PROCESS SCHEDULING IN LONGEST JOB FIRST (LJF) ALGORITHM:
A Proposed framework for Starvation Problem

BY

ABDULLAHI, Ibrahim

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA.

OCTOBER, 2012.

PROCESS SCHEDULING IN LONGEST JOB FIRST (LJF) ALGORITHM:
A Proposed framework for Starvation Problem

BY

ABDULLAHI, Ibrahim

M.Sc/SCIE/03388/2008-09

A Thesis submitted to the School of Postgraduate Studies, Ahmadu Bello University, Zaria

In partial fulfilment of the requirements for the Award of

Masters Degree in Computer Science.

Department of Mathematics, Faculty of Science

Ahmadu Bello University, Zaria, Nigeria

October, 2012.

Declaration

I hereby declare that the work in this thesis titled "Process Scheduling in Longest Job First (LJF) Algorithm: A Proposed framework for starvation problem" was performed by me in the Department of Mathematics, under the supervision of Prof. S. B Junaidu and Dr. S.E Abdullahi.

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.

ABDULLAHI, Ibrahim _____ _____

Name of Student Signature Date

Certification

This thesis titled "Process Scheduling in Longest Job First (LJF) Algorithm: A Proposed framework for starvation problem" meets the regulations governing the award of the degree of Master of Science of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Prof. S. B Junaidu	_____	_____
Major Supervisor	(Signature)	(Date)
Dr. S.E Abdullahi	_____	_____
Minor Supervisor	(Signature)	(Date)
_____	_____	_____
External Examiner	(Signature)	(Date)
Dr. B. Sani	_____	_____
Head of Department	(Signature)	(Date)
Prof Adebayo A. Joshua	_____	_____
Dean, School of Postgraduate	(Signature)	(Date)

Dedication

This research is dedicated to my family Surv. (Dr) A.Y Mahmouds.

Acknowledgement

Glory is to Allah (SWT) the Lord of the World, for giving me the gift of life and health to make this work a success.

My wholehearted appreciation and immense gratitude goes to my computer paragon Professor S.B Junaidu for diligently going through this work to make it a success and worthwhile. This research would have been difficult if not impossible without the likes of my supervisors. Dr S.E Abdullahi my Minor supervisor who showed me the interest of choosing Operating System as an area of research.

I must acknowledge the scholarly preparations given to me by my Masters Degree instructors: Prof. S. B Junaidu, Prof. O.S Adewale, Prof. (Dr) M.I Seyed Buhari and Dr. S.E Abdullahi whom I owe a debt of gratitude and pray Gods guidance as their rewards.

I wouldn't have been where I yearn to be without the support of my role model, my father Surv. (Dr) A.Y Mahmoud whose encouragement never stopped to see the completion of this work; Daddy, I say may Jannah be your final abode. To my beloved Mother, my siblings I owe you a big gratitude for standing by me through out.

I acknowledge my encourager Dr A. A. Obiniyi for making me set a time for the completion of the work. My brother, classmate and a friend Mal. Abdullahi Muhammad (ABU, Zaria); without you, the java codes wouldn't have been easy to handle; I pray Allah to bless you and reward your efforts, Mal. Barroon I.A, Mr. Fred H.K, Hajiya Hindatu Muhammad, Mustapha Bagiwa, Yusuf Y, Mr. D.F Kana, Khalil, Maj. G.E Ugwu, Mal U.I Abdullahi, Mal S. Dishing ('threshold I must remember'), Nnamani whom all amid your tight schedules you took time to criticize and made relevant encouragements to my work, my entire M.Sc classmates; I say a big thank you.

To all staff of Iya Abubakar computer center, ABU Zaria for providing the benevolent environment. Mal Nura Dari, Mal Abdulrahman, Mr Yakubu Friday to mention but a few.

Dr. M.S Ramadan, Salisu M.K Nusa, Lt. M.B, Abduljalal, Kabir Mali, Muhammad Kudu, Dr Raji Y, Dr. Musa Mustapha, Dr Aisha, Amit Mishra, U.M Gana, David.I L, Abdulganiyu and Sadiq Gado, lest I forget, the entire staff of the Department of Mathematics/Computer Science of IBB University, Lapai.

My unreserved and golden appreciation goes to my wife Fatima Binta Adamu; who stood to see that my coming to Zaria must be a success. Thank you and May Allah reward you.

Abstract

Process as an individualistic program in execution forms the bases of everything in the computer system functionality, Central Processing Unit (CPU) becomes the main target of every process execution. The best ordering and sequence of assigning these processes to the CPU becomes the most difficult problem to obtain best performances. This thesis work in the field of CPU scheduling by carefully studying all popular scheduling algorithms thereby proposing an option to the most uncommon scheduling algorithm: Longest Job First (LJF), to compete along side others. The major problem leading to starvation was reduced by proposing a model into the LJF. The model was designed and tested thereby suggesting ways by which LJF could be enhanced to solve parts of the starvation problems, waiting times and context switches.

Table of Content

Declaration i

Certification ii

Dedication iii

Acknowledgement iv

Abstract v

Table of Content vi

List of Tables ix

List of Figures x

List of Abbreviations xi

CHAPTER ONE 1

GENERAL INTRODUCTION 1

1.0 INTRODUCTION 1

1.1 OBJECTIVES OF STUDY 3

1.2 SIGNIFICANCE OF STUDY 3

1.3 RESEARCH QUESTIONS 3

1.4 SCOPE 3

1.5 METHODOLOGY 3

1.6 ORGANIZATION OF THESIS 4

CHAPTER TWO 5

LITERATURE REVIEW 5

2.0 INTRODUCTION 5

2.1 PROCESSES/ MULTIPROGRAMMING 6

2.2 SCHEDULING 7

2.3 SCHEDULERS 9

2.3.1 Long-term Scheduler/admission Scheduler 9

2.3.2 Mid/Medium-term Scheduler 10

2.3.3 Short-term Scheduler/CPU scheduler 10

2.4 DISPATCHER 11

2.5 SCHEDULING ALGORITHMS IN OPERATING SYSTEM. 12

2.5.1 First Come First Served (FCFS) 12

2.5.2 Shortest Job First/ Shortest Remaining Processing Time (SJF/SRPT) 14

2.5.3 Priority scheduling 17
2.5.4 Round-Robin (RR) 18
2.5.5 Longest Job First (LJF) 18
2.6 RELATED WORK 19
CHAPTER THREE 23
STARVATION PROBLEM FRAMEWORK MODEL 23
3.0 INTRODUCTION 23
3.1 APPROACH OF THE MODEL 24
3.1.1 Analysis of the CBT model 26
3.2 CBT ALGORITHM 27
CHAPTER FOUR 28
IMPLEMENTATION AND TEST OF THE CBT MODEL 28
4.0 INTRODUCTION 28
4.1 CASE ASSUMPTION 28
4.2 SIMULATION FRAME WORK 28
4.2.1 Processes Case 1 29
4.3 CBT PERFORMANCE EVALUATION 31
4.4 EXPERIMENT CASE 32
4.4.1 First Come First Served (FCFS) 32
4.4.2 Shortest Job First (SJF) 34
4.4.3 Round Robin 36
4.4.4 Priority Scheduling 38
4.4.5 Longest Job First (LJF) 40
4.4.6. Longest Job First + CBT 42
4.5. COMPARING EXISTING SCHEDULING ALGORITHMS WITH LJF+CBT 45
4.5.1 Comparing FCFS and LJF+CBT 45
4.4.2 Comparing SJF and LJF+CBT 46
4.4.3. Comparing LJF and LJF+CBT 46
4.4.4 Comparing RR and LJF+CBT 46
4.4.5 Comparing Priority Scheduling and LJF+CBT 46
4.5. RESULTS DISCUSSION AND CONCLUSION 47
CHAPTER FIVE 53
5.0 SUMMARY, CONCLUSION AND RECOMMENDATION 53
5.1 Summary 53

5.2 Future work 54
5.3 Conclusion 54
References 55
APPENDIX A: RESULTS OF THE SIMULATION IN LOG.TXT 58
APPENDIX B: JAVA SOURCE CODES OF THE LJF+CBT 63

List of Tables

Table 2.1: Process Snap shot 12

Table 2.2: Process Snap-shot with arrival times 15

Table 4.1: First 25 processes$Xi\delta i$ submitted to the pool N 29

Table 4.2: Processes $Xi\delta i$ on sorting at $\lambda 1$ higher burst times order. 29

Table 4.3: Processes with burst time less than Cwa at $\lambda \mathbf{2}$ 30

Table 4.6: Process with burst time 40

Table 4.7: Processes sorted in decreasing order of burst time 42

Table 4.8: Longer Jobs + CBT 43

Table 4.9: Average Waiting Time 43

Table 4.10: Average Turn-around Time 58

Table 4.11: Results of the Average Waiting Times obtained 59

Table 4.12: Results of the Average Turn Around Time obtained 60

Table 4.13: Results of the Context Switches obtained 62

List of Figures

Figure 1.1: Process state diagram 1

Figure 2.1 Memory Layout in multiprocessing system ..... …19

Figure 2.2 Process Control Block 20

Figure 2.3 Queuing diagram of a process scheduling 22

Figure 2.4: Mid-term scheduler 23

Figure 4.1: Gantt chart FCFS 45

Figure 4.2: Gantt chart SJF 47

Figure 4.3: Gantt chart for RR with q=10 36

Figure 4.4: Gantt chart for Priority Scheduling 38

Figure 4.5: Gantt chart for LJF Scheduling 40

Figure 4.6 Gantt chart for LJF+CBT 43

Figure 4.7 Process Scheduling in execution 47

Figure 4.8 Graph of 1500 sample processes 50

Figure 4.9 Graph of average waiting time for sample processes 61

Figure 4.10 Graph of average turn-around times for sample processes 61

Figure 4.11 Graph of Context Switches for sample processes 62

List of Abbreviations

ATAT: Average Turn-Around Time

AWT: Average Waiting Time

BW: Burst Time Priority Weight

CBT: Combinational Burst Time

CPU: Central Processing Unit

CS: Context Switch

Cwa: Combined Weighted Average

EDD: Estimated Due Date

FB: Foreground Background

FCFS: First Come First Served

FIFO: First In First Out

FJFDRR: Fittest Job First Dynamic Round Robin

HJS: Hierarchical Job Scheduling

HRN: Highest Response Next

I/O : Input Output

JSQ: Joint Shortest Queue

LAS: Least Attained Service

LJF: Longest Job First

LJF+CBT: Longest Job First + Combinational Burst Time

MST: Minimum Slack Time

ORC: Optimal Resource Constraint

OS: Operating System

PBSRR: Priority Based Static Round Robin

PC: Personal Computers

PCB: Process Control Block

POW: Pre-empted On Wait

PS: Processor Sharing

PSO: Particle Swarm Optimization

Q: Quantum number

RCSTD: Resource CoAllocation for Scheduling Tasks with Dependencies

RR : Round Robin

SET: Shortest Elapsed Time

SJF: Shortest Job First

SJN: Shortest Job Next

SP: Shorter Burst Time Priority

SPT: Shortest Processing Time

SRPT: Shortest Remaining Processing Time

TAT: Turn-Around Time

TFT: Total Finish Time

TTAT: Total Turn-Around Time

UP: User Priority

UW: User Priority Weight

WSPT: Weighted Shortest Processing Time

WT: Waiting Time

CHAPTER ONE

GENERAL INTRODUCTION

1.0 INTRODUCTION

Process scheduling algorithms has been an interesting field of study in Operating Systems. Scheduling is a key concept in computer multitasking, multiprocessing and real-time operating system designs.

Scheduling refers to the way processes are assigned to run on available Central Processing Unit (CPU) (Galvin, 2004). Each Process (a program in execution) passes through some stages in their execution and allocation to CPU as will be illustrated in process-state diagram. A Process is started at arrival time which gets into the Ready Queue based on some scheduling algorithm and waits for the Job scheduler/ dispatcher which decide on what scheduling algorithm to use. The process state diagram of Fig 1.1 illustrates this.

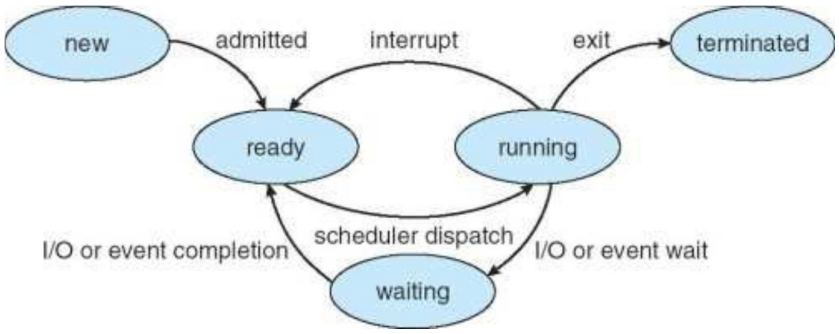


Figure 1.1: Process state diagram

The process state is as follows:

a. New: Where processes enters the system.

b. Ready: It is a stage where queues are used to order the manner in which jobs arrived

c. Running: The Scheduler and Dispatcher, based on selected algorithms, move Ready jobs to Running, a job may be pre-empted back to the ready depending on the scheduling algorithm in use.
d. Waiting: On a request for an I/O and later rescheduled on completion of request.
e. End/Exit: Process terminates.

At the moment, based on literature, there isn't a world most preferable scheduling algorithm (Stallings, 2000) as most operating systems use the most common scheduling algorithms which include: First Come First Serve (FCFS), Shortest Job Next (SJN), Shortest Job First (SJF), Round Robin (RR) and Priority Scheduling as an extension or the combination of two or more scheduling algorithm.

In this research, some of the existing scheduling algorithms basically the Shortest Job First (SJF) and the Longest Job First (LJF) algorithm will be looked into while putting latency and minimising total waiting time to achieve maximum utilization of the Central Processing Unit (CPU).

Furthermore, we propose a frame work to deal with the *starvation* (A condition that make processes waits stagnantly and denied allocation to the CPU) problems to curtail the *convoy* (A long queue of waiting processes) problem through fairness to shorter processes.

This research will also be focusing on making the Longest Job First (LJF) the approach to test and suggest how best the Shorter Jobs gets fairer allocation to the CPU through backfilling and combination of shorter jobs through *ageing* and *burst-times merging*. The process will be gaining access through comparing the merged burst-time and the incoming longer jobs.

1.1 OBJECTIVES OF STUDY

The objectives of this work is to minimise the overall waiting time and minimise the numbers of context switching of processes in a multiprocessing system; thereby coming up with a framework that will make fair consideration to shorter jobs on queue and suggest efficient time closer to the Longer Jobs in execution to minimise starvation in LJF.

1.2 SIGNIFICANCE OF STUDY

The research hopes to make a contribution to the existing problem faced by operating system by minimizing the starvation problem to achieve maximum *efficiency*, *throughput*, and CPU utilization by minimizing the overall waiting time of processes.

1.3 RESEARCH QUESTIONS

a. When does LJF compete with other algorithms?

b. Under what environment does LJF perform better than SJF, RR and FCFS?

c. Can Starvation problem be minimised in a multiprocessor environment using LJF?

1.4 SCOPE

The research will be focusing on the scheduling algorithm for Longer Job processes and propose a framework towards reducing long delays in queues. Shorter jobs will be merged up to make them a bit considerable to reduce starvation through fairness. The scope of this work is limited to reducing starvation problems and minimizing the overall waiting times and average turn-around time.

1.5 METHODOLOGY

These are approaches and series of steps to be taken in actualising our research objectives:

a. The first consideration on this work will be the generation of the processes using Random Number generators with an appropriate statistical distribution (Poisson and Exponential average).
b. The burst-time for each process will also be generated through a distribution.
c. Analysing the processes generated on Gantt charts and calculations of general process properties will be acquired and kept for comparison at the final Chapter.
d. Conditions for merging processes will be stated and designed by some algorithm and tested as implementation using Java and later a dispatch and schedule of the merged short jobs will be moved into the CPU by our algorithm.

1.6 ORGANIZATION OF THESIS

The thesis will be focusing on proposing a framework that will minimise starvation of processes in Longest Job First (LJF) scheduling algorithm through merging shorter jobs (Combinational Burst Time CBT) to maximise total utilization of the CPU to minimize total waiting time of the processes in the system.

Chapter two will be discussing on the existing literatures and researches conducted in the field of process scheduling.

Chapter three discusses the methodologies adopted to design the proposed framework for solving starvation problems in the Longest Job First (LJF) scheduling algorithm.

Chapter four will focus on the implementation and testing of the proposed algorithm.

Chapter five will summarise the work and propose areas for future work.

CHAPTER TWO

LITERATURE REVIEW

2.0 INTRODUCTION

An introduction to operating system as an area of study will lay better foundations to our research as Galvin, (2004) defined Operating system as a program that manages the computer hardware. Operating system provides foundations and serves as intermediary between the computer system hardware and the user. Different operating systems as they relate to functions they perform will also be discussed. For Mainframes, the operating system was designed primarily to optimize utilization of hardware, while Personal Computers (PC) supports complex games, business applications and everything in between the gaming and applications. (Stallings, 2000)

Operating system can also be looked at to perform three (3) principal functions which are:-

a. Allocating and assigning of system resources that are I/O devices, soft wares, Central Processing Units (CPU).
b. Scheduling: - The function that coordinates resources and jobs using some selected algorithms.
c. Monitoring: - The function that monitors and keep tracks of activities in the computer system.

Generally, one of the major goals of the operating system is the maximisation of the CPU utility and to achieve this, the concept of multiprogramming was conceived (Dalibor *et al.*, 2005) as illustrated in Fig 2.1

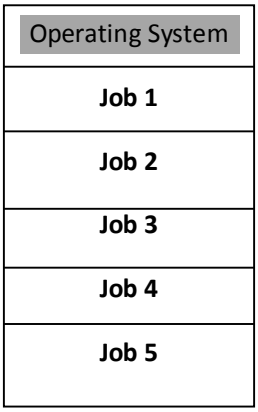


Figure 2.1 Memory layouts in multiprocessing system

2.1 PROCESSES/ MULTIPROGRAMMING

A multiprogramming environment is an environment where more than one program runs. Process is defined as a program in execution and has some properties such as process type, process identification (Galvin, 2004). Examples of such programs include a Time-Shared user program such as a compiler and a word processing program being run by an individual user on a PC.

Furthermore, a process needs certain resources for its functionality. These include CPU-time, memory, files and Input/output devices. A process can further be defined as the unit of work in a system; where all management and manipulations of this process is handled by the operating system (Galvin, 2004) as shown in Fig 2.2 representing a Process Control Block (PCB).

Operating system makes all the characteristics of the processes achievable through the following:

a. Creating and deleting of both user and system processes.
b. Suspends and resumes processes (as in cases of interrupts, I/O request and pre-emption).
c. Provide mechanisms for process synchronization.
d. Provide mechanism for process communications.

e. Provide mechanism for deadlock handling.

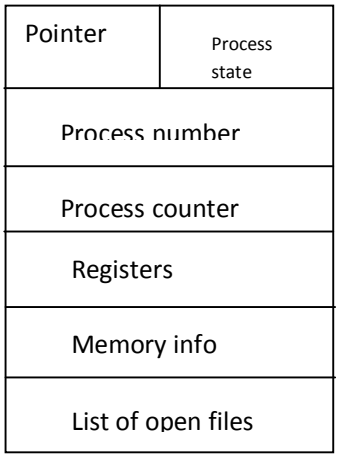


Figure 2.2 Process Control Block

Each process exhibits the conditions of the PCB (see Fig 2.2) such as

a. Process Counter**:** Indicates the address and location of the next instruction to be executed.
b. CPU registers**:** These include accumulators, index registers, stack pointers, general purpose registers and any condition code information.
c. CPU scheduling information: Information includes a process priority, pointers to scheduling queues and other scheduling parameters.
d. Memory management information: Information about the limit registers page tables and segments.
e. Accounting: The information of the amount of CPU, real-time used, time limits, process numbers.
f. I/O status information**:** Keeps information on the I/O devices allocated to the processes and the lists of open files. (Galvin, 2004)

2.2 SCHEDULING

Scheduling is the method or actions by which threads, processes or data flows are given access to the system resources (e.g. process time, CPU, communication, bandwidth) which is usually

done to balance a system and achieve best quality of service (Ernst *et al*, 2007). Scheduling also refers to the series of actions performed on a system to distribute processes to resources using the long, short and the mid-term scheduler.

Scheduler is the module that moves jobs from queue to queue as shown in Fig 2.3 of the Queuing diagram of process scheduling. The scheduler evaluation can be measured using the following criteria.

a. Throughput: The number of processes that complete their execution per unit time.
b. Latency: This is defined using the turn-around and response-time.
    - Turn-around time is the total time between the submission of a process and its completion.
    - Response time is the amount of time it takes from when a request was submitted to the first response produced.
c. Fairness/ waiting time: Equal the CPU time to each process and the time a process waits before been scheduled from its time of arrival. It could also be define as the total time a process waits in the system on suspension or pre-emption.
d. CPU utilization: Percentage of time that the CPU is busy.

From the scheduler's perspective, the four distinct goals must be considered for a good scheduler.

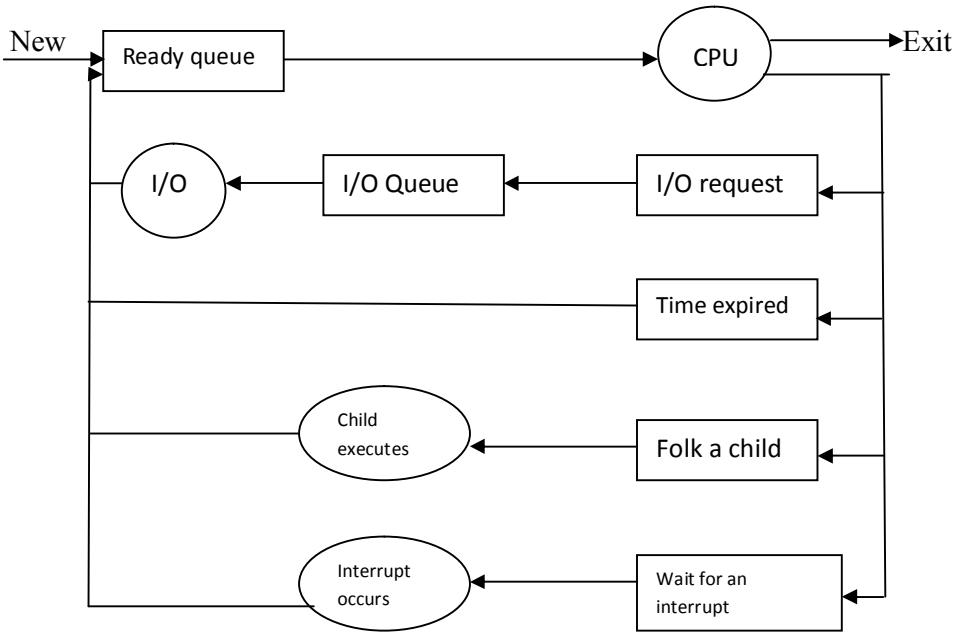


Figure 2.3 Queuing diagram of a process scheduling

2.3 SCHEDULERS

Long term, Mid-term/medium-terms and short-term schedulers are the main types we will be looking at in this research. In some literatures, there only exist the long-term and the short-term schedulers. From the definition of scheduling, using Fig 2.3 to illustrate, schedulers are a module that chooses and selects the jobs in and out of the CPU and orders jobs to be admitted to the system.

2.3.1 Long-term Scheduler/admission Scheduler

From the concept of memory in operating system (Stallings, 2011), the Long-term scheduler decides which jobs are to be given and taken to the Ready queue; this scheduler is mainly concerned with the main memory. In scheduling concepts, the first and second phase of the process state diagram in Fig.1.1 will be analysed. The Long-term scheduler also dictates to the system its degree of concurrency and how the processes will be split. Modern operating systems such as Windows have high priority to real-time process than others (Krishnan, 1995). The Long-term queue for this scheduling type exists in the hard disk or virtual memory (Stallings,

2011). Some existing examples of the Long-term scheduling are implemented in large scale system such as batch processing, computer clusters and super computers.

2.3.2 Mid/Medium-term Scheduler

Mid-term scheduler is best understood when related to the memory hierarchy concept in operating system, as such, the memory scale shows that from the main memory to the disk drives, the medium scheduler temporarily removes the processes from main memory to the secondary memory or vice versa as shown on Fig 2.4. This process is known as "swapping in and swapping out" also called "paging in and paging out" in some books (Galvin, 2004) and (Prashant, 2008). The mid scheduler works in a way that it swaps out processes that have not been active or lower in priority measures, faulted processes, large acquiring memory process, request processes and later swaps them in when ready. Conclusively, the mid- term scheduler can be discussed as the middle module for swapping in and out of the memory. In modern operating systems, the mid-term scheduler performs the job of the long-term scheduler using the virtual address space technology of the memory.

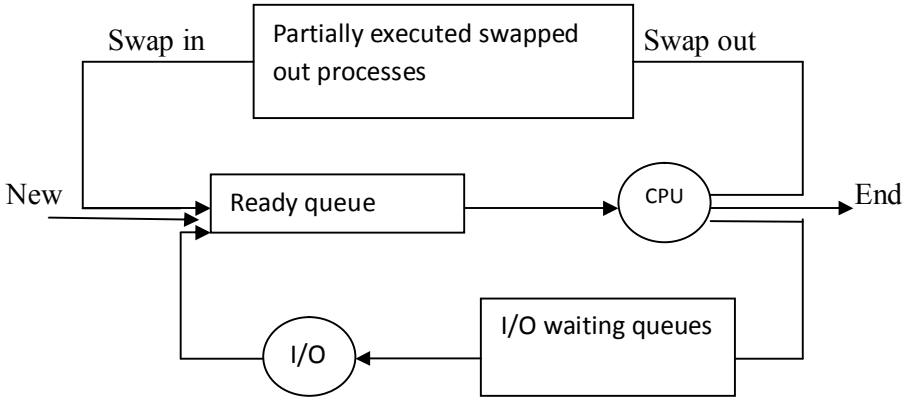


Figure 2.4: Mid-term scheduler

2.3.3 Short-term Scheduler/CPU scheduler

This is the scheduler that deals with the Ready Queue as seen in Fig 2.4. It chooses the processes to be executed and assigned to the CPU. Going by the process state diagram in Fig 1.1 and the Queuing diagram in Fig 2.3, the short-term scheduler decides more than the long-term scheduler.

It work based on clock interrupt, I/O interrupts, system calls. It does the pre-emption (forcibly removing process from the CPU) depending on the algorithm, priority or time slice (Gerald *et al.,* 2004).

2.4 DISPATCHER

In every operating system, scheduling is best treated along side the dispatcher. The dispatcher is a module that gives privileges/control of the CPU to the process selected by the Short-term scheduler. The functions of the dispatcher include:

a. Context Switching
b. User mode Switching
c. Program location jumping

The time it usually takes to stop a process and start another running program is known as the *dispatcher latency*.

Scheduling as a concept of operating system migrates this work to its algorithm which are used to distribute resources amongst processes/jobs. Some major implementations of scheduling areas of computer science are: Routing (packet traffic in networking). Operating system (share CPU time among processes and threads), disk drives (I/O operations), printers (print spooler), and gas station machines (embedded). Scheduling algorithm enhance system performance by minimizing resource starvation and fairness among processes to achieve efficiency of the system.

According to Achim (2006), some common algorithms for better evaluation include the First Come, First Served (FCFS), Shortest Job First (SJF), Longest Job First (LJF), Priority scheduling and the Round Robin (RR). This research will analyse each of the above algorithm and propose the frame work that reduces starvation in the Longest Job First (LJF) algorithm.

2.5 SCHEDULING ALGORITHMS IN OPERATING SYSTEM.

Scheduling algorithm in operating system aides in actualising the goal of maximum utilization of CPU; in the multiprogramming system, barely all processes are busy at all time. When a process enters into the job queue, it requires a scheduling algorithm to decide on the mode and nature of the execution process (Boleslaw, 1990). See Process state diagram in Fig 1.1, and Fig 2.3 Queuing- diagram of a process scheduling.

Table 2.1: Process Snap shot

Processes	Burst-time
P1	10
P2	29
P3	3
P4	7
P5	12
P6	14

The process snap shots are pictorial representations of processes alongside burst times and other properties such as arrival time, priority numbers for easy evaluation metrics calculations.

2.5.1 First Come First Served (FCFS)

This is the most popular form of the scheduling algorithm as several systems name it as the fairer algorithm. It operates based on the basis of the first process/job that arrive should be the first to be scheduled. This makes it logically possible for the first job in queue will be the first to leave (FIFO: First in First Out). This is the implemented in packet- switching in computer networks.

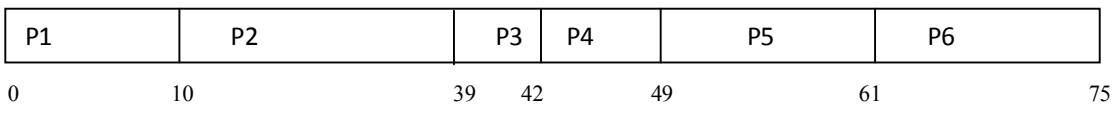


Figure 2.5: Gantt chart FCFS

Waiting Time (WT) for FCFS

WT = (time first schedule – arrival time)

Average waiting time (Awt) for FCFS

Awt = (time first schedule – arrival time)/n

P1= (0), where P1-P6 represents the distinct identification for each process.

By the equation *WT*= (time first schedule – arrival time),

P2= (10-0),

P3= (39-0), P4= (42-0), P5= (49-0), P6= (61-0)

Assume all processes arrived at time= 0 according to the process numbers.

So the *AWT* = (0 +10+39+42+49+61)/6 =33.2

A. Properties of FCFS

a. Scheduling overhead (Cost) is minimal since context switch is absent in most cases.
b. Throughput can be slow when longer processes are the first. Hog the CPU
c. High turnaround time.
d. High waiting time for shorter jobs.
e. High response time.
f. No priority; which could also result to processes not meeting deadline.
g. Based on queuing.

B. Advantages of FCFS

a. Fairness: Each process is treated fairly depending on the time of arrival.
b. Less costly to implement.

c. No pre-emption.

d. Better for shorter jobs.

C. Disadvantages/problems of FCFS

a. High turnaround time.

b. Starvation of shorter jobs when longer job exist.

c. Un-attainable due dates.

d. Reduces CPU and I/O device utilization.

e. Could result in convoy problem.

2.5.2 Shortest Job First/ Shortest Remaining Processing Time (SJF/SRPT)

The SJF scheduling algorithm is the form that takes into consideration the burst-time of all processes as they arrive into the system. This is an algorithm that is referred to as 'selfish' algorithm since longer jobs wait indefinitely for shorter jobs to use up their burst times (Cheng *et al.,* 2006). The SJF takes two forms namely: SJF with pre-emption and the SJF without pre-emption. The SJF with pre-emption checks and grant the Shorter jobs the CPU and also keeps a check of when shorter jobs arrive, and has lesser burst-time than the one in the CPU, it is forcibly taken out of the CPU for the shortest remaining process. While the SJF without pre emption assumes all processes arrived at same time and doesn't take away the CPU once allocated to a process based on the shortest available time. Generally, the SJF algorithm requires advance knowledge of burst time. The following Gantt-chart is computed using Table 2.1

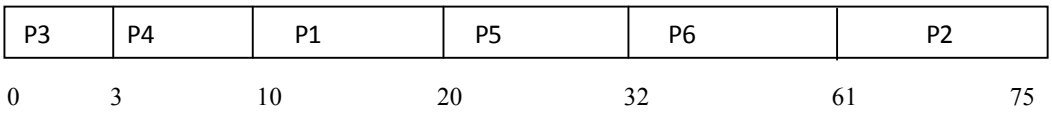


Figure 2.6: Gantt chart SJF

Waiting Time (WT) for SJF

WT = (time first schedule – arrival time)

Average waiting time (Awt) for SJF

Awt = (time first schedule – arrival time)/number of processes

P3= (0), where P1-P6 represents the distinct identification for each process.

By the equation WT= (time first schedule – arrival time), P1-P6 is presented thus:

P1= (10-0), P2= (61-0), P3= (0-0), P4= (20-0), P5= (32-0);

Awt = (10-0) + (61-0) + (0-0) + (3-0) + (20-0) + (32-0)/6

(10+61+0+3+20+32)/6 = 21.0

Table 2.2: Process Snap-shot with arrival times

Processes	Arrival time	Burst-time
P1	0	10
P2	1	29
P3	6	3
P4	7	7
P5	10	12
P6	15	14

The Table 2.2 shows processes with respective burst time alongside arrival time, this involves pre-emption and context switches.

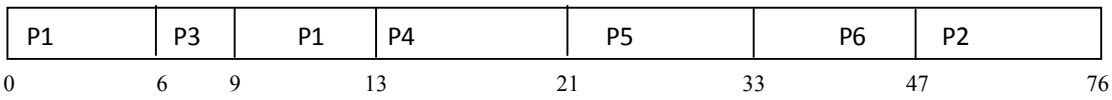


Figure 2.7: Gantt chart SJF with pre-emption

Waiting Time (WT) for SJF

WT = (time first schedule – arrival time)

Average waiting time (Awt) for SJF

Awt = (time first schedule – arrival time)/number of processes

Where n = total number of processes in the system.

By the equation *WT= (time first schedule – arrival time)*, P1-P6 is presented thus:

P1 arrived at Time= 0, it was taken out of the system as P3 arrived.

Thus: P1= (0+9-0), P2= (47-1), P3= (6-6), P4= (13-7), P5= (21-10), P6= (33-15).

Awt = ((9-0) + (46) + (0) + (6) + (11) + (18))/6 = 15

A. Properties of SJF

a. It has the ability to pre-empt jobs with longer burst-time.

b. The algorithm mostly makes CPU busy at all time resulting to a better through put value.

c. Waiting time is normally small as constant checks are made on shorter jobs.

d. It has the ability of re-scheduling the suspended job from the queue.

B. Advantages of SJF

a. Maximum throughput.

b. Low waiting time compared to FCFS.

c. Response time is better as jobs are treated in a faster manner.

d. Maximum utilization of CPU (Goal of operating system).

C. Disadvantages/problems of SJF

a. Difficult in checking job pool every time for the next schedule.

b. Next burst-time is usually predicted by the OS using the past burst-time.

c. Interrupts running process.

d. Could lead to starving longer jobs.

e. High cost of implementation because of the run-time estimation. It is not a widely used scheduling algorithm.

f. Selfish an unrealistic.

2.5.3 Priority scheduling

Priority scheduling algorithm is an algorithm that allocates CPU to processes according to number called 'Priority mark' score where the lower the number, the higher the priority. Priority scheduling is sometimes referred to as 'Fixed Priority Pre-emptive Scheduling'.

A. Properties of Priority Scheduling

a. Overhead is not minimal.

b. All evaluation metric depends on the priority order.

c. High priority processes have smaller waiting time

d. High priority processes have smaller response time.

e. Deadlines are not met by assigning deadlines to jobs with lower priority.

f. Starvation is likely to occur to processes with lower priority.

B. Advantages of Priority Scheduling

a. Throughput is easily predicted and enhanced as priority are set.

b. High turnaround time.

c. High response time.

C. Disadvantages/problems of Priority Scheduling

a. Starvation as a result of lower priority score.

b. Costly as involves context switching between processes of lower priority scores.

c. Expensive to implement.

2.5.4 Round-Robin (RR)

The round-robin is a scheduling algorithm that assigns fixed time-slices called 'Quantum' to processes as they are scheduled. Each process in the queue is allocated the CPU through the amount of Quantum set by the designer (Rakesh, 2011). It is generally referred to as a starvation-free algorithm; where each process is considered continuously until it finishes its available burst-time.

A. Properties of RR

a. Only one process runs at a time.

b. Processes give up the CPU when quantum is reached.

c. All processes are assumed to be treated equally with fairness.

B. Advantages of RR

a. Easy to implement

b. Each process gets equal share of the CPU.

c. CPU utilization is almost 100% as the CPU never gets idle.

C. Disadvantages/problem of RR

a. Bad overall result.

b. Uniformly inefficient.

c. Longer jobs keeps switching when Quantum is low.

2.5.5 Longest Job First (LJF)

This is the algorithm that will be introduced as an opposite of the SJF; it differs from the SJF by considering longer burst time jobs before the shorter ones. From a logical point of view, we can predict that the LJF can challenge side-by-side with the SJF because of some hidden advantages. In a multiprocessing environment, where a general result is submitted after the entire process execution, all processes need to be completed almost at the same time. In this algorithm, the longer jobs are admitted first and the shorter jobs wait. Its overall advantage includes the ease of

computing longer jobs and makes it less tedious when handling shorter jobs (Raksha *et al.*, 2010).

A. Properties of LJF

a. The Longer jobs are treated and given more consideration than the shorter ones.

b. It has the ability to pre-empt jobs with shorter burst-time.

c. The overall turn-around time is sometimes high.

d. All processes spend great time in the system because of the context switching.

B. Advantages of LJF

a. It corrects some of the weakness of both Shortest Job First (SJF) and First Come First Serve (FCFS).

b. It tries to reduce overall turnaround time and maximize system utilization for high system loads.

C. Disadvantages of LJF

a. Starvation of shorter jobs.

b. Low response time among several jobs

2.6 RELATED WORK

Rakesh *et al.*, (2011), in his work titled "Design and performance evaluation of a new proposed fittest job first dynamic round robin" (FJFDRR) where he adopted a method by testing the popular algorithms. In the work, he suggested a fit factor 'f' which is computed by f = UP * UW * SP * BW where the UP is the User priority set by the designer, UW is the User Priority weight, SP is the Shorter Burst time priority and the BW as the Burst time priority weight. A method which was used as a diligent presentation of pseudo-code was produced for the algorithm. The approach compared the performance of Fittest Job First Dynamic Round Robin (FJFDRR) and the Priority Based Static Round Robin (PBSRR) which they later concluded their approach performs better.

Ali *et al.,* (2011) studied different scheduling and proposed backtracking scheduling algorithms for processes using the Particle Swarm Optimization (PSO) algorithm. The work argued that since birds, fishes move in swarm methods, backtracking could be done using same. Their goal was achieved as having a good minimal Total Finish Time (TFT) as position and speed of each element for reaching optimized state changed continually.

Vissarion (2011) looked at the cost policies of convergence of selfish strategies in Pure Nash equilibria. Here jobs are scheduled as selfish as possible in a pseudo-polynomial order. He used game theory to prove selfishness i.e. sole objectives are the maximization of personal benefit. He used the results generated to take priority into consideration to test cost he used: Shortest Job First (SJF), Longest Job First (LJF), and First Come First Served (FCFS).

Raksha *et al.,* (2010) studied all existing scheduling algorithm and checked the implementation on the grid environment. "A survey of Job Scheduling and resource management in Grid Computing". The work took into consideration the major challenges faced by the scheduling concept calling it an NP-complete problem. Their findings suggested there is no best scheduling algorithm as it all depends on the problems to solve. The research was done on some listed scheduling algorithm which includes: Highest Response Next (HRN), Optimal Resource Constraint (ORC), Hierarchical Job Scheduling (HJS), and Resource CoAllocation for Scheduling Tasks with Dependencies in grid (RCSTD). A method adopted in their research was to highlight the advantages and some set-backs for each scheduling algorithm. In the analysis phase of the research, problems were simulated using 50 to 300 processing time records.

Hoeksma (2010) looked at price of Anarchy for Machine Scheduling Games with sum of completion Times objectives. The work is done with the help of a central administrator who schedules Jobs and aims at reducing waiting time and maximises CPU scheduling busy. The

work classified problems into 3 categories namely: Machine environment, Job properties and objective of the scheduler.

Erust *et al.,* (2007), scheduling in practice was a work that tested critically scheduling problems and queues and further suggested ways of overcoming long queues and minimising waiting times in relation to Processor-Sharing (PS), using 2 applications namely the Connection Scheduling in web servers and Packet Scheduling in network routers. Goal was Shorter mean response time across all request in a system. In this research, Job scheduling problem in databases were also captured and considered as Lock algorithm was discussed as POW (Pre-empted-On-Wait). The work couldn't use the SRPT (Shortest Remaining Process Time) but came up with alternative called LAS (Least Attained Service) which mimics the SRPT. The LAS gives priority to least attended jobs. LAS also called Foreground Background (FB) or Shortest Elapsed Time (SET) in their work.

Mike, (2006) looked at solving starvation problem in the scheduler to test his suggestions. In Scheduling domains fixed many of the problems on larger system interactivity heuristics made desktop work better. His work is based on the notion of *Sleep Average* which he calculates some time and calls it sleeping time relative to time it runs specifying some priorities. Since Linux scheduler maintains 2 arrays, Mike (2006) called Array one "Active Array" where the original process competes for CPU using its given time slices, once the time slice runs out and the process couldn't finish, it is moved to the second array called "Expired". It relinquishes there until all executable process have used up their time slices. Once all processes are through the 2 arrays are switched and process begins again. Mike (2006) tested using a system with heavily-loaded APACHE server running. His solution was a bit less generous with the interactivity bonuses, based on his findings, there was a step-up and starvation was reduced to a bearable minimum.

Guangzhi and Gianfranco (2000), worked on the "Approximate analysis of a fault-tolerant join-the-shortest-queue policy" to combine two methods of scheduling to provide good performance and high reliability. A matrix-geometric method was used to carry out their analysis. They designed a Petri net Simulation model to obtain the average number of jobs in the system. Their scheduling algorithm was functioning based on some criteria set for failed servers during processing. An Algorithm was designed that used a birth-death Markov process and thus, an approximation to analyze the performance of Joint Shortest Queue (JSQ) was developed.

RIOT (1999), focussed on some notable algorithms which included the **List Algorithm** amongst others, the processes are scheduled according to some permutations. Longest process concluded test to finding the minimum make span of a schedule while Shortest Processing Time (SPT) is optimal for finding the minimum total completion time and weighted completion time, while ***Weighted Shortest Processing Time*** (*WSPT*) is a variation of the SPT; this inequality is used where t[i] and w[i] denotes the processing time and weighted respectively, as thus: $t[1]/w[1] \leq t[2]/w[2] \leq \ldots, \leq t[n]/w[n]$. The Goal was also to minimise delay and long completion times. ***Earliest Due Date (****EDD)* is also an algorithm reviewed and it works based on a time t set at Zero (0) for all jobs, the earliest due date sequence jobs to be done using the earliest due date jobs. Where d[i] denotes due date of the i*th* job, it will be as $d[1] \leq d[2], \leq \cdots d[n]$. The major goal is minimising the maximum lateness and minimise the maximum tardiness. ***Minimum Slack Time*** (*MST*) measures urgency of a job by its slack time using d[i] and t[i] as due date and process time as represented on the inequality:

$d[1] - t[1] \leq d[2] - t[2] \leq \cdots \leq d[n] - t[n]$ The goal was to maximise the minimum lateness. Other Algorithms includes ***Hodgson*** which minimises tardy jobs with ready time = 0 and ***Johnsons*** which minimises lateness and over stay using 2 jobs to test.

CHAPTER THREE

STARVATION PROBLEM FRAMEWORK MODEL

3.0 INTRODUCTION

In every multiprocessing environment, jobs/process scheduling is a deterministic factor to the system efficiency and effectiveness measure. This research looked into various forms of process scheduling algorithms, but focused mainly on the fair considerations for shorter jobs/processes in a Longest-Job First (LJF) algorithm. Logically, the shorter jobs are *starved* (waiting indefinitely) as longer jobs hog unto the CPU endlessly. As part of the solution to the case of starvation, Rakesh *et al.,* (2011) proposed a Fittest Job First Dynamic Round Robin (FJFDRR). In our research, we are proposing a Combinational Burst Time (CBT) for shorter jobs in the system so as to equalise the Total Turn-Around Time (TTAT) and minimise the overall Waiting Time (WT) of the processes in the Longer Job First (LJF) process scheduling algorithm. Using logical reasoning, for cases with due date and pipelining operations, the LJF+CBT will be useful.

The challenge here is to assign different conditions for the merger and processing (access to the CPU). The research shall be considering some factors which will make the scheduling possible. Generally, processes are associated with three (3) factors namely:-

a. User Priority,
b. Burst Time
c. Arrival Time.

For this research with the CBT, the work will be focusing on the Job identity, the Burst time, arrival time and the most distinct difference the 'phase and stage of execution' which will be taken care of by a counter in progress. The idea of the phase of execution inclusion stands our

contribution out as a new technique which this research hopes it will curtail and minimise the existing starvation problems faced by Shorter Jobs in Longest Job First (LJF) algorithm.

In First Come First Served (FCFS), Priority and Shortest Job First (SJF) algorithms, only a factor is set as a mark timer (Rakesh *et al*,. 2011); whereas setting four factors listed to actualise the Combinational Burst Time (CBT) will reduce the Average Waiting Time (*AWT*), reduce the overall Context Switch numbers and also minimise the problem of starvation in the system there by enhancing performance in due date systems and pipeline of cooperating processes.

3.1 APPROACH OF THE MODEL

The major priority is set at minimising the case of starvation to shorter jobs by stating some factors as the major requirement to reducing the problem. Combinational Burst Time (CBT) will be calculated on meeting some specified conditions in the propose algorithm. In the FCFS algorithm, which is most times believed to be the fairest algorithm (Cheng *et al*, 2006); the only determining factor is the arrival time. SJF on the other hand sounds easy to implement but difficult in practice which is mostly termed the most unfair algorithm using the factor burst time as the evaluator. For Round Robin (RR), its major disadvantage is the larger numbers of context switches when given a less number of Quantum (q) to meet maximum utilization of the CPU. An approach to solving context switch problem was proposed by Rakesh *et al.*, (2011) and Raksha *et al.,* (2010) for Round Robin scheduling.

The approach in this work combines several properties of the process scheduling to make up the proposed algorithm.

The Model Architecture

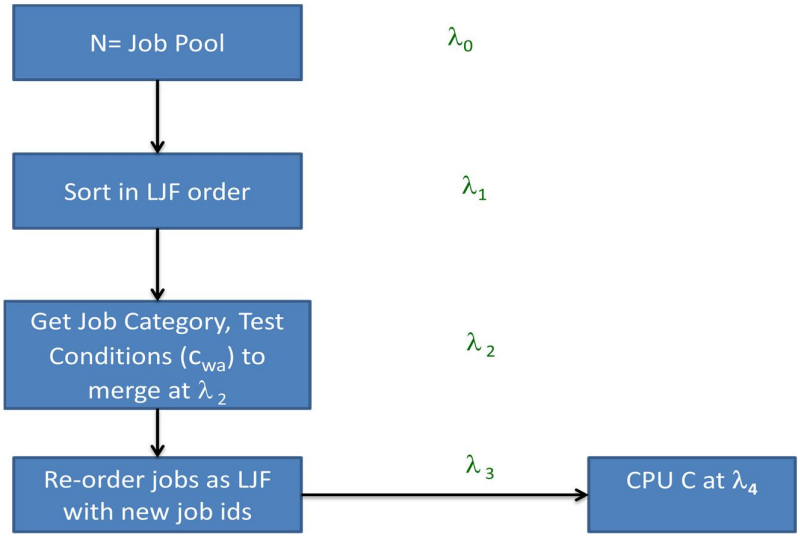


Figure 3.1: Model Architecture

COMBINATIONAL BURST TIME (CBT) MODEL

Consider a Job pool as a set containing various processes with distinct burst times for each, there exist as a phase of process execution.

burst-times …, for every associated with at

Using the LJF concept, sort all jobs in decreasing order of their respective burst times at

The sorting provides a queue of jobs from the pool N.

Let be a threshold condition at defined as = ——— , jobs are categorised into longer and shorter based on calculated as follows:-

After the condition of categorization, sort resulted jobs in descending order of δ_i

Let α_i be defined as a new burst time at a phase of execution λ_3 .

Where $\alpha_i = \begin{cases} \delta_i;\ \delta_i > c_{wa} \\ \delta_i + \delta_j;\ \delta_i, \delta_j \leq c_{wa} \end{cases} \ldots (4)$

On completion of the phase λ_3, the model has tested the conditions of merger (CBT), the new LJF+CBT now admits processes into the CPU as:-

$$C = \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n \ldots (5)\ ;$$

Provided it satisfies the inequality $\alpha_i \geq \alpha_{i+1}$, *i= 0, 1, 2,..., n. where* $\delta_i + \delta_j = \alpha_i$

3.1.1 Analysis of the CBT model

In the experiment of the CBT model, for a discrete case it is assumed that processes have a burst-times of several sizes in the lower and upper bound as limits (non zero). The conditions that all processes with burst time less or equal to a threshold value known as Combined Weighted Average (*Cwa*), forms the category of either a longer job or short. According to equation (3), jobs are categorized based on the test of the values obtained from the *Cwa* and the logical comparison of its size. For all jobs less than *Cwa* are termed as shorter jobs while jobs that are greater than the *Cwa* are termed long. The λ_0 now serves as the condition that checks the phase of execution and the model hierarchy for each short process to be merged with any other available process in the queue all belonging to N. Once all conditions are satisfied, the Combinational Burst Time (CBT) takes place and result a new process with a high burst time(δ). Order and precedence such as the arrival of processes in the queue is very vital in this concept, an N number of sample processes each with its burst timeδi is given to a job pool, order of the processes are needed as all processes have a process id that takes care of the arrangement at λ_0.

On the first computation and sorting, the generated process Burst- time gives a summary of all Processes with a Burst time greater than Cwa and shorter processes with less or equal to the threshold which are further denoted as α_i after the CBT.

Normalising the given processes, an algorithm to reduce the process to satisfy the merging condition CBT is defined thus: A set N,

$$N = \{ \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n \} \dots (6)$$

3.2 CBT ALGORITHM

Let $Xi\delta i$ be a process (X) with id number (i) with its burst time(δi),

a. Submit Job to pool N at λ_0
b. Sort all processes in descending order of larger burst time (δi) at λ_1.
c. Set threshold by executing Cwa.
d. If $Xi\delta i > Cwa$; categorise as Long job, else categorise as short at λ_2.
e. Move $Xi\delta i$ to the new queue for re-ordering at λ_3.
f. If $Xi\delta i$ exist and less than Cwa; then add $\delta i + \delta j = \alpha i$
g. Reduce until $Xi\delta i \leq Cwa$ have no $Xi\delta i$ to merge.
h. Compare $Xi\delta i$ with the result αi
i. If $\alpha i \geq Xi\delta i$, place before $Xi\delta i$
j. Else place after αi
k. End.

After following the above Normalisation and sorting, the following order into the CPU is obtained:

C = $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$, where $\alpha_i \geq \alpha_{i+1}$, *i= 0, 1, 2,..., n*

CHAPTER FOUR

IMPLEMENTATION AND TEST OF THE CBT MODEL

4.0 INTRODUCTION

Different cases of implementation of the CBT model was tested after the model designed in java to create a graphical user interface where processes are entered and the ranges of burst times are specified. The collective results for each running of the program is written to a text file called log.txt where sample results are analysed and compared with the results obtained from the popular existing scheduling algorithms.

4.1 CASE ASSUMPTION

In a multiprocessing environment, processes of different characteristics are submitted to the CPU, this research assumed a system comprising of 10-5000 processes to be assigned the CPU based on the Combinational Burst Time (CBT) model. The burst time and identification of each process was carefully generated and assigned.

4.2 SIMULATION FRAME WORK

The frame work will present different cases of the generated processes in chunks and groups for evaluation and comparisons of different results obtained. Each case shall present the actual processes involved thereby testing with the metrics of performance by computing the Average Waiting times (*AWT*) and each waiting time of the process. Following all steps of the proposed CBT framework in Chapter Three, results will be collated and compared to the existing scheduling algorithms (FCFS, SJF, RR and the Priority Scheduling).

This frame work creates new process identification for the combined burst time through adding the address of the processes as one. Example: a process P17 combine with P2 gives a new

identification as *P17,2. Where the '*,' implies it is a CBT process and the numbers represents the original identification number of the process separated by a comma (,).

4.2.1 Processes Case 1.

From the results of the generated processes, 25 processes are presented thus:

Table 4.1: First 25 processes $\boldsymbol{Xi\delta i}$ submitted to the pool N.

P1=43	P2=22	P3=48	P4=25	P5=5
P6=16	P7=21	P8=18	P9=9	P10=5
P11=38	P12=47	P13=6	P14=47	P15=22
P16=44	P17=23	P18=17	P19=32	P20=28
P21=26	P22=33	P23=33	P24=49	P25=28

Executing the CBT model along side the Longest Job First (LJF) algorithm, the following results are obtained and sorting by the highest burst time process preceding the shorter processes.

Table 4.2: Processes $\boldsymbol{Xi\delta i}$ on sorting at $\boldsymbol{\lambda_1}$ higher burst times order**.**

P24=49	P3=48	P12=47	P14=47	P16=44
P1=43	P11=38	P22=33	P23=33	P19=32
P20=28	P25=28	P21=26	P4=25	P17=23
P2=22	P15=22	P7=21	P8=18	P18=17
P6=16	P5=14	P9=9	P13=6	P10=5

Based on the CBT model, the threshold known as the Combined Weighted Average (Cwa) is calculated by the formula $c_{wa}= \frac{\sum_{i=1}^{n} \delta i}{n}$, from equation (2).

$$c_{wa} = \frac{672}{25}$$

$$= 26.88$$

From the Cwa calculation, the jobs are then categorized as short and long by the test of burst times with the Cwa.

By equation (3), the processes are grouped into two categories as thus presented in Table 4.3

Table 4.3: Processes with burst time less than Cwa at $\boldsymbol{\lambda_2}$

P21=26	P4=25	P17=23	P2=22	P15=22
P7=21	P8=18	P18=17	P6=16	P5=14
P9=9	P13=6	P10=5		

Table 4.4: Processes with burst time greater than Cwa at $\boldsymbol{\lambda_2}$

P24=49	P3=48	P12=47	P14=47	P16=44	P1=43
P11=38	P22=33	P23=33	P19=32	P20=28	P25=28

Combinational Burst time CBT in Case 1.

On evaluating the CBT model, 10 processes combine to form 5 new processes as presented thus:

Table 4.5: CBT Case 1

*P21,4=51	*P17,2=45	*P15,7=43	*P8,18=35	*P6,5=30	*P9,13=15	P10=5

$$\delta i + \delta i = \alpha_i$$

The result of the CBT in Table 4.4 is thereby stored in a temporal location to await the Longest Job First Scheduling algorithm by the Job Scheduler.

The results shows that a new order of the processes was obtained and the total number of processes merged were twelve resulting into six (6) chunks with new identity and burst values.

Table 4.6: Summary Processes with CBT at λ_3 .

*P21,4=51	P24=49	P3=48	P12=47	P14=47	*P17,2=45	P16=44	P1=43
*P15,7=43	P11=38	*P8,18=35	P22=33	P23=33	P19=32	*P6,5=30	P20=28
P25=28	*P9,13=15	P10=5	-	-	-	-	-

The results obtained from Table 4.5 provide the new order which the scheduler will assign processes to the CPU C for execution. Such that

$$C = \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n \text{ , where } \alpha_i \geq \alpha_{i+1} \text{ , } i = 0, 1, 2, \ldots, n$$

For deterministic analysis of process scheduling algorithms, Gantt charts representation and the Evaluation metrics are shown for clear understanding of the concepts.

4.3 CBT PERFORMANCE EVALUATION

As defined by Rakesh *et al.*, (2011), performance metrics of an algorithm can be best tested using the *Turn-Around Time (TAT)* to check the algorithm performance. The *Average Turn Around Time (ATAT)* of the proposed algorithm should be less. Furthermore, *Waiting Time(WT)* for each process needs to be calculated, which specifies the time a process needs to wait in the system before it is scheduled by the scheduler or the amount of time a process waits as a result of suspension, I/O request or pre-emption. The *Average Waiting Time (AWT)* therefore provides the score for the algorithm's performance. Lesser *AWT*; implies better performance of the algorithm. See Fig 1.1 Process state diagram. *Context Switches (CS)* defines the total number of the switches among processes in the CPU; the best performance of a system should reduce the *CS* by a minimum. The CBT model has worked on reducing the total *CS* by introducing the mergers of processes as one.

4.4 EXPERIMENT CASE

To experiment the arguments of this research, 10 processes shall be selected in random all of which shall contain the properties of Short burst times, long burst time, and each with distinct identification. Thereafter, the evaluation shall be tested using the existing algorithms of *First Come First Served (FCFS)*, *Shortest Job First (SJF)*, *Priority Scheduling* and *Round Robin (RR)*. A snap shot of the processes is presented as thus:

Table 4.6: Process snap shot

Processes	**Arrival Time**	**Burst Time**	***User Priority**
P1	0	9	7
P2	0	27	9
P3	0	14	8
P4	0	30	3
P5	0	22	5
P6	0	19	10
P7	0	45	2
P8	0	39	1
P9	0	7	4
P10	0	10	6

*The User Priority is used in the Priority Scheduling algorithm

4.4.1 First Come First Served (FCFS)

For the FCFS, the ordering of the Processes by the identification gives the form of the execution, since all processes are assumed to arrived at Time = 0, therefore, P1, P2, P3, P4, P5, P6, P7, P8, P9 and P10 is maintained. See table 4.6

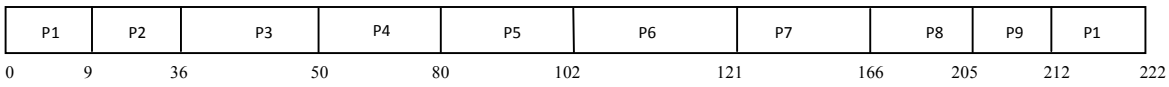


Figure 4.1: Gantt chart FCFS

From the Fig 4.1, Processes 1-10 are represented and P1 starts the CPU allocation at time = 0 while Process P10 finished at time = 222. The total number of Context switch = 9 as the processes move to completion.

The evaluation shall use the formula of *Waiting Time (WT)* to obtain the *Average Waiting Time (AWT)* for each scheduling algorithm. The *Turn-Around Time (TAT)* shall also be presented which will aid in calculating the *Average Turn-Around Time (ATAT)*.

FCFS:

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

P1: (0-0) = 0, where P1 is the first Process what arrived at Time = 0 and thus didn't wait

P2: (9-0) = 9, where P2 arrived at Time = 0 and scheduled at 9.

P3: (36-0) = 36, P4: (50-0) = 50, P5: (80-0) = 80, P6: (102-0) = 102, P7: (121-0) = 121, P8: (166-0) = 166, P9: (205-0) = 205, and P10: (212-0) = 212.

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus *AWT* = (0 + 9 + 36 + 50 + 80 + 102 + 121 + 166 + 205 + 212) / 10

= 981/ 10

= 98.1

Context Switch = 9

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus :TAT in the system will be thus: P1: (9-0) = 9, P2: (36-0) =36, P3: (50-0) = 50, P4: (80-0) =80, P5: (102-0) = 102, P6: (121-0) = 121, P7: (166-0) = 166, P8: (205-0) = 205, P9: (212-0) = 212, P10: (222-0)= 222.

ATAT = (9 + 36 + 50 + 80 + 102 + 121 + 166 + 205 + 212 + 222) / 10

= 1203/10

= 120.3

4.4.2 Shortest Job First (SJF)

Based on the SJF scheduling algorithm, the CPU is allocated to the processes after scanning through all the processes thereby following the order of the least burst time. This algorithm is also referred to as the *Shortest Remaining Time First (SRTF).*

Using the information on Table 4.19, the Gantt chart for SJF is presented thus:



Figure 4.2: Gantt chart SJF

SJF:

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

P1: (7-0) = 7, where P1 is the first Process what arrived at Time = 7 and thus had to wait for the shortest process to proceed.

P2: (81-0) = 81, where P2 arrived at Time = 0 and scheduled at 81.

P3: (26-0) = 26, P4: (108-0) = 108, P5: (59-0) = 59, P6: (40-0) = 40, P7: (177-0) = 177, P8: (138-0) = 138, P9: (0-0) = 0, and P10: (16-0) = 16.

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus *AWT* = (7 + 81 + 26 + 108 + 59 + 40 + 177 + 138 + 0 + 16) / 10

= 652/ 10

= 65.2

Context Switch = 9

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus: TAT in the system will be thus: P1: (16-0) = 16, P2: (108-0) =108, P3: (40-0) = 40,

P4: (138-0) =138, P5: (81-0) = 81, P6: (59-0) = 59, P7: (222-0) = 222, P8: (177-0) = 177,

P9: (7-0) = 7, P10: (26-0) = 26.

ATAT = (16 + 108 + 40 + 138 + 81 + 59 + 222 + 177 +7+ 26) / 10

= 874/10

= 87.4

4.4.3 Round Robin

A Round- Robin scheduling algorithm provides the mechanism for scheduling in a cyclic order using a metric measure referred to as *Quantum (q)*. The quantum is usually specified by the system designer. When the quantum is low, the CPU is usually utilized almost at optimal level with the disadvantage of so many switches. Using the Table 4.19, a quantum q = 10 is thus set.

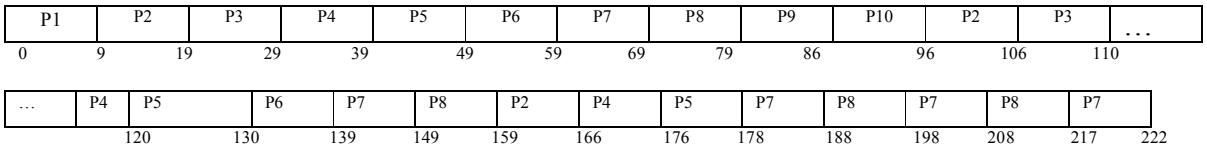


Figure 4.3: Gantt chart for RR with q=10

Context Switch = 24

AWT and ATAT shows that Round Robin has higher values as compared to the FCFS and SJF scheduling algorithm.

RR:

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

P1: (0-0) = 0, where P1 is the first process and the Quantum Q = 10, thus the context switch will work to the maximum of 10. It will be noticed that P1 has a burst time of 9.

P2: ((9+77+53)-0) =139, where P2 arrived at Time = 0 and waits on two occasions between processes.

P3: ((19+77)-0) = 96, P4: ((29+71+46)-0) = 146, P5: ((39+79+46)-0) = 164, P6: ((49+71)-0) = 120, P7: ((59+70+29+10+9)-0) = 177, P8: ((69+70+29+10)-0) = 178, P9: ((79)-0) = 79, P9 and P10 used up burst times on the first round of RR and P10: (86-0) = 86.

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus *AWT* = (0 + 139 + 96 + 146 + 164 + 120 + 177 + 178 +79 + 86) / 10

= 1185/ 10

= 118.5

Context Switch = 24

Defines the numbers of time the CPU switch between processes.

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus: TAT in the system will be thus: P1: (9-0) = 9, P2: (166-0) =166, P3: (110-0) = 110,

P4: (176-0) =176, P5: (178-0) = 178, P6: (139-0) = 139, P7: (222-0) = 222, P8: (217-0) = 217,

P9: (86-0) = 86, P10: (96-0) = 96.

ATAT = (9 +166 + 110 + 176 + 178 + 139 + 222 + 217 +86+ 96) / 10

= 1399/10

= 139.9

4.4.4 Priority Scheduling

Priority Scheduling is a scheduling algorithm that defines the order of process execution based on the priority values set (Randomly generated using the math.random in java). The major problem faced by the priority scheduling is *indefinite blocking (starvation)* (Ida *et al*, 1993). The lower number implies a higher the priority as seen on Table 4.19.

Priority Scheduling:

On evaluation using the Table 4.19 Process snap shot

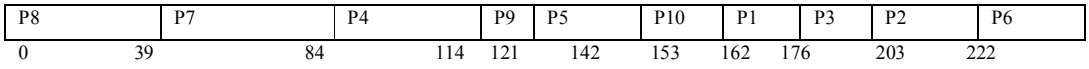


Figure 4.4: Gantt chart for Priority Scheduling

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

P1: (153-0) = 153, where P1 is the firs process but its priority value is 7, thus it makes it the seventh process to be assigned the CPU.

P2: (176-0) =176, where P2 arrived at Time = 0 and has a priority value of 9.

P3: (162-0) = 162, P4: (84-0) = 84, P5: (121-0) = 121, P6: (203-0) = 203, P7: (39-0) = 39,

P8: (0-0) = 0, this was as a result of having a priority value = 1, P9: (114-0) = 114,

P10: (142-0) = 142.

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus *AWT* = (153 + 176 + 162 + 84 + 121 + 203 + 39 + 0 +114 + 142) / 10

= 1194/ 10

= 119.4

Context Switch = 9

Defines the numbers of time the CPU switch between processes.

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus: TAT in the system will be thus: P1: (162-0) = 162, P2: (203-0) =203, P3: (176-0) = 176,

P4: (114-0) =114, P5: (142-0) = 142, P6: (222-0) = 222, P7: (84-0) = 84, P8: (39-0) = 39,

P9: (121-0) = 121, P10: (153-0) = 153.

ATAT = (162 +203 + 176 + 114 + 142 + 222 + 84 + 39 +121+ 153) / 10

= 1416/10

$= 141.6$

4.4.5 Longest Job First (LJF)

The LJF algorithm is the most uncommon among the scheduling algorithm; it schedules the process with the highest burst time before shorter processes. It is seen as the reverse of the SJF; the major disadvantage includes *starvation* of the shorter processes (Ida *et al*, 1993) (Cheng *et al*, 2006).

This form the bases for this research as the new CBT frame work aim at solving the problems of *starvation* and *convoy*.

Using the Table 4.19 of the Process snap shot, a Gantt chart is used as thus:

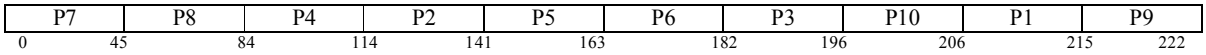


Figure 4.5: Gantt chart for LJF Scheduling

For the LJF, a rearrangement of the snap shot is necessary to show its ordering. Processes are presented in the following order.

Table 4.4: Process with burst time

Processes	Burst time
P7	45
P8	39
P4	30
P2	27
P5	22
P6	19
P3	14
P10	10
P1	9
P9	7

LJF:

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

P1: (206-0) = 206, where P1 is the first Process what arrived at Time = 0 and thus had to wait for the longer processes to proceed.

P2: (114-0) = 114, where P2 arrived at Time = 0 and scheduled at 114.

P3: (182-0) = 182, P4: (84-0) = 84, P5: (141-0) = 141, P6: (163-0) = 40, P7: (0-0) = 0, P8: (45-0) = 45, P9: (215-0) = 215, and P10: (206-0) = 206.

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus AWT = (206 + 114 + 182 + 84 + 141 + 163 + 0 + 45 + 215 + 206) / 10

= 1356/ 10

= 135.6

Context Switch = 9

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus :TAT in the system will be thus: P1: (215-0) = 215, P2: (141-0) =141, P3: (196-0) = 196, P4: (114-0) =114, P5: (163-0) = 163, P6: (182-0) = 182, P7: (45-0) = 45, P8: (84-0) = 84, P9: (222-0) = 222, P10: (206-0) = 206.

ATAT = (215 + 141 + 196 + 114 +163 + 182 + 45 + 84 +222+ 206) / 10 = 1568 / 10

= 156.8

4.4.6. Longest Job First + CBT

This is the research contribution to the field of operating system scheduling design, the CBT model is now evaluated and the results will be compared to the entire existing scheduling algorithm concept. The model designed on Chapter Three is fully tested and the procedures are presented as follows:

Let all processes be sorted and merge as conditions in the model suggest.

Table 4.5: Processes sorted in decreasing order of burst time.

Processes	Burst time
P7	45
P8	39
P4	30
P2	27
P5	22
P6	19
P3	14
P10	10
P1	9
P9	7

From the CBT model, some processes satisfied the condition of merging resulting a Longer processes because their burst times are less than $c_{wa}=\frac{\sum_{i=1}^{n}\delta i}{n}$, from equation (2). Consequently,

Processes P5 and P6 merge by the CBT model resulting into *P5, 6 = 41, Process P3 merge with Process P10 to generate *P3, 10 = 24 and lastly the Processes P1 and P7 to obtain *P1, 7= 16.

Sorting and presenting the result shows a new identification and burst times of the CBT.

Table 4.6: Longer Jobs + CBT

Processes	Burst time
P7	45
*P5,6	41
P8	39
P4	30
P2	27
*P3,10	24
*P1,9	16

Results shows that the total processes are reduced to seven (7).

The Gantt chart and evaluation calculations is shown on Fig 4.6 Gantt chart for the *LJF+CBT*

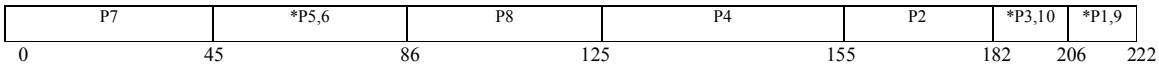


Figure 4.6 Gantt chart for LJF+CBT

LJF + CBT:

Waiting Time (WT) = Time first Scheduled – Arrival time

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

For each process, WT= *Time first Scheduled – Arrival Time*

*P1, 9: (206-0) = 206, where *P1, 9 is the result of merging P1 and P9, at Time = 0 and thus had to wait for the longer processes to proceed.

P2: (155-0) = 155, where P2 arrived at Time = 0 and scheduled at 155.

*P3,10: (182-0) = 182, P4: (125-0) = 125, *P5,6: (45-0) = 45, P7: (0-0) = 0, P8: (86-0)= 86

From the formula

Average Waiting Time (AWT) = Sum of all Processes Waiting Time/ Total numbers of Processes.

Thus AWT = (206 + 155 + 182 + 125 + 45 + 0 + 86) / 10

= 799/ 10

= 79.9

Context Switch = 6

In summary, the LJF+CBT gives an *Average Waiting Time (AWT)* far less than all scheduling algorithms presented. The *Context Switch (CS) = 6.*

Turn Around Time (TAT) = Overall Time a process spent in the system

= Time of Process completion – Arrival Time

Average Turn-Around Time = Sum of each Process / Total numbers of Processes.

Thus: TAT in the system will be thus: *P1, 9: (222-0) = 222, P2: (182-0) =182,

*P3, 10: (206-0) =206, P4: (155-0) =155, *P5, 6: (86-0) = 86, P7: (45-0) = 45,

P8: (125-0) = 125.

ATAT = (222 +182 + 206 + 155 +86 + 45 +125) / 10

= 1021/10

= 121.1

4.5. COMPARING EXISTING SCHEDULING ALGORITHMS WITH LJF+CBT

Graphical comparison will be discussed as each scheduling algorithm shall be compared to the proposed LJF+CBT. Operating system which serves as an interface between the user and the hardware components or devices (Galvin, 2004) will function more diligently through its process scheduling when it has minimal overall time, least context switches, and minimised average turn around time and gives a fairer consideration to shorter jobs. By achieving all the listed advantages, this research compared the following scheduling algorithms with the proposed LJF+CBT results:

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin (RR)
4. Longest Job First (LJF)
5. Priority Scheduling

Average Waiting Time (AWT), Average Turn-Around Time (ATAT), Context Switches (CS) is used as the evaluation metrics.

4.5.1 Comparing FCFS and LJF+CBT

The First Come First Served (FCFS) scheduling algorithm produced the results of an *Average Waiting Time (AWT)* and *Average Turn-Around Time (ATAT)* better than the LJF+CBT after the program ran for about 100 times varying the sizes of the job and the nature of the burst time generation. It was also found out that for a smaller number of processes involving shorter jobs, the model has less effect in performance. For the FCFS, the numbers of Context *Switch (CS)* are always higher than the LJF+CBT.

4.4.2 Comparing SJF and LJF+CBT

Also using the evaluation calculation results obtained in the Shortest Job First algorithm, LJF+CBT results are compared. These proved that the SJF performance can be equally competed against LJF+CBT. The SJF always has a lower AWT and ATAT than the LJF+CBT; its disadvantage against the proposed LJF+CBT is the large number of Context Switches. The program was also run for about 100 different cases varying the numbers of processes and the ranges of the burst time for proper documentation of the behaviour of the CBT.

4.4.3. Comparing LJF and LJF+CBT

Results of the LJF Scheduling algorithm proved that the LJF+CBT performed far better than the ordinary LJF. The numbers of Context Switches and the large values of AWT and ATAT proved that LJF+CBT are a good solution to the problems that LJF could not fully solve. In the experiment and program analysis, the proposed model designed in Java was put to test for about a 100 sample cases of burst times and process numbers variation.

4.4.4 Comparing RR and LJF+CBT

Results of the RR Scheduling algorithm obtained using the proposed LJF+CBT; proves that the LJF+CBT performed better that the RR. Careful selection of the Quantum time and the numbers of the processes against different ranges proved that the large numbers of context switch, AWT and ATAT makes RR perform worse than all tested scheduling algorithms.

4.4.5 Comparing Priority Scheduling and LJF+CBT

Results of the Priority Scheduling algorithm obtained using prove that the LJF+CBT performed better that the Priority Scheduling. Using 100 variations of processes ranging numbers between 1 and 10,000 numbers of processes. The Priority scheduling had Context switches lager than the LJF+CBT, thus a large number of CS implies the CBT performed better as compared. For fewer numbers of process burst time range, the Priority also defeated the CBT model.

4.5. RESULTS DISCUSSION AND CONCLUSION

From the results of five scheduling algorithms (FCFS, SJF, LJF, RR and Priority Scheduling) compared with the LJF+CBT, it has shown that the CBT has lesser *Context switches (CS), Average Waiting Time (AWT)* and *Average Turn Around-Time (ATAT)* against LJF and RR. Generally, the SJF proved to be the best in times of performance with a slight disadvantage in numbers of context switches. It can further be suggested that CBT will perform better when implemented in any operating system design as a scheduling type.

The figure 4.7 shows the capture of the program environment and some results of the computation. Elaborate details of the source code and results obtained are presented in the appendix page.

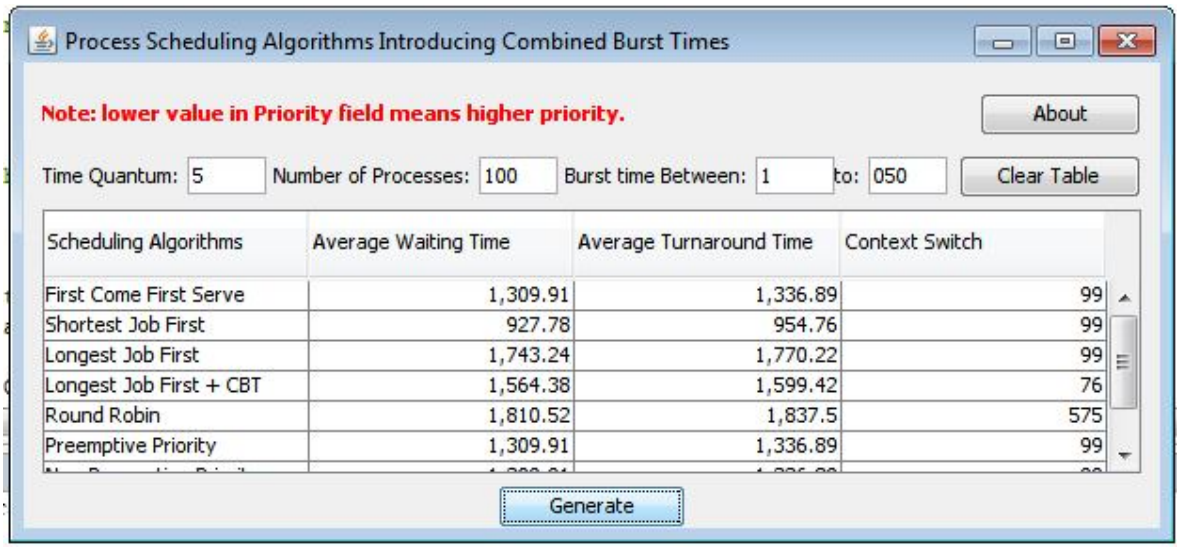


Figure 4.7 Process Scheduling in execution

The following shows a sample program and the relationships between the scheduling algorithms.

Table 4.9: Results showing Statistics for: 2000 Processes, Range of Burst times = 10-50.

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	30054.84	30084.79	1999
Shortest Job First	23168.64	23198.59	1999
Longest Job First	36692.4	36722.35	1999
Longest Job First + CBT	33485.68	33525.23	1513
Round Robin	45712.24	45742.18	12772
Preemptive Priority	30054.84	30084.79	1999
Non-preemptive Priority	30054.84	30084.79	1999

Results from Table 4.9, shows that the implemented program in java was ran for 2000 processes along side burst times of values 10 to 50. The First Come First Served, Pre-emptive priority and non-premptive priority scheduling has values of Average Waiting Time 30054.84. For Longest Job First scheduling algorithm, the values 36692.4 as against 33485.68 with the CBT inclusion. The results prove that the LJF+CBT produced lower values in context switches.

Table 4.10: Results showing Statistics for: 1500 Processes, Range of Burst times = 1-100

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	37511.85	37561.97	1499
Shortest Job First	25165.06	25215.18	1499
Longest Job First	49965.81	50015.93	1499
Longest Job First + CBT	44824.66	44891.19	1129
Round Robin	49923.5	49973.62	15644
Preemptive Priority	37511.85	37561.97	1499
Non-preemptive Priority	37511.85	37561.97	1499

Results from Table 4.10, shows that the implemented program in java was ran for 1500 processes along side burst times of values 1 to 100. The First Come First Served, Pre-emptive priority and non-premptive priority scheduling has values of Average Waiting Time and Average Turnaround Times as 37511.85 and 37561.97 respectively. For Longest Job First scheduling algorithm, the values 49965.81 as against 44824.66 with the CBT inclusion. The results prove that the LJF+CBT produced lower values in context switches.

Table 4.11: Results of the Average Waiting Times obtained.

			Average Waiting Time				
No of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	30	19.2	42.9	39	37	30	30
15	148.19	103.06	218	199.08	204.06	148.19	148.19
25	255.2	183.88	403.64	358.58	354.12	255.2	255.2
50	674.68	468.76	866	787.44	930.02	674.68	674.68
100	1250.2	828.94	1658.93	1494.53	1631.95	1250.2	1250.2
150	4048.04	2745.22	5056.41	4525.55	5454.49	4048.04	4048.04
200	5342.59	3846.69	6835.63	6164.7	7537.13	5342.59	5342.59
250	2115.08	1760.96	2444.15	2307.18	3457.25	2115.08	2115.08
500	6329.22	4209.31	8351.51	7464.96	8303.04	6329.22	6329.22
1000	25267.13	16694.75	33561.93	29980.91	330001.93	25267.13	25267.13
1500	37964.17	25347.79	49800.07	44690.48	50369.89	37964.17	37964.17
2000	30267.47	23243.22	36942.66	33607.94	45831.61	30267.47	30367.47

Table 4.11 presents the results obtained for the different times of runs and different processes generated. Major significant proved that the SJF still remains the algorithm with the best numbers in terms of the Average waiting time. The research even though focused solely on reducing the starvation problem contained in the LJF by introducing the CBT; as such the LJF+CBT algorithm produced values less than the ordinary LJF on all occasions.

Table 4.12: Results of the Average Turn-around times obtained by average

Number of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	43.4	39.2	58.7	56	59.7	43.4	43.4
15	171.13	126	240.93	227.75	227	171.13	171.13
25	279.68	208.36	428.12	391.05	378.6	279.68	279.68
50	662.24	437.7	874.02	800.95	834.92	662.24	662.24
100	1275.33	854.07	1684.06	1527.16	1657.08	1275.33	1275.33
150	4124.7	3075	5323.61	4828.02	5953.41	4124.7	4124.7
200	5565.4	4016.41	7072.75	6376.25	7827.01	5565.4	5565.4
250	2131.97	1777.84	2461.04	2329.76	3474.14	2131.97	2131.97

500	4237.39	3598.56	4923.44	4688.64	7023.88	4237.39	4237.39
1000	24762.54	17025.77	33528.72	30138.03	33330.02	24762.54	24762.54
1500	37764.61	25678.72	50610.1	45545.35	50846.75	37764.61	37764.61
2000	29626.98	22786.09	36491.52	33229.31	44826.49	29626.98	29626.98

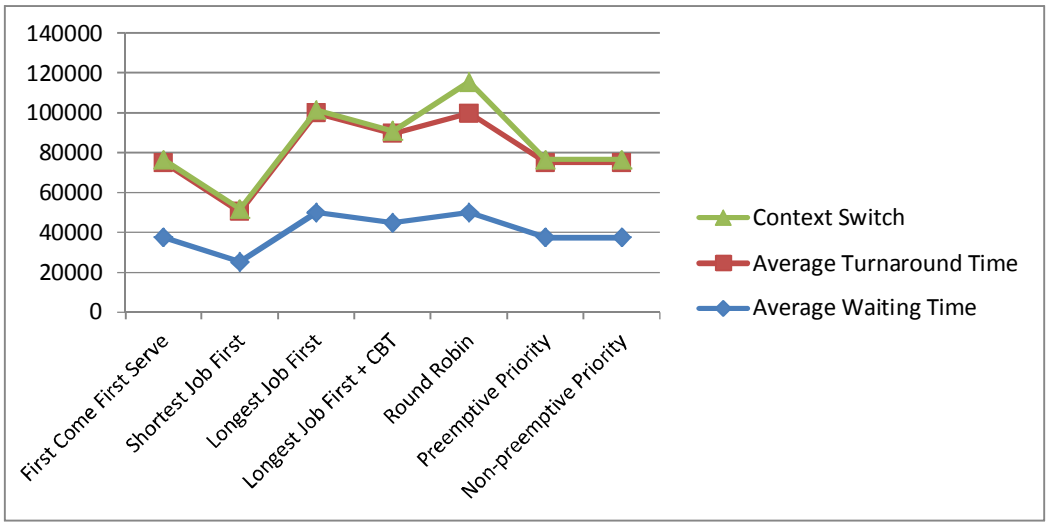


Figure 4.8 Graph of 1500 sample processes with burst time 1-100

The graph in Figure 4.8 shows the overall summary of the standard metrics against the tested scheduling algorithms as the series guide through representation of the graph.

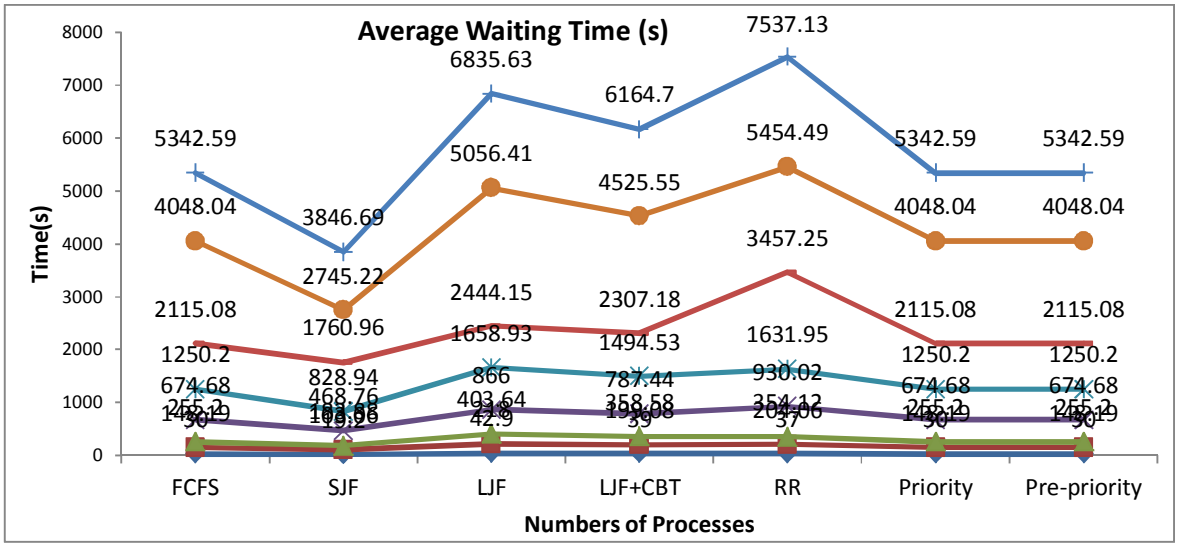


Figure 4.9: Graph of average waiting time for sample processes 10-2000

The figure 4.9 shows the overall result of the average waiting time for all cases of values taken. It is observed from the graph that the performance of LJF, RR is less than the LJF+CBT; as such the average waiting time in LJF and RR is reduced.

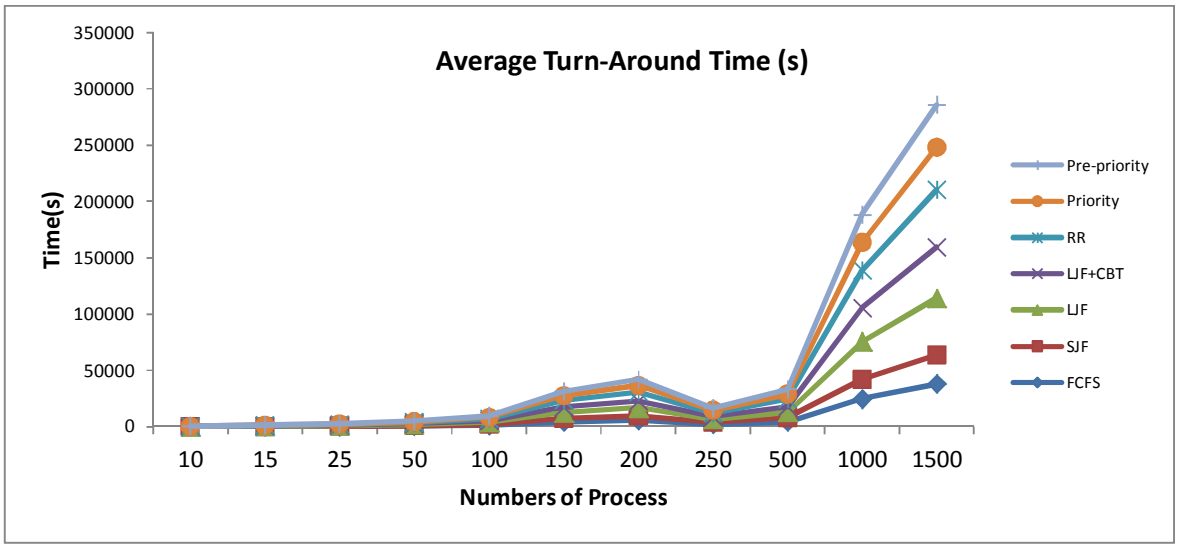


Figure 4.10: Graph of average turn-around times for sample processes 10-2000

The graph shows the average turn-around times of the process samples.

Table 4.13: Results of the Context Switches obtained.

Number of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	9	9	9	6	24	9	9
15	14	14	14	11	75	14	14
20	19	19	19	14	66	19	19
25	24	24	24	18	127	24	24
30	29	29	29	21	91	29	29
35	34	34	34	25	91	34	34
40	39	39	39	29	132	39	39
45	44	44	44	35	142	44	44
50	49	49	49	37	269	49	49
100	99	99	99	76	545	99	99
150	149	149	149	113	591	149	149
200	199	199	199	152	803	199	199

250	249	249	249	182	1473	249	249
500	499	499	499	370	1106	499	499
1000	999	999	999	753	3155	999	999
1500	1499	1499	1499	1138	15841	1499	1499
2000	1999	1999	1999	1491	12820	1999	1999

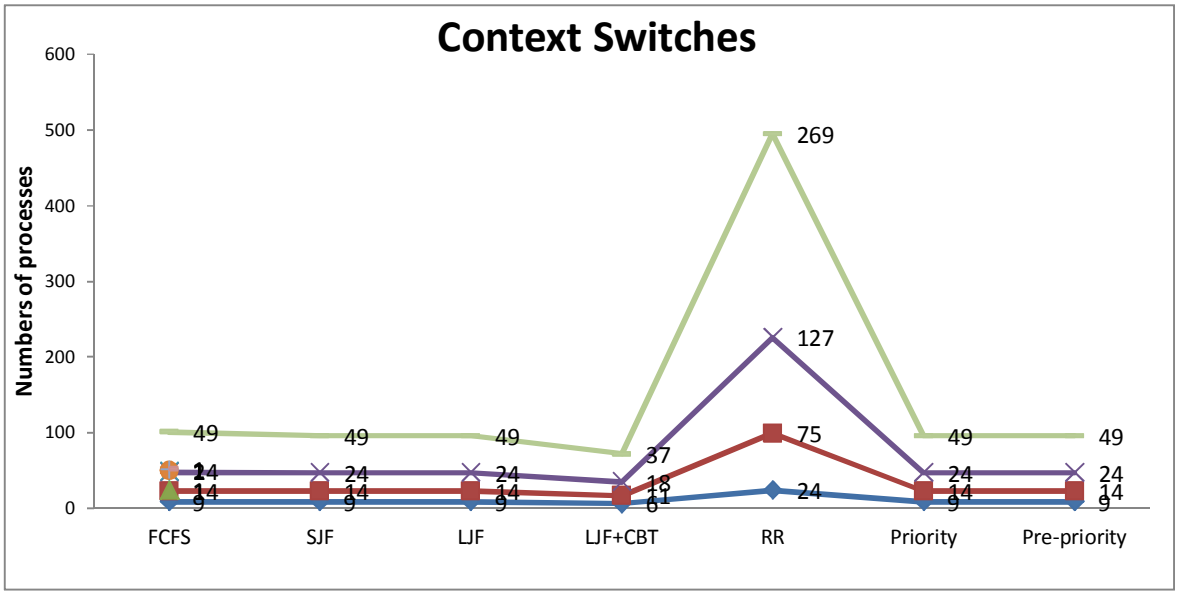


Figure 4.11: Graph of Context Switches for sample processes 10-2000

Operating system efficiency is measured by the maximum utilization of the CPU, the utilization of the CPU is always obtained when there exist lesser numbers of context switches. From our research, the LJF+CBT have the best of values against the tested scheduling algorithms for all numbers of process cases. In the first category comprising of 10 processes, all algorithms except the RR and LJF+CBT has 9 context switches and the RR and LJF+CBT has the value 24 and 6 respectively. For 50 processes, the context switch value for the LJF+CBT was less as it produced 37 context switches against 49 for FCFS, SJF, Priority and 269 for RR.

CHAPTER FIVE

5.0 SUMMARY, CONCLUSION AND RECOMMENDATION

Process scheduling remain the key to achieving good performance of every system and as such the choice of choosing the type that best suit problems cannot be over emphasized, for a system to choose its scheduling algorithm, the designers must take into consideration the nature and order of the processes involved. In so doing, in multiprocessing systems involving lots of shorter jobs with due dates, it can be suggested that the research proved Longest Job First (LJF) cannot be fully thrown away as it has some parts which it could be very helpful.

From practical human activities in the hospitals, LJF is used as cases of emergency are termed cases with the highest burst time due to the life saving situation involved. Therefore, LJF+CBT will be a great option to minimise the starvation of the shorter processes in the hospital system.

5.1 Summary

The objective of the research was to propose a frame work that would minimise the Average waiting times, the Average turn-around time and reduce the numbers of context switches thereby reducing the starvation problems faced by shorter jobs in the Longest Job First (LJF). The model was developed as stated and implemented using Java to test different occurrences. A user specifies the nature of the processes, the ranges of the bust time and run the program as many times as possible to compare and analyse the results written to a log file.

From the series of problems ran using the proposed model, the research has found out that the LJF+CBT can compete along other scheduling algorithms in the cases where due dates are fixed, and larger processes exist in the system. It will be noticed that the results obtained showed that if a system has a close range interval in the burst time, the LJF is also a good algorithm for use.

5.2 Future work

For future work as related to this research, it is suggested that the following areas need to be looked into:

a. A test of the LJF with processes assigned different arrival times.
b. A continuous arrival to the job pool queue with job prediction using exponential averages
c. A new condition for selecting the threshold of job categorization
d. Better merging order different from the one used.
e. Representing the LJF+CBT in memory.

5.3 Conclusion

Sequel to the problems definitions in chapter one, the research has answered the listed problems it set out to answer. The major problem of starvation faced in the LJF has been minimised.

Processes were randomly generated along the burst times and conditions set were carefully computed as the result of the work supports the conclusion that the starvation problem in LJF was reduced. Users and designers of operating system will find LJF+CBT helpful in implementing aging and the multi-level feedback queues in their designs.

Based on the results of the experiments carried out, LJF+CBT will be very useful in mechanical and electrical engineering machine buildings; since machine utilization and strength decreases as the usage persist.

References

A Scheduling: Computer Scheduling. Retrieved October 27, 2011, from Scheduling: http://www.cs.sunysb.edu/~algorith/files/scheduling.shtml

Achim, S. (2005). *On Performance Evaluation of Slackness option for the Self-turning dynP scheduler.* Central Institute for Applied Mathematics. Julich, Germany: RSJ.

Ali Bagherinia, Ali Jaharpour and Sohrab Hojatkhah (2011). *Backtracking Algorithm for Process Scheduling in Multi Processor Systems.*International Conference on Computational Techniques and Artificial Intelligence (ICCTAI). pp 17-18.

Baptiste, P. (1994). *Constraint- Based Scheduling : Two Extensions.* University of Strathclyde, Department of Manufacturing and Engineering . Scotland:

Boleslaw, S. K. (1990). Mutual Exclusion Revisited. *Fifth Jerusalem Conference in Information Technology* (pp. 110-117). Israel, Jerusalem: IEEE Computer Society Press.

Cheng, T., & Kahlbacher, H. (2006). *A proof for the Longest Job First Policy in one machine Scheduling*. CA: Naval Research Logistics Vol 38, Issues 5, pp 715-720.

CS/CPE408. (2001, June 4). *Projects*. Retrieved October 29, 2011, from Bridge Port: http://stargazer.bridgeport.edu/sed/projects/cs503/Spring_2001/kode/os/

Dalibor, K., Rudova, H., Raneiri, B., Marco, P., & Capannini, G. (2005). Comparison of Multi-Criteria Scheduling. Retrieved October 29, 2011, http://hpc.isti.cnr.it/~gabriele/pdf/1319569468.pdf

Dong, H. (2005) Retrieved October 27, 2011 *MSc Thesis*. *Delaying Convoy.* Naval Postgraduate School, Computer Science. California
: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA443502

EE442. (2222, 2 121).[online]. Retrieved October 29, 2011, from Schoool web site: www.eee.metu.edu.tr/~halici/courses/442/442index.html

Ernst, B. W., Schroeder, B., & Urvoykella, G. (2007). Scheduling in Practice. *Journal of Parallel Processing , 34* (4), 2-7.

Fluhrer, S., Itsik, M., & Adi, S. (2001). Weakness in the Key scheduling Algorithm of RC4. *Proceedings 01 SAC Revised papers from the 8th Annual workshop in Cryptography* (pp. 1-24). London: Springer.

Galvin, P. B., Abrahm, S., & Gagne, G. (2004).Ch.5 *Operating System Concepts* (7th ed.). (B.Zobrist, K. Santor, & M. Lesure, Eds.) NJ, USA: John Wiley and Sons Inc.

Gerald, S., Garima, K., & P, S. (2004). Job Fairness in Non-preepmtive Job Scheduling. *International Journal on Parallel Processing* , 186-194.

Guangzhi Li and Gianfranco Ciardo, (1998). Approximate analysis of Fault tolerant Join Shortest Queue Policy. http://citeseerx.ist.psu.edu/viewdoc/versions;jsessionid=189C5AE602AE20C1A822DF9E7642A78E?doi=10.1.1.16.2108

Haziza, F. (2007).Spring Lecture Slides(Scheduling Algorithms) *Uppsal University.* Uppsala Univeristy, Department of Computer System. Sweden.pp1-33

Herve, M. (2007). On Scheduling fees to prevent merging, splitting and transfering Jobs. *Journal of Mathematics of Operational Research* , 266-283

Hoeksma, R. (2010). Retrieved September 10, 2011. *Price of Anarchy for Machines Scheduling with sum of Completion times Objectives.* University of Twente, Computer Engineering. Twente: http://essay.utwente.nl/59756/1/MA_theisis_R_Hoeksma.pdf

Ida, F. A., & McIver, A. M. (1993). *PWS Publishing Company.* Retrieved Octorber 29, 2011, from OCities Website: http://www.oocities.org/ftcc_operatingsystems/chapter_5a.html

Idris, R. A., & Okopa, M. (2011). Modeling and Evaluation of SWAP scheduling Policy under varying Job size distribution. *Network Conference* (pp. 115-120). Uganda: IARIA.

Javier, C., Fatos, X., & Ajith, A. (2007). Genetic Algorithm Based Schedulers for grid Computing Systems. *International Journal of Innovative Computing, Information and Control , 3* (6), 25-32.

Krishnan P, (1995). Online Prediction for Databases and Operating Systems.*Phd Thesis from Brown University, Providence Rhode Island* Retrieved September, 2011 ftp://128.148.32.111/pub/techreports/95/cs95-24.pdf

Lee, I. (2007). *Software System, OS Overview.* University of Pennsylvania, Department of Computer and information Science. Pennsylvania: unpublished.

Ludmila, C., Lau, R., Burose, H., Subramanian, V., Bernhard, K., & Kuttiraja, V. (2011). Run-Time Performance Optimization and Job Management in Data Protection Solution. *IFIP/IEEE Symposium in Integrated Management* (pp. 20-27). Dublin: IEEE.

Matthias, B., Anne, H. C., Melanie, B., Andrea, J., Jens, J., Hubert, S., et al. (2004, February 3). *Physiol Genomics*. Retrieved October 29, 2011,

Nagesh, L., Rao, S., & Kim, H. (2009). Asymmetry Aware scheduling Algorithms for asymmetric Multiprocessors. *LJFFF* http://www.ideal.ece.ufl.edu/workshops/wiosca08/paper6.pdf

Palem, H., & Karamcheti, N. (1999, March 4). *school.* Retrieved November 21, 2011, from G22: 3250 Honors Operating System: http://www.scs.stanford.edu/nyu/03sp/notes/l1.pdf

Prashant Shenoy, (Fall,2008).*CPU Scheduling Lecture Slides*. Retrieved November 21, 2011, from UMASS Operating System http://lass.cs.umass.edu/~shenoy/courses/fall08/lectures/Lec07.pdf

Raksha, S., Vishnu, K., Mishra, M. K., & Bhuyan, P. (2010). A Survey of Job Scheduling and Resource Management in Grid Computing. *World Academy of Science Engineering and Technology Journal* ,pp 461-466.

Rakesh Mohanty, Manas Das, M. Lakshmi Prasanna and Sudhashree. (2011). Design and Performance Evaluation of a New Proposed Fittest Job First Dynamic Round Robin Scheduling Algorithm .*International Journal of Computer Information Systems* , Vol2, No.2 pp 23-93.

Raz, D. (2007). Retrieved October 11,2011. *Quantifying Job Fairness in Queuing Systems.* Tel-Aviv University, Computer Science. Tel-Aviv: http://www.cs.tau.ac.il/thesis/thesis/Raz-Thesis.pdf

Reinder, B. J. (2007, Februrary 7). *System Architecture Networks.* Retrieved November 29, 2011, from SAN

RIOT. (2000). *RIOT Co*. Retrieved October 29, 2011, from http://riot.ieor.berkeley.edu/riot/Applications/scheduling/algorithms.html

Schedulers. (n.d.). *Scheduling*. Retrieved March 12, 2011, from:Kohlers Harvard http://www.read.seas.harvard.edu/~kohler/class/05s-osp/notes/notes5.html

Scheduling Algorithm. (n.d.). Retrieved March 12, 2011, from GITAM http://www.gitam.edu/eresource/comp/gvr(os)/5.3.htm

Scheduling(Computing). (n.d.). *Wikipedia*. Retrieved October 27, 2011, from Wiki: http://en.wikipedia.org/wiki/scheduling_(computing)

Stallings, W. (2001). *Operating Systems, Internals and design Principles* (4th ed.). Prentice Hall 2001, ISBN 0-13-031999-6 CA, USA:

Understanding Operating Systems. Retrieved octorber 29, 2011, from Process management: http://www.darkridge.com/~jpr5/archive/alg/node148.html

Vissarion, F. (2011). Study of the Effect of Cost Policies in the convergence of selfish strategies in Pure Nash equilibrium in Congestion Games. Retrieved October 29, 2011 http://users.uoa.gr/~vfisikop/projects/thesis.pdf

APPENDIX A: RESULTS OF THE SIMULATION IN LOG.TXT

			Average Waiting Time				
No. of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	30	19.2	42.9	39	37	30	30
15	148.19	103.06	218	199.08	204.06	148.19	148.19
25	255.2	183.88	403.64	358.58	354.12	255.2	255.2
50	674.68	468.76	866	787.44	930.02	674.68	674.68
100	1250.2	828.94	1658.93	1494.53	1631.95	1250.2	1250.2
150	4048.04	2745.22	5056.41	4525.55	5454.49	4048.04	4048.04
200	5342.59	3846.69	6835.63	6164.7	7537.13	5342.59	5342.59
250	2115.08	1760.96	2444.15	2307.18	3457.25	2115.08	2115.08

			Average Turn-Around Time				
No. of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	43.4	39.2	58.7	56	59.7	43.4	43.4
15	171.13	126	240.93	227.75	227	171.13	171.13
25	279.68	208.36	428.12	391.05	378.6	279.68	279.68
50	662.24	437.7	874.02	800.95	834.92	662.24	662.24
100	1275.33	854.07	1684.06	1527.16	1657.08	1275.33	1275.33
150	4124.7	3075	5323.61	4828.02	5953.41	4124.7	4124.7
200	5565.4	4016.41	7072.75	6376.25	7827.01	5565.4	5565.4
250	2131.97	1777.84	2461.04	2329.76	3474.14	2131.97	2131.97
500	4237.39	3598.56	4923.44	4688.64	7023.88	4237.39	4237.39
1000	24762.54	17025.77	33528.72	30138.03	33330.02	24762.54	24762.54
1500	37764.61	25678.72	50610.1	45545.35	50846.75	37764.61	37764.61
2000	29626.98	22786.09	36491.52	33229.31	44826.49	29626.98	29626.98

			Context Switches				
No. of Processes	FCFS	SJF	LJF	LJF+CBT	RR	Priority	Pre-priority
10	9	9	9	6	24	9	9
15	14	14	14	11	75	14	14
25	24	24	24	18	127	24	24
50	49	49	49	37	269	49	49
100	99	99	99	76	545	99	99

150	149	149	149	113	591	149	149
200	199	199	199	152	803	199	199
250	249	249	249	182	1473	249	249
500	499	499	499	370	1106	499	499
1000	999	999	999	753	3155	999	999
1500	1499	1499	1499	1138	15841	1499	1499
2000	1999	1999	1999	1491	12820	1999	1999

SAMPLE
PROCESSES

Statistics for: 15 Processes, Range of Burst times = 1-50

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	148.19	171.13	14
Shortest Job First	103.06	126	14
Longest Job First	218	240.93	14
Longest Job First + CBT	199.08	227.75	11
Round Robin	204.06	227	75
Preemptive Priority	148.19	171.13	14
Non-preemptive Priority	148.19	171.13	14

Statistics for: 25 Processes, Range of Burst times = 1-50

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	255.2	279.68	24
Shortest Job First	183.88	208.36	24
Longest Job First	403.64	428.12	24
Longest Job First + CBT	358.84	391.05	18
Round Robin	354.12	378.6	132
Preemptive Priority	255.2	279.68	24
Non-preemptive Priority	255.2	279.68	24

Statistics for: 50 Processes, Range of Burst times = 1-50

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	636.52	662.24	49
Shortest Job First	411.98	437.7	49
Longest Job First	848.3	874.02	49

Longest Job First + CBT	768.8	800.95	39
Round Robin	809.2	834.92	276
Preemptive Priority	636.52	662.24	49
Non-preemptive Priority	636.52	662.24	49

Statistics for: 100 Processes, Range of Burst times = 1-50

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	1137.22	1160.6	99
Shortest Job First	785.24	808.63	99
Longest Job First	1530.37	1553.76	99
Longest Job First + CBT	1375.2	1406.38	74
Round Robin	1525.06	1548.45	503
Preemptive Priority	1137.22	1160.6	99
Non-preemptive Priority	1137.22	1160.6	99

Statistics for: 150 Processes, Range of Burst times = 10-100

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	4048.04	4100.39	149
Shortest Job First	2745.22	2797.58	149
Longest Job First	5056.41	5108.77	149
Longest Job First + CBT	4525.55	4595.67	111
Round Robin	5454.49	5506.85	2664
Preemptive Priority	4048.04	4100.39	149
Non-preemptive Priority	4048.04	4100.39	149

Statistics for: 200 Processes, Range of Burst times = 10-100

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	5434.85	5488.35	199
Shortest Job First	3782.03	3835.52	199
Longest Job First	6863.47	6916.97	199
Longest Job First + CBT	6166.73	6236.66	152
Round Robin	7450.75	7504.24	803
Preemptive Priority	5434.85	5488.35	199
Non-preemptive Priority	5434.85	5488.35	199

Statistics for: 250 Processes, Range of Burst times = 10-15

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	1547.44	1559.88	249
Shortest Job First	1424.01	1436.44	249
Longest Job First	1671.55	1683.98	249
Longest Job First + CBT	1740.22	1756.85	186
Round Robin	2757.56	2770	1122
Preemptive Priority	1547.44	1559.88	249
Non-preemptive Priority	1547.44	1559.88	249

Statistics for: 250 Processes, Range of Burst times = 1-15

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	1021.13	1029.38	249
Shortest Job First	732.49	740.74	249
Longest Job First	1321.26	1329.5	249
Longest Job First + CBT	1189.09	1200.12	186
Round Robin	1385.67	1393.92	774
Preemptive Priority	1021.13	1029.38	249
Non-preemptive Priority	1021.13	1029.38	249

Statistics for: 500 Processes, Range of Burst times = 10-25

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	4431.48	4449.1	499
Shortest Job First	3754.73	3772.35	499
Longest Job First	5038.64	5056.27	499
Longest Job First + CBT	4842.1	4865.66	373
Round Robin	7363.19	7380.81	3110
Preemptive Priority	4431.48	4449.1	499
Non-preemptive Priority	4431.48	4449.1	499

Statistics for: 1000 Processes, Range of Burst times = 1-100

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	24712.04	24762.54	999
Shortest Job First	16975.27	17025.77	999
Longest Job First	33478.22	33528.72	999
Longest Job First + CBT	30071.05	30138.03	753

Round Robin	33279.52	33330.02	3155
Preemptive Priority	24712.04	24762.54	999
Non-preemptive Priority	24712.04	24762.54	999

Statistics for: 1500 Processes, Range of Burst times = 1-100

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	37713.79	37764.61	1499
Shortest Job First	25627.89	25678.72	1499
Longest Job First	50559.28	50610.1	1499
Longest Job First + CBT	45478.41	45545.35	1138
Round Robin	50795.92	50846.75	15841
Preemptive Priority	37713.79	37764.61	1499
Non-preemptive Priority	37713.79	37764.61	1499

Statistics for: 2000 Processes, Range of Burst times = 10-50

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	29800.12	29829.97	1999
Shortest Job First	23038.13	23067.97	1999
Longest Job First	36623.02	36652.86	1999
Longest Job First + CBT	33354.37	33394.06	1503
Round Robin	45322.55	45352.39	12723
Preemptive Priority	29800.12	29829.97	1999
Non-preemptive Priority	29800.12	29829.97	1999

Statistics for: 22 Processes, Range of Burst times = 1-12

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time	Context Switch
First Come First Serve	70.72	77.27	21
Shortest Job First	51.27	57.81	21
Longest Job First	86.18	92.72	21
Longest Job First + CBT	76.29	84.76	16
Round Robin	101.36	107.9	57
Preemptive Priority	70.72	77.27	21
Non-preemptive Priority	70.72	77.27	21

APPENDIX B: JAVA SOURCE CODES OF THE LJF+CBT

1. Author.java

```
package scheduling;

public class About extends javax.swing.JDialog {

    /** Creates new form About */

    public About(java.awt.Frame parent, boolean modal) {

        super(parent, modal);

        initComponents();

    }

    @SuppressWarnings("unchecked")

    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents

    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();

        jButton1 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        setTitle("Process Scheduling Algorithms");

        jLabel1.setText("<html><h2><i>Process Scheduling Algorithms
CBT</i></h2><p><b>Developed By:</b><br><font color=blue> 1.  Sharafat Ibn Mollah
Mosharraf <br> and Modified By  2.   Ibrahim Abdullahi</font><br>Department
of Computer Science & Engineering,<br>University of Dhaka.<br>Department of
Mathematics<br>Ahmadu Bello University,
Zaria<br>sharafat_8271@yahoo.co.uk<br>ibrojay01@yahoo.co.uk<br>http://www.sharafat.inf
o</p></html>");

        jButton1.setMnemonic('C');

        jButton1.setText("Close");

        jButton1.addActionListener(new java.awt.event.ActionListener() {

            public void actionPerformed(java.awt.event.ActionEvent evt) {

                jButton1ActionPerformed(evt);

            }
```

```
        });
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jLabel1))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(116, 116, 116)
                        .addComponent(jButton1)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 34,
Short.MAX_VALUE)
                .addComponent(jButton1)
                .addContainerGap())
        );
        pack();
    }// </editor-fold>//GEN-END:initComponents
```

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
        this.dispose();
    }//GEN-LAST:event_jButton1ActionPerformed
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration//GEN-END:variables


}
```

2. Main.java

```
package scheduling;
import java.io.*;
import java.util.ArrayList;
import javax.swing.*;
import java.awt.*;
import javax.swing.table.DefaultTableModel;
public class Main extends javax.swing.JFrame {


    /** Creates new form Main */
    public Main()
    {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
```

```
private void initComponents() {
    jScrollPane1 = new javax.swing.JScrollPane();
    table = new javax.swing.JTable();
   // add = new javax.swing.JButton();
   // delete = new javax.swing.JButton();
    Generate = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    fcfs = new javax.swing.JPanel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    tt = new javax.swing.JLabel();
   statlabel = new javax.swing.JLabel();
    wtFinal = new javax.swing.JLabel();
    ttFinal = new javax.swing.JLabel();
    wt = new javax.swing.JLabel();
    jLabel15 = new javax.swing.JLabel();
    jLabel16 = new javax.swing.JLabel();
    gantt = new javax.swing.JPanel();
    title = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    quantum = new javax.swing.JTextField();
    numofpro = new javax.swing.JTextField();
    from = new javax.swing.JTextField();
    to = new javax.swing.JTextField();
    numofprolabel = new javax.swing.JLabel();
    fromlabel = new javax.swing.JLabel();
    tolabel = new javax.swing.JLabel();
```

```
        calcSJF = new javax.swing.JButton();
        calcLJF = new javax.swing.JButton();
        calcLJFCBT = new javax.swing.JButton();
        calcRR = new javax.swing.JButton();
        calcPP = new javax.swing.JButton();
        calcNPP = new javax.swing.JButton();
        clearTable = new javax.swing.JButton();
        about = new javax.swing.JButton();
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Process Scheduling Algorithms Introducing Combined Burst Times");
        table.setAutoCreateRowSorter(true);
        table.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
            },
            new String [] {
                "Scheduling Algorithms", "Average Waiting Time", "Average Turnaround
Time", "Context Switch"
            }
        ) {
```

```
        Class[] types = new Class [] {
            java.lang.String.class, java.lang.Double.class, java.lang.Double.class,
java.lang.Integer.class
        };
        public Class getColumnClass(int columnIndex) {
            return types [columnIndex];
        }
    });
    table.setFillsViewportHeight(true);
table.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_INTERVAL_SELEC
TION);
    jScrollPane1.setViewportView(table);
    Generate.setMnemonic('G');
    Generate.setText("Generate");
    Generate.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            GenerateActionPerformed(evt);
        }
    });
    jLabel1.setText("<html><b><font color=\"red\">Note: lower value in Priority field
means higher priority.</font></b></html>");
    jLabel2.setDisplayedMnemonic('T');
    jLabel2.setLabelFor(quantum);
    jLabel2.setText("Time Quantum:");
    quantum.setText("0");
    numofprolabel.setDisplayedMnemonic('N');
    numofprolabel.setLabelFor(numofpro);
    numofprolabel.setText("Number of Processes:");
```

```
        numofpro.setText("0");
        fromlabel.setDisplayedMnemonic('E');
        fromlabel.setLabelFor(from);
        fromlabel.setText("Burst time Between:");
        from.setText("0");
        tolabel.setDisplayedMnemonic('w');
        tolabel.setLabelFor(to);
        tolabel.setText("to:");
        to.setText("0");
        clearTable.setMnemonic('C');
        clearTable.setText("Clear Table");
        clearTable.addActionListener(new java.awt.event.ActionListener() {
           public void actionPerformed(java.awt.event.ActionEvent evt) {
              clearTableActionPerformed(evt);
           }
        });
        about.setMnemonic('A');
        about.setText("About");
        about.addActionListener(new java.awt.event.ActionListener() {
           public void actionPerformed(java.awt.event.ActionEvent evt) {
              aboutActionPerformed(evt);
           }
        });
        //jButton1.setText("LJF");
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
```

```
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(188, 188, 188)
                        .addComponent(about, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addComponent(jScrollPane1)
                    .addGroup(javax.swing.GroupLayout.Alignment.CENTER,
layout.createSequentialGroup()
                        .addComponent(Generate,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE)
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
javax.swing.GroupLayout.PREFERRED_SIZE, 69,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        )
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
```

```
                .addComponent(jLabel2)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                .addComponent(quantum,
javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)

 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                .addComponent(numofprolabel)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                .addComponent(numofpro,
javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                .addComponent(fromlabel)

  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                .addComponent(from, javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                .addComponent(tolabel)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                .addComponent(to, javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(clearTable, javax.swing.GroupLayout.PREFERRED_SIZE, 97,
javax.swing.GroupLayout.PREFERRED_SIZE)

javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                        ))
```

```
            .addContainerGap())

        );

        layout.setVerticalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

                .addContainerGap()
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE
)

                    .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE, 32,
Short.MAX_VALUE) 32, Short.MAX_VALUE)

                    .addComponent(about))

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE
)

                    .addComponent(jLabel2)

                    .addComponent(quantum, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                    .addComponent(numofprolabel)

                    .addComponent(numofpro, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                    .addComponent(fromlabel)

                    .addComponent(from, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                    .addComponent(tolabel)

                    .addComponent(to, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addComponent(clearTable))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
140, javax.swing.GroupLayout.PREFERRED_SIZE)

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)


.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE
)

                )

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

            //.addComponent(fcfs, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

            .addContainerGap())

      );

      pack();

   }

   private void GenerateActionPerformed(java.awt.event.ActionEvent evt)

         {

                numberofp = getNumberOfProcesses();

          lower = getlower();

         upper = getUpper();

                statlabel.setText("<html><b><font color=\"red\">Statistics for:
"+numberofp+" Processes, Range of Burst times = "+lower+"-"+upper+", Quantum:
</font></b></html>");

                String str = "Statistics for: "+numberofp+" Processes, Range of Burst
times = "+lower+"-"+upper+"\n\nScheduling Algorithms\t\tAverage Waiting
Time\t\tAverage Turnaround Time\t\tContext Switch";

         //GEN-FIRST:event_GenerateActionPerformed
```

```
            burstT = new double[numberofp];
            generateBurstTimes(numberofp,lower,upper);
        ArrayList<Process> process = getProcesses();
        Scheduling algo = new Scheduling(process);
        PaintPanel p = new PaintPanel(algo);
        algo.fcfs();
        str+="\nFirst Come First
Serve\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t\t"+(int)((algo.avgTurnaro
undTime)*100)/100.0+"\t\t\t\t\t\t"+p.getContextSwitch(algo);
        displayResults(algo, 0);
        process = getProcesses();
        algo = new Scheduling(process);
        algo.sjf();
        str+="\nShortest Job
First\t\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t\t"+(int)((algo.avgTurnaro
undTime)*100)/100.0+"\t\t\t\t\t\t"+p.getContextSwitch(algo);
        displayResults(algo, 1);
        process = getProcesses();
        algo = new Scheduling(process);
        algo.ljf();
        str+="\nLongest Job
First\t\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t\t"+(int)((algo.avgTurnaro
undTime)*100)/100.0+"\t\t\t\t\t\t"+p.getContextSwitch(algo);
        displayResults(algo, 2);
        process = getProcesses2();
        algo = new Scheduling(process);
        algo.ljf();
        str+="\nLongest Job First +
CBT\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t\t"+(int)((algo.avgTurnarou
ndTime)*100)/100.0+"\t\t\t\t\t\t"+p.getContextSwitch(algo);
```

```
        displayResults(algo, 3);
        process = getProcesses();
        algo = new Scheduling(process);
        try {
           if (Integer.parseInt(quantum.getText()) > 0) {
              algo.rr(Integer.parseInt(quantum.getText()));
                    str+="\nRound
Robin\t\t\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t"+(int)((algo.avgTur
naroundTime)*100)/100.0+"\t\t\t\t\t"+p.getContextSwitch(algo);
              displayResults(algo, 4);
           } else {
            JOptionPane.showMessageDialog(this, "Time quantum must be greater than 0.",
            "Process Scheduling Algorithms", JOptionPane.WARNING_MESSAGE);
              quantum.requestFocusInWindow();
           }
        } catch (NumberFormatException e) {
           JOptionPane.showMessageDialog(this, "Invalid time quantum.",
                 "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
              quantum.requestFocusInWindow();
        }
        process = getProcesses();
        algo = new Scheduling(process);
        algo.preemptivePriority();
        str+="\nPreemptive
Priority\t\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t"+(int)((algo.avgTurn
aroundTime)*100)/100.0+"\t\t\t\t\t"+p.getContextSwitch(algo);
        displayResults(algo, 5);
             process = getProcesses();
```

```
        algo = new Scheduling(process);

        algo.nonpreemptivePriority();

        str+="\nNon-preemptive
Priority\t\t\t"+(int)((algo.avgWaitingTime)*100)/100.0+"\t\t\t\t\t\t"+(int)((algo.avgTurnar
oundTime)*100)/100.0+"\t\t\t\t\t\t"+p.getContextSwitch(algo);

        displayResults(algo, 6);

        str+="\n\n";

           try

        {

          out = new PrintWriter(new FileWriter("temp.txt", true));
           out.println(str);

        }

        catch (IOException e)

        {

           System.out.println("I/O Error");

           System.exit(0);

        }

    }

    private void clearTableActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_clearTableActionPerformed

        for (int i = 0; i < table.getModel().getRowCount(); i++) {

           for (int j = 0; j < table.getModel().getColumnCount(); j++) {

              table.setValueAt(null, i, j);

           }

        }

    }

    private void aboutActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_aboutActionPerformed

        About about = new About(this, true);
```

```
        Dimension screenDim = Toolkit.getDefaultToolkit().getScreenSize();
        about.setLocation(((screenDim.width - about.getSize().width) / 2),
                ((screenDim.height - about.getSize().height) / 2));
        about.setVisible(true);
    }//GEN-LAST:event_aboutActionPerformed

                    private int getNumberOfProcesses() {//GEN-
FIRST:event_calcRRActionPerformed
                     int numprocesses = 0;
                    try{
          if (Integer.parseInt(numofpro.getText()) > 0) {
                       numprocesses = Integer.parseInt(numofpro.getText());
          } else {
              JOptionPane.showMessageDialog(this, "Number of processes must be greater
than 0.",
                  "Process Scheduling Algorithms", JOptionPane.WARNING_MESSAGE);
             numofpro.requestFocusInWindow();
          }
        } catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(this, "Invalid processes number.",
                  "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
             numofpro.requestFocusInWindow();
        }
        return numprocesses;
    }
          private int getlower() {
                    int lower = 0;
                    try{
         if (Integer.parseInt(from.getText()) > 0) {
```

```
            lower = Integer.parseInt(from.getText());
          } else {
            JOptionPane.showMessageDialog(this, "lowererer limit of burstimes must be
greater than 0.",
                "Process Scheduling Algorithms", JOptionPane.WARNING_MESSAGE);
            from.requestFocusInWindow();
          }
        } catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(this, "Invalid lowererer limit.",
                "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
            from.requestFocusInWindow();
        }
        return lower;
          }
          private int getUpper(){
                int upper = 0;
                try{
        if (Integer.parseInt(to.getText()) > 0) {
           upper = Integer.parseInt(to.getText());
          } else {
            JOptionPane.showMessageDialog(this, "Upper limit of burstimes must be
greater than 0.",
                "Process Scheduling Algorithms", JOptionPane.WARNING_MESSAGE);
            to.requestFocusInWindow();
          }
        } catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(this, "Invalid upper limit.",
                "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
```

```
            to.requestFocusInWindow();
        }
        return upper;
    }//GEN-LAST:event_calcRRActionPerformed
     public void generateBurstTimes(int numofprocesses, int begining, int end){
            int counter = 0;
            int t;
            while(counter <  numofprocesses ){
                    t = (int)(begining+Math.random()*end);
                    if(t<=end){
                            burstT[counter] = t;
                            counter++;
                    }
                    }
        }
    private ArrayList<Process> getProcesses() {
        ArrayList<Process> process = new ArrayList<Process>(5);
             numberofp = getNumberOfProcesses();
        lower = getlower();
        upper = getUpper();
        String[] nam = new String[numberofp];
        double[] arrivalTim = new double[numberofp];
        double[] burstTim = new double[numberofp];
        double[] sortedBurstTim = new double[numberofp];
        int[] priorit = new int[numberofp];
        try {
            for (int i = 0; i < numberofp; i++) {
```

```
At(i, 0)).equals("")) {
                String name = "p" + i;
                 double arrivalTime = 0;
                double burstTime = burstT[i];
                 int priority = 0;
                 Process p = new Process(name, arrivalTime, burstTime, priority);
                 process.add(p);
              }
        } catch (NullPointerException e) {
           JOptionPane.showMessageDialog(this,
                "<html>Each process must have a burst time associated.<br>" +
                "Check if you've omitted any burst time. <br><br>" +
                "Also, ensure that no cell is in editing state before " +
                "pressing button for calculating any algorithm.<br>" +
                "To ensure it, just click on any cell other than the one in editing state.",
                "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
        }
        return process;
    }
    private ArrayList<Process> getProcesses2() {
        ArrayList<Process> process = new ArrayList<Process>(5);
        numberofp = getNumberOfProcesses();
         lower = getlower();
         upper = getUpper();
        String[] name = new String[numberofp];
        double[] arrivalTime = new double[numberofp];
        double[] burstTime = new double[numberofp];
```

```
double[] sortedBurstTime = new double[numberofp];
int[] priority = new int[numberofp];
double cwa = 0.0;
int numberofprocesses = 0;
try {
   for (int i = 0; i < numberofp; i++) {
                        numberofprocesses++;
          name[i] = "p"+i;
             arrivalTime[i] = 0;
          burstTime[i] = burstT[i];
           cwa += burstTime[i];
              priority[i] = 0;
   }
} catch (NullPointerException e) {
   JOptionPane.showMessageDialog(this,
          "<html>Each process must have a burst time associated.<br>" +
          "Check if you've omitted any burst time. <br><br>" +
          "Also, ensure that no cell is in editing state before " +
          "pressing button for calculating any algorithm.<br>" +
          "To ensure it, just click on any cell other than the one in editing state.",
          "Process Scheduling Algorithms", JOptionPane.ERROR_MESSAGE);
}
String nam = "" ;
double arrivalTim, burstTim;
double  tempBt = 0;
int priorit = 0;
String tempname = null;
```

```
cwa /= numberofprocesses;
java.util.Arrays.sort(burstTime, 0, numberofprocesses);
for (int i = 0, j = numberofprocesses-1; i < numberofprocesses; i++, j--){
  sortedBurstTime[i]=burstTime[j];
}
for (int i = 0; i < numberofprocesses; i++){
    arrivalTim = arrivalTime[i];
    priorit = priority[i];
   if (sortedBurstTime[i]<=cwa){
                        if ((tempBt == 0.0) && (tempname == null)){
                                tempBt = sortedBurstTime[i];
                                tempname = name[i];
                          if (i == numberofprocesses-1){
                         Process p = new Process(tempname, arrivalTim, tempBt, priorit);
                                process.add(p);
                          }
                        }
                        else{
                burstTim =  tempBt + sortedBurstTime[i];
                nam = tempname + name[i];
                Process p = new Process(nam, arrivalTim, burstTim, priorit);
                        process.add(p);
                tempBt = 0;
                tempname = null;
                }
    }
    else{
```

```
            burstTim = sortedBurstTime[i];
            nam = name[i];
            Process p = new Process(nam, arrivalTim, burstTim, priorit);
                process.add(p);
        }
    }
    return process;
  }
  private void displayResults(Scheduling algo, int algoNo) {
        PaintPanel p = new PaintPanel(algo);
    switch (algoNo) {
        case 0: //FCFS
            table.getModel().setValueAt("First Come First Serve",0, 0);
                table.getModel().setValueAt((int)((algo.avgWaitingTime)*100)/100.0,0,
1);
        table.getModel().setValueAt((int)((algo.avgTurnaroundTime)*100)/100.0,0, 2);
            table.getModel().setValueAt(p.getContextSwitch(algo),0, 3);
            break;
        case 1: //SJF
                table.getModel().setValueAt("Shortest Job First",1, 0);
table.getModel().setValueAt((int)((algo.avgWaitingTime)*100)/100.0,1, 1);
        table.getModel().setValueAt((int)((algo.avgTurnaroundTime)*100)/100.0,1, 2);
            table.getModel().setValueAt(p.getContextSwitch(algo),1, 3);
            break;
        case 2: //LJF
            table.getModel().setValueAt("Longest Job First",2, 0);
                table.getModel().setValueAt((int)((algo.avgWaitingTime)*100)/100.0,2,
1);
table.getModel().setValueAt((int)((algo.avgTurnaroundTime)*100)/100.0,2, 2);
```

```
                table.getModel().setValueAt(p.getContextSwitch(algo),2, 3);
            break;
        case 3:
            table.getModel().setValueAt("Longest Job First + CBT",3, 0);

        table.getModel().setValueAt((int)((algo.avgWaitingTime)*100)/100.0,3,1);
            table.getModel().setValueAt((int)((algo.avgTurnaroundTime)*100)/100.0,3,2);
        table.getModel().setValueAt(p.getContextSwitch(algo),3, 3);
            break;
        case 4:
            table.getModel().setValueAt("Round Robin",4, 0);

        table.getModel().setValueAt((int)((algo.avgWaitingTime)*100)/100.0,4,1);
            table.getModel().setValueAt((int)((algo.avgTurnaroundTime)*100)/100.0,4,2);
                table.getModel().setValueAt(p.getContextSwitch(algo),4, 3);
            break;
        case 5:
        case 6:
      }
   }
   public static void main(String args[])  {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {}
            Main main = new Main();
            main.table.getTableHeader().setReorderingAllowed(false);
```

```
            Dimension screenDim = Toolkit.getDefaultToolkit().getScreenSize();
            main.setLocation(((screenDim.width - main.getSize().width) / 2),
                    ((screenDim.height - main.getSize().height) / 2));
            main.setVisible(true);
        }
      });
   }

}
```

3. PaintPanel.java

```
package scheduling;

import javax.swing.*;
import java.awt.*;
public class PaintPanel extends JPanel {
   Scheduling algo;
   int ct;
   final int TIME_UNIT = 10;
   final boolean DISPLAY_GANTT_TIME_IN_INTEGER = true;
   public Main m;
   PaintPanel(Scheduling algo) {
      this.algo = algo;
      setPreferredSize(new Dimension(580,580));
   }
        public void init(){
        ct = 0;
                }
```

```
    protected void paintComponent(Graphics g) {
        doSomtin(g);
    }
                }
            }
            }
        ct--;
        System.out.println("tt"+ct);
    }
}
```

4. Process.java

```
package scheduling;
import java.util.ArrayList;
public class Process {
    String name;
    double arrivalTime;
    double burstTime;
    int priority;
    double remainingTime;
    ArrayList<BurstPosition> burstPos = new ArrayList<BurstPosition>();
    Process(String name, double arrivalTime, double burstTime, int priority) {
        this.name = name;
        this.arrivalTime = arrivalTime;
        this.burstTime = burstTime;
        this.priority = priority;
        this.remainingTime = burstTime;
    }
```

```
    void addBurst(double start, double end) {
        burstPos.add(new BurstPosition(start, end));
    }
    class BurstPosition {
        double start;
        double end;
        BurstPosition(double start, double end) {
            this.start = start;
            this.end = end;
        }
    }
}
```

5. Scheduling.java

```
package scheduling;
import java.util.ArrayList;
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() {
        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() {
        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();
        int index = waitingTime.indexOf('+');
        waitingTime = waitingTime.substring(0, index) + waitingTime.substring(index +2);
        return avgWaitTime;
    }
    private double getAvgTurnaroundTime() {
        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();
        int index = turnaroundTime.indexOf('+');
        turnaroundTime = turnaroundTime.substring(0, index) +
turnaroundTime.substring(index + 2);
        return avgTurnTime;
    }
```