things first.

Thus priority scheduling improves the performance of the computer considerably. However this improvement comes at a cost as the added computational complexity adds to the execution time of the tasks. This way, waiting time for tasks increases as well.

Round Robin Scheduling is the next step in the development of scheduling algorithms. In this scheme all tasks are assigned a preselected quota of time to be processed, otherwise known as quantum. After one task executes for its quantum, it is preempted and the processor moves on to the next task. CPU resources get evenly distributed this way.

Round Robin and priority can be mixed effectively to assign more quanta to higher priority tasks. This breakdown of tasks can significantly improve CPU performance.

Finally, Multilevel Queue Scheduling is the process where the process queues themselves are divided into different arrays and dealt with using different algorithms. These way appropriate algorithms can be applied to suitable jobs to maximize CPU performance.

Chapter 3

User Priority Based CPU Efficient CPU Scheduler Algorithm for Real Time Systems

In the previous chapters we have discussed the basic idea behind scheduling and its necessities. We have also discussed the motivations behind this thesis.

Furthermore, we have gone on to discuss the theory and algorithms behind scheduling as well.

In the first chapter we have shown why the study of scheduling is important and where the areas of improvement are. We have followed that lead and extensively studied scheduling over the course of the second chapter.

As such we have already discussed the basic types of scheduling algorithms that are already in use at the moment. In this thesis, a new scheduling algorithm is proposed and tested against the other algorithms available at random jobs.

Coming up with this hybrid algorithm and calculating and analyzing its performance are to be the main goals of the thesis. In order to do so however, the system in question has to be described first.

The design of the system revolves around the creation of random processes of different priorities and burst times. The priorities and burst times of each are taken as inputs and these parameters are used to simulate the designed hybrid user-centric scheduler, as well as with a conventional scheduler.

The performance of both in terms of wait time and execution time as well as utilization of other resources is measured. These measurements provide key insight into the scheduling algorithms and their strengths and weaknesses.

3.1 Overview

The scheduler that is proposed in this thesis is called the User Priority Based CPU Scheduler.

It is a hybrid of all the algorithms used in scheduling.

The main purpose of this scheduler is to take the best of everything and make maximum utilization of CPU resources while simultaneously providing the best possible performance.

It is termed as user centric for this very reason. The scheduler exhibits strategic exploitation FCFS, SJF, Priority, Round Robin and Multilevel Queue.

The algorithm is designed around a code base that can implement and compare all the different systems used in scheduling.

It does so by preparing a stream of random data and using each scheme to deal with that data.

The data is in fact used to model different processes which have to be run.

The data is sorted into three queues residing in three different threads to ensure implementation of all the algorithms and to guarantee maximum utilization of resources.

The three stacks or queues store the ID's and priorities of the processes as well as other relevant data. The three queues are the tertiary or initiation stack, the secondary or job queue (also referred to hence forth as the filtered process queue) and finally the most important primary queue or the processing queue.

The processes can be readily moved from one stack to another depending on the needs of the system. The various algorithms applied confirm that best scheduling performance can be achieved.

A brief description of the system is given in the next section.

3.2 System Description

The user-centric scheduler has 3 main stacks. However, before the stacks come into play, random data on the simulation processes have to be generated. These data are generated using a random data generator with uniform distribution of values.

Therefore the system can be simply described as a block diagram with the following elements:

i. Process Data Generator
ii. Initiation Stack
iii. Secondary Queue (Job Queue)
iv. Primary Queue (Processing Queue)

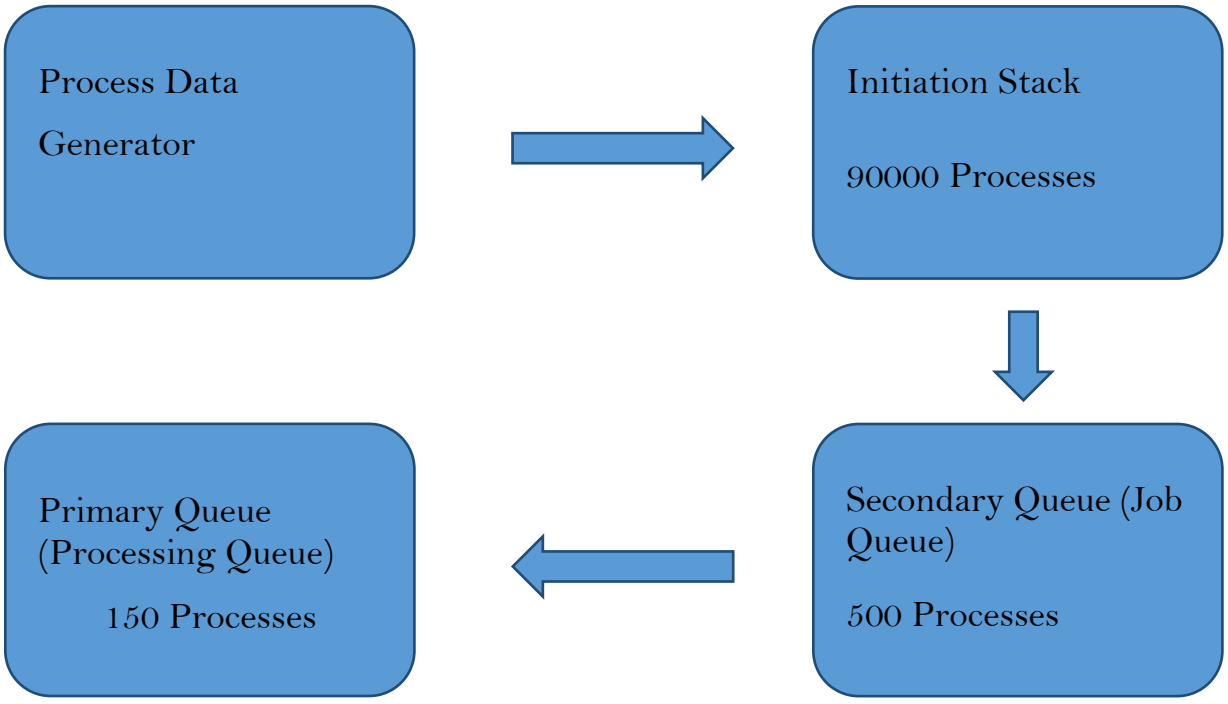


Fig 3.1: Flow Chart of the System

(i) Process Generator

The process generator is a random number generator which generates the necessary parameters for each process. The distribution of the generated numbers follows the exponential random variable pattern of the actual empirical values of process parameters, thus making the study condition similar to the initial conditions of the experiment.

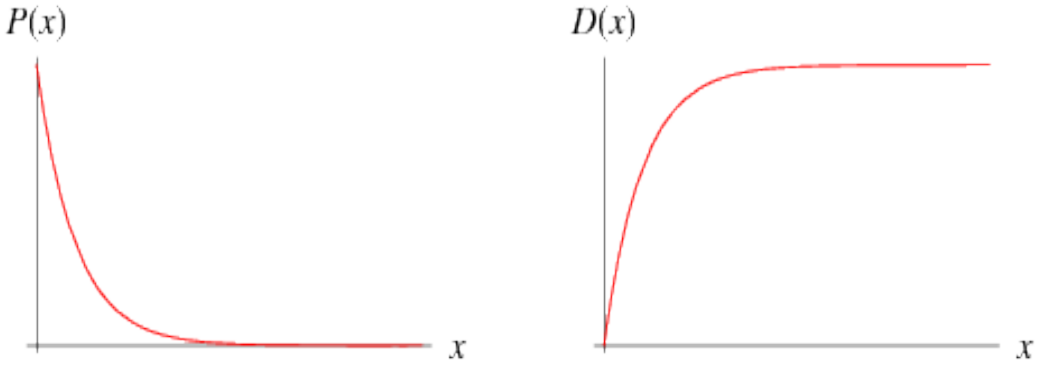


Fig 3.2: Probability and Cumulative Distributions of Burst Time for Generated Processes

This generator provides all the data for the actual simulator which simulates different scheduling algorithms.

(ii) Initiation Stack

The initiation stack is the first stack among the three queues used by the algorithm.

This stack holds data for 90000 processes. This is the total number of processes we use in our experiment. The number is large enough to simulate for larger and smaller numbers or processes.

In this way statistical significance can be realized. Furthermore this stack can be pushed or popped anytime if necessary.

Because we need the three stacks to be different from each other, this stack has its own separate thread. This makes it very versatile.

A pictorial representation of the initiation stack is given below.

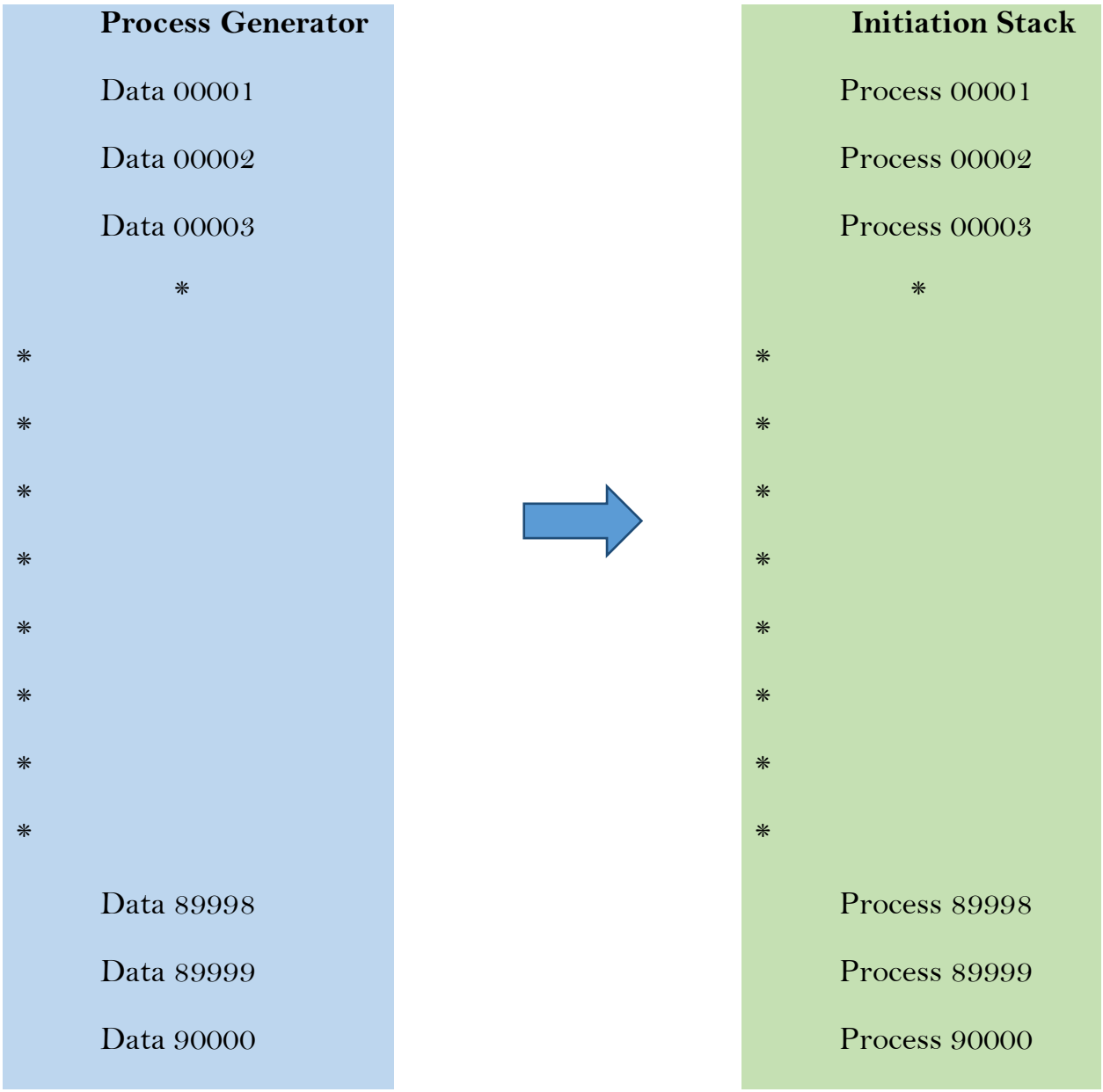


Fig 3.3: Process Generator to Initiation Stack

As is clear from the explanation whenever the initiation stack requires new processes, the data or process generator generates it for the process itself. This data reaches the initiation stack and is interpreted as a new process.

(iii) Secondary Queue/ Job Queue/ Job Stack

This is the next stack in our execution line. It is not the final stack, i.e. the stack from which the processes are sent to be executed. Rather, it is the stack which holds the processes right before they are sent to be executed. It acts as a buffer between the initiation and primary queues. This is very important for the proper execution of the scheduling algorithm.

The first thing about the secondary stack is that it is much less densely populated than the previous initiation stack. This makes it enormously quicker to handle than the initial stack itself. Therefore this is effectively the actual reservoir for working process data for the CPU, noticeably the initiation stack is large and cumbersome when faced with fast calculations, which are a vital part of CPU scheduling. Therefore, the secondary queue is rather more suited for this action than the initiation stack.

It serves the important purpose of holding the highest priority jobs or tasks that the processor has to handle at any given real-time moment. Once this stack is initially filled, the processor is ready to fill the primary task handling queue or the processing queue.

The top 500 high priority tasks are then shifted from the secondary stack to the primary queue. These jobs automatically go into execution on a round robin basis.

The 500 empty seats in the job queue are then filled up by 500 highest priority jobs in the initiation stack.

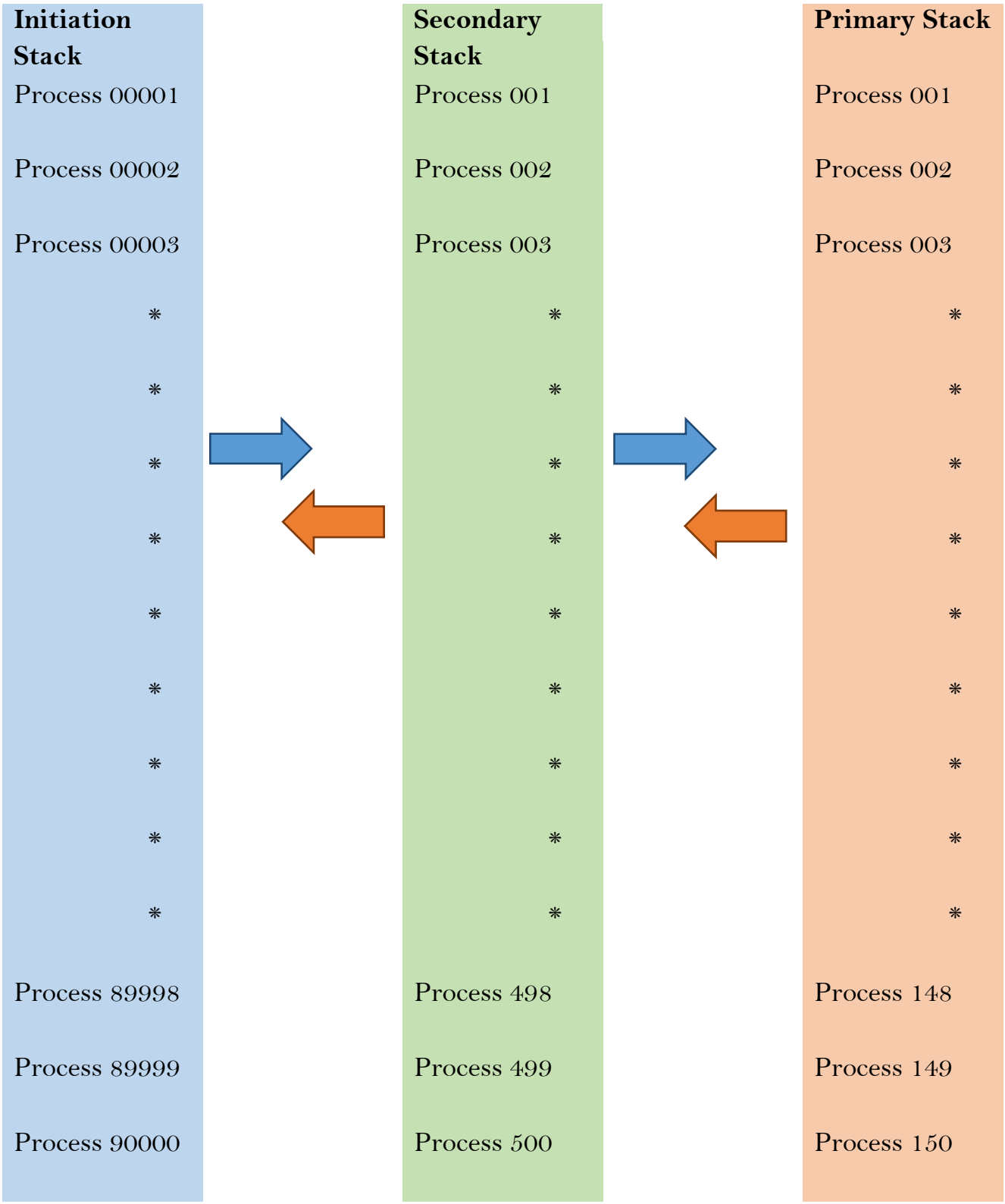


Fig 3.4: Exchange of Process ID's through secondary stack.

(iv) Primary Queue/ Processing Queue/ Primary Stack

The primary queue is the final stack in our execution line. This is where all the processes are sent to be executed. This stack is the final step from process ID arrival at the CPU andthe process being executed according to order. It acts as a buffer between the initiation and primary queues. This is the most important queue for the proper execution of the scheduling algorithm.

The processes selected based on burst time and priority is executed here.

The execution is carried out in the round robin format. Even so higher priority jobs get more rounds in the round robin format. But this happens only in the case when there are multiple jobs to handle that is the CPU is overburdened for resources. The round robin format is ideal for multitasking. In fact it is the ultimate multitasking algorithm.

However, when the CPU is idle, i.e. it does not have many jobs to handle, the primary queue executes its processes in the SJF or Shortest Job First fashion. This action minimizes wait time and increases CPU performance manifold, making it work faster and appear smooth.

The application of the round robin technique makes sure that every job is handled and the processor does not fall into deadlock. Somehow managing to minimize average wait time and CPU loading while maximizing efficiency is the salient upshot of using two different algorithms in the primary stack.

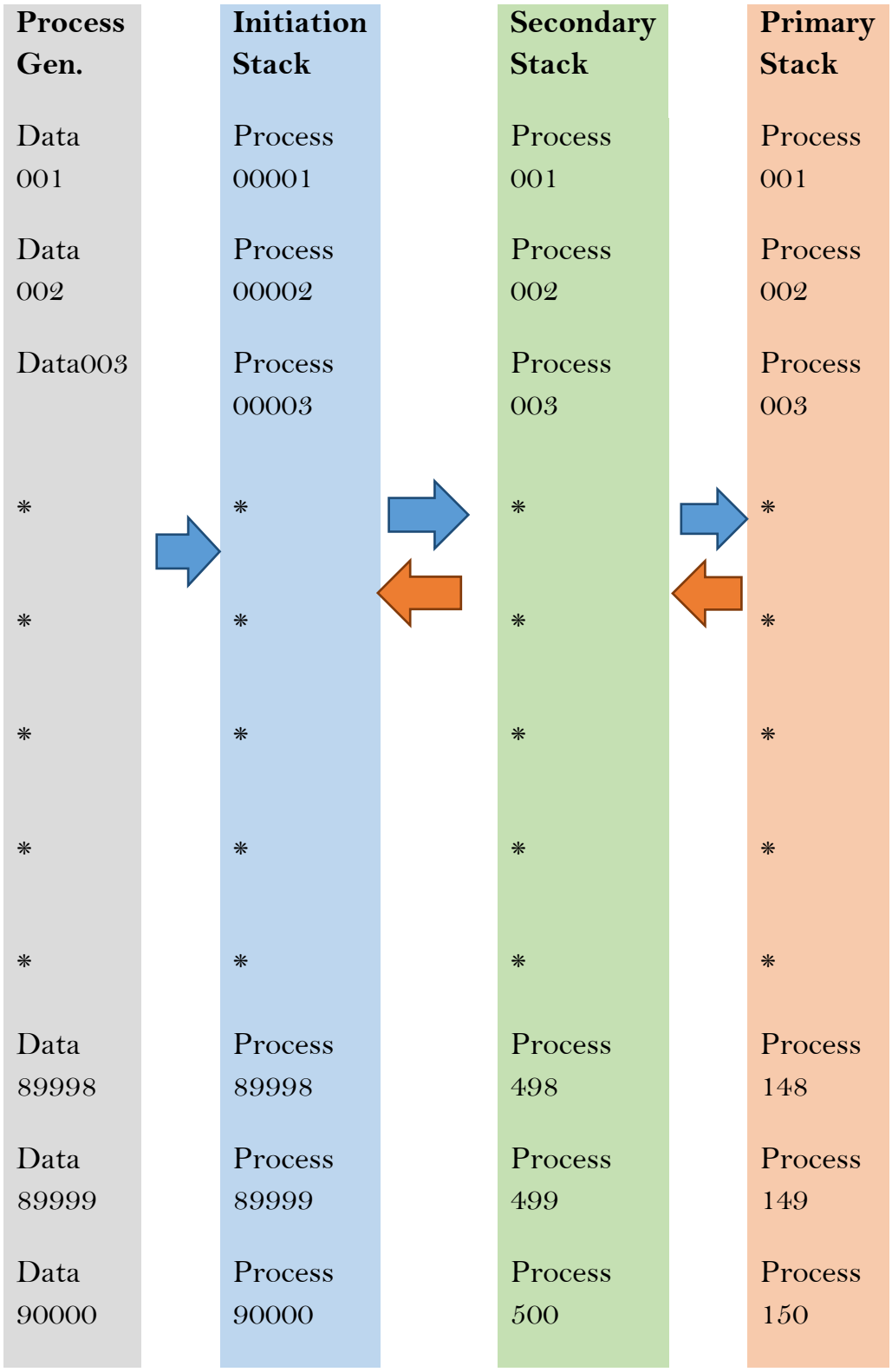


Fig 3.5: Interchange of process data between all the components of the user-centric scheduler

3.3 System Specifications

The system works by firstly following the rules below:

1. The tasks are data generated from a random stream by the data generator.
2. The data are initiated in the initiation stack as different tasks.
3. Both these stacks carry 90,000 tasks or processes.
4. Initial priorities of the tasks are set to 2. Maximum bound of priority is infinity and minimum is 1. Priority can be also set manually.
5. Once the initiation stack is filled, the top 500 priority processes are siphoned and put into the secondary queue.
6. These processes in turn, are sorted by priority and the top 150 are put into the primary stack.
7. The primary stack keeps executing automatically when it has any jobs contained within it.
8. The primary stack executes in round robin format for multitasking (many tasks) and SJF format for idle mode of the PC.
9. Once a job is done in the primary stack, a new job takes its place from the secondary stack.
10. In turn, a job from the initiation stack fills the empty space created in the secondary stack.
11. Subsequently the data stack pushes a new process into the initiation stack.

Exceptions:

1. If the priority of a new job (in the initiation stack) is 1.5 times greater than the currently executing process, the system executes an interrupt and the high-priority job is immediately pushed to execution.

2. The lowest priority job in the primary stack is pushed into the secondary stack.
3. Similarly, the lowest priority job in the secondary stack is pushed back into the initiation stack.

The following table illustrates the use of different algorithms in the different queues of the operation.

Data	Initiation	Secondary	Primary
None	1, FCFS (First Come First Serve)	1.Priority (Whenever different Priorities are present) 2. FCFS (When all tasks have same priority)	1. Round Robin (Whenever there is a multitude of tasks) 2. Shortest Job First (When there are not too many tasks to handle)

Fig 3.6: Algorithms used in various queues of the user-centric scheduler.

3.4 Chapter Summary

This chapter has been about describing the scheduler that has been proposed in this thesis.

The User Priority Based CPU Scheduler is versatile and robust scheduler which can be used to handle different types of loads on the CPU.

The scheduler consists of four main stacks. The first, data generator is not actually a stack but a data stream which generates data that results into processes in the initiation stack.

All of the data is assigned priorities in the data stack and are treated at the initiation stack as they arrive (First Come First Serve).

Afterwards, the data is taken on the basis of priority into the secondary or job stack. This stack may send data to both the initiation and primary stacks. The main purpose of this stack is to lessen the number of jobs to handle and make the scheduler leaner in terms of computing power consumption.

The execution stack of the scheduler is known as the primary queue or the processing queue. It handles priority jobs on a round robin basis. It switches to shortest job first (SJF) mode when the system is idle. This is a conservationist minimalistic and yet powerful scheme.

In case a rather high priority job is received at the initiation stack, that task is directly sent to execution, making the system less prone to becoming stuck. The SJF algorithm used in the processing stack significantly reduces waiting time as well.

Overall the system is basically a strategically developed hybrid of the available schedulers. This is all done in aneffort to get the best of all the worlds into one package.

Chapter 4

Results

So far we have touched upon the topics of scheduling, why it necessary and all of its different aspects.

In the first chapter we discussed the motivations behind conducting the thesis on the topic of scheduling. The objectives of the thesis were also clarified along with the organization of this narrative.

Then in the second chapter, the thesis topic, namely scheduling was described in detail. Scheduling is a topic that has been extensively researched and given significant important. Therefore the important factors relating to and from scheduling, as well as the different criteria to base the performance of schedulers on- are topics that require attention before delving into the design of new algorithms for the task.

Therefore, that chapter was dedicated to framing the essential information required to study scheduling and arranging them in a top-down manner so as to make the analysis and formulation of new algorithms simpler.

After the second chapter, we were ready to move on to the discussion on the thesis itself. This was done in the third chapter, 'System Model'. The new algorithm proposed, named the User Priority Based CPU Scheduler, was at first shown as a summation of its main components and then the components themselves were analyzed and the motivations behind making them the way they were was explained in detail.

Thus the first three chapters dealt with the initial motivation, analysis, and proposed solution on the topic of scheduling. However the main part of the thesis still remains to be discussed. All the analysis and thought behind the new idea would go in vain but for its field test in practical conditions, with its performance measured and thoroughly checked against the standards set by the current systems.

In this chapter, the results of the thesis are discussed and their implications are analyzed. From this part we come to a conclusion about the proposed hybrid algorithm on whether it merits application in the tasks it has been designed to do.

My scheduling will not give the best performance always .But most of the time it will try to give us the best performance .Some of the results are given below

4.1 FCFS vsUser Priority Based CPU Scheduler

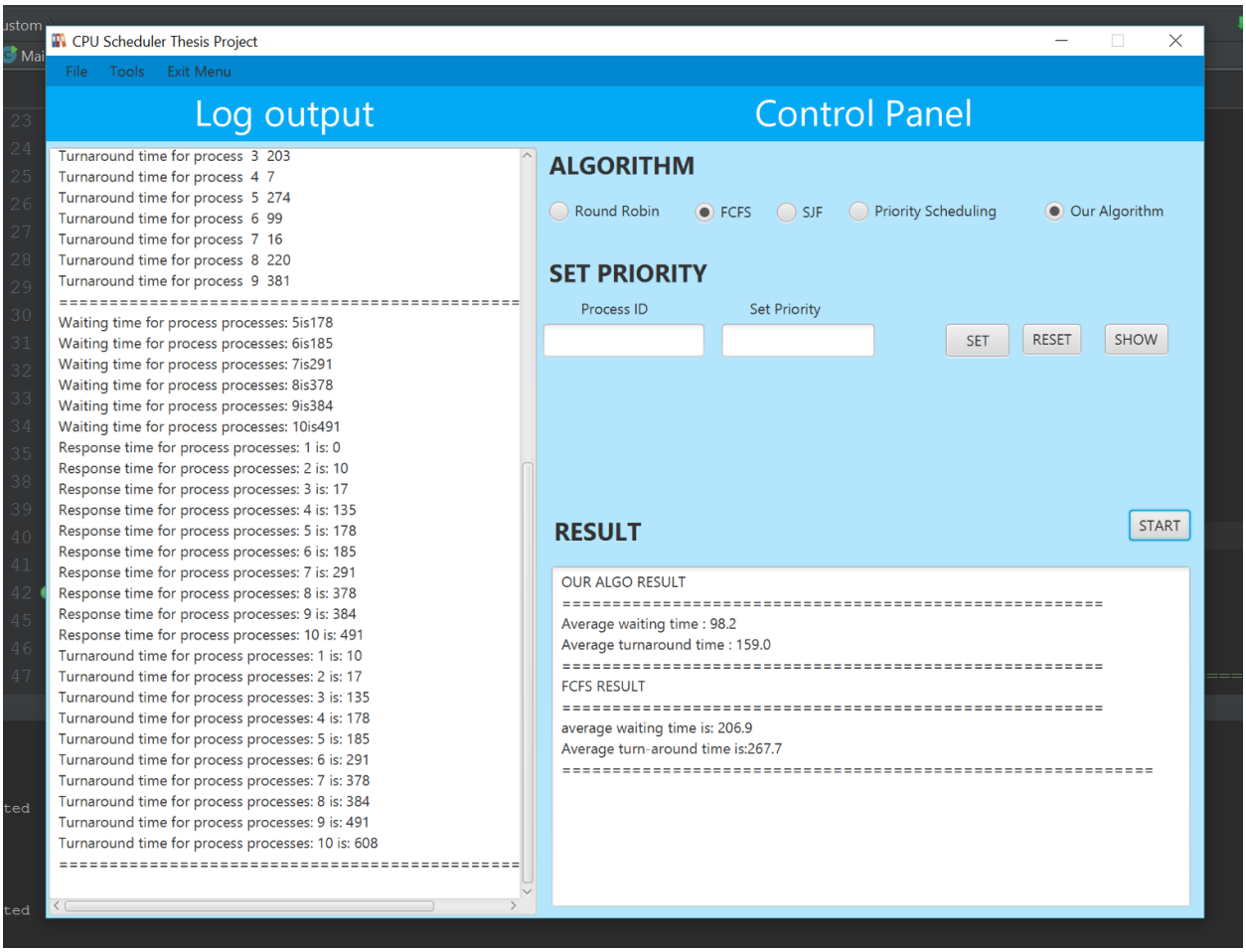


Fig 4.1: FCFS vsUser Priority Based CPU Scheduler

Here we can see our CPU scheduling give more better performance than FCFS .
Average waiting time for FCFS is 206.9s
Average turn-around time for FCFS is 267.7s

Average waiting time for User Priority Based CPU Scheduleris 98.2s
Average turn-around time for User Priority Based CPU Scheduleris159.0s

4.2 SJF vsUser Priority Based CPU Scheduler

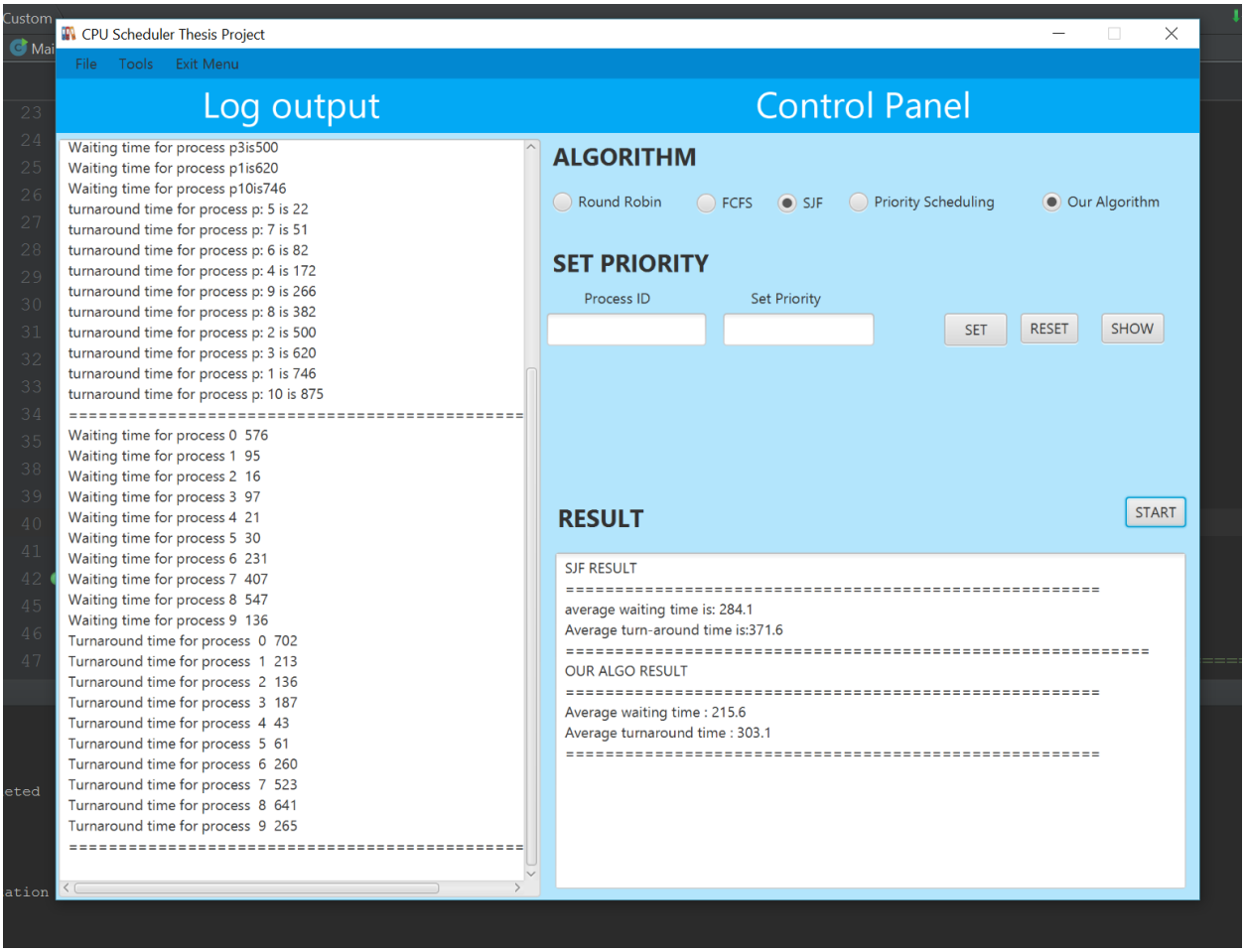


Fig 4.2: SJF vsUser Priority Based CPU Scheduler

Here we can see User Priority Based CPU Scheduler is more efficient than SJF
Average waiting time for SJF is 284.1s
Average turn-around time for SJFis 371.6s

Average waiting time for User Priority Based CPU Scheduleris 215.6s
Average turn-around time for User Priority Based CPU Scheduleris303.1s

4.3 Priority Scheduling vsUser Priority Based CPU Scheduler

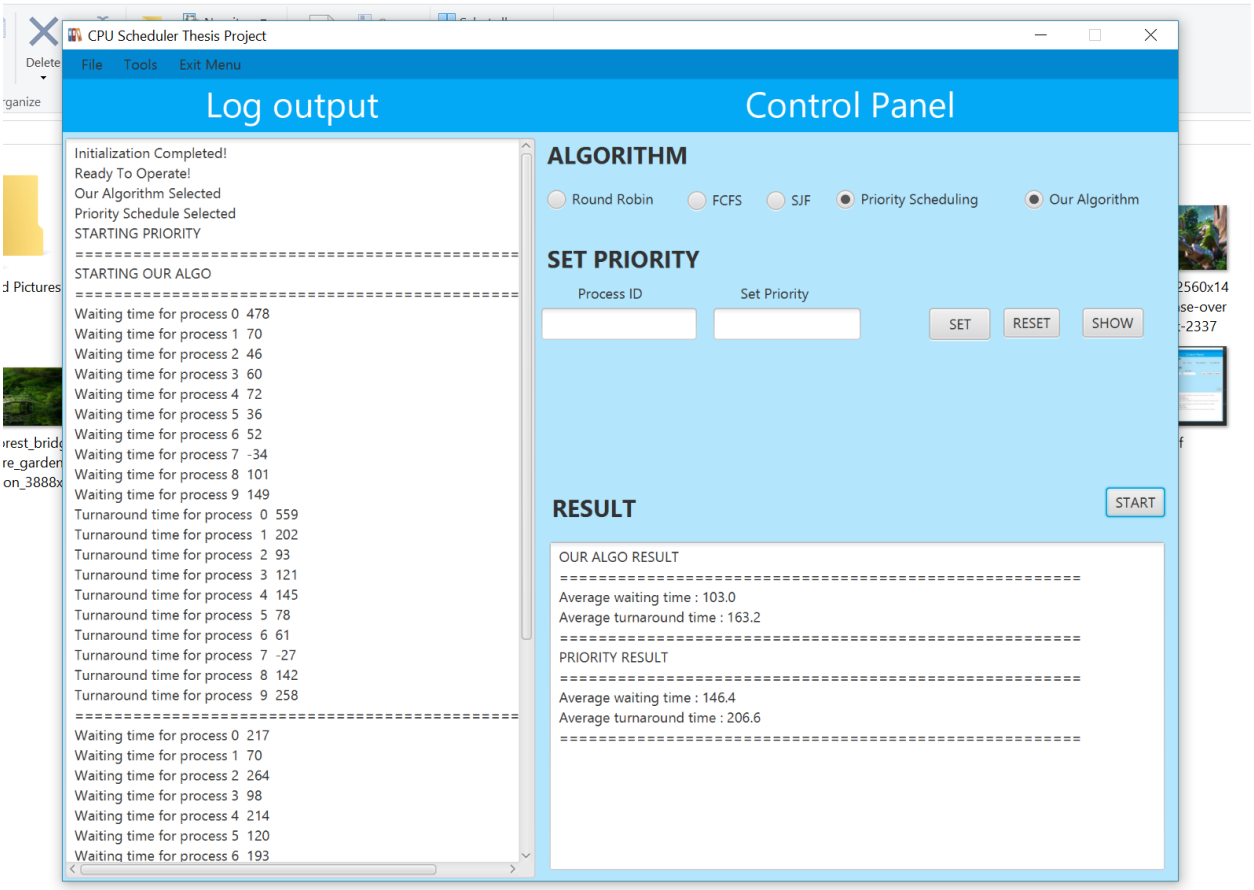


Fig 4.3: Priority schedulingvsUser Priority Based CPU Scheduler

Here we can see our processor scheduling gives better performance than Priority Scheduling.
Average waiting time for Priority Scheduling is 146.4s
Average turn-around time for Priority Scheduling is 206.6s

Average waiting time for User Priority Based CPU Scheduleris 103.0s
Average turn-around time for User Priority Based CPU Scheduleris163.2s

4.4 Round Robin vsUser Priority Based CPU Scheduler

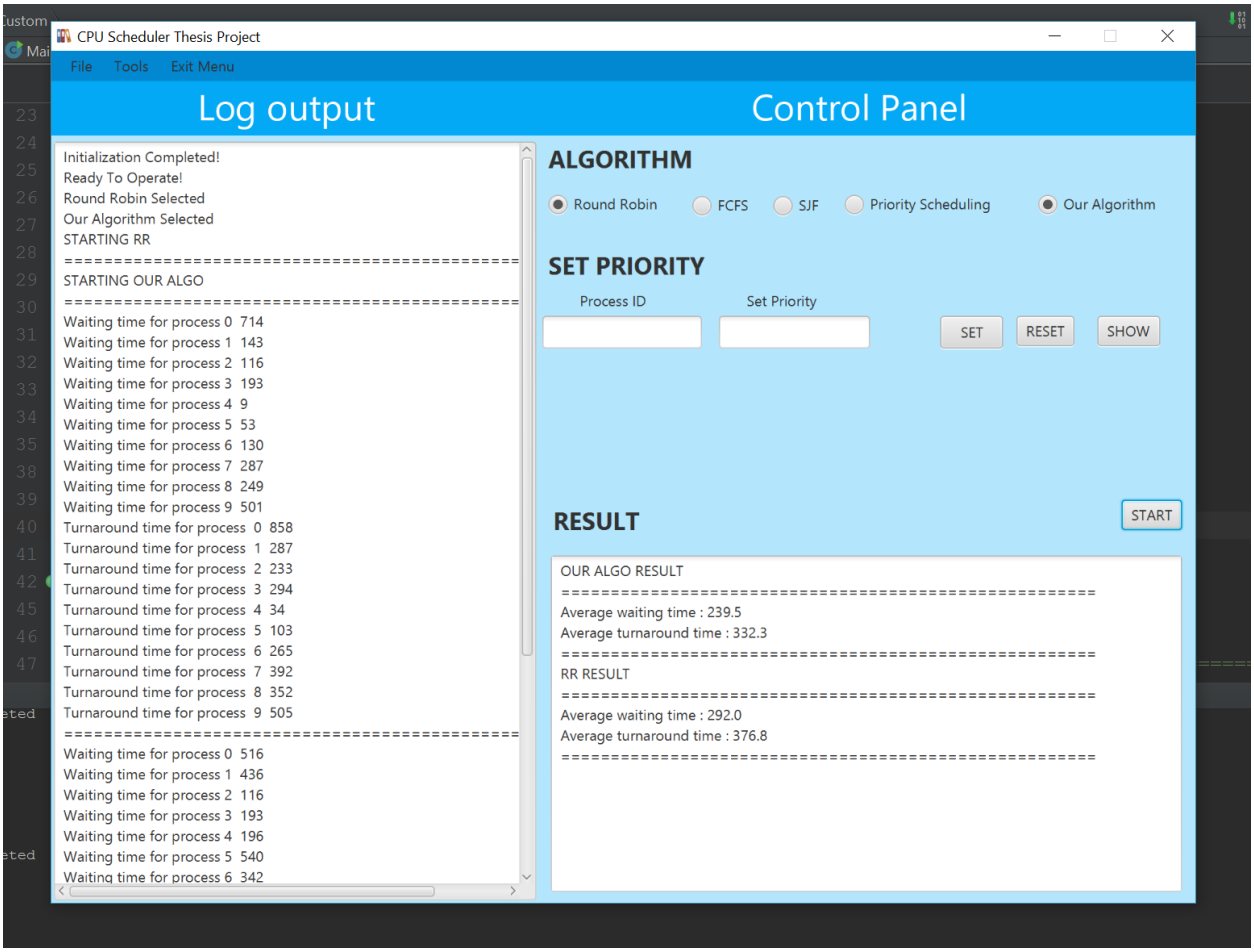
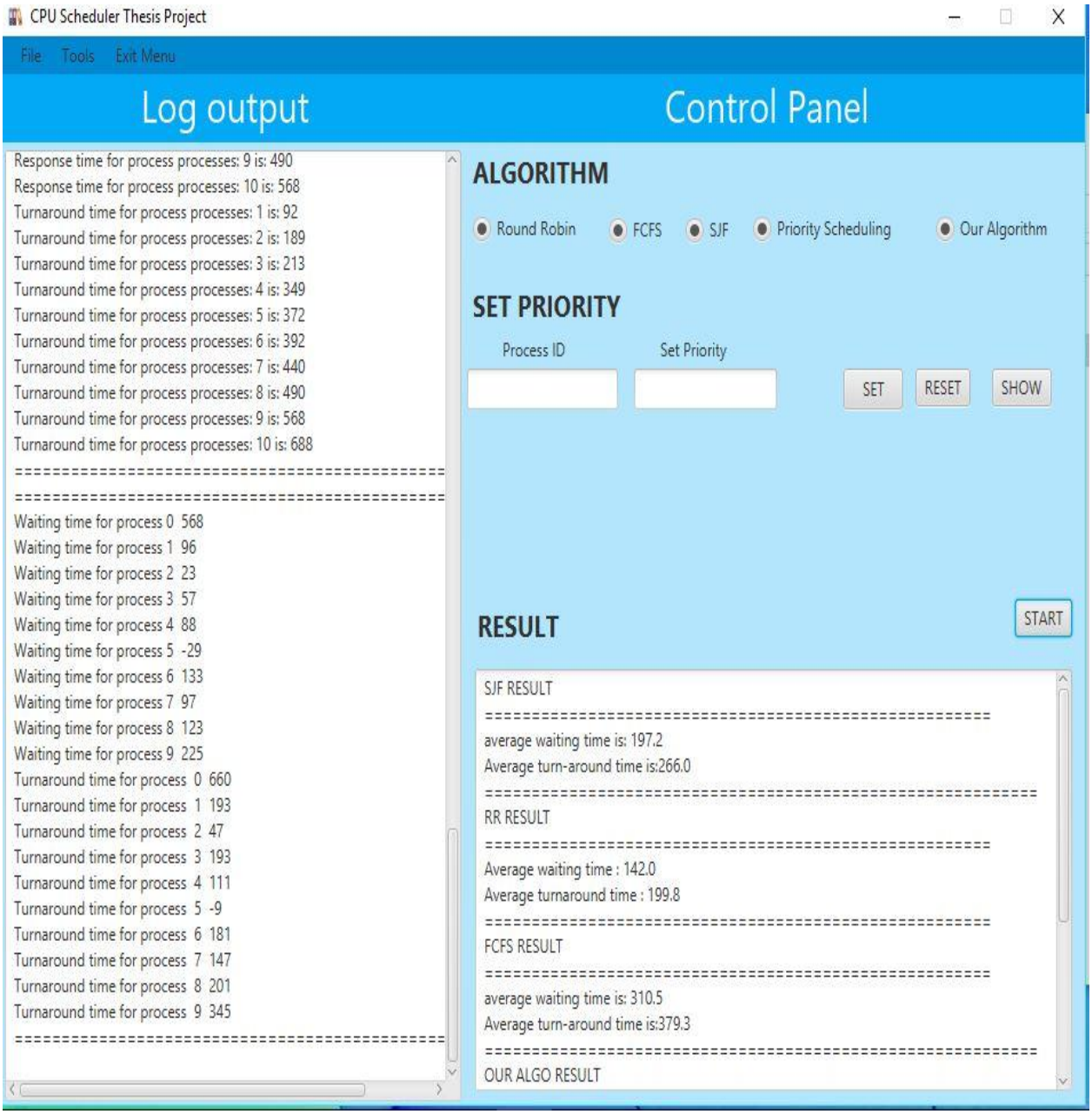


Fig 4.4: Round Robin vsUser Priority Based CPU Scheduler

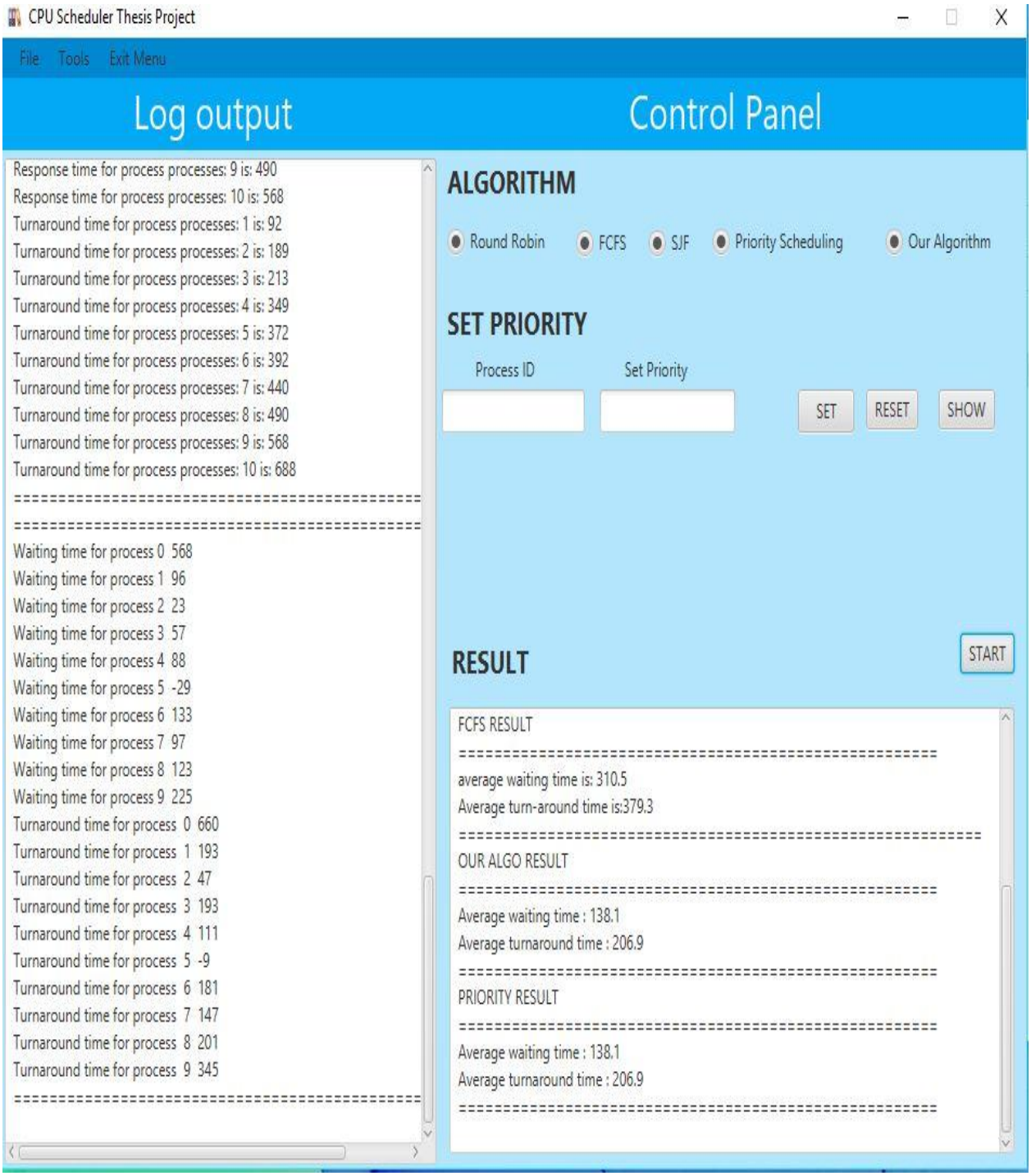
Here we can see User Priority Based CPU Scheduler is more efficient than SJF
Average waiting time for RR is 292.0s
Average turn-around time for RR is 376.8s

Average waiting time for User Priority Based CPU Scheduleris 239.5s
Average turn-around time for User Priority Based CPU Scheduleris376.8s

4.5 User Priority Based CPU Schedulervs FCFS vs SJF vs Priority Scheduling vs Round Robin



(a)



(b)

Fig 4.5: User Priority Based CPU Schedulervs FCFS vs SJF vs Priority Scheduling vs Round Robin

Here we can see User Priority Based CPU Scheduler is more efficient than SJF
Average waiting time for FCFS is310.5s
Average turn-around time for FCFS is 379.3.8s

Average waiting time for SJF is 197.2s
Average turn-around time for User Priority Based CPU Scheduleris266.0s

Average waiting time for Priority Scheduling is 138.1s
Average turn-around time for Priority Scheduling is 206.9s
Average waiting time for RR is 142.0s
Average turn-around time for RR is199.8s

Average waiting time for User Priority Based CPU Scheduleris 138.1s
Average turn-around time for User Priority Based CPU Scheduleris206.9s

4.6 Chapter Summary

In this chapter we observe the performance of User Priority Based CPU Scheduler compared with different scheduling algorithms.

In a few isolated cases it does not perform as well as the other algorithms.

But in most of situations it does give us the best results.

Therefore it can be said to be more effective than the other algorithms mentioned here.

Chapter 5

Conclusion and Future Prospects

In this thesis, a new complex scheduling algorithm, comprising of all the conventional scheduling processes, has been initially proposed and subsequently implemented.

The resulting User Priority Based CPU Scheduler is a comprehensive algorithm for scheduling CPU tasks. It supports pseudo-parallel execution multiple tasks according to assigned priority. In this way it is designed to handle the important jobs first, therefore this algorithm ensures proper performance of the system.

Furthermore, it can perform equally well for a diverse range of loading of the CPU. For example its performance is consistently better than other schedulers for processor loading ranging from 30 to 100%.

All this and reduced waiting times makes the User Priority Based CPU Scheduler a very attractive option for modern computers.

However, if there is something that can be improved for this algorithm it is its own complexity. Thus the scheduler itself takes more processing time than other scheduling algorithms, and can perform slower when the processor does not have that many tasks to take care of.

This minute weakness is a possible area of improvement for the user-centric scheduler. Future researchers are invited to shed more light into the matter.

References

[1] Abraham Silberschatz , Peter Baer Galvin , Greg Gagine , "Operating System Concepts", 7th edition 153- 166

[2] Leo J. Cohen , " Operating System", 5th edition 309- 373

[3] Michael Kifer , Scott A. Smolka , "Introduction To Operating System Design AndImplementation" 3rd edition 54- 72

[4] M. Naghibzadeh" Concepts And Techniques" 101- 134

[5] Sudhir Kumar "Encyclopedia Of Operating System" 160- 205

[6] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "Operating System Concepts", Sixth Edition.

[7] Milan Milenkovic, "Operating Systems Concepts and Design", McGRAM-HILL, Computer Science Series, second edition.

[8] P. Balakrishna Prasad, "Operating Systems" Second Edition.

[9] A. Dhore "Opeating Systems", Technical Publications.

[10] M. Dietel, "Operating Systems", Pearson Education, Second Edition.

[11] http://en.wikipedia.org/wiki/Scheduling

[12] M Gary Nutt, "Operating systems – A Modern Perspective, Second Edition, Pearson Education,