ENHANCED PROCESS SCHEDULING IN LONGEST JOB FIRST ALGORITHM WITH COMBINATIONAL BURST TIME

By

ONWUKWE, Chioma Christiana

DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF PHYSICAL SCIENCE,
AHMADU BELLO UNIVERSITY, ZARIA NIGERIA.

MAY 2017

ENHANCED PROCESS SCHEDULING IN LONGEST JOB FIRST ALGORITHM WITH COMBINATIONAL BURST TIME

By

ONWUKWE, Chioma Christiana

P16PSCS8001

A THESIS SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF MASTER DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE, AHMADU BELLO UNIVERSITY, ZARIA NIGERIA.

DECLARATION

I hereby declare that the work in this thesis titled “Enhanced Process Scheduling In Longest Job First Algorithm With Combinational Burst Time” was performed by me in the Department of Computer Science, under the supervision of Prof. S. B Junaidu and Prof. A. A Obiniyi. The information derived from the literatures has been duly acknowledged in the text and a list of references provided. No part of this work has been presented for another degree or diploma at any institution.

ONWUKWE, Chioma Christiana _____ _____

Name of Student Signature Date

CERTIFICATION

This dissertation titled “Enhanced Process Scheduling In Longest Job First Algorithm With Combinational Burst Time” by Onwukwe Chioma Christiana (**P16PSCS8001**) meets the regulations governing the award of the degree of M.Sc. Computer Science of Ahmadu Bello University (ABU), Zaria, and it is approved for its contribution to knowledge and literary presentation.

Prof. S. B Junaidu

……. …… ……

Chairman, Supervisory Committee Signature Date

Prof. A. A. Obiniyi

……. …… ……

Member, Supervisory Committee Signature Date

……. …… ……

External Examiner Signature Date

Prof. S. B Junaidu

……. …… ……

Head of Department Signature Date

Prof. Kabir Bala

……. …… ……

Dean, School of Postgraduate Signature Date

DEDICATION

I dedicate this work to God Almighty.

ACKNOWLEDGEMENT

I am grateful to God Almighty for sustaining me throughout the period of this thesis and for His numerous provisions.

I have to sincerely appreciate my husband, for initiating and encouraging me to embark on this prestigious journey to acquire a Masters degree. My many thanks go to my children for supporting and bearing with me especially when I had to travel for days and also to Vicky for taking care of the children while I was away.

I want to thank my supervisors, Prof. S. B Junaidu and Prof. A. A Obiniyi; they are the best dual anybody can have as supervisors. They are principled and thorough to the core, their commitments and contributions made the work much easier.

I will not forget to mention Mr. Olaide Oyelade for his immense support during this work. I want to sincerely extend my thanks to all staff of Computer Science Department of A.B.U Zaria, especially the H.O.D; Prof. S. B Junaidu and P.G Coordinator, for their selfless effort to keep the young department going. All staff of Mathematics Department are not left out.

I want to also express my gratitude to all my course mates for encouraging each other to keep on going and all the people that made my stay in A.B.U Zaria worthwhile like Aminat Bola, Sikirat Keyinde and Ruqqiya.

I want to also thank my parents, brothers, sister, sisters-in-law, friends and colleagues for their prayers and support during the course of this work.

I pray that God Almighty will richly reward each and every one of you in Jesus name.

TABLE OF CONTENTS

DECLARATION ii
CERTIFICATION iii
DEDICATION iv
ACKNOWLEDGMENT v
TABLE OF CONTENTS vi
LIST OF FIGURES ix
LIST OF TABLES xi
LIST OF EQUATIONS xiii
LIST OF ABBREVIATIONS xiv
ABSTRACT xv
CHAPTER ONE 1
INTRODUCTION 1
1.1 Background of the Study 1
1.2 Problem Statement 4
1.3 Aim and Objectives 5
1.4 Research Methodology 5
1.5 Contribution to Knowledge 6
1.6 Organization of the dissertation 6
CHAPTER TWO 8
LITERATURE REVIEW 8
2.1 INTRODUCTION 8
2.2 Operating System 8
2.2.1 The Operating System as a User/Computer Interface 8
2.2.2 The Operating System as Resource Manager 10
2.3 Process 11
2.4 Scheduling 14
2.4.1 Long-Term Scheduling 15
2.4.2 Medium-Term Scheduling 15
2.4.3 Short-Term Scheduling 16
2.4.4 Dispatcher 17
2.5 Context Switch 18

2.6 Scheduling Algorithms....18
2.6.1 Shortest Jobs First....18
2.6.2 Longest Job First....19
2.6.3 First-Come First-Served....20
2.6.4 Round Robin....21
2.7 Related Works....22
CHAPTER THREE....30
DESIGN OF THE ENHANCED LONGEST JOB FIRST ALGORITHM WITH COMBINATIONAL BURST TIME....30
3.1 Introduction....30
3.2 The Proposed Algorithm....30
3.2.1 The Architecture of the Proposed ELJF+CBT....33
3.2.2 Pseudo Code of the Proposed ELJF+CBT....34
3.3 Design for the algorithm of the proposed ELJF +CBT....35
3.4 Tools and Programming Languages Used....35
CHAPTER FOUR....36
IMPLEMENTATION AND RESULTS PRESENTATION OF THE ELJF+CBT ALGORITHM....36
4.1 Introduction....36
4.2 Analytical Example....36
4.3 Performance Evaluation Metrics....42
4.4 Experimental Cases of the Scheduling Algorithm....43
4.4.1 First Come First Serve (FCFS)....43
4.4.2 Shortest Job First (SJF)....45
4.4.3 Longest Job First (LJF)....46
4.4.4 Longest Job First +CBT (LJF+CBT)....48
4.4.5 Enhanced Longest Job First with CBT (ELJF+CBT)....50
4.5 Comparing ELJF+CBT with some Existing Scheduling Algorithms....53
4.5.1 Comparing ELJF+CBT and FCFS....53
4.5.2 Comparing ELJF+CBT and SJF....54
4.5.3 Comparing ELJF+CBT and LJF....54
4.5.4 Comparing ELJF+CBT and LJF +CBT....54
4.6 Result Discussion and Conclusion....55

CHAPTER FIVE 70
Summary, Conclusion and Recommendation 70
5.1 Summary 70
5.2 Conclusion 70
5.3 Future work 71
5.4 Recommendation 71
REFERENCES 72
APPENDIX I 75

LIST OF FIGURES

Figure 1.1: Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process (Tanenbaum, 2009). 2

Figure 2.1: Computer Hardware and Software Structure (Stallings, 2012).....9

Figure 2.2: Five-state Process model (Stallings, 2012). 13

Figure 2.3: Scheduling and Process State Transition (Forum, 2010) 15

Figure 2.4: Queuing Diagram for Scheduling (Stallings, 2012) 17

Figure 2.5: Model Architecture (Abdullahi and Junaidu, 2013)..... 23

Figure 3.1: ELJF+CBT Architecture 33

Figure 4.1: Comparison of AWT for LJF order and interleaved order..... 40

Figure 4.2: Comparison of ATAT for LJF order and interleaved order 41

Figure 4.3: Gantt chart for FCFS 43

Figure 4.4: Gantt chart for SJF 45

Figure 4.5: Gantt chart for LJF 47

Figure 4.6: Gantt chart for LJF+CBT 49

Figure 4.7:Gantt chart for ELJF+CBT 52

Figure 4.8: Snapshot of 500 processes after one execution 56

Figure 4.9: Snapshot of execution of process scheduling using windows batch scripting ... 57

Figure 4.10: Snapshot of processes with their Burst Times and Arrival Times 57

Figure 4.11: Graph of Average Waiting Times for a sample process (50-500) 58

Figure 4.12: Graph of Average Turnaround Times for a sample process (50-500) 60

Figure 4.13: Graph of Context Switches for sample processes (50-500) 62

Figure 4.14: Graph of average waiting time for LJF, LJF+CBT and ELJF+CBT 64

Figure 4.15: Graph of average turnaround time for LJF, LJF+CBT and ELJF+CBT 66
Figure 4.16: Graph of context switch for LJF, LJF+CBT and ELJF+CBT 68

LIST OF TABLES

Table 2.1: Types of Scheduling (Stallings, 2012) 14

Table 4.1: 20 processes with their burst times (X_i,δ_i) submitted to the pool **N** 37

Table 4.2: Processes ($X\delta$) sorted from highest burst time to the lowest..... 38

Table 4.3: Processes with burst times greater than the threshold H 38

Table 4.4: Processes with burst times less than or equal to the threshold H 39

Table 4.5: New formed processes with their burst times 39

Table 4.6: Collation of processes after evaluation of CBT model..... 39

Table 4.7: processes arranged in interleaved order 41

Table 4.8: 10 Processes with process id, burst time and arrival time 43

Table 4.9: 10 processes and their Burst Times, Sorted in SJF order of their burst times 45

Table 4.10: 10 processes and their Burst Times, Sorted in LJF order of their burst times... 46

Table 4.11: 8 Processes formed after merging shorter processes 48

Table 4.12: showing Table 4.11 sorted in LJF order 49

Table 4.13: 7 Processes formed after merging shorter processes 51

Table 4.14: showing Table 4.13 sorted in alternate order of burst time 51

Table 4.15: Comparison of ELJF+CBT and FCFS for 500 processes after 12 executions .. 53

Table 4.16: Comparison of ELJF+CBT and SJF for 500 processes after 12 executions 54

Table 4.17: Comparison of ELJF+CBT and LJF for 500 processes after 12 executions 54

Table 4.18: Comparison of ELJF+CBT and LJF+CBT for 500 processes after 12 executions 55

Table 4.19: Cumulative result of 500 processes after 12 executions. 55

Table 4.20: Average Waiting Time data 59

Table 4.21: Average Turnaround Time data 61

Table 4.22: Context Switches data 63

Table 4.23: Average waiting time data for LJF, LJF+CBT and ELJF+CBT 65

Table 4.24: Average turnaround time data for LJF, LJF+CBT and ELJF+CBT 67

Table 4.25: Context switch data for LJF, LJF+CBT and ELJF+CBT 69

LIST OF EQUATIONS

Equation 3.1: To calculate median value **H** 31

Equation 4.1: waiting time 42

Equation 4.2: average waiting time 42

Equation 4.3: turn-around time 42

Equation 4.4: average turn-around time 42

LIST OF ABBREVIATIONS

SJF	Shortest Job First
LJF	Longest Job First
RR	Round Robin
WT	Waiting Time
TWT	Total Waiting Time
AWT	Average Waiting Time
TAT	Turn-Around Time
TTAT	Total Turn-Around Time
ATAT	Average Turn-Around Time
CS	Context Switch
RT	Response Time
CBT	Combinational Burst Time
LJF+CBT	Longest Job First + Combinational Burst Time
ELJF+CBT	Enhanced Longest Job First + Combinational Burst Time
FCFS	First Come First Serve
CWA	Combined Weighted Average
CPU	Central Processing Unit
I/O	Input/Output
TQ	Time Quantum
OS	Operating System
PCB	Process Control Block

ABSTRACT

Longest Job First (LJF), the opposite of Shortest Job First, has been an unpopular process scheduling algorithm. The idea of *combinational burst time* was used to avoid the starvation problem associated with the LJF algorithm and to make it compete with other scheduling algorithms with respect to some performance metrics. Although the combinational burst time proposal led to improved performance, the work suffers from the limitations that the average waiting time and the average turnaround time were slightly higher, and the number of context switches was a little higher also. The research reported in this dissertation addressed these shortcomings by proposing a new scheduling algorithm that enhanced the combinational burst time model by using median as a statistics of central tendency and assigning the processors to the CPU in interleave order. The proposed algorithm was implemented and compared with First Come First Serve (FCFS), Longest Job First (LJF), Longest Job First with combinational model (LJF+CBT) and Shortest Job First (SJF) scheduling algorithms using varying number of processes and burst times. Results from the experiments showed that the enhanced LJF+CBT outperformed the existing LJF+CBT producing 26.69% better average waiting time (AWT), 21.77% better average turnaround time (ATAT) and 14.29% better number of context switches (CS). In Longest Job First (LJF) scheduling this algorithm drastically reduced the average waiting time by 46.5%, average turnaround time by 39.39% and number of context switching between processes by 33.33% for all the number of processes used. Sequel to these results, a better solution to starvation problem in Longest Job First scheduling algorithm was proffered.

CHAPTER ONE
INTRODUCTION

This chapter discusses the introductory part of this thesis which includes the background of the study, problem statement, the research aim and objectives, the methodology that was used in solving the problems stated and finally the summary of the thesis contribution to knowledge.

1.1 Background of the Study

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive (Silberschatz *et al.*, 2005).

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization, so whenever the CPU becomes idle the operating system must select one of the processes in the ready queue to be executed, the short-term scheduler (or CPU scheduler) carries out the selection process. The scheduler selects a process from the processes in memory that are ready to be executed and allocates the CPU to that process (Silberschatz *et al.*, 2005).

Nearly all processes alternate bursts of computing with (disk) I/O requests, as shown in Fig. 1.1

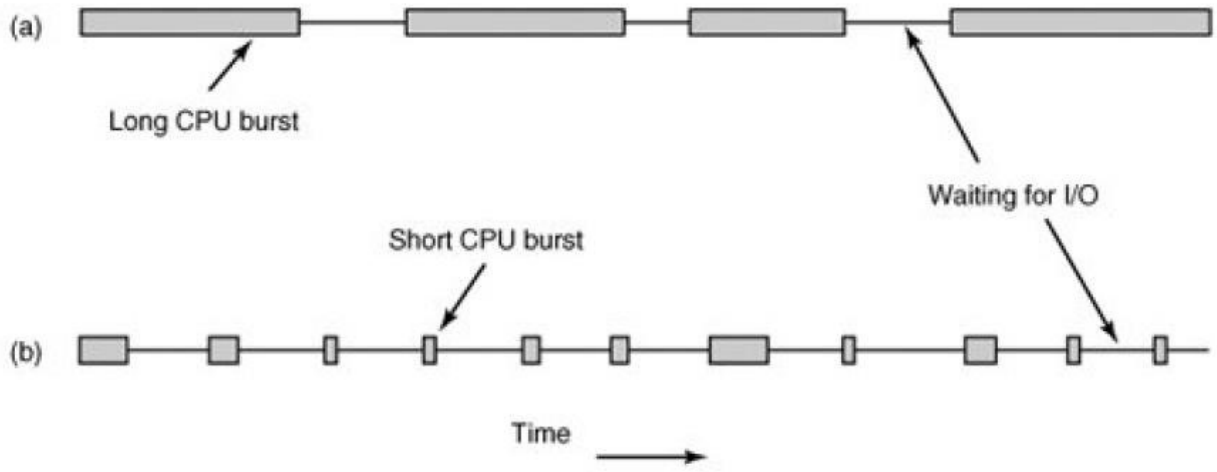


Figure 1.1: Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process (Tanenbaum, 2009).

Typically the CPU runs for a while without stopping, then a system call is made to read from a file or write to a file. When the system call completes, the CPU computes again until it needs more data or has to write more data. Note that some I/O activities count as computing. For example, when the CPU copies bits to a video RAM to update the screen, it is computing, not doing I/O, because the CPU is in use. I/O in this sense is when a process enters the blocked state waiting for an external device to complete its work (Tanenbaum, 2009).

Some scheduling algorithms goals include the following (Tanenbaum, 2009), (Silberschatz *et al.*, 2009):

1. Fairness: giving each process a fair share of the CPU, comparable processes should get comparable service. Giving one process much more CPU time than an equivalent one is not fair. Of course, different categories of processes may be treated differently.
2. Throughput: Maximize jobs per hour; one measure of work is the number of processes that are completed per time unit.

3. Turnaround time: Minimize time between submission and termination, the interval from the time of submission of a process to the time of completion.
4. CPU utilization: Keep the CPU busy all the time, Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
5. Response time: is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device. respond to requests quickly
6. Meeting deadlines: avoid losing data, they are characterized by having deadlines that must or at least should be met.

A non-preemptive scheduling algorithm picks a process to run and then just lets it run until it blocks (either on I/O or waiting for another process) or until it voluntarily releases the CPU. In contrast, a preemptive scheduling algorithm picks a process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run (if one is available). Doing preemptive scheduling requires having a clock interrupt occur at the end of the time interval to give control of the CPU back to the scheduler. (Tanenbaum, 2009).

Unfortunately, preemptive scheduling incurs a cost associated with access to shared data. Consider the case of two processes that share data. While one is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state. In such situations, there is need for a new mechanisms to coordinate access to shared data (Silberschatz *et al.*, 2009).

Longest Job First scheduling algorithm is a (non-preemptive) scheduling algorithm that will be introduced as an opposite of the Shortest Job First (SJF); (non-preemptive) scheduling algorithm; it differs from the SJF by considering longer burst time jobs before the shorter ones (Abdullahi and Junaidu, 2013).

The major problem with Longest Job First is the starvation of processes with shorter CPU burst time. This research work introduces a new scheduling algorithm called Enhanced Process Scheduling in Longest Job First Algorithm with Combinational Burst Time (ELJF+CBT), which will further minimize Average Waiting Time (AWT), Average Turn-Around Time (ATAT) and reduce the number of Context Switch (CS) of LJF.

1.2 Problem Statement

Scheduling as a key to multiprogramming affects the performance of a system because it determines which processes will wait and which will be processed. Several scheduling algorithms have been developed for efficient multiprogramming with each having its own pros and cons. Longest Job First being one of the scheduling algorithms has some shortcomings like starvation of shorter process, high average waiting time and average turnaround time. Appreciably, there was an effort made by Abdullahi and Junaidu (Abdullahi and Junaidu, 2013) to find a solution to the shortcomings of Longest Job First to some extent. However, the average waiting time (AWT), average turnaround time (ATAT), context switch (CS) are still slightly high. Although the combinational model did minimize the average waiting time, average turnaround time and number of context switches in LJF, the questions are as follows can the:

1. Average waiting time be minimized further?

2. Average turnaround time be minimized further?
3. Number of context switches be reduced?

1.3 Aim and Objectives

The aim of this research is to develop an algorithm that will improve the performance of Longest Job First scheduling algorithm of Abdullahi and Junaidu (Abdullahi and Junaidu, 2013).

The objectives of this thesis work are to:

1. Enhance the LJF+CBT algorithm of Abdullahi and Juniadu (2013) with a view to overcoming its shortcomings of high AWT, ATAT and context switch,
2. Implement the enhanced LJF+CBT algorithm and
3. Evaluate performance of the enhanced algorithm side-by-side the existing LJF algorithm with respect to average waiting time, average turnaround time and number of context switches.

1.4 Research Methodology

1. Generate processes and their CPU burst times using a Java random number generator,
2. Determine a threshold, H, that distinguishes short and long processes. The threshold will be calculated using the median of burst times of the processes in the ready queue,
3. Processes will be categorized as shorter or longer jobs based on the threshold value **H**,
4. Shorter jobs will be merged to form a new longer job dynamically while comparing burst time of new formed process with threshold value **H**,
5. A special flag **F** will be used to mark shorter or longer jobs to identify them for CPU scheduler,

6. CPU will be allocated to jobs in alternation order of burst times: long job, short job, long job, short job, etc
7. Gantt charts and graphical illustrations will be used to analyze and measure evaluation criteria of the algorithms, and
8. The new algorithm (ELJF+CBT) will be compared with FCFS, LJF, SJF and LJF+CBT.

1.5 Contribution to Knowledge

This thesis contributed the following:

1. Developed an algorithm that drastically reduced average waiting time by 46.5%, average turnaround time by 39.39% and number of context switching between processes by 33.33% in Longest Job First (LJF) and reduced average waiting time by 26.69%, average turnaround time by 21.77% and number of context switching between processes by 14.29% in Longest Job First with Combinational Burst Time (LJF+CBT).
2. Introduction of a specialized array **F**, which served as a flag that was set and reset each time the CPU scheduler allocated the CPU to a process; which enabled scheduling to be done in interleave order.

1.6 Organization of the dissertation

Chapter one provides a general introduction of the dissertation, the problem statement, the aim and objectives, research methodology and the contribution to knowledge.

Chapter two presents review of some related literature and studies conducted in the area of process scheduling.

Chapter three provides a detailed methodology followed to design the proposed enhanced process scheduling in Longest Job First Algorithm with CBT.

Chapter four presents the implementation and analysis of the result of the proposed algorithm

Chapter five provides the conclusion, summary and future area of research of the work.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter reviews the foundation literature for the thesis, that is, the operating system, process, some scheduling algorithms and their limitations and also discusses some works that are related to this thesis.

2.2 Operating System

An Operating System (OS) is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware, (Stallings, 2012). It can be thought of as having three objectives:

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to evolve:** An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

2.2.1 The Operating System as a User/Computer Interface

The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion, as depicted in Figure 2.1. The user of those applications, the end user, generally is not concerned with the details of computer hardware. Thus, the end user views a computer system in terms of a set of applications.

An application can be expressed in a programming language and is developed by an application programmer (Stallings, 2012). If one were to develop an application program as a set of machine

instructions that is completely responsible for controlling the computer hardware, one would be faced with an overwhelmingly complex undertaking.

To ease this chore, a set of system programs is provided. Some of these programs are referred to as utilities, or library programs. These implement frequently used functions that assist in program creation, the management of files, and the control of

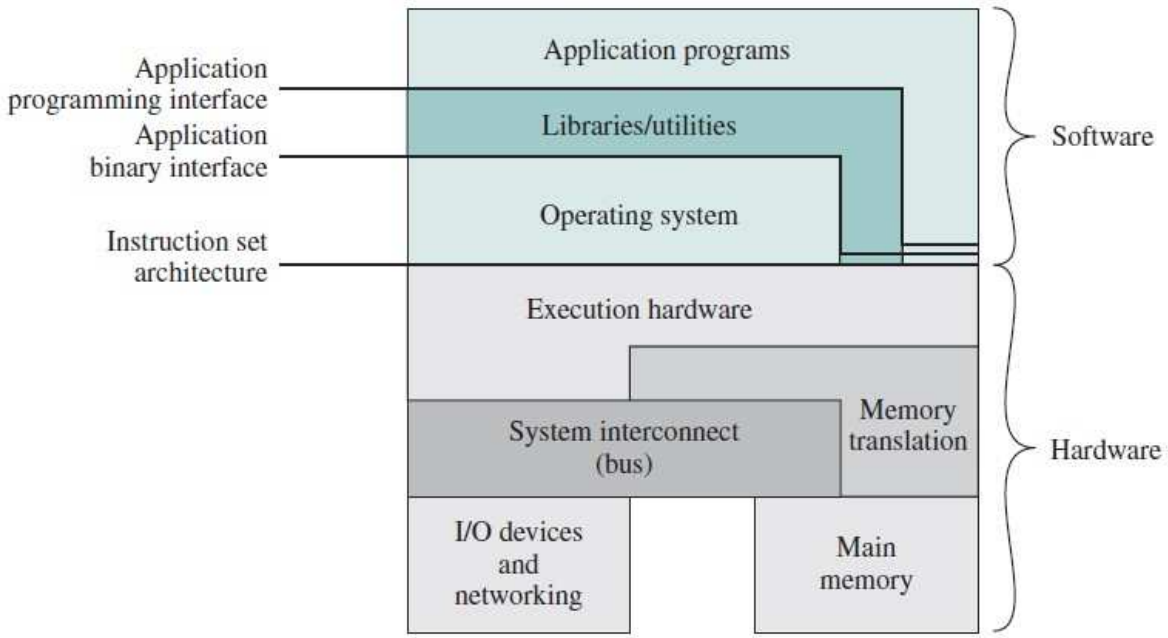


Figure 2.1: Computer Hardware and Software Structure (Stallings, 2012)

I/O devices. A programmer will make use of these facilities in developing an application, and the application, while it is running, will invoke the utilities to perform certain functions. The most important collection of system programs comprises the operating system (OS). The OS masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system. It acts as mediator, making it easier for the programmer and for application programs to access and use those facilities and services.

Briefly, the OS according to (Stallings, 2012) typically provides services in the following areas:

1. Program development
2. Program execution: A number of steps need to be performed to execute a program. Instructions and data must be loaded into main memory, I/O devices and files must be initialized, and other resources must be prepared. The OS handles these scheduling duties for the user.
3. Access to I/O devices
4. Controlled access to files
5. System access
6. Error detection and response
7. Accounting
8. Instruction set architecture (ISA)
9. Application binary interface (ABI)
10. Application programming interface (API)

2.2.2 The Operating System as Resource Manager

A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions. The OS is responsible for managing these resources.

Can we say that it is the OS that controls the movement, storage, and processing of data? From one point of view, the answer is yes: By managing the computer's resources, the OS is in control of the computer's basic functions. But this control is exercised in a curious way. Normally, we think of a control mechanism as something external to that which is controlled, or at least as something that is a distinct and separate part of that which is controlled. (For example, a residential heating system is controlled by a thermostat, which is separate from the heat-generation and heat-

distribution apparatus.) This is not the case with the OS, which as a control mechanism is unusual in two respects (Stallings, 2012):

1. The OS functions in the same way as ordinary computer software; that is, it is a program or suite of programs executed by the processor.
2. The OS frequently relinquishes control and must depend on the processor to allow it to regain control.

Like other computer programs, the OS provides instructions for the processor.

The key difference is in the intent of the program. The OS directs the processor in the use of the other system resources and in the timing of its execution of other programs. But in order for the processor to do any of these things, it must cease executing the OS program and execute other programs. Thus, the OS relinquishes control for the processor to do some “useful” work and then resumes control long enough to prepare the processor to do the next piece of work.

2.3 Process

Central to the design of operating systems is the concept of *process.* This term was first used by the designers of Multics in the 1960s (Daley and Dennis, 1968). It is a somewhat more general term than *job.* Many definitions have been given for the term *process*, including

1. A program in execution
2. An instance of a program running on a computer
3. The entity that can be assigned to and executed on a processor
4. A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Three major lines of computer system development created problems in timing and synchronization that contributed to the development of the concept of the process: multiprogramming batch operation, time sharing, and real-time transaction systems. As we have seen, multiprogramming was designed to keep the processor and I/O devices, including storage devices, simultaneously busy to achieve maximum efficiency. The key mechanism is this: In response to signals indicating the completion of I/O transactions, the processor is switched among the various programs residing in main memory (Stallings, 2012).

A second line of development was general-purpose time sharing. Here, the key design objective is to be responsive to the needs of the individual user and yet, for cost reasons, be able to support many users simultaneously. These goals are compatible because of the relatively slow reaction time of the user. For example, if a typical user needs an average of 2 seconds of processing time per minute, then close to 30 such users should be able to share the same system without noticeable interference. Of course, OS overhead must be factored into such calculations.

A third important line of development has been real-time transaction processing systems. In this case, a number of users are entering queries or updates against a database. An example is an airline reservation system. The key difference between the transaction processing system and the timesharing system is that the former is limited to one or a few applications, whereas users of a timesharing system can engage in program development, job execution, and the use of various applications, (Stallings, 2012). In both cases, system response time is paramount.

The principal tool available to system programmers in developing the early multiprogramming and multiuser interactive systems was the interrupt. The activity of any job could be suspended by the occurrence of a defined event, such as an I/O completion. The processor would save some sort of context (e.g., program counter and other registers) and branch to an interrupt-handling routine,

which would determine the nature of the interrupt, process the interrupt, and then resume user processing with the interrupted job or some other job (Stallings, 2012).

Figure 2.2 shows the process state diagram.

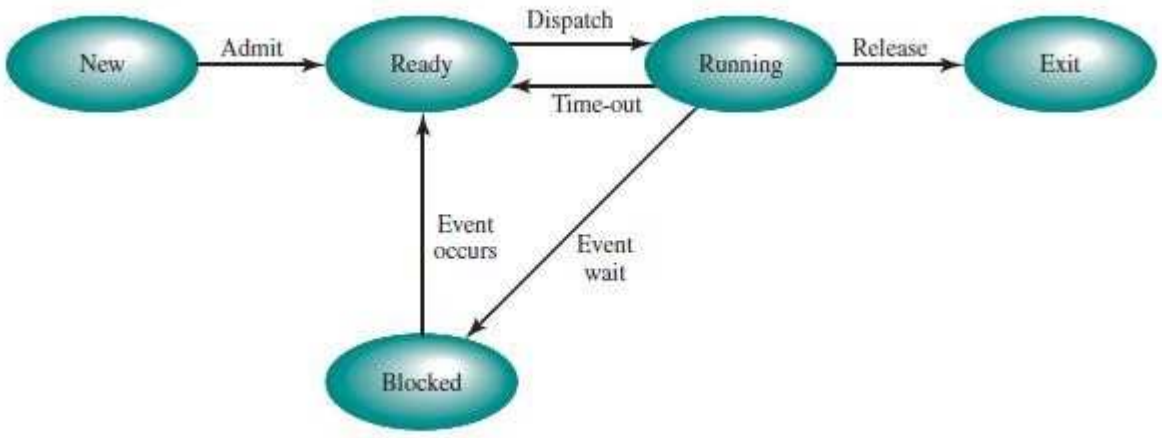


Figure 2.2: Five-state Process model (Stallings, 2012).

As a process; executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states (Silberschatz *et al.*, 2009):

New: The process is being created.

Running: Instructions are being executed.

Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).

Ready: The process is waiting to be assigned to a processor.

Terminated: The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also more finely delineate process states.

It is important to realize that only one process can be running on any processor at any instant. Many processes may be ready and waiting, however.

2.4 Scheduling

The method by which the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU is known as scheduling. Scheduling is gone to meet the objective of multiprogramming, which is to have some process running at all times to maximize CPU utilization and the objective of time sharing which is to switch the CPU among processes so frequently that users can interact with each program while it is running (Silberschatz *et al.*, 2009).

The key to multiprogramming is scheduling. In fact, four types of scheduling are typically involved (Table 2.1). One of these, I/O scheduling, will not be discussed in this thesis. (Stallings, 2012)

Table 2.1: Types of Scheduling (Stallings, 2012)

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

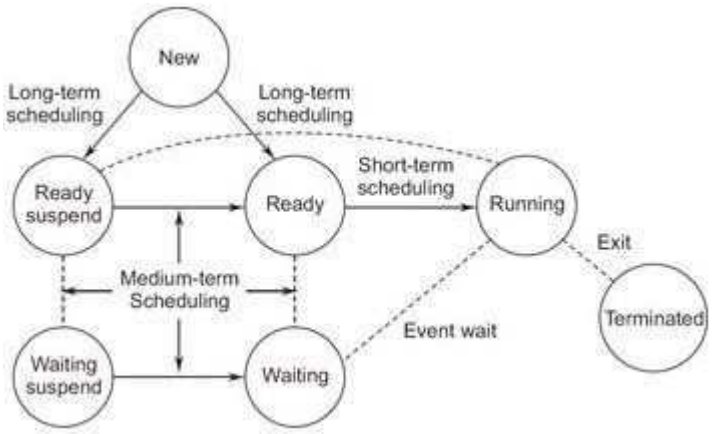


Figure 2.3: Scheduling and Process State Transition (Forum, 2010)

2.4.1 Long-Term Scheduling

The long-term scheduler, or admission scheduler, decides which jobs or processes are to be admitted to the ready queue (in main memory); that is, when an attempt is made to execute a program, its admission to the set of currently executing processes is either authorized or delayed by the long-term scheduler. Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time – whether many or few processes are to be executed concurrently, and how the split between I/O-intensive and CPU-intensive processes is to be handled. The long-term scheduler is responsible for controlling the degree of multiprogramming (Wikipedia, 2016).

2.4.2 Medium-Term Scheduling

Medium-term scheduling is part of the swapping function and determines when a program is brought partially or fully into main memory so that it may be executed (Stallings, 2012). The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of

multiprogramrning. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called swapping. The process is swapped out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up (Silberschatz *et al.*, 2009).

2.4.3 Short-Term Scheduling

The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them (Silberschatz *et al.*, 2009).

In terms of frequency of execution, the short-term scheduler, also known as the dispatcher, executes most frequently and makes the fine-grained decision of which process to execute next. The long-term scheduler executes relatively infrequently and makes the coarse-grained decision of whether or not to take on a new process and which one to take. The medium-term scheduler is executed somewhat more frequently to make a swapping decision.

The short-term scheduler is invoked whenever an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another. As can be seen in Figure 2.4. Examples of such events include:

1. Clock interrupts
2. I/O interrupts
3. Operating system calls
4. Signals (e.g., semaphores) (Stallings, 2012).

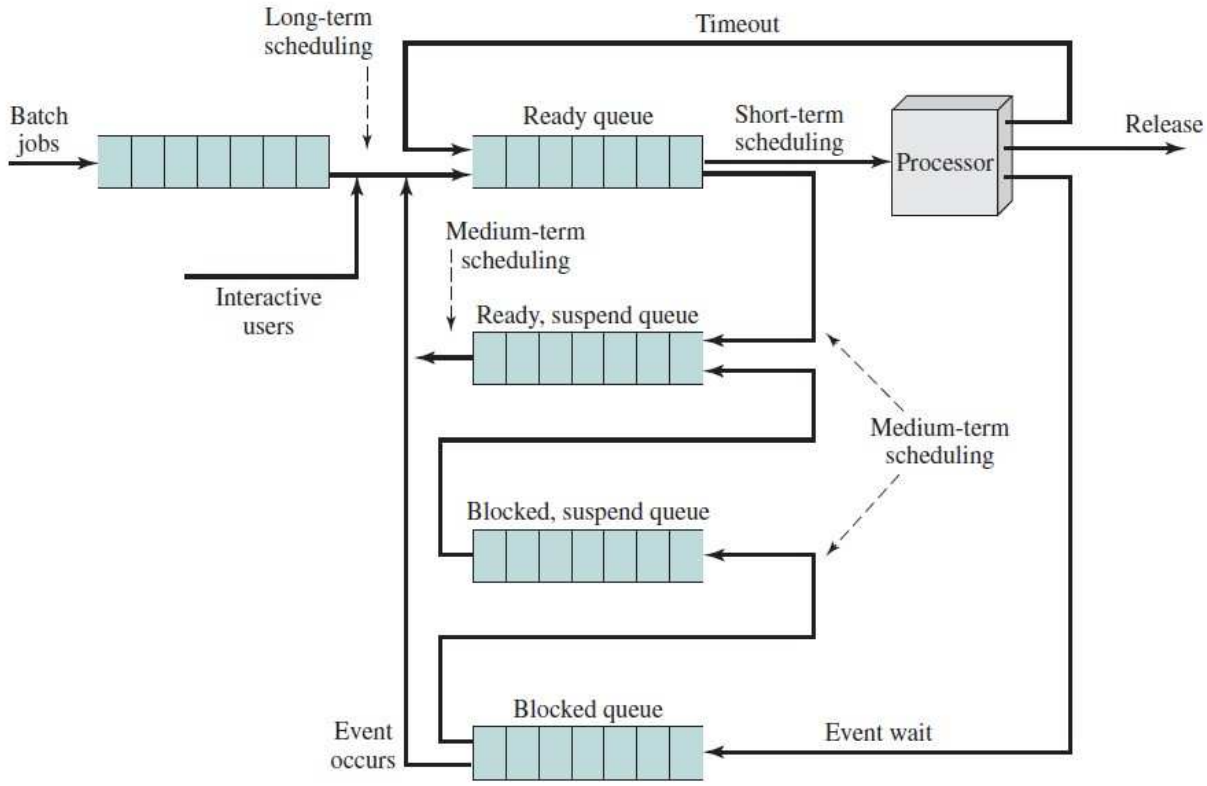


Figure 2.4: Queuing Diagram for Scheduling (Stallings, 2012)

2.4.4 Dispatcher

Another component involved in the CPU-scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

1. Switching context;
2. Switching to user mode;
3. Jumping to the proper location in the user program to restart that program.

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency (Silberschatz *et al.*, 2009).

2.5 Context Switch

Process switch occur when a running process is interrupted and the OS assigns another process to the Running state and turns control over to that process (Stallings, 2012).

When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. The context is represented in the Process Control Block (PCB) of the process; it includes the value of the CPU registers, the process state, and memory-management information.

Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers).

Context-switch times are highly dependent on hardware support. Also, the more complex the operating system, the more work must be done during a context switch. (Silberschatz *et al.*, 2009).

2.6 Scheduling Algorithms

2.6.1 Shortest Jobs First

(Tanenbaum, 2009), explained that this scheduling algorithm as a non-preemptive batch algorithm that assumes the run times are known in advance. When several equally important jobs are sitting in the input queue waiting to be started, the scheduler picks the shortest job first.

Shortest job first is provably optimal, in that it gives the minimum average waiting time for a given set of processes. Moving a short process before a long one decreases the waiting time of the short

process more than it increases the waiting time of the long process. Consequently, the average waiting time decreases. It is worth pointing out that Shortest Job First is only optimal when all the jobs are available simultaneously.

(Silberschatz *et al.*, 2009), this algorithm associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

They noted that a more appropriate term for this scheduling algorithm would be the shortest-next-CPU-burst algorithm because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Limitation: knowing the length of the next CPU burst and all job must have the same arrival time.

2.6.2 Longest Job First

(Dipti and Mittal, 2013), In Longest Job First scheduling, the job with large execution time is executed first in comparison to job with shortest execution time. It means a job whose execution time is small will have to wait because of a job which has large execution time than that job.

Longest Processing Time (LPT), (Dorit, 1999), The Longest Processing Time rule orders the jobs in the order of decreasing processing times. Whenever a machine is freed, the largest job ready at the time will begin processing. This algorithm is a heuristic used for finding the minimum makespan of a schedule. It schedules the longest jobs first so that no one large job will "stick out" at the end of the schedule and dramatically lengthen the completion time of the last job.

Decreasing-Time Algorithm, (Bowen, 2015), the Decreasing-Time Algorithm (DTA) is based on a seemingly simple strategy: Do the longer jobs first and save the shorter jobs for last.

Simply put, the DTA creates a Priority List by listing the tasks in decreasing order of processing times (longest task first, shortest task last). Tasks with equal processing times can be listed in any order.

A Priority List created by the DTA is often called a decreasing-time list.

Limitation: starvation of processes with shorter CPU burst time, high average waiting time and average turnaround time.

2.6.3 First-Come First-Served

By far the simplest CPU-scheduling algorithm is the first-come first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue (Silberschatz *et al.*, 2009).

First-come first-served scheduling algorithm is easy to understand and equally easy to program. It is also fair in the same sense that allocating CPU to a process that arrived first to the queue is fair. With this algorithm, a single linked list keeps track of all ready processes. Picking a process to run just requires removing one from the front of the queue. Adding a new job or unblocked process just requires attaching it to the end of the queue. First-come first-served is simple to understand and implement (Tanenbaum, 2009)

Limitations:

There is a convoy effect as all the other processes wait for the one big process to get off the CPU, poor utilization of the CPU, and turnaround time, waiting time and response time can be high, since long processes can hold the CPU.

2.6.4 Round Robin

Round Robin is one of the oldest, simplest, fairest, and most widely used algorithms. Each process is assigned a time interval, called its quantum, during which it is allowed to run. If the process is still running at the end of the quantum, the CPU is preempted and given to another process. If the process has blocked or finished before the quantum has elapsed, the CPU switching is done when the process blocks. Round Robin is easy to implement (Tanenbaum, 2009).

The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. To implement Round Robin scheduling, the ready queue is kept as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process. (Silberschatz *et al.*, 2009).

Limitations:

Setting the quantum too short causes too many process switches resulting to extensive overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short interactive requests and high average waiting time.

2.7 Related Works

Empirical Framework to Mitigate Problems in Longer Job First Scheduling Algorithm LJF+CBT: (Abdullahi and Junaidu, 2013) in their work proposed a framework which was developed on a model called Combinational Burst Time (CBT), see Figure 2.5.

The model reduced the Average Waiting Time (AWT), Average Turn-Around Time (ATAT) and number of Context Switches of LJF.

In the model,

N is a set of process X in a pool, i.e $N=X_i, X_j, \ldots, X_n$

δ is the burst associated with each process

λ is the time a process is assigned to the CPU

$Y_i\delta_i = X_i\delta_i + X_j\delta_j$ are shorter jobs that will be merged if they satisfy the merging condition.

n is the threshold measure.

In the model below, a pool N contains jobs. The jobs are sorted in descending order of the process burst time, this is in stage 1. The threshold **n** is the Combined Weighted Average (Cwa) of burst time of the processes. Any process that has burst time less or equal to **n** is categorized as a short job which are combined to form a longer job, this is done in stage 2. The new merged job and the longer job will be sorted again in descending order in the ready queue this is done in stage 3. The jobs are allocated to the CPU in stage 4.

They conducted experiment with the CBT model and without the CBT model, they used evaluation metrics of Average Waiting Time (AWT), Average Turn-Around Time (ATAT) and number of Context Switch.

They carried out simulation using empirical data analysis. It was found that LJF+CBT performed better than FCFS and LJF scheduling algorithms using evaluation metrics of AWT, ATAT and CS.

Limitation: Average Waiting Time and Average Turn-around Time are moderately high.

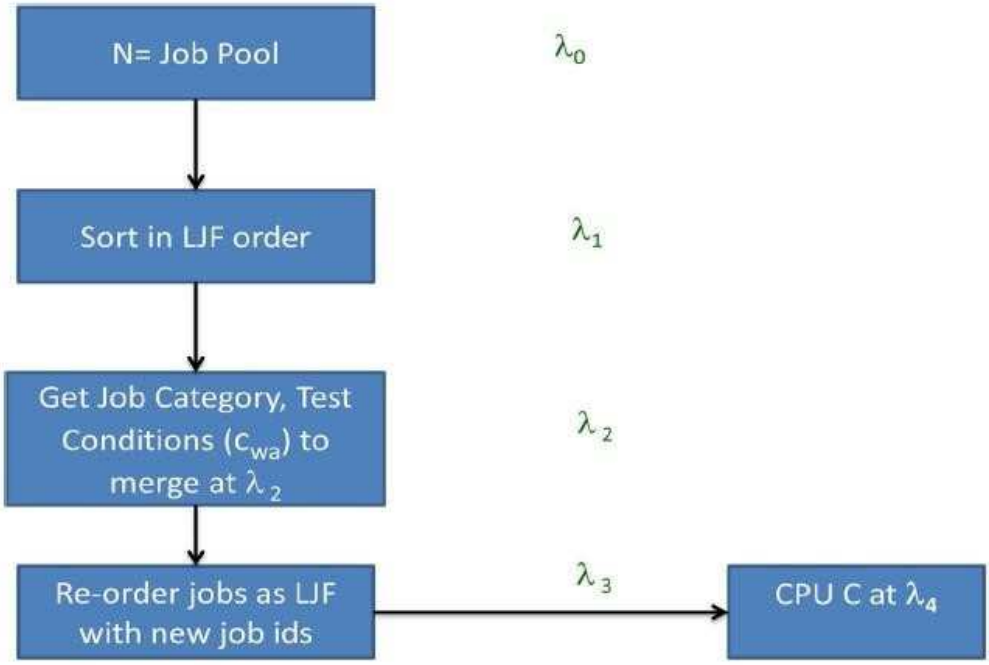


Figure 2.5: Model Architecture (Abdullahi and Junaidu, 2013)

(Shyam and Nandal, 2014), in their paper proposed a new algorithm that uses Round Robin with Shortest Job First scheduling. The Time Quantum(TQ) was studied to improve the efficiency of RR and to reduce the number of context switching, average wait time and average turnaround time.

This approach was proposed to calculate the Time Quantum, known as square root of mean and Highest Burst multiplied values, $TQ = \sqrt{mean * Highest\ Burst\ Time}$. The processes sorted in ascending order of their shortest remaining burst time and then Time Quantum are given to the process. Then the proposed algorithm performs better than Round Robin (RR), Improved Round Robin (IRR), FCFS and some other scheduling algorithm in terms of reducing the number of context switches, average turnaround time and average waiting time.

This approach is similar to the approach that was adopted in this thesis work in the sense that it used Longest Job First with the non-preemptive Shortest Job First in allocating the CPU to the processes in the ready queue.

(Behera *et al.*, 2010) in their proposed approach, arranged the processes present in the ready queue in ascending order of their burst times. The time quantum was calculated. For finding an optimal time quantum, they used the median method.

$$median\ \tilde{x} = \begin{cases} Y_{(n+1)/2}, if\ n\ is\ odd \\ 1/2\ \left[y_{n/2} + y_{1+\frac{n}{2}}\right],\ if\ n\ is\ even \end{cases} \quad 2.1$$

Where, $\tilde{x}$ = median

y = number located in the middle of a group of numbers arranged in ascending order

n = number of processes.

The time quantum was assigned to the processes, then after each cycle the time quantum of the remaining burst times was recalculated and assigned to the processes. In the next step the sorted processes were rearranged such that the process which needs minimum CPU burst time was placed as the first process and then the process with highest CPU burst time from the queue, was placed as the second process and so on in that order.

Their proposed algorithm performed better than RR scheduling algorithm with respect to average waiting time, turnaround time and context switching.

The median method approach is similar to the approach that was adopted in this thesis work in the sense that it calculates the threshold using median method and used it to separate shorter jobs form longer jobs which are arranged in longest job order.

Mohan in (Mohan, 2009) used the job mix order for non-preemptive scheduling First Come First Serve (FCFS) and Shortest Job First (SJF). According to job mix order, from a list of N processes, the process which needs minimum CPU time is executed first and then the highest from the list and so on till the nth process.

This approach is similar to the approach that was adopted in this thesis work in the sense that it used the job mix order of non-preemptive Longest Job First and Shortest Job First in allocating the CPU to the processes in the ready queue.

In Behera *et al*., 2012, the processes are arranged in ready queue in ascending order according to their scheduling time. To get optimised result for time quantum, median of the burst time of processes on the ready queue is calculated. The median is calculated as follows:

$$median = \begin{cases} 1/2(Y_{(N+1)} + Y_{(N+1)/2} & \text{if N is even} \\ Y_{(N+1)/2} & \text{if N is odd} \end{cases} \qquad 2.2$$

The scheduling time is calculated as follows:

Scheduling time= (burst time* burst task component) + (priority * priority task component);

The determinant factor is calculated as follows:

Determinant factor = (max of scheduling time of processes + min of scheduling time of processes)/2; 2.3

Time quantum is calculated as follows:

Time quantum = (median + determinant factor)/2;

Time quantum is assigned to each process. Median is recalculated with remaining scheduling time after each execution of each cycle. In the next step, the process which needs minimum CPU burst time was placed as the first process and then the process with highest CPU burst time from the queue, was placed as the second process and so on in that order.

The median method approach is similar to the approach that was adopted in this thesis work in the sense that this work calculates the threshold using median method and used it to separate shorter jobs form longer jobs which are arranged in longest job order.

Rami in (Rami, 2009) proposed a new algorithm based on a new approach called dynamic-time-quantum; the idea was to make the time quantum repeatedly adjusted according to the burst time of the now-running processes. This was done by analyzing the process to identify its burst time, this analysis was carried out only once when the process executed for the first time, without the need to be replicated, except in rare cases such as the program had been changed, modified, or updated since the last analysis.

The analysis determined the burst time of the process and accordingly the operating system adapts itself by readjusting the value of the time-slice or time quantum Q to commensurate with the set of the programs in the ready queue. When an operating system is installed for the first time, it begins with a default time quantum value, which is subject to change after a period of time through which the operating system can identify the burst time for a subset of the programs used by the

user. So, he assumed that the system will not immediately take advantage of this method because it needs a short period of time to learn user behaviour through the analysis of the burst time of the new processes. The determined time quantum represents real and optimal value because it based on real burst time.

Repeatedly, when a new process loaded to be executed the operating system tests the status of the specified program which can be either 1 or 0. When the status equals to 0 this means that the process is either being executed for the first time or it has been modified or updated since the last analysis. In this case the operating system assigns a counter to find the burst time of the process and continues executing the processes in the ready queue on the current round including the new arrived process using the current time quantum Q, otherwise and when status is equal to 1, then the operating system recalculates the time quantum Q depending on the remaining burst time of all ready processes including the newly arrived process. He used median to get the optimal time quantum for the set of processes in the ready queue, if the median was less than 25 then its value must be modified to 25 to avoid the overhead of the context switch. Formula 1 represents the value of time quantum Q consequences for the median $\tilde{x}$:

$$Q = \tilde{x} \equiv \begin{cases} Y_{(N+1)/2} & \text{if N is odd} \\ \frac{1}{2}\left(Y_{(N+1)/2}\right) + (Y_{(1+N/2)}) & \text{if N is even} \end{cases} \quad 2.4$$

where, Y is the number located in the middle of a group of numbers arranged in ascending order. Because the value of Q should not be less than 25, he rewrite formula (1) in more specific form to fit with the allowed range:

$$Q = \begin{cases} \tilde{x}\,, if\ \ \tilde{x} \geq 25 \\ 25, if\ \ \tilde{x} < 25 \end{cases} \quad 2.5$$

The approach is similar to the approach that was adopted in this thesis work in the sense that this work calculates the threshold using median method and used a flag 0 and 1 to indicate the status of the processes in the ready queue.

The aim of processor scheduling is to assign processes to be executed by the processor or processors over time, in a way that meets system objectives, such as waiting time, turnaround time, response time, throughput, and processor efficiency. Fundamentally, scheduling is a matter of managing queues to minimize queueing delay and to optimize performance in a queueing environment (Stallings, 2012). An efficient scheduling algorithm will result in high CPU utilization, throughput and response time while sustaining a low average waiting time, average turnaround time and number of context switches. This research focused on reducing average waiting time, average turnaround time and context switches in Longest Job First scheduling algorithm. Mohan, (Mohan, 2009) proposed 'Mixed Scheduling (A New Scheduling Policy)' and Behera *et al*., (Behera *et al.*, 2012) proposed 'Advanced Mix Job With Dynamic Quantum Round Robin (AMDRR)', said that there is no universal "best" scheduling algorithm, and many operating systems are using extended or combinations of the scheduling algorithms. Mohan and Behera's works offered part of the solution to high average waiting time and average turnaround time. Rami, in (Rami, 2009) proposed 'Self-Adjustment-Round-Robin (SARR)' as part of the solution to high number of context switches. This research proposed an Enhanced Combinational Burst Time which enhanced the Abdullahi and Junaidu's (Abdullahi and Junaidu, 2013) Combinational Burst Time model with the aim of reducing the average waiting time, average turnaround time and number of context switches in Longest Job First scheduling algorithm.

The task was to make the number of shorter jobs merged dynamic instead of a static number and to devise a different way of assigning CPU to the jobs in the waiting queue. The research considered some factors which included arrival time, burst time, job identity and a special flag **F**. The special flag was used to signify the type of job been executed by the CPU whether it was a short or long job, which helped the CPU scheduler to identify the job being executed and to know the next job to allocate to the CPU. This special flag is a new technique and stand this contribution to knowledge out, these will help to reduce the average waiting time, turnaround time and minimize the number of context switching.

Median was used to calculate the threshold value because the median is a robust statistic of central tendency, that is, it is a robust estimator capable of coping with outliers (Wikipeadia, 2016), while the mean is not since outliers can skew the mean (Taylor, 2015). The burst time used in this research was randomly generated therefore there are bound to be outliers in the data set, so median is the most appropriate statistic for this data set.

CHAPTER THREE

DESIGN OF THE ENHANCED LONGEST JOB FIRST ALGORITHM WITH COMBINATIONAL BURST TIME

3.1 Introduction

This chapter presents the proposed Enhanced Longest Job First CPU scheduling algorithm with CBT, an illustration on how it works, its Architecture and Pseudo code.

3.2 The Proposed Algorithm

The proposed ELJF+CBT CPU scheduling algorithm is the enhancement of the Longest Job First with Combinational Burst Time (LJF+CBT) CPU scheduling algorithm proposed by (Abdullahi and Junaidu, 2013) which was aimed at reducing the average waiting time, average turuaround time and also minimize the number of context switches.

This proposed (ELJF+CBT) algorithm introduced a special flag **F** used to indicate job order, LJF (if flag is **0**) and SJF (if flag is **1**) to identify them the CPU scheduler to select. When the flag **F** was set to **0** signified that the job being executed by the CPU was a long job, which helped the CPU scheduler to identify that the job being executed was a long job and to know that the next job to allocate to the CPU would be a short job. In the same way, when the flag **F** was set to **1** signified that the job being executed by the CPU was a short job, which helped the CPU scheduler to identify that the job being executed was a short job and to know that the next job to allocate to the CPU would be a long job.

The algorithm first of all sorts a pool of **N** processes X in descending order of their burst times δ into another pool **Q**.

N = $\{X_i\delta_i, X_{i+1}\delta_{i+1}, \ldots, X_n\delta_n\}$, where i starts from 0 and n can be any number.

At the second phase, the threshold value **H**, that is, the median of the sorted burst times of the processes in the pool **Q** was calculated and used as a test condition for the processes to be categorized as long job or short job, see Equation 3.1.

$$\text{medianValue H} = \begin{cases} \delta_{\frac{n+1}{2}}, & \text{if n is odd} \\ {}^{1}/_{2}\left[\delta_{n/2} + \delta_{1+\frac{n}{2}}\right], & \text{if n is even} \end{cases} \qquad 3.1$$

At the third phase, processes were categorized as short job or long job based on the result from the comparison of the threshold value **H** and the burst times, $\delta_i \leq$ **H**. All Processes with burst time less than or equal to **H** were categorized as short jobs and were moved to a temporary pool **temp** while processes with burst time greater than **H** were categorized as long jobs and remained in the pool **Q**.

temp ={ $X_h\delta_h, X_{h+1}\delta_{h+1}, \ldots, X_k\delta_k$ }, short jobs were defined in term of H

At the fourth phase, two adjacent jobs burst times were added together to form a new job in the temporary pool **temp** ($Y_i\delta_i = X_h\delta_h + X_{h+1}\delta_{h+1}$) and the new formed burst time was compared with the threshold value **H** ($\delta_i <$ **H**). If the burst time was less than the threshold value **H**, there would be another merging between the new process burst time and the next job burst time in the **temp** ($Y_i\delta_i = Y_i\delta_i + X_{h+2}\delta_{h+2}$), and there would be another comparison between the new process burst time and the threshold value **H**, until the new formed burst time was greater than or equal the threshold value **H** or the **temp** was empty then it would be inserted into the pool **Q** and another new merging process would start.

On the other hand if after the first comparison the new process burst time is equal or greater than the threshold value **H** ($\delta_i \geq$ **H**), the new process formed would be inserted into the pool **Q** and another new merging process would start.

There may be a situation where one process is remaining in the **temp** after merging the other processes burst times, in that case, that process would still be inserted to the pool **Q**.

The comparison of the new formed process burst time with the threshold value **H** was carried out here to make the number of processes merged to be dynamic. Therefore it served as a constraint to the number of merging to be done, so that merging of processes burst times would not be without limit or fixing the number of merging to be done to a static number.

The Combinational Burst Time (CBT) takes place in the **temp** which results to new processes with a high burst time and new process id. This is the fourth phase.

All the processes in **temp** with their new ids will be moved to **Q**. This is the fifth phase.

Finally in phase six, the CPU was assigned to the processes in the pool **Q** in interleave order.

All the processes in the pool **Q**, the new combined processes inclusive, are assigned the CPU in alternate burst time order, i.e. the process with the highest burst time would be the first followed by the process with the shortest burst, then followed by the process with the next highest burst time and process with next shortest burst time, in that order till all the processes in the pool **Q** are assigned. This is achieved with the use of the Special flag **F** which is set to 0 when long job was assigned to CPU and 1 when short job was assigned to CPU. (Mohan, 2009), (Behera *et al.*, 2012).

3.2.1 The Architecture of the Proposed ELJF+CBT

Figure 3.1 shows the architecture of the proposed Enhanced Longest Job First + CBT scheduling algorithm, the red shaded figures are the enhancements made to Abdullahi and Junaidu's architecture.

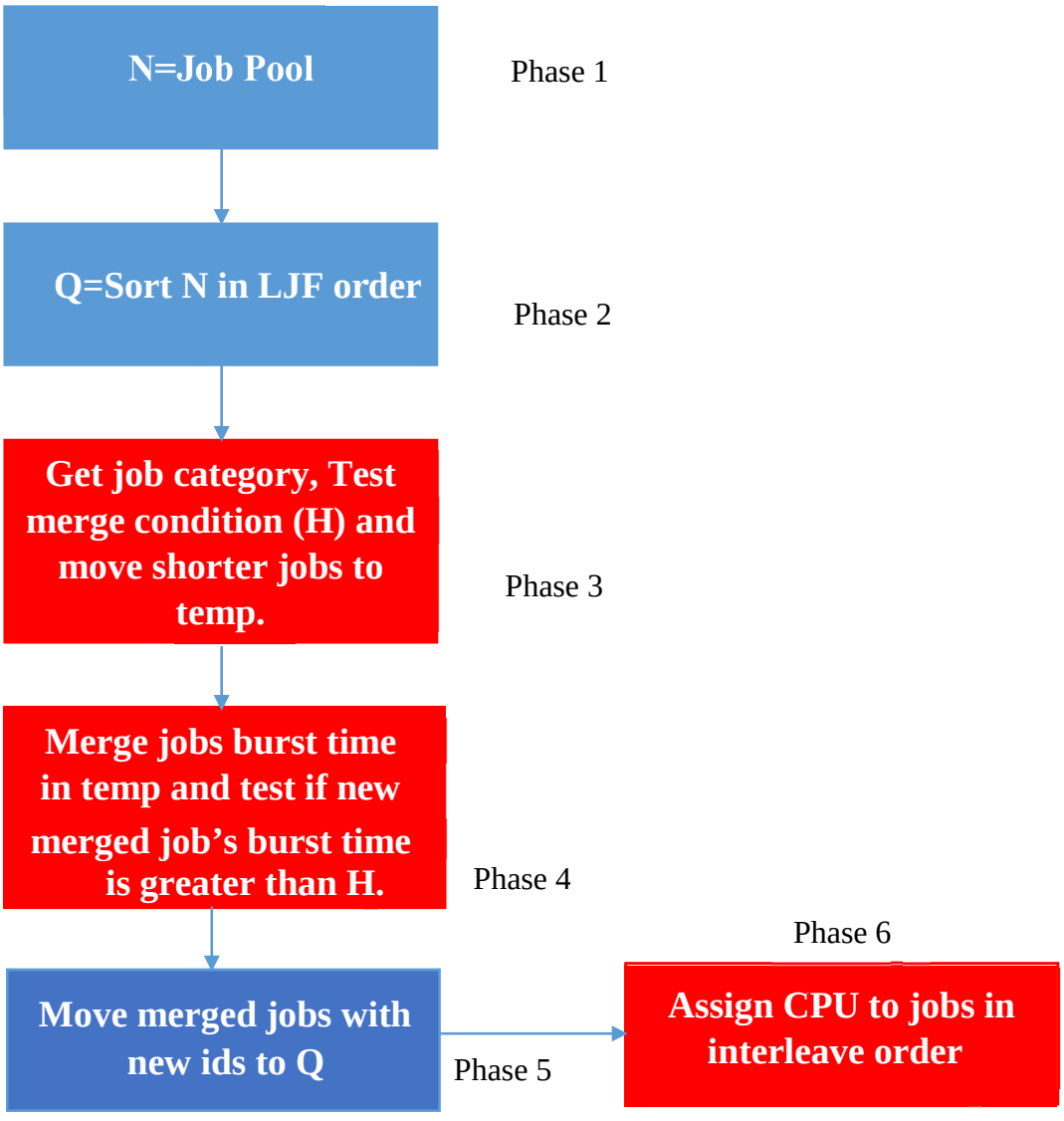


Figure 3.1: ELJF+CBT Architecture

3.2.2 Pseudo Code of the Proposed ELJF+CBT

// INPUT: AN ARRAY N [(X_0, δ_0), (X_1, δ_1), …, (X_{n-1}, δ_{n-1})]//

//OUTPUT: AN ARRAY Q[(X_0, δ_0), (X_1, δ_1), …(X_{m-1}, δ_{m-1})]//

1. Q = SORT(N) in descending order;
2. temp //a temporary array;
3. H=median; //median of the sorted burst times of the processes in the pool Q

   i = Q.length-1; F = 0;
4. for n = 0 to Q.length-1

   { if Q[n]< H

   j=n;

   break};

   end for;
5. while j ≤ i {

   temp.insert (Q[j]) // move jobs with burst time less or equal to H into temp

   Q[j] = null

   j++ };
6. t = null

   for i = 0 to temp.length-1

   {

   t = t + temp(i).δ_i //merge t with job burst time in temp(i)

   if(t.δ_i >= H) or (i = temp.length-1) { // and test if t.δ_i greater or

   Q.insert(t) // equal to median value **H** or end of the temp

   t = null

};

};

7. for i = 0 to Q.Length-1 {

   If **F**=0

   {assign longer job to the CPU and reset F to 1}; //reset flag to 1

   else

   {assign shortest job to the **CPU** and set **F** to 0}; //set flag is 0

};

8. End.

3.3 Design for the algorithm of the proposed ELJF +CBT

1. Generated CPU burst time of processes and assigned ids to them using Java random number generator,
2. The threshold was calculated using the median of burst times of the processes in the ready queue,
3. Process was categorized as short or long job based on the threshold value **H**,
4. Shorter jobs burst times were merged and compared with the threshold value **H** to form a new longer job with new id, and with burst time greater than threshold **H**,
5. A special flag **F** was used to indicate job order,
6. Jobs were assigned to the CPU using mix job order i.e. longer job first, shorter job then followed by the next longer job,
7. Gantt's chart was used to analyze and measure evaluation criteria of the algorithms,
8. The new algorithm (ELJF+CBT) was compared with FCFS, SJF, LJF and LJF+CBT.

3.4 Tools and Programming Languages Used

1. Java: as Programming language,
2. NetBeans IDE: as the integrated development environment,
3. Windows Script: to run the program in batch,
4. Hewlett Packard (HP) laptop with Intel core i5 processor running at 2.50GHz

CHAPTER FOUR

IMPLEMENTATION AND RESULTS PRESENTATION OF THE ELJF+CBT ALGORITHM

4.1 Introduction

This chapter presents the implementation analysis of the Enhanced Longest Job First with Combinational Burst Time CPU scheduling algorithm. It starts with the assumptions made in the system design; followed by the description of how the various scheduling algorithms under study work with the aid of an illustrative example. Finally, the system implementation follows; which was tested using processes that vary between 10 and 500. The results of this implementation were analyzed.

Assumptions

1. Our experiments were performed in a uni-processor environment,
2. The processes considered were CPU bound processes only and
3. All processes arrived at the same time.

4.2 Analytical Example

This frame work creates new process identification for the combined burst time through joining the address of the processes as one. Example: a process P19 combines with P14 gives a new identification as *P19,14. Where the '*' implies it is a CBT process and the numbers represents the original identification number of the processes separated by a comma (,) (Abdullahi and Junaidu, 2013).

*Table 4.1: 20 processes with their burst times (X_i,δ_i) submitted to the pool **N***

Process name	Arrival Time	Burst Time
P0	0	35
P1	0	38
P2	0	35
P3	0	22
P4	0	13
P5	0	17
P6	0	38
P7	0	38
P8	0	37
P9	0	14
P10	0	35
P11	0	34
P12	0	40
P13	0	26
P14	0	26
P15	0	36
P16	0	20
P17	0	5
P18	0	14
P19	0	28

By sorting the processes in Table 4.1 by their burst times in Longest Job First order, the following result were obtained, see Table 4.2.

Table 4.2: Processes (Xδ) sorted from highest burst time to the lowest

P12=40	P7=38	P6=38	P1=38	P8=37
P15=36	P2=35	P10=35	P0=35	P11=34
P19=28	P14=26	P13=26	P3=22	P16=20
P5=17	P9=14	P18=14	P4=13	P17=5

To execute the CBT model by (Abdullahi and Junaidu, 2013), the threshold **H** of the processes burst times in Table 4.2 was calculated using the median, see Equation 3.1. This was used to categorize jobs into long jobs, which are processes with burst time greater than the threshold **H;** and short jobs, which are processes with burst time less than or equal to the threshold **H**.

Using Equation 3.1,

$$medianValue = \begin{cases} \delta_{\frac{n+1}{2}}, & if\ n\ is\ odd \\ {}^{1}/_{2}\left[\delta_{n/2} + \delta_{1+\frac{n}{2}}\right], & if\ n\ is\ even \end{cases}$$

Since n is even,

$$medianValue = {}^{1}/_{2}\left[\delta_{n/2} + \delta_{1+\frac{n}{2}}\right]$$

$$medianValue = {}^{1}/_{2}\left[34 + 28\right]$$

$$medianValue = 31$$

the threshold value **H** is 31

Table 4.3: Processes with burst times greater than the threshold H

P12=40	P7=38	P6=38	P1=38	P8=37
P15=36	P2=35	P10=35	P0=35	P11=34

Table 4.4: Processes with burst times less than or equal to the threshold H

P19=28	P14=26	P13=26	P3=22	P16=20
P5=17	P9=14	P18=14	P4=13	P17=5

The 10 process in Table 4.4 were merged to form 5 new processes. P9, P18 and P4 were merged based on 'methodology 4' of chapter 3. After merging the burst time of the processes categorize as shorter processes, the new processes and their burst times are shown in Table 4.5.

Table 4.5: New formed processes with their burst times

*P19,P14=54	*P13,P3=48	*P16,P5=37	*P9,P18,P4=41	P17=5

Collate the new (merged) processes and the unmerged processes, see Table 4.6.

Table 4.6: Collation of processes after evaluation of CBT model

*P19,P14=54	*P13,P3=48	*P9,P18,P4=41	P12=40	P7=38
P6=38	P1=38	P8=37	*P16,P5=37	P15=36
P2=35	P10=35	P0=35	P11=34	P17=5

Result obtained from Table 4.6 will be rearranged using mix job order (i.e. longer job first, shorter job then followed by the next shorter job), based on 'methodology 6' of chapter 3, and then will be assigned to the CPU. See Table 4.7.

Merging more than two processes and assigning the CPU to the processes in LJF order gives a good AWT and ATAT. However, merging more than two processes and assigning the CPU to the processes in interleaved order gives a better AWT and ATAT. This informed our assigning the CPU to the processes in interleaved order. See Figures 4.1 and 4.2 respectively.

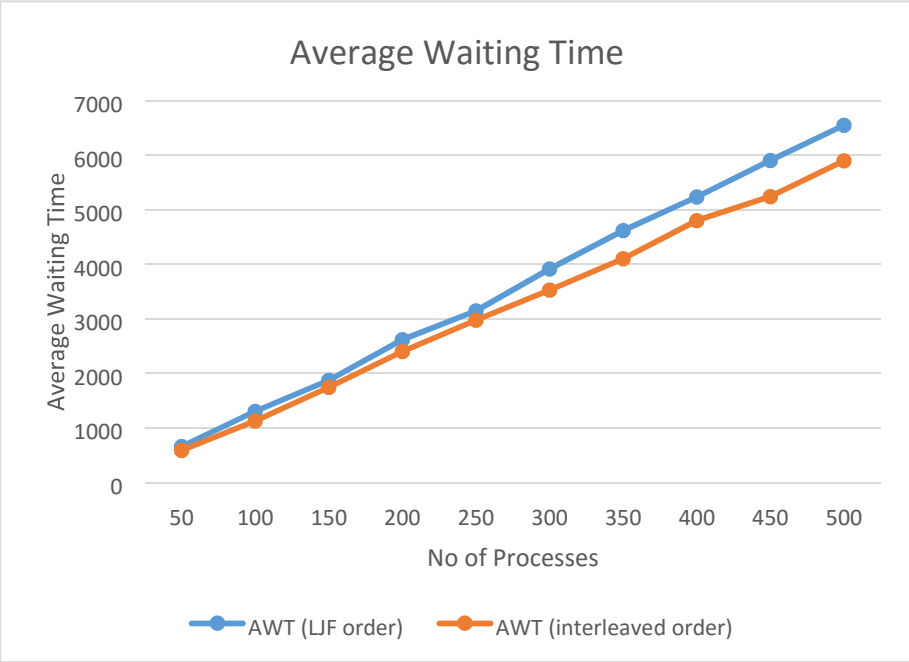


Figure 4.1: Comparison of AWT for LJF order and interleaved order

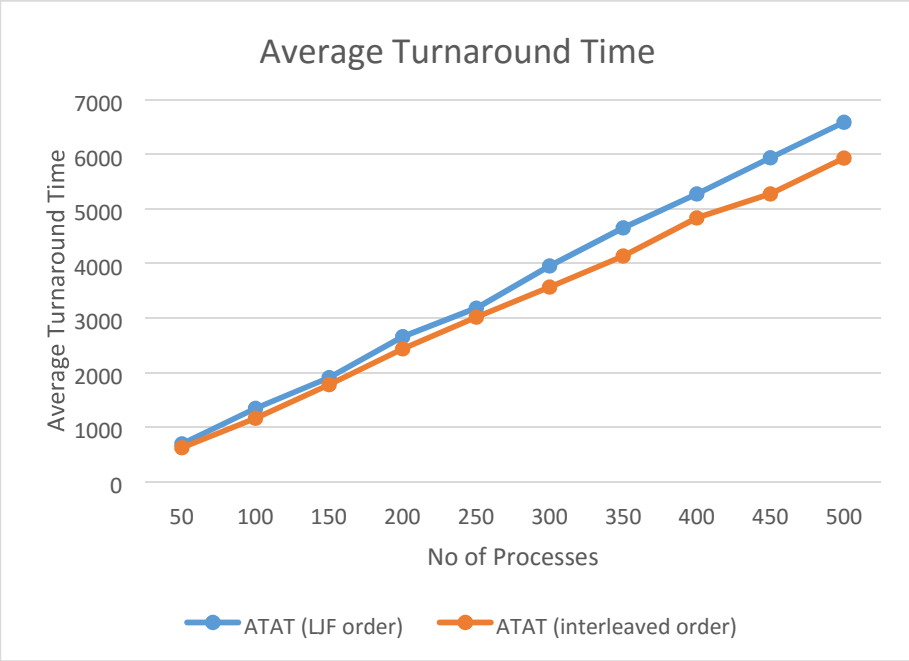


Figure 4.2: Comparison of ATAT for LJF order and interleaved order

Table 4.7: processes arranged in interleaved order

*P19,P14=54	P17=5	*P13,P3=48	P11=34	*P9,P18,P4=41
P0=35	P12=40	P10=35	P7=38	P2=35
P6=38	P15=36	P1=38	*P16,P5=37	P8=37

4.3 Performance Evaluation Metrics

To evaluate the performance of the proposed scheduling algorithm (Enhanced Longest Job First+CBT), the Average Waiting Time, Average Turn-Around Time and Context Switch would be calculated.

Waiting Time: according to (Silberschatz *et al.*, 2009), The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/0; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is sum of periods spent waiting in the ready queue.

$$Waiting\ Time\ (WT) = first\ scheduled\ time - arrived\ time \quad 4.1$$

Every good scheduling algorithm always has the aim of reducing the average waiting time of processes in the ready queue.

$$\boldsymbol{Average\ Waiting\ Time\ (AWT) = Total\ Waiting\ Time/\ total\ number\ of\ processes} \quad 4.2$$

Turn-Around Time: according to (Silberschatz *et al.*, 2009), the interval from the time of submission of a process to the time of completion. From the point of view of a particular process, how long it takes to execute that process is an important criterion.

$$TurnAround\ Time\ (TAT) = completion\ time - Arrival\ time \quad 4.3$$

$$Average\ Total\ TurnAround\ Time\ (ATAT) = \frac{Total\ TurnAround\ Time}{Total\ number\ of\ processes} \quad 4.1$$

Minimum average turn-around time is good performance index.

4.4 Experimental Cases of the Scheduling Algorithm

To demonstrate the proposed algorithm, the following example were considered, 10 processes were used, and each process had its process name/id, burst and arrival time as shown in Table 4.8 below. All processes were considered to arrive at the same time.

Table 4.8: 10 Processes with process id, burst time and arrival time

Process Name	Arrival Time	**Burst Time**
P0	0	8
P1	0	17
P2	0	40
P3	0	11
P4	0	8
P5	0	21
P6	0	6
P7	0	24
P8	0	19
P9	0	30

4.4.1 First Come First Serve (FCFS)

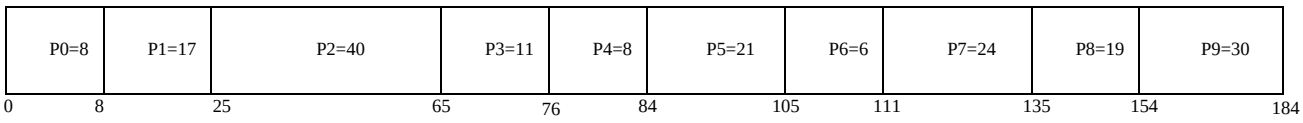


Figure 4.3: Gantt chart for FCFS

To find the average waiting time (AWT), equation 4.1 was used.

Waiting Time (WT) = (P8:0-0), (P17: 8-0), (P2:25-0), (P3:65-0), (P4:76-0), (P5:84-0), (P6:105-0), (P7:111-0), (P8:135-0), (P9:154-0)

Total Waiting Time (TWT)= ((0-0)+(8-0)+(25-0)+(65-0)+(76-0)+(84-0)+(105-0)+(111-0)+(135-0)+(154-0))

Total Waiting Time (TWT) = 763

Applying Equation 4.2, Average Waiting Time (AWT) =763/10

Average Waiting Time (AWT) =76.3

Using Equation 4.3,

Turn-around Time (TAT) = (P8:8-0), (P17: 25-0), (P2:65-0), (P3:76-0), (P4:84-0), (P5:105-0), (P6:111-0), (P7:135-0), (P8:154-0), (P9:184-0)

Total Turn-around Time (TTAT) = ((8-0)+(25-0)+(65-0)+(76-0)+(84-0)+(105-0)+(111-0)+(135-0)+(154-0)+ (184-0))

Total Turn-around Time (TTAT) = 947

Applying Equation 4.4, Average Turn-around Time (ATAT) = 947/10

Average Turn-around Time (ATAT) = 94.7

Context Switch=9

4.4.2 Shortest Job First (SJF)

Table 4.9: 10 processes and their Burst Times, Sorted in SJF order of their burst times

Process Name	Burst Time
P6	6
P0	8
P4	8
P3	11
P1	17
P8	19
P5	21
P7	24
P9	30
P2	40

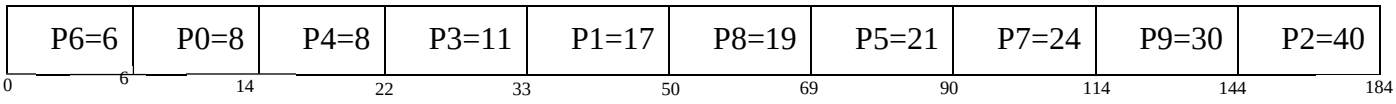


Figure 4.4: Gantt chart for SJF

To find the average waiting time (AWT), equation 4.1 was used.

Waiting Time (WT) = (P6:0-0), (P0:6-0), (P4:14-0), (P3:22-0), (P1:33-0), (P8:50-0), (P5:69-0), (P7:90-0), (P9:114-0), (P2:144-0)

Total Waiting Time (TWT) = ((0-0)+(6-0)+(14-0)+(22-0)+(33-0)+(50-0)+(69-0)+(90-0)+(114-0)+(144-0))

Total Waiting Time (TWT) = 542

Applying Equation 4.2, Average Waiting Time (AWT) = 542/10

Average Waiting Time (AWT) = 54.2

Using Equation 4.3,

Turn-around Time (TAT) = (P6:6-0), (P0: 14-0), (P4: 22-0), (P3: 33-0), (P1: 50-0), (P8: 69-0), (P5: 90-0), (P7: 114-0), (P9: 144-0), (P2:184-0)

Total Turn-around Time (TTAT)= ((6-0)+(14-0)+(22-0)+(33-0)+(50-0)+(69-0)+(90-0)+(114-0)+(144-0)+ (184-0))

Total Turn-around Time (TTAT) = 726

Applying Equation 4.4, Average Turn-around Time (ATAT) = 726/10

Average Turn-around Time (ATAT) = 72.6

Context Switch=9

4.4.3 Longest Job First (LJF)

Table 4.10: 10 processes and their Burst Times, Sorted in LJF order of their burst times

Process Name	Burst Time
P2	40
P9	30
P7	24
P5	21
P8	19
P1	17
P3	11
P0	8
P4	8
P6	6

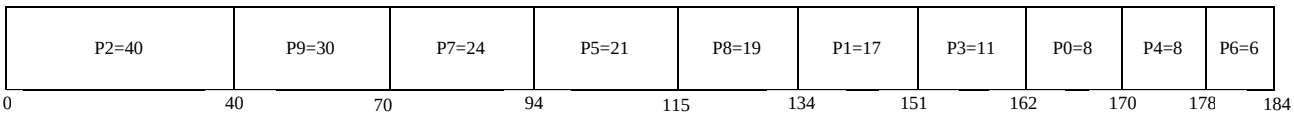


Figure 4.5: Gantt chart for LJF

To find the average waiting time (AWT), equation 4.1 was used.

Waiting Time (WT) = (P2:0-0), (P9: 40-0), (P7:70-0), (P5:94-0), (P8:115-0), (P1:134-0), (P3:151-0), (P0:162-0), (P4:170-0), (P6:178-0)

Total Waiting Time (TWT) = ((0-0)+ (40-0)+(70-0)+ (94-0)+ (115-0)+ (134-0)+ (151-0)+(162-0)+ (170-0)+(178-0))

Total Waiting Time (TWT) = 1114

Applying Equation 4.2, Average Waiting Time (AWT) =1114/10

Average Waiting Time (AWT) =111.4

Using Equation 4.3,

Turn-around Time (TAT) = (P2:40-0), (P9: 70-0), (P7:94-0), (P5:115-0), (P8:134-0), (P1:151-0), (P3:162-0), (P0:170-0), (P4:170-0), (P6:184-0)

Total Turn-around Time (TTAT) = ((40-0)+(70-0)+ (94-0)+ (115-0)+ (134-0)+ (151-0)+(162-0)+ (170-0)+(178-0)+(184-0))

Total Turn-around Time (TTAT) = 1298

Applying Equation 4.4, Average Turn-around Time (ATAT) = 1298/10

Average Turn-around Time (ATAT) = 129.8

Context Switch=9

4.4.4 Longest Job First +CBT (LJF+CBT)

Using (Abdullahi and Junaidu, 2013) CBT model, $C_{wa} = \frac{\sum_i^n \delta_i}{n}$

from Table 4.10, Cwa = (40+30+24+21+19+17+11+8+8+6)/10

Cwa =184/10

Cwa =18.4

Testing for merging condition,

Table 4.11: 8 Processes formed after merging shorter processes

Process Name	Burst Time
P2	40
P9	30
P7	24
P5	21
P8	19
*P1, P3	28
*P0, P4	16
P6	6

Processes P1, P3, P0, P4 and P6 met the condition merging i.e. their burst times are less than or equal to the Cwa (≤ 18), and resulted to '*P1, P3', '*P0, P4' and P6 was not merged because there is no other processes that met the merging condition in the pool.

Table 4.12: showing Table 4.11 sorted in LJF order

Process Name	Burst Time
P2	40
P9	30
*P1, P3	28
P7	24
P5	21
P8	19
*P0, P4	16
P6	6

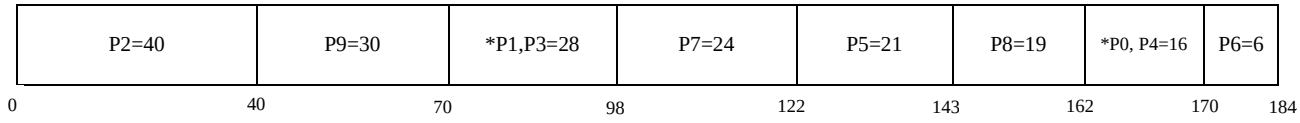


Figure 4.6: Gantt chart for LJF+CBT

To find the average waiting time (AWT), equation 4.1 was used.

Waiting Time (WT) = (P2:0-0), (P9: 40-0), (*P1,P3:70-0), (P7:98-0), (P5:122-0), (P8:143-0), (*P0,P4:162-0), (P6:178-0)

Total Waiting Time (TWT) = ((0-0)+(40-0)+(70-0)+(98-0)+(122-0)+(143-0)+ (162-0)+(178-0))

Total Waiting Time (TWT) = 813

Applying Equation 4.2, Average Waiting Time (AWT) =813/10

Average Waiting Time (AWT) =81.3

Using Equation 4.3,

Turn-around Time (TAT) = (P2:40-0), (P9:70-0), (*P1,P3:98-0), (P7:122-0), (P5:143-0), (P8:162-0), (*P0,P4:178-0), (P6:184-0)

Total Turn-around Time (TTAT) = ((40-0)+(70-0)+(98-0)+(122-0)+(143-0)+ (162-0)+(178-0)+(184-0))

Total Turn-around Time (TTAT) = 997

Applying Equation 4.4, Average Turn-around Time (ATAT) = 997/10

Average Turn-around Time (ATAT) = 99.7

Context Switch=7

4.4.5 Enhanced Longest Job First with CBT (ELJF+CBT)

Using (Abdullahi and Junaidu, 2013) CBT model, Equation 3.1 which is the median of the burst times of the processes in the ready queue was used to calculate the threshold value **H**.

Using Equation 3.1,

Since n is even form Table 4.10,

$$medianValue = {}^{1}/_{2}\left[\delta_{n/2} + \delta_{1+\frac{n}{2}}\right]$$

$$medianValue = {}^{1}/_{2}\,[19 + 17]$$

$$medianValue = 18$$

the threshold value **H** is 18, processes with burst time less than or equal to 18 are merged to form new processes.

Table 4.13: 7 Processes formed after merging shorter processes

Process Name	Burst Time
P2	40
P9	30
P7	24
P5	21
P8	19
*P1,P3	28
*P0, P4, P6	22

Processes P1, P3, P0, P4 and P6 met the condition merging i.e. their burst times are less than or equal to the threshold (≤ 18), and resulted to '*P1, P3' and '*P0, P4, P6' were merged based on 'methodology 4' of chapter three.

Result obtained from Table 4.13 are rearranged using mix job order (i.e. longer job first, shorter job then followed by the next shorter job), see methodology 6 of chapter three.

Table 4.14: showing Table 4.13 sorted in alternate order of burst time

Process Name	Burst Time
P2	40
P8	19
P9	30
P5	21
*P1,P3	28
*P0, P4, P6	22
P7	24

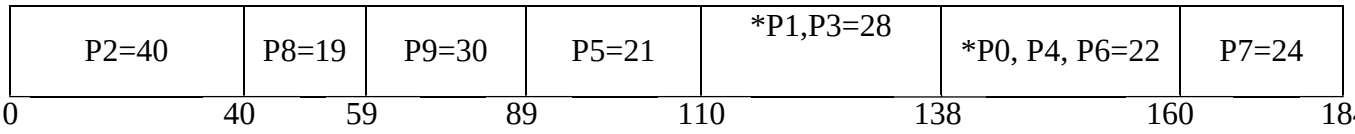


Figure 4.7:Gantt chart for ELJF+CBT

To find the average waiting time (AWT), equation 4.1 was used.

WT for P2 (0-0)= 0, WT for P8 (40-0)= 40, WT for P9 (59-0) = 59, WT for P5(89-0)= 89, WT for *P1,P3(110-0)= 110, WT for *P0, P4, P6 (138-0)= 138 and WT for P7(160-0) = 160.

Total Waiting Time (TWT) = 0+40+59+89+110+138+160

Total Waiting Time (TWT) = 596

Applying Equation 4.2,

Average Waiting Time (AWT) = 596/10

Average Waiting Time (AWT) =59.6

Using Equation 4.3,

TAT for P2 (40-0) = 40, TAT for P8 (59-0) = 59, TAT for P9 (89-0) = 89, TAT for P5 (110-0) = 110,

TAT for *P1,P3(138-0)= 138, TAT for *P0, P4, P6 (160-0)= 160 and TAT for P7(184-0) = 184.

Total Turn-Around Time (TTAT) = 40+59+89+110+138+160+184

Total Turn-Around Time (TTAT) = 780

Applying Equation 4.4,

Average Turn-Around Time (ATAT) =780/10

Average Turn-Around Time (ATAT) =78.0

Context Switch=6

4.5 Comparing ELJF+CBT with some Existing Scheduling Algorithms

This new proposed scheduling algorithm was compared with the following existing scheduling algorithm:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Longest Job First (LJF)
4. Longest Job First +CBT (LJF+CBT)

The evaluation metrics used were average waiting time (AWT), average turn-around time (ATAT) and context switch (CS).

4.5.1 Comparing ELJF+CBT and FCFS

ELJF+CBT performed better than First Come First Serve scheduling algorithm in terms of AWT and ATAT because it has a lower AWT, ATAT and number of context switches. ELJF+CBT's smaller number of context switches is a big advantage because context switch has a high overhead cost as shown in Table 4.15.

Table 4.15: Comparison of ELJF+CBT and FCFS for 500 processes after 12 executions

Name of Algorithm	AWT	ATAT	CS
FCFS	5633	5655	499
Enhanced LJF+CBT	5122	5153	327

4.5.2 Comparing ELJF+CBT and SJF

ELJF+CBT performed better than SJF in terms of context switch because it has a lower number of context switches. This is a big advantage over SJF because context switch has a high overhead cost as shown in Table 4.16.

However, Shortest Job First scheduling algorithm has a better AWT and ATAT than ELJF+CBT, this is because from the literature in 'chapter two' and other literatures SJF has a provably optimal average waiting time when arrival time is the same.

Table 4.16: Comparison of ELJF+CBT and SJF for 500 processes after 12 executions

Name of Algorithm	AWT	ATAT	CS
Enhanced LJF+CBT	5122	5153	327
SJF	4098	4120	499

4.5.3 Comparing ELJF+CBT and LJF

ELJF+CBT performed far better than Longest Job First scheduling algorithm in terms of AWT and ATAT. It also has a much lower number of context switches which is a big advantage because context switch has a high overhead cost as shown in Table 4.17.

Table 4.17: Comparison of ELJF+CBT and LJF for 500 processes after 12 executions

Name of Algorithm	AWT	ATAT	CS
Enhanced LJF+CBT	5122	5153	327
Longest Job First	7120	7143	499

4.5.4 Comparing ELJF+CBT and LJF+CBT

ELJF+CBT performed better than Longest Job First+CBT scheduling algorithm in terms of AWT,

ATAT and context switch. ELJF+CBT has a lower AWT, ATAT and number of context switches which means that ELJF+CBT has really enhanced LJF+CBT thereby offering a better solution to the problems of LJF as shown in Table 4.18.

Table 4.18: Comparison of ELJF+CBT and LJF+CBT for 500 processes after 12 executions

Name of Algorithm	AWT	ATAT	CS
Enhanced LJF+CBT	5122	5153	327
Longest Job First + CBT	6411	6441	374

4.6 Result Discussion and Conclusion

The results from the execution of the scheduling algorithms for 12 runs showed that ELJF+CBT performed better than LJF+CBT, LJF, and FCFS in terms of AWT, ATAT, and CS.

ELJF+CBT has a lower CS than SJF which is a major advantage for ELJF+CBT over SJF for the fact that CS has an overhead cost associated with it. Though SJF has a lower AWT and ATAT. See Table 4.19.

Table 4.19: Cumulative result of 500 processes after 12 executions.

Name of Algorithm	AWT	ATAT	CS
FCFS	5633	5655	499
Enhanced LJF+CBT	5122	5153	327
SJF	4098	4120	499
Longest Job First + CBT	6411	6441	374
Longest Job First	7120	7143	499

The Table 4.19 showed cumulative result of execution of the scheduling algorithm program for 500 processes and with burst time ranging from 5-50 executed for 12 times. The shaded row showing the result of the proposed algorithm.

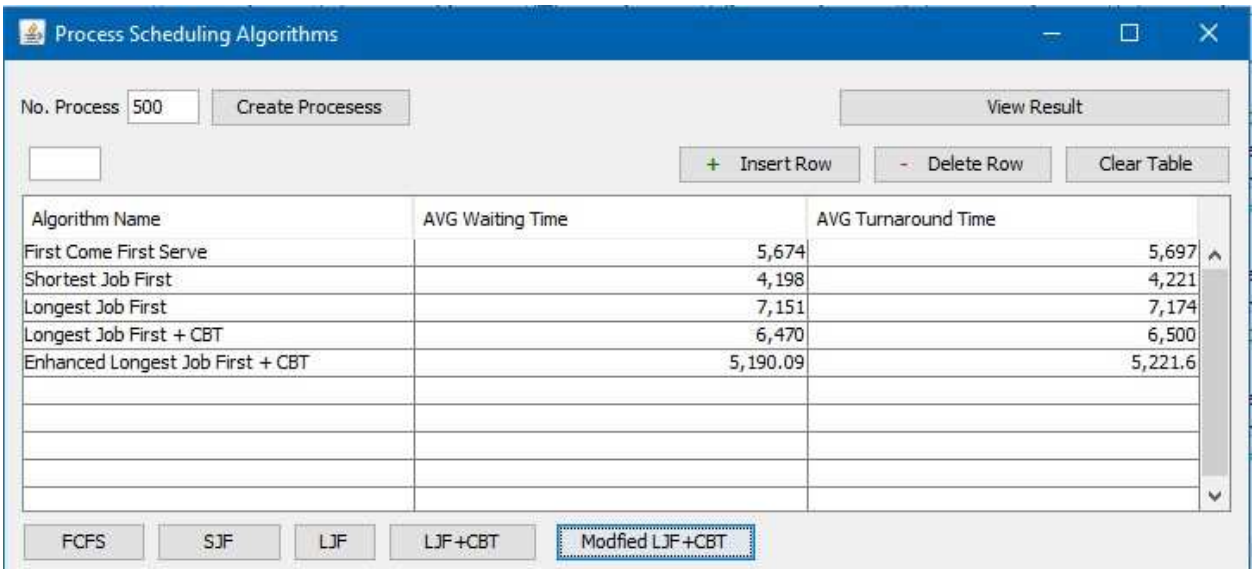


Figure 4.8: Snapshot of 500 processes after one execution

Windows batch script was used for execution of different ranges of process burst time for a number of times at once. (Figure 4.9).

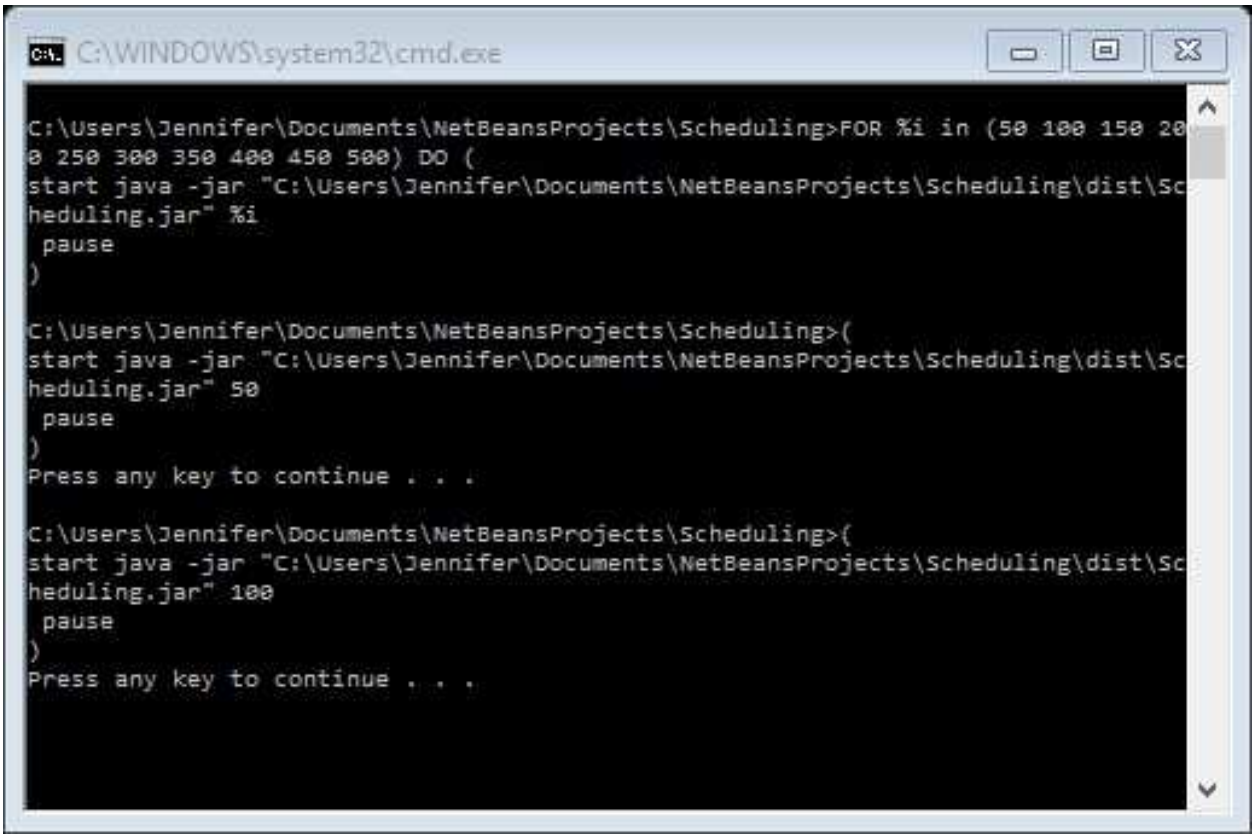
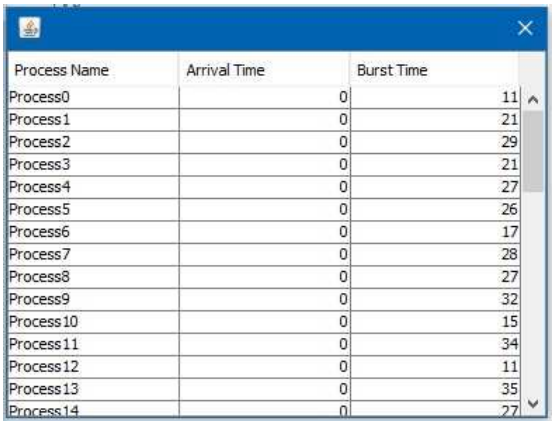


Figure 4.9: Snapshot of execution of process scheduling using windows batch scripting



Process Name	Arrival Time	Burst Time
Process0	0	11
Process1	0	21
Process2	0	29
Process3	0	21
Process4	0	27
Process5	0	26
Process6	0	17
Process7	0	28
Process8	0	27
Process9	0	32
Process10	0	15
Process11	0	34
Process12	0	11
Process13	0	35
Process14	0	27

Figure 4.10: Snapshot of processes with their Burst Times and Arrival Times

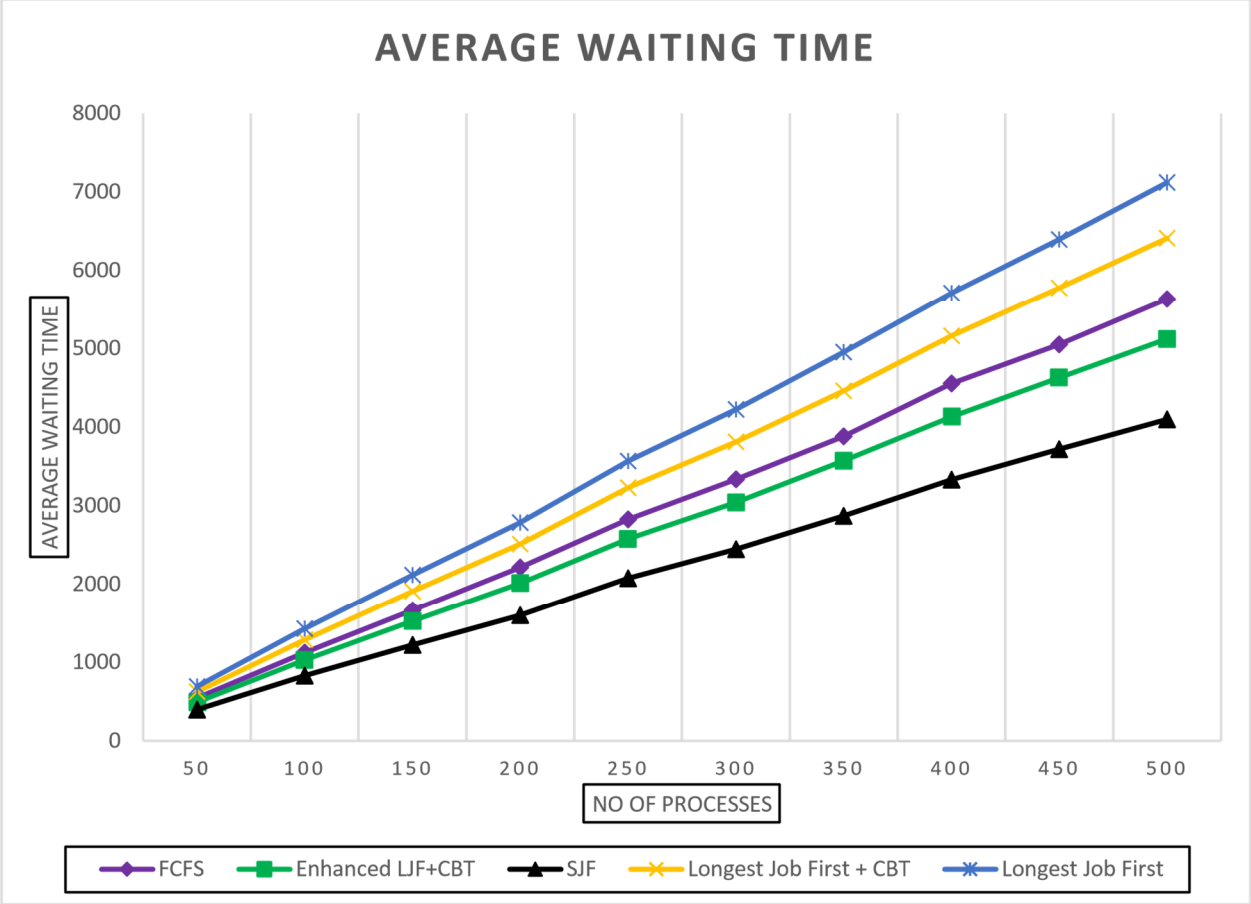


Figure 4.11: Graph of Average Waiting Times for a sample process (50-500)

The Figure 4.11 shows a graphical representation of the all the result of average waiting time for all the scheduling algorithms used, using the data from Table 4.20.

As can be seen from the graph, ELJF+CBT performed better than FCFS, LJF and LJF+CBT in terms of average waiting time, and SJF had lower AWT.

Table 4.20: Average Waiting Time data

Name of Algorithm	No of Processes									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
FCFS	545	1124	1665	2213	2825	3336	3881	4553	5049	5633
Enhanced LJF+CBT	491	1027	1526	2010	2578	3042	3571	4131	4626	5122
SJF	398	831	1220	1600	2078	2445	2871	3330	3717	4098
Longest Job First + CBT	624	1285	1908	2510	3227	3811	4462	5161	5776	6411
Longest Job First	691	1429	2116	2785	3567	4224	4954	5711	6396	7120

Table 4.20 shows the results generated form the execution of the scheduling algorithm for different ranges of processes between 50 and 500 and executed 12 times. The first objective of the research which was enhancing the LJF+CBT algorithm was achieved, with the value of AWT produced for ELJF+CBT lower than the value for LJF+CBT.

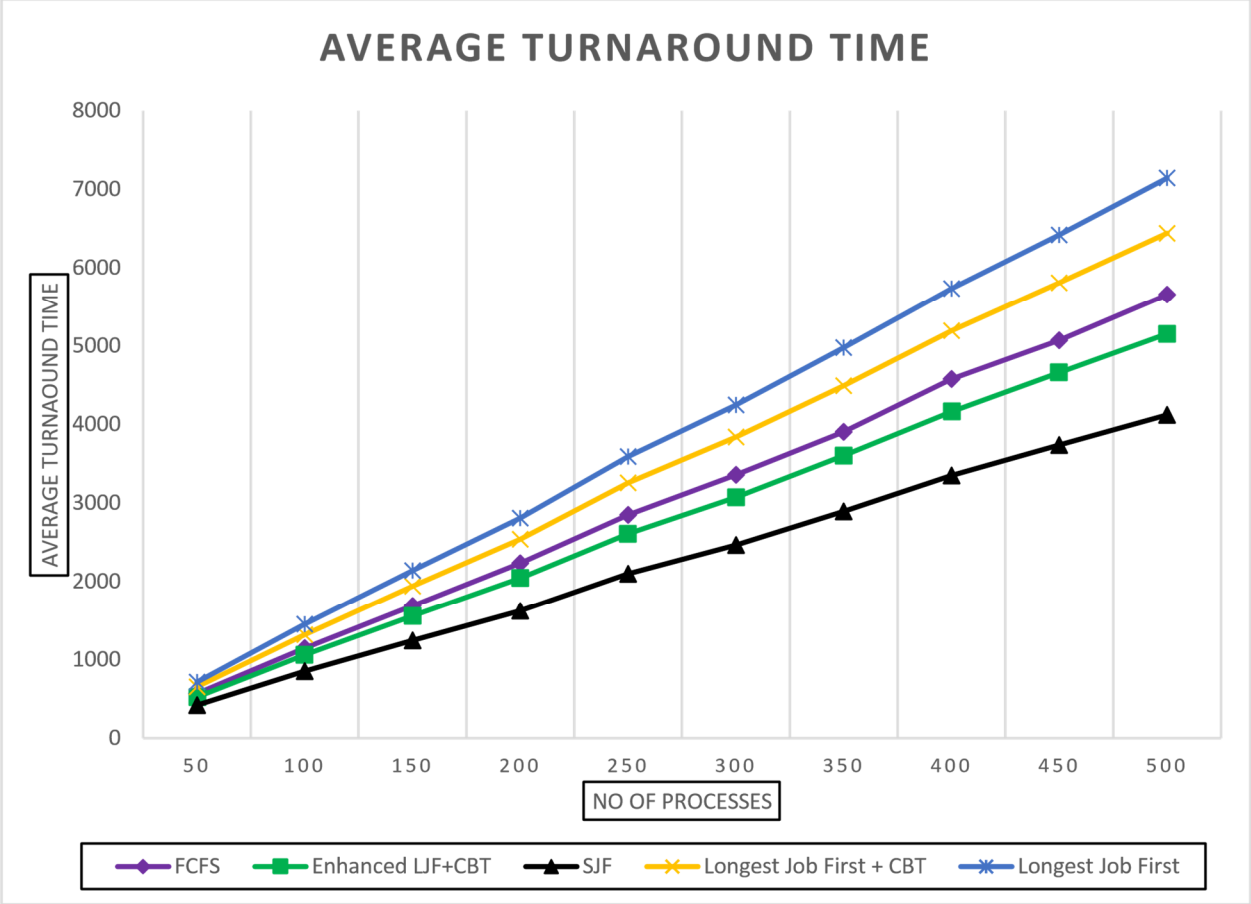


Figure 4.12: Graph of Average Turnaround Times for a sample process (50-500)

The Figure 4.12 shows a graphical representation of the all the result of average turn-around time for all the scheduling algorithms used, using the data from Table 4.21.

As can be seen from the graph, ELJF+CBT performed better than FCFS, LJF and LJF+CBT, and SJF had lower ATAT.

Table 4.21: Average Turnaround Time data

Name of Algorithm	**No of Processes**									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
FCFS	567	1147	1687	2235	2847	3358	3903	4576	5072	5655
Enhanced LJF+CBT	523	1059	1558	2040	2609	3073	3602	4163	4657	5153
SJF	420	853	1242	1622	2101	2468	2893	3352	3739	4120
Longest Job First + CBT	653	1315	1938	2539	3258	3841	4492	5191	5806	6441
Longest Job First	713	1451	2138	2807	3590	4246	4976	5733	6419	7143

Table 4.21 shows the results generated form the execution of the scheduling algorithm for different ranges of processes between 50 and 500 executed 12 times. One of the objectives of the research which was enhancing the LJF+CBT algorithm was achieved, with the value of ATAT produced for ELJF+CBT lower than the value for LJF+CBT.

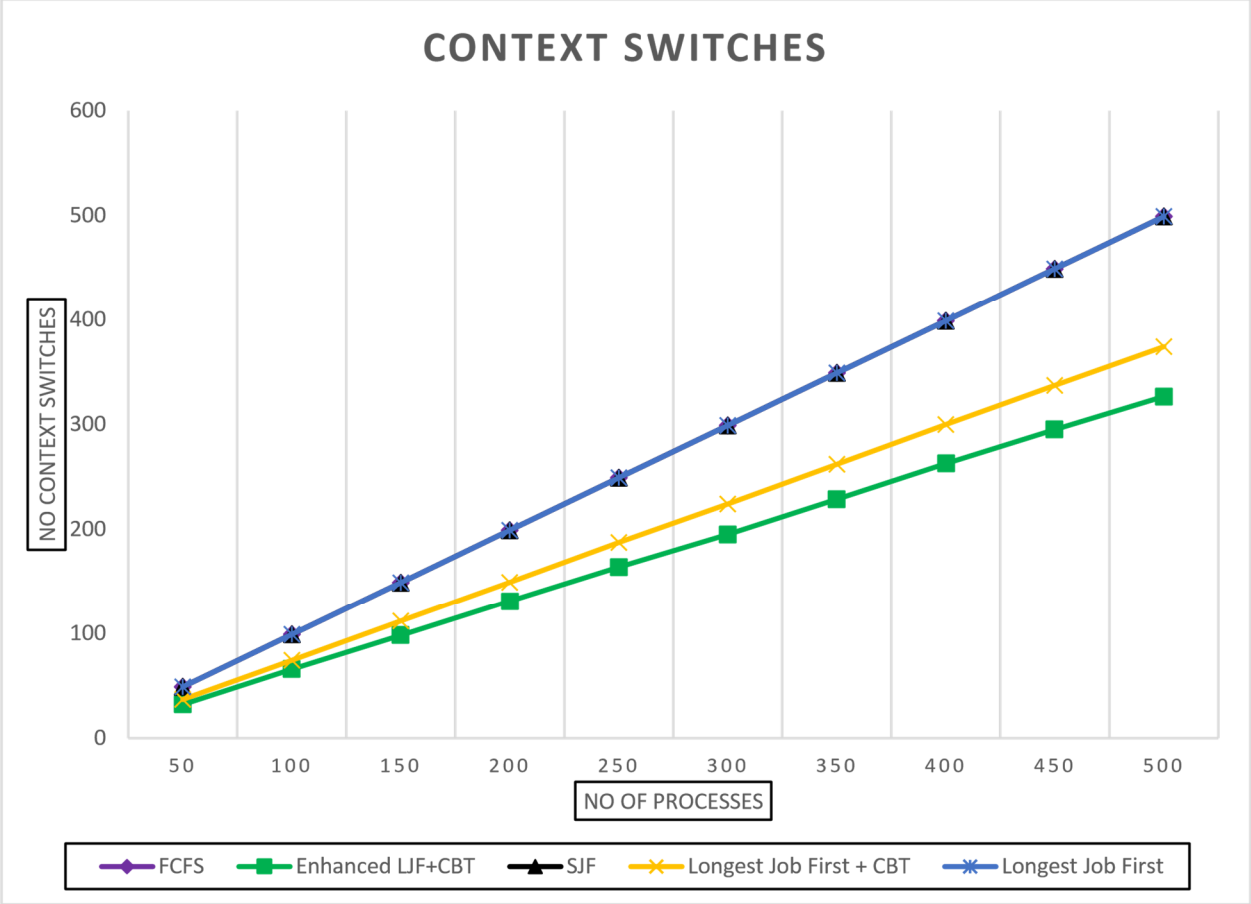


Figure 4.13: Graph of Context Switches for sample processes (50-500)

The Figure 4.13 shows a graphical representation of the all the result of context switch for all the scheduling algorithms used, using the data from Table 4.22.

As can be seen from the graph, ELJF+CBT performed better than SJF, FCFS, LJF and LJF+CBT.

Table 4.22: Context Switches data

Name of Algorithm	No of Processes									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
FCFS	49	99	149	199	249	299	349	399	449	499
Enhanced LJF+CBT	32	66	98	131	164	195	229	263	295	327
SJF	49	99	149	199	249	299	349	399	449	499
Longest Job First + CBT	37	74	112	149	187	224	262	300	337	374
Longest Job First	49	99	149	199	249	299	349	399	449	499

Table 4.22 shows the results generated form the execution of the scheduling algorithm for different ranges of processes between 50 and 500 executed 12 times. One of the objectives of the research which was enhancing the LJF+CBT algorithm was also achieved, with the value of Context Switches produced for ELJF+CBT lower than the value for LJF+CBT.

ELJF+CBT also has an advantage over SJF algorithm in terms of CS, because ELJF+CBT had the lowest CS for all the number of processes executed and all the scheduling algorithm considered in this research.

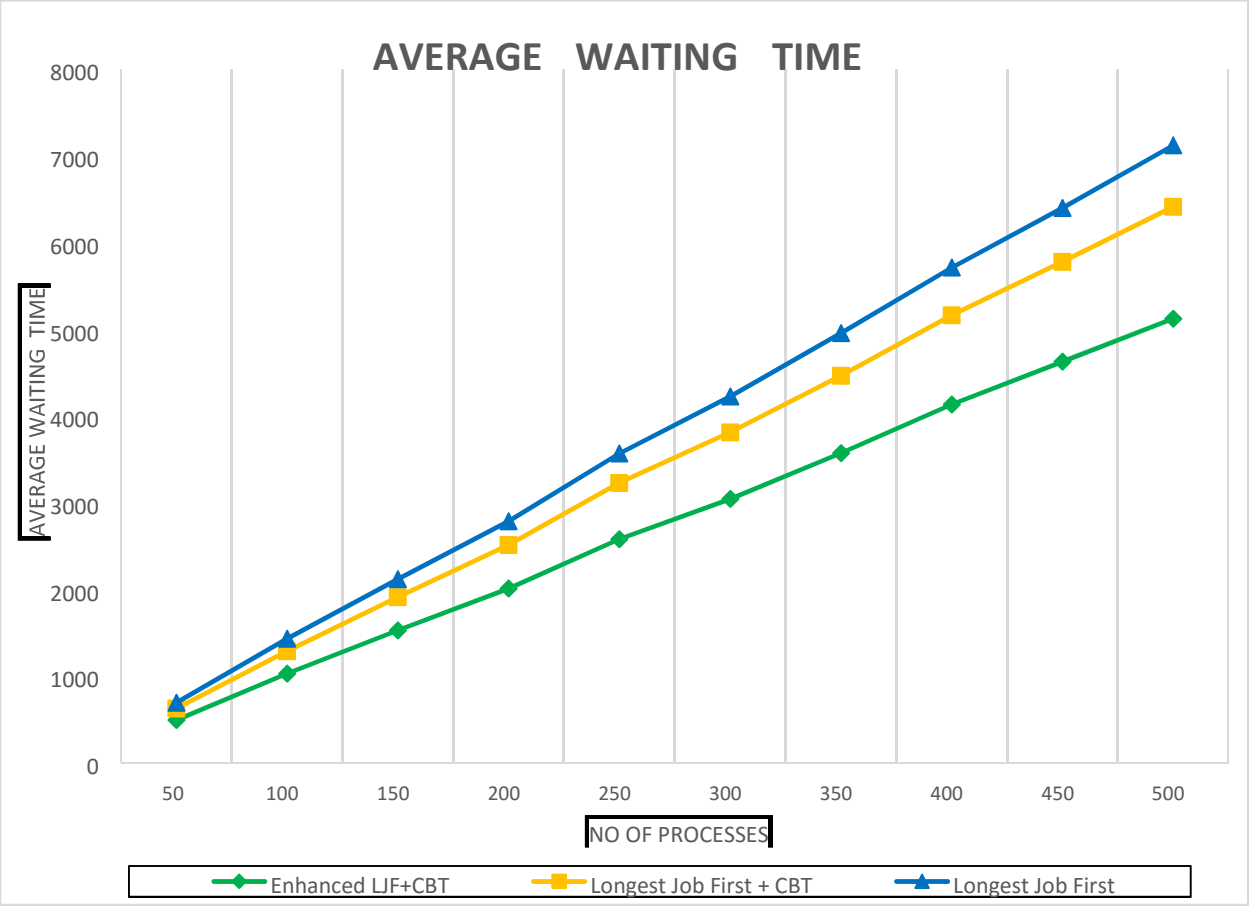


Figure 4.14: Graph of average waiting time for LJF, LJF+CBT and ELJF+CBT

The Figure 4.14 shows a graphical representation of the result of average waiting time for Enhanced LJF+CBT, LJF+CBT and LJF scheduling algorithms, using the data from Table 4.23.

As can be seen from the graph, ELJF+CBT performed better than LJF+CBT, and far better than LJF. This shows that the average waiting time of LJF was greatly reduced.

Table 4.23: Average waiting time data for LJF, LJF+CBT and ELJF+CBT

Name of Algorithm	No of Processes									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Enhanced LJF+CBT	491	1027	1526	2010	2578	3042	3571	4131	4626	5122
Longest Job First + CBT	624	1285	1908	2510	3227	3811	4462	5161	5776	6411
Longest Job First	691	1429	2116	2785	3567	4224	4954	5711	6396	7120

Table 4.23 shows the results generated form the execution of Enhanced LJF+CBT, LJF+CBT and LJF scheduling algorithm for different ranges of processes between 50 and 500 executed 12 times. The first objective of the research which was enhancing the LJF+CBT algorithm was achieved, with the value of AWT produced for ELJF+CBT lower than the value for LJF+CBT and LJF.

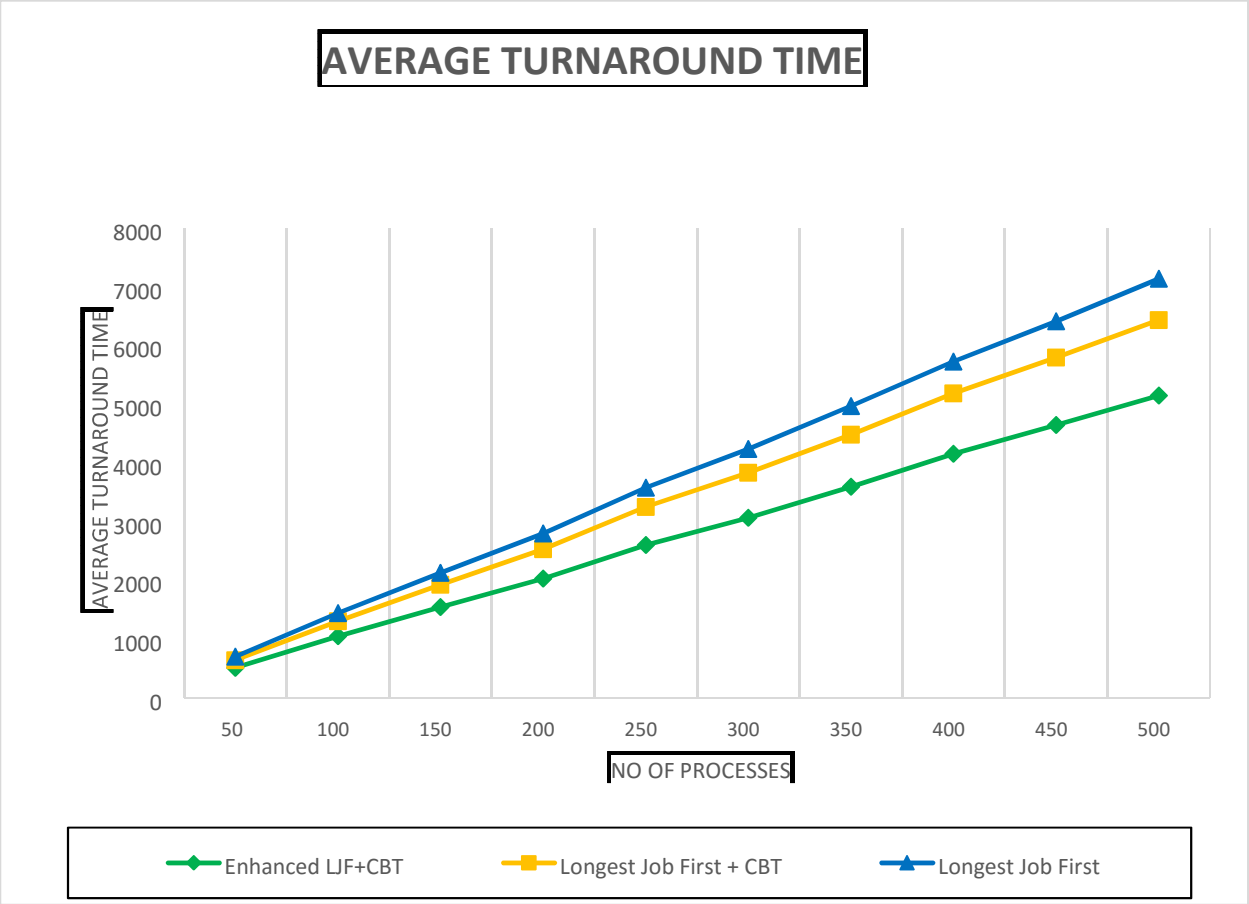


Figure 4.15: Graph of average turnaround time for LJF, LJF+CBT and ELJF+CBT

The Figure 4.15 shows a graphical representation of the result of average turnaround time for ELJF+CBT, LJF+CBT and LJF scheduling algorithms, using the data from Table 4.24.

As can be seen from the graph, ELJF+CBT performed better than LJF+CBT, and far better than LJF. This shows that the average turnaround time of LJF was greatly reduced.

Table 4.24: Average turnaround time data for LJF, LJF+CBT and ELJF+CBT

Name of Algorithm	**No of Processes**									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Enhanced LJF+CBT	523	1059	1558	2040	2609	3073	3602	4163	4657	5153
Longest Job First + CBT	653	1315	1938	2539	3258	3841	4492	5191	5806	6441
Longest Job First	713	1451	2138	2807	3590	4246	4976	5733	6419	7143

Table 4.24 shows the results generated form the execution of Enhanced LJF+CBT, LJF+CBT and LJF scheduling algorithm for different ranges of processes between 50 and 500 executed 12 times. The objective of enhancing the LJF+CBT algorithm was achieved, with the value of ATAT produced for ELJF+CBT lower than the value for LJF+CBT and LJF.

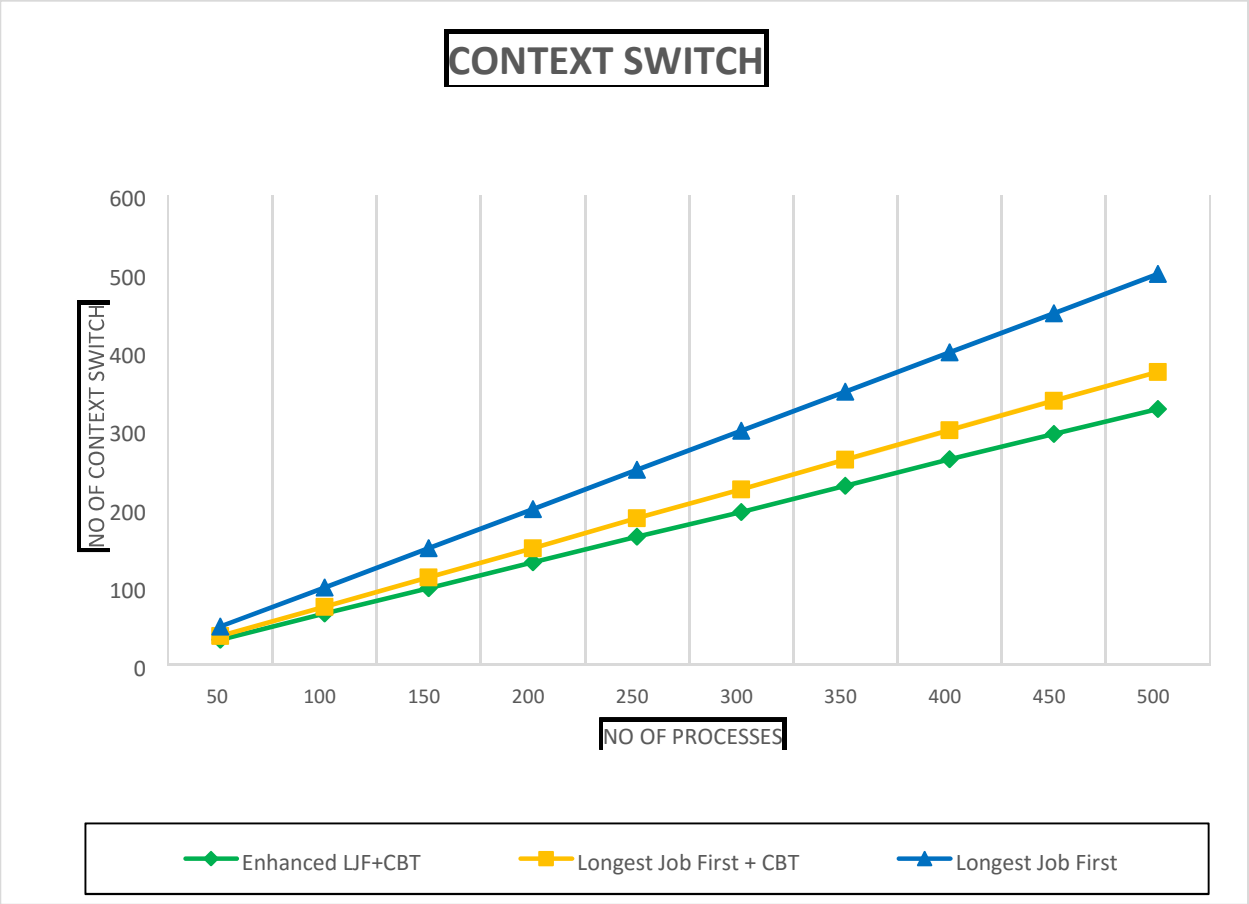


Figure 4.16: Graph of context switch for LJF, LJF+CBT and ELJF+CBT

The Figure 4.16 shows a graphical representation of the result of context switch for Enhanced LJF+CBT, LJF+CBT and LJF scheduling algorithms, using the data from Table 4.25.

As can be seen from the graph, ELJF+CBT performed better than LJF+CBT, and far better than LJF. This shows that the number of context switches of LJF was greatly reduced.

Table 4.25: Context switch data for LJF, LJF+CBT and ELJF+CBT

Context Switch										
Name of Algorithm	**No of Processes**									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Enhanced LJF+CBT	32	66	98	131	164	195	229	263	295	327
Longest Job First + CBT	37	74	112	149	187	224	262	300	337	374
Longest Job First	49	99	149	199	249	299	349	399	449	499

Table 4.25 shows the results generated form the execution of Enhanced LJF+CBT, LJF+CBT and LJF scheduling algorithm for different ranges of processes between 50 and 500 executed 12 times. The enhancing the LJF+CBT algorithm was reducing the number of contest switches was achieved, with the value of CS produced for ELJF+CBT lower than the value for LJF+CBT and far lower for LJF.

CHAPTER FIVE

Summary, Conclusion and Recommendation

This chapter presents the summary, conclusion and recommendation of this thesis work.

5.1 Summary

One of the most important components of the computer resource is the CPU. CPU scheduling involves a careful examination of pending processes to determine the most efficient way to service the requests. Many CPU scheduling algorithms have been presented with varying advantages and disadvantages.

In this research an Enhanced Longest Job First with Combinational Burst Time (ELJF+CBT) CPU scheduling algorithm was proposed. Simulation results showed that the proposed ELJF+CBT CPU scheduling algorithm gave better performance than the LJF+CBT in terms of average waiting time, average turnaround time and context switch.

After the enhancement of LJF+CBT, it was found that the average waiting time, average turnaround time and context switch were reduced.

5.2 Conclusion

The proposed algorithm (ELJF+CBT) was be able to minimize the Average Waiting Time by 26.69%, Average Turn-Around Time by 21.77% and also reduce the number of context switches by 14.29% in LJF+CBT. In Longest Job First (LJF) scheduling this algorithm drastically reduced the average waiting time by 46.5%, average turnaround time by 39.39% and number of context switching between processes by 33.33%.

Following these results obtained, the three objectives set by this work were achieved.

5.3 Future work

For future work as related to this research, a test of the ELJF+CBT with different arrival times for processes should be looked into.

5.4 Recommendation

We recommend that this proposed scheduling algorithm, ‘Enhanced Longest Job First with CBT’, should be implemented in an operating system especially those used in manufacturing industries, since as it was seen from the simulation results that it can compete favourably with other scheduling algorithms in a batch system.

REFERENCES

Abdullahi, I., and Junaidu, S. B. (2013). Empirical Framework to Mitigate Problems in Longer Job First Scheduling Algorithm LJF+CBT. *International Journal of Computer Applications (0975 – 8887), 75*(14), 9-14.

Behera, H. S., Bijayalaxmi, P., and Sreelipa, C. (2012). Design and Performance Evaluation of the Advanced Mix Job with Dynamic Quantum Round Robin Scheduling Algorithm for Real Time Systems. *Journal of Global Research in Computer Science, 3*(3), 46-53.

Bowen, L. (2015). *Scheduling Algorithms.* Retrieved May 23, 2016, from The University of Alabama Website: http://www.ctl.ua.edu/math103/scheduling/scheduling_algorithms.htm

Daley, R., and Dennis, R. (1968). Virtual Memory, Processes, and Sharing in MULTICS. *Communications of the ACM, 11*(5, Pages 306-312), 306-312.

Dipti, S., and Mittal, P. (2013). Job Scheduling Algorithm for Computational Grid in Grid Computing Environment. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(5), 735-743.

Dorit, S. H. (1999). *RIOT- The Scheduling Problem.* Retrieved January 2016, from Remote Interactive Optimization Testbed: http://people.tuke.sk/jan.paralic/Scheduling/algorithms.html

Forum, G. -P. (2010). *www.go4expert.com/articles/types-of-scheduling-t22307/.* (go4expert) Retrieved June 3, 2016, from www.go4expert.com: http://www.go4expert.com/articles/types-of-scheduling-t22307/

H.S.Behera, Mohanty, R., and Nayak, D. (2010). A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. *International Journal of Computer Applications, 5*(5), 10-15.

Mohan, S. (2009). Mixed Scheduling (A New Scheduling Policy). *Proceedings of Insight'09.*

Rami, J. M. (2009). Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes. *American Journal of Applied Sciences, 6*(10: 1831-1837), 1831-1837.

Shyam, R., and Nandal, S. K. (2014). Improved Mean Round Robin with Shortest Job First Scheduling. *International Journal of Advanced Research in Computer Science and Software Engineering, 4*(7), 170-179.

Silberschatz, A., Galvin, and Gagne. (2005). *Operating System Concepts* (Vol. Seventh Edition). hoboken, NJ: John Wiley and Sons Inc.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2009). *Operating System Concepts* (Vol. 8th Edition). hoboken, NJ: John Wiley and Sons Inc.

Stallings, W. (2012). *Operating Systems: Internals and Design Principles* (Vol. 7th Edition). NJ: Prentice Hall.

Tanenbaum, A. S. (2009). *Modern Operating Systems, Third Edition.* Upper Saddle River, NJ: Pearson Education International.

Taylor, C. (2015). *Finding the Average Mean, Median, Mode, and Midrange*. (About.com) Retrieved December 10, 2015, from About.com: http://statistics.about.com/od/HelpandTutorials/a/Ways-To-Find-The-Average.htm

Wikipeadia. (2016). *Outliers- wikipeadia, the free encyclopedia*. (Wikimedia Foundation, Inc) Retrieved July 13, 2016, from Wikipeadia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Outlier

Wikipedia. (2016). *Scheduling (computing)*. (Wikimedia Foundation, Inc) Retrieved June 1, 2016, from Wikipedia, The free Encylopedia: https://en.wikipedia.org/wiki/Scheduling_(computing)

APPENDIX I

```
//package scheduling;

import java.util.ArrayList; import
java.util.Random;

public class Scheduling {
   ArrayList<Process> process = new ArrayList<Process>(5);
double time;
   double avgWaitingTime, avgTurnaroundTime;
   String waitingTime = "( ", turnaroundTime = "( ";

   Scheduling(ArrayList<Process> process) {
this.process = process;        sortProcesses();
   }

   private void sortProcesses() {
      //sort the processes according to arrival time in ascending order
for (int i = 0; i < process.size() - 1; i++) {            for (int j = i + 1; j
< process.size(); j++) {
            if (process.get(i).arrivalTime > process.get(j).arrivalTime) {
Process p = process.get(j);                  process.set(j, process.get(i));
process.set(i, p);
            }
         }
      }
   }
```

```
  private double getAvgWaitingTime(ArrayList<Process> process) {
double[] waitTime = new double[process.size()];

    for (int i = 0; i < waitTime.length; i++) {
waitTime[i] = 0;
    }

    for (int i = 0; i < process.size(); i++) {
Process p = process.get(i);
        for (int j = 0; j < p.burstPos.size(); j++) {
try {
            waitTime[i] += p.burstPos.get(j).start - p.burstPos.get(j - 1).end;
            waitingTime += "+ (" + p.burstPos.get(j).start + " - " + p.burstPos.get(j - 1).end + ") ";
        } catch (ArrayIndexOutOfBoundsException e) {
waitTime[i] += p.burstPos.get(j).start - p.arrivalTime;
            waitingTime += "+ (" + p.burstPos.get(j).start + " - " + p.arrivalTime + ") ";
        }
      }
    }

    double avgWaitTime = 0;
    for (int i = 0; i < waitTime.length; i++) {
avgWaitTime += waitTime[i];
    }
    avgWaitTime /= process.size();
    waitingTime += ") / " + process.size();
```

```
        //remove the first '+' sign from the waitingTime string
int index = waitingTime.indexOf('+');
        waitingTime = waitingTime.substring(0, index) + waitingTime.substring(index + 2);

        return avgWaitTime;
    }

    private double getAvgTurnaroundTime(ArrayList<Process> process) {
int[] turnTime = new int[process.size()];

        for (int i = 0; i < turnTime.length; i++) {
turnTime[i] = 0;
        }

        double avgTurnTime = 0;         for
(int i = 0; i < process.size(); i++) {
Process p = process.get(i);
            avgTurnTime += p.burstPos.get(p.burstPos.size() - 1).end - p.arrivalTime;
turnaroundTime += "+ (" + p.burstPos.get(p.burstPos.size() - 1).end + " - " + p.arrivalTime + ") ";
        }
        avgTurnTime             /=             process.size();
turnaroundTime += ") / " + process.size();

        //remove the first '+' sign from the turnaroundTime string
int index = turnaroundTime.indexOf('+');         turnaroundTime =
turnaroundTime.substring(0, index) +
turnaroundTime.substring(index + 2);
```

```
        return avgTurnTime;
    }

    private    void    reset()    {
time = 0;
        avgTurnaroundTime = 0;
avgWaitingTime = 0;
waitingTime = "( ";
turnaroundTime = "( ";         for
(Process p : process) {
p.burstPos.clear();
        }
    }
    private void sortProcessesBurstDescending(ArrayList<Process> process) {
//sort the processes according to arrival time in ascending order         for (int i
= 0; i < process.size() - 1; i++) {             for (int j = i + 1; j < process.size();
j++) {
                if (process.get(i).burstTime < process.get(j).burstTime) {
Process p = process.get(j);                     process.set(j,
process.get(i));                    process.set(i, p);
                }
            }
        }
    }
    void modifiedLJFCBT() {}
```