

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
JNANASANGAMA, BELAGAVI - 590018



**Mini Project Report**  
**on**  
**ROUND-ROBIN SCHEDULING WITH OPENGL**

*Submitted in partial fulfillment for the award of degree of*

**Bachelor of Engineering**  
**in**  
**COMPUTER SCIENCE AND ENGINEERING**

Submitted by  
**DANIYAL PARVEEZ**  
1BG15CS026

Guide  
**Smt. Ranjana S. Chakrasali**  
Assistant Professor, Dept. of CSE  
BNMIT



*Vidyaya Amrutham Ashruthe*

***B.N.M. Institute of Technology***

(Approved by AICTE, Affiliated to VTU, ISO 9001:2008 certified  
and Accredited as grade A Institution by NAAC)

Post box no. 7087, 27<sup>th</sup> cross, 12<sup>th</sup> Main,  
Banashankari 2<sup>nd</sup> Stage, Bengaluru- 560070, INDIA  
Ph: 91-80- 26711780/81/82 Email: [principal@bnmit.in](mailto:principal@bnmit.in) www.bnmit.org

**Department of Computer Science and Engineering**

2017 – 2018

# *B.N.M. Institute of Technology*

Approved by AICTE, Affiliated to VTU, ISO 9001:2008 certified  
and Accredited as grade A Institution by NAAC)

Post box no. 7087, 27<sup>th</sup> cross, 12<sup>th</sup> Main,  
Banashankari 2<sup>nd</sup> Stage, Bengaluru- 560070, INDIA  
Ph: 91-80- 26711780/81/82 Email: [principal@bnmit.in](mailto:principal@bnmit.in) www.bnmit.org

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



*Vidyaya Amrutham Ashnutho*

## **CERTIFICATE**

Certified that the Mini Project entitled **Round-Robin Scheduling with OpenGL** carried out by Mr. **Daniyal Parveez**, USN **1BG15CS026**, a bonafide student of VI Semester B.E., **B.N.M Institute of Technology** in partial fulfillment for the Bachelor of Engineering in **COMPUTER SCIENCE AND ENGINEERING** of the **Visvesvaraya Technological University**, Belagavi during the year 2017-18. It is certified that all corrections / suggestions indicated for internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of technical seminar prescribed for the said degree.

**Smt. Ranjana S. Chakrasali**  
Assistant Professor  
Department of CSE  
BNMIT, Bengaluru

**Dr. Sahana D. Gowda**  
Professor and HOD  
Department of CSE  
BNMIT, Bengaluru

**Name & Signature:-**

**Examiner 1:**

**Examiner 2:**

# ACKNOWLEDGEMENT

I put a tremendous amount of effort into this project over the course of the past few months, and seeing the results brings a huge amount of joy and satisfaction. Credit should be given where it is due, and I'd like to thank all those who were responsible for helping me realize this project.

I would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMIT, Bengaluru for providing an excellent academic environment in the college.

I would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his support and encouragement during the course of the work.

I would like to express my gratitude to **Dr. M S Suresh**, Dean, BNMIT, Bengaluru for his relentless support, guidance and assistance.

I extend my sincere gratitude to **Dr. Krishnamurthy G.N.**, Principal, BNMIT, for providing us with the facilities required for the project.

I express my heartfelt gratitude to **Dr.Sahana D. Gowda**, Professor and H.O.D, Department of Computer Science and Engineering, BNMIT, Bengaluru , for her valuable suggestions and support.

I extend my heartfelt thanks to **Smt. Ranjana S. Chakrasali**, Asst.Prof., Department of Computer Science and Engineering, BNMIT, Bengaluru , for her guidance and support.

Finally, I would like to thank all the faculty members of Department of Computer Science and Engineering, BNMIT, Bengaluru.

I would like to thank my family members and friends for their unfailing moral support and encouragement.

# ABSTRACT

A computer, being an electronic machine, is made up of several pieces of integrated circuits, transistors, other chips and lots of wiring. Some of these pieces constitute what is known as the ‘hardware’. The CPU (Central Processing Unit) is the most important piece of hardware in a computer and is responsible for handling the execution of machine instructions that run on a system in the form of various processes. Being an important resource, it is essential that the CPU is managed and shared efficiently and fairly as a resource.

A mechanism known as CPU scheduling ensures efficient, fast and fair management of the CPU in a system. CPU scheduling allows one process to use the CPU while another process is on hold (in the waiting state), thus maximizing CPU utilization.

A type of CPU scheduling known as preemptive scheduling allows a running process to be preempted (stopped temporarily), to start a new one. Such an event of preemption can occur in case:

- A higher priority process arrives in the ready queue.
- An interrupt is raised.

This project aims to simulate how a popular pre-emptive scheduling algorithm called Round Robin works. Here, each process is executed for a fixed time slice called a quantum. If the process finishes execution, it terminates. Otherwise, it is preempted so that the next process in line can be executed. The set of processes is thus executed in a circular fashion till all processes have run to completion.

This simulation is achieved using a graphics API known as OpenGL. It allows portable and interactive 2-D and 3-D applications to be developed.

# TABLE OF CONTENTS

<b>CONTENTS</b>	<b>PAGE</b>
<b>1. INTRODUCTION</b>	<b>1-4</b>
1.1 Overview	1
1.2 Problem statement	1
1.3 Motivation	2
1.4 Computer Graphics	2
1.5 OpenGL	2
1.6 Applications of Computer Graphics	4
<b>2. SYSTEM REQUIREMENTS</b>	<b>5</b>
2.1 Hardware and Software Requirements	5
<b>3. SYSTEM DESIGN</b>	<b>6-7</b>
3.1 Proposed System	6
3.2 Flowchart	6
<b>4. IMPLEMENTATION</b>	<b>8-12</b>
4.1 Module Description	8
4.2 High-level Code (Main functions)	8
<b>5. RESULTS</b>	<b>14-16</b>
<b>6. CONCLUSION AND FUTURE ENHANCEMENT</b>	<b>17</b>
<b>BIBLIOGRAPHY</b>	<b>18</b>

# LIST OF FIGURES

<b>FIGURE NO.</b>	<b>FIGURE CAPTION</b>	<b>PAGE NO.</b>
1.1	OpenGL Pipeline	3
3.1	Flowchart	7
5.1	Opening Command-line screen	18
5.2	Intro screen	18
5.3	Main Screen	19
5.4	Animation Loop	19
5.5	Animation Loop	20

# **CHAPTER 1**

## **INTRODUCTION**

# CHAPTER 1

## INTRODUCTION

### 1.1 Overview

This project aims to simulate how Round Robin works on a limited set of processes. The number of processes, as well as their attributes – burst time and arrival time, are inputted by the user. The OpenGL Utility Toolkit (GLUT) is used to implement the graphical application. The various features provided by GLUT are used to handle I/O, display primitives and text, and animate some text across the screen, resulting in the creation of an interactive and educational graphical application.

### 1.2 Problem Statement

Round Robin is a fairly popular pre-emptive CPU scheduling algorithm that's is used in many multiprogramming operating systems. Given a fixed time slice called a quantum, the algorithm executes all processes in the ready queue in a circular fashion for the duration of the quantum, before moving on to the next process. This continues until all processes in the ready queue have been executed to completion.

Simulation of this algorithm on a limited set of processes will give viewers a rough idea of how this algorithm works in real-time. Furthermore, this application serves as a demonstration of what exactly can be achieve using the OpenGL API.

### 1.3 Motivation

OpenGL is a low-level API that has no notion of objects. Everything is specified in terms of coordinates.

As such, simulating something as abstract as an algorithm working on a set of processes is not only challenging, but gives room for freedom and creativity. This desire for adventure in experimentation lead me to take up this project.



## **1.4 Computer graphics**

Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began 50 years ago, with the display of a few lines on a cathode-ray tube (CRT). Classical graphics techniques arose as a medium to convey information among people.

The field of information visualization is becoming increasingly more important as we have to deal with understanding complex phenomena from problems in bioinformatics to detecting security threats. Medical imaging, flight simulation, computer aided design (CAD) and VLSI design are some areas that pose complex problems relating to visualization and data-analysis. <sup>[1]</sup>

It is seen as a sub-field of Computer Science and therefore deals with mathematical and computation issues.

Like any other science, it has revolutionized the way the world works and runs.

## **1.5 OpenGL**

OpenGL (for “Open Graphics Library”) is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically colour images of three-dimensional objects.

To the programmer, OpenGL is a set of commands that allow the specification of geometric objects in two or three dimensions, together with commands that control how these objects are rendered into the framebuffer.

To the implementor, OpenGL is a set of commands that affect the operation of graphics hardware.

We view OpenGL as a pipeline having some programmable stages and some state-driven stages that control a set of specific drawing operations. <sup>[2]</sup>

Silicon Graphics Inc., (SGI) started developing OpenGL in 1991 and released it in January 1992. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

The following image gives a high-level overview of the OpenGL rendering pipeline: -

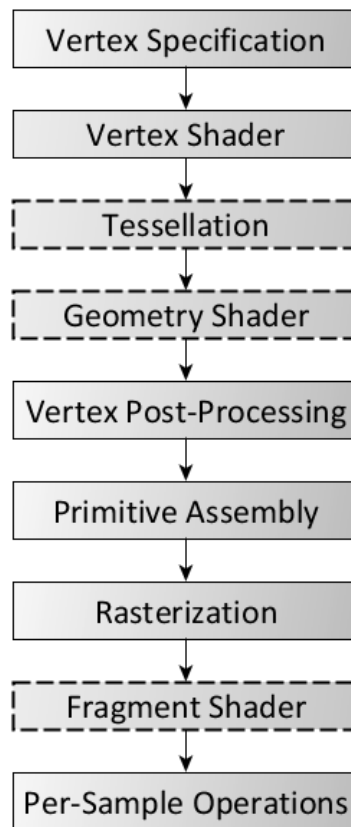


Figure 1.1 OpenGL Pipeline

The following steps are involved in the pipeline: -

- The process of vertex specification is where the application sets up an ordered list of vertices to send to the pipeline. These vertices define the boundaries of a primitive. Vertex shaders perform basic processing of each individual vertex. Vertex shaders receive the attribute inputs from the vertex rendering and converts each incoming vertex into a single outgoing vertex.
- Primitives can be tessellated using two shader stages and a fixed-function tessellator between them.
- Geometry shaders are user-defined programs that process each incoming primitive, returning zero or more output primitives.
- After the shader-based vertex processing, vertices undergo a number of fixed-function processing steps, known as vertex post-processing.
- Primitive assembly is the process of collecting a run of vertex data output from the prior stages and composing it into a sequence of primitives.

- Primitives are then rasterized to create fragments.
- The data for each fragment from the rasterization stage is processed by a fragment shader.
- The fragment data output from the fragment processor is then passed through a sequence of steps before being written to the frame-buffer.

## **1.6 Applications of Computer Graphics**

As a theoretical and practical science, computer graphics has found application in a variety of fields and has transformed our world as we know it. Broadly speaking, computer graphics finds its use in 4 major areas – Simulation, UI (User Interfaces), Design and Information Display.

These areas have seen the emergence and evolution of several different types of applications that include: -

- Virtual Reality (VR)
- Computer-aided Design (CAD)
- Computer Modelling and Simulation
- Information Visualization
- Digital Art
- Scientific Visualization

## **CHAPTER 2**

### **System Requirements**

## CHAPTER 2

# SYSTEM REQUIREMENTS

## 2.1 Hardware & Software Requirements

### Hardware Requirements

Requirement	Minimum	Recommended
Memory	1 GB	4 GB
Free Disk Space	5 GB	10 GB or more
Processor Speed	1.5 GHz	2.0 GHz and above
GPU	OpenGL 2.0 compliant GPU with 128 MB VRAM	OpenGL 2.0 compliant GPU with 256 MB VRAM and Hardware Acceleration

### Software Requirements

- Windows 7 (32-bit/64-bit) or above Operating System
- Microsoft Visual Studio 2008 and above
- Microsoft Visual C++ Redistributable

# **CHAPTER 3**

## **System Design**

## CHAPTER 3

### SYSTEM DESIGN

#### 3.1 Proposed System

The proposed application is a graphical interactive application. It should be able to handle colourful three-dimensional and two-dimensional visuals and should be able to respond to user input. Specifically speaking, the proposed **functional requirements** allow the system in concern be able to: -

- Accept input regarding the number of processes to be represented in the simulation.
- Use data computed from the round robin algorithm to setup the animation – the number of quanta each process requires is used to determine the number of times each process passes through the CPU during animation.
- Accept burst time and arrival time for each of the processes.
- Execute the round robin algorithm on the set of processes to compute how many quanta each process requires to proceed to completion.

The proposed **non-functional** requirements allow the system to: -

- Respond effectively to user input. For instance, pressing the ‘+’ key should increase the animation speed immediately.
- Ensure that user input does not break any kind of logic in the program before actually processing it. For instance, pressing the ‘-’ key should decrease the animation speed. However, the value of the ‘animationSpeed’ variable must be checked, before executing the snippet of code that decrements that variable.
- To handle simulation of a decent number of processes (up to about 25) without any apparent issues.

#### 3.2 Flowchart

The following flowchart gives a high-level overview of the flow of information of a system. It should give brief idea of how the system works, and how information is passed between different stages of the system: -

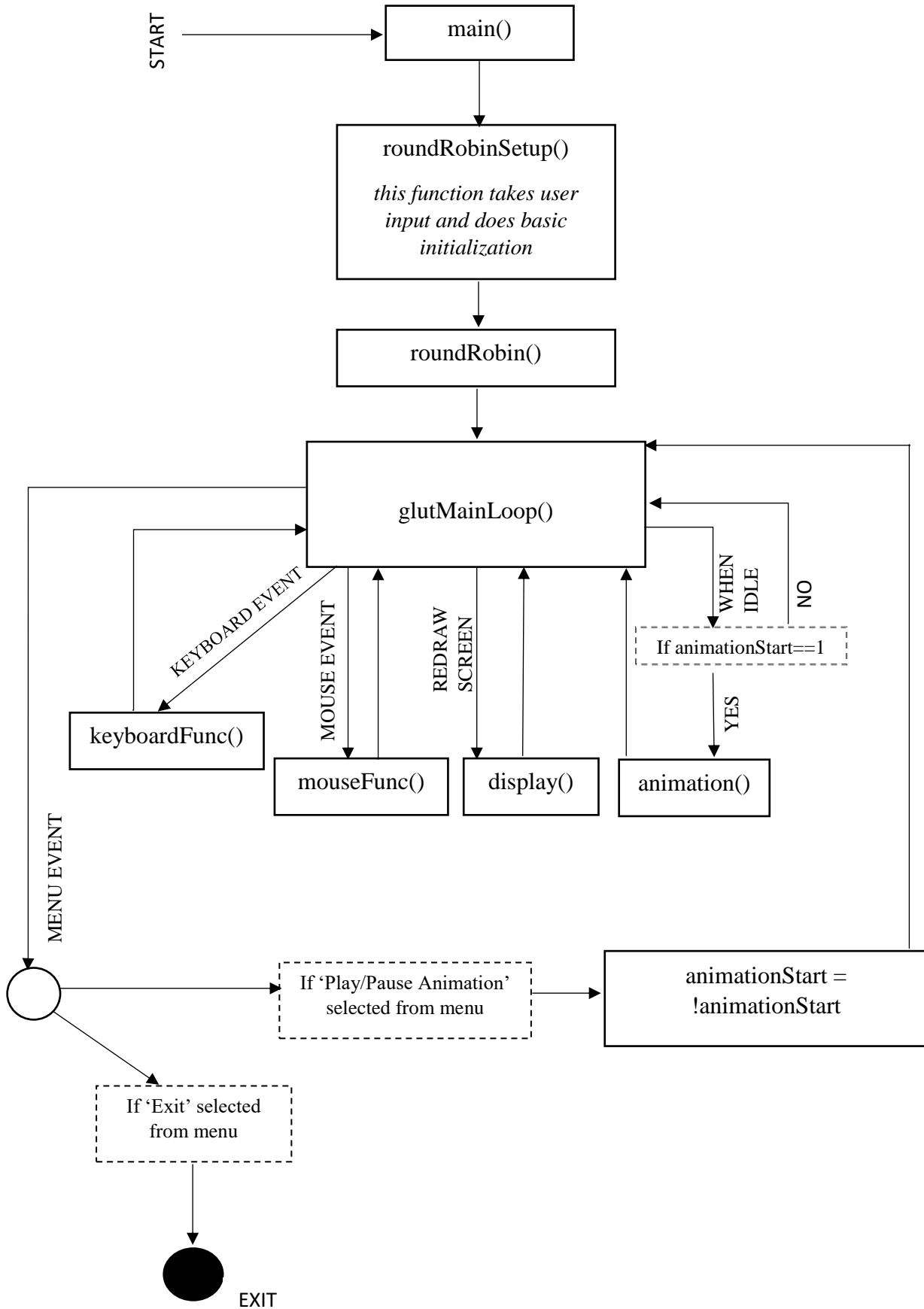


Figure 3.1 Flowchart



# **CHAPTER 4**

## **IMPLEMENTATION**

## CHAPTER 4

# IMPLEMENTATION

### 4.1 Module Description

The proposed system is divided into several modules in order to adhere to best programming practices and to keep the code maintainable. These modules are implemented via functions, each of which has a distinct functionality.

The major modules that constitute the system include: -

- **Main:** Registers all callbacks and puts the program into an infinite event processing loop.
- **Display:** This module is one of the most important modules in the program. It is responsible for drawing text and primitives to screen.
- **Mouse Function:** Handles mouse events.
- **Keyboard Function:** Handles keyboard events.
- **Menu Function:** Handles menu events.
- **Animation:** Manipulates necessary variables to animate required objects.
- **Round Robin:** Calculates the quanta required by each process to run to termination. This data is used during animation.

### 4.2 High-Level Code

The program is composed of several functions and variables that work together to create a working simulation. While it is not possible to cover each of the intricacies that constitute the code in detail, the most important functions (and their associated code) that are enough to gain a brief understanding of the program are listed below.

main function:-

```
int main(int argc, char **argv)

{

roundrobinSetup();

getch();
```

```
glutInit(&argc, argv);

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH |
GLUT_MULTISAMPLE);

glutInitWindowSize(800, 600);

glutInitWindowPosition(0, 0);

glutCreateWindow("Preemptive Scheduling with Round Robin");

glutFullScreen();init_mygl();

glEnable(GL_DEPTH_TEST);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyHandler);

glutSpecialFunc(specialKeyHandler);

glutMouseFunc(mouseMotionHandler);

glutIdleFunc(animate);

glutMainLoop();

getch();

return 0;

}
```

display function: -

```
void display()

{

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

if (finishedIntro == 0)

{
```

```
draw_intro();

}

else

{

    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    draw_boxes();

    draw_lines();

    draw_screen2_texts();

    if (animationStarted)

        draw_proc_labels();

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);}

    glutSwapBuffers();

}
```

#### Round Robin function: -

```
void roundrobin()

{

    int *temp, *wt, tat = 0; //burst, wait and turnaoround times

    temp = (int*)malloc(n_proc * sizeof(int));

    wt = (int*)malloc(n_proc * sizeof(int));

    int sum_wt = 0, sum_tat = 0;

    int n = n_proc;

    int i = 0;

    int flag = 0; //set to 1 when a process has completed
```

```
for (; i < n_proc; i++)

{

temp[i] = rq[i].proc.bt;

wt[i] = 0;

}

printf("\nProcess ID\tArrival Time\tBurst Time\tTurnaround Time\tWaiting
Time\n");

i = 0;

while (n)

{

if (temp[i] <= quantum && temp[i] > 0)

{

flag = 1;

tat += temp[i];

if (tat - proc[i].at - proc[i].bt < 0 && flag == 1)

{

tat += temp[i];

sum_tat += temp[i];

}

temp[i] = 0;

rq[i].iter++;

temp_rq[i].iter++;

}
```

```
else if (temp[i] > 0)

{

tat += quantum;

temp[i] -= quantum;

rq[i].iter++;

temp_rq[i].iter++;

}

if (temp[i] == 0 && flag == 1)

{

n--;

sum_wt += tat - proc[i].at - proc[i].bt;

sum_tat += tat - proc[i].at;

flag = 0;

}

i++;

if (i > n_proc - 1)

i = 0;

}

avg_wt = sum_wt * 1.0 / n_proc;

avg_tat = sum_tat * 1.0 / n_proc;

printf("\n\nAverage Waiting Time:\t%f", avg_wt);printf("\nAvg Turnaround\nTime:\t%f\n", avg_tat);

printf("\n");

}
```

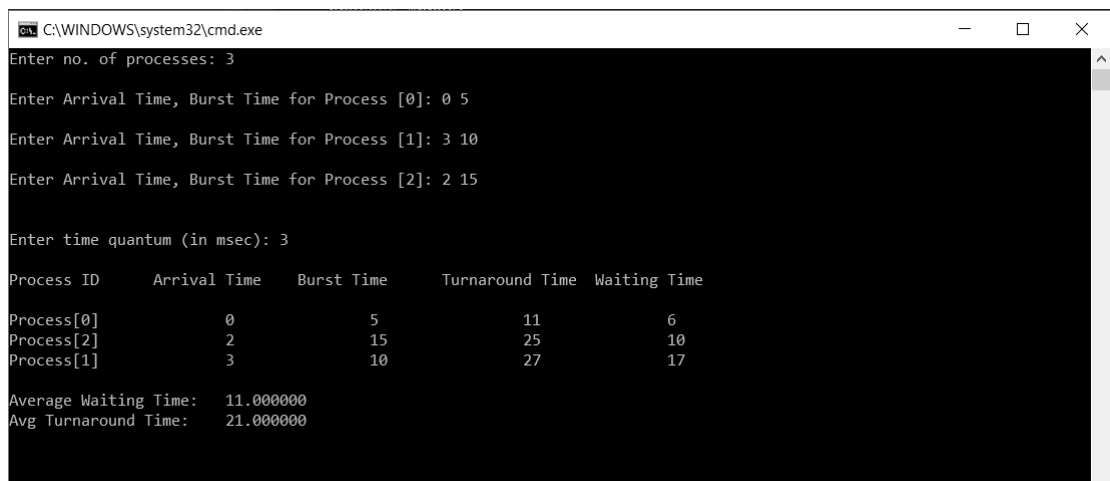
# **CHAPTER 5**

## **RESULTS**

## CHAPTER 5

### RESULTS

The application implemented seems to meet the proposed requirements. It response effectively and immediately to user input. The animation works as expected. The primitives and text look as intended and are displayed in the correct locations.



```
C:\WINDOWS\system32\cmd.exe
Enter no. of processes: 3
Enter Arrival Time, Burst Time for Process [0]: 0 5
Enter Arrival Time, Burst Time for Process [1]: 3 10
Enter Arrival Time, Burst Time for Process [2]: 2 15
Enter time quantum (in msec): 3
Process ID      Arrival Time    Burst Time    Turnaround Time    Waiting Time
Process[0]      0              5             11                6
Process[2]      2              15            25                10
Process[1]      3              10            27                17
Average Waiting Time: 11.000000
Avg Turnaround Time: 21.000000
```

Figure 5.1 Opening command-line screen

When the program is run, a command-line screen is first displayed. This takes in input from the user about the number of processes to be simulated, as well as input about their respective arrival times and burst times.

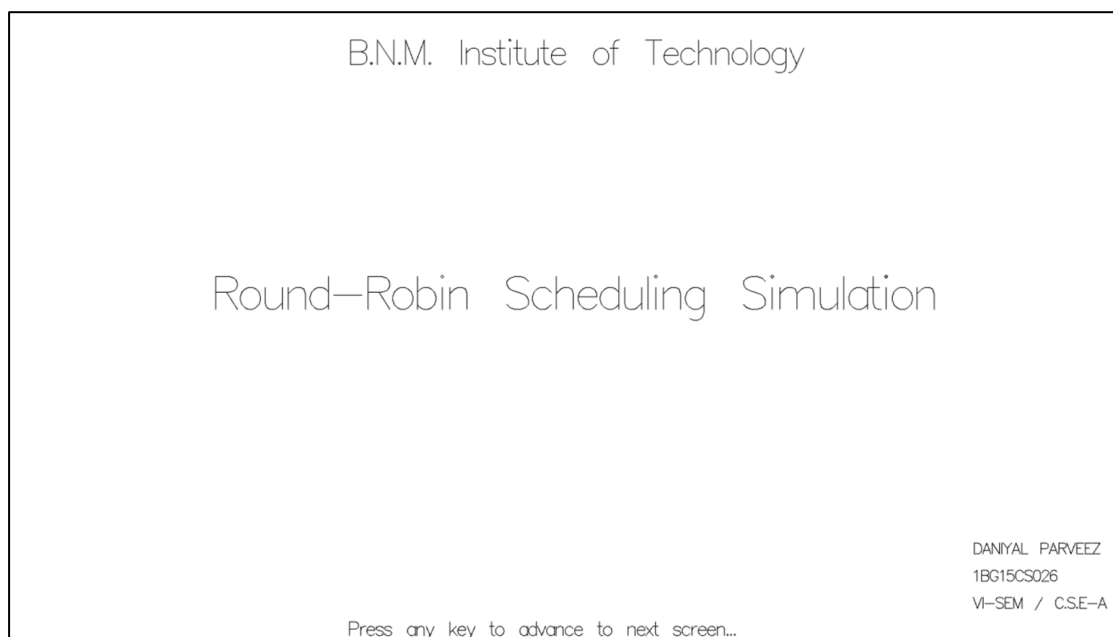


Figure 5.2 Intro Screen



The next screen displayed is the intro screen. It displays some basic information – name of the project, name of the institute, and student details. On accepting a key from the user, it advances to the next screen.

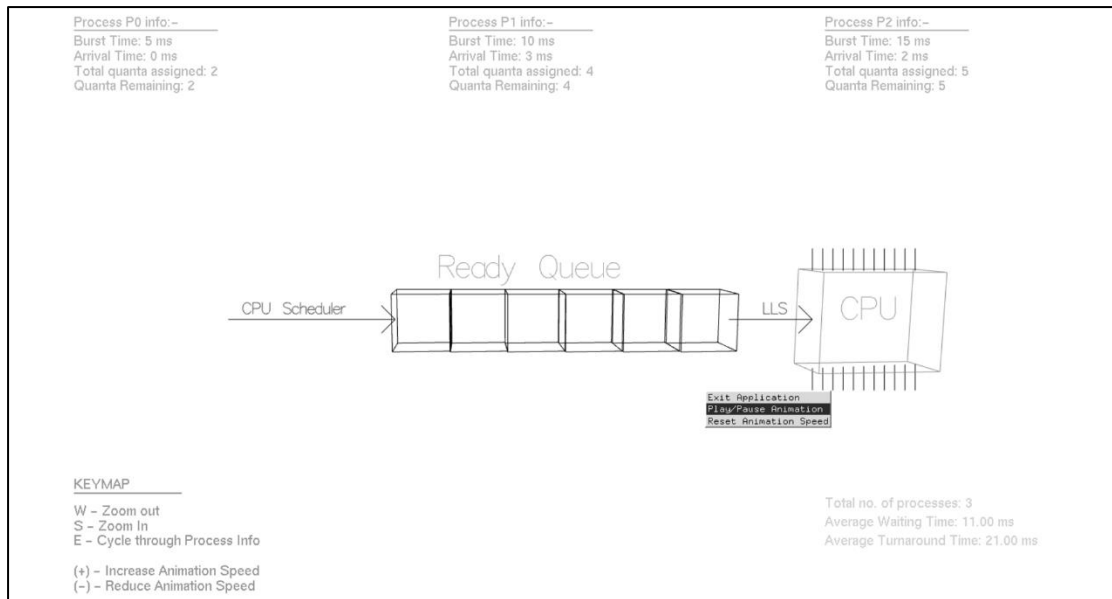


Figure 5.3 Main Screen

The main screen displays all primitives and text that are a part of the actual simulation. Upon selecting the ‘Play / Pause Animation’ from the menu, the animation begins.

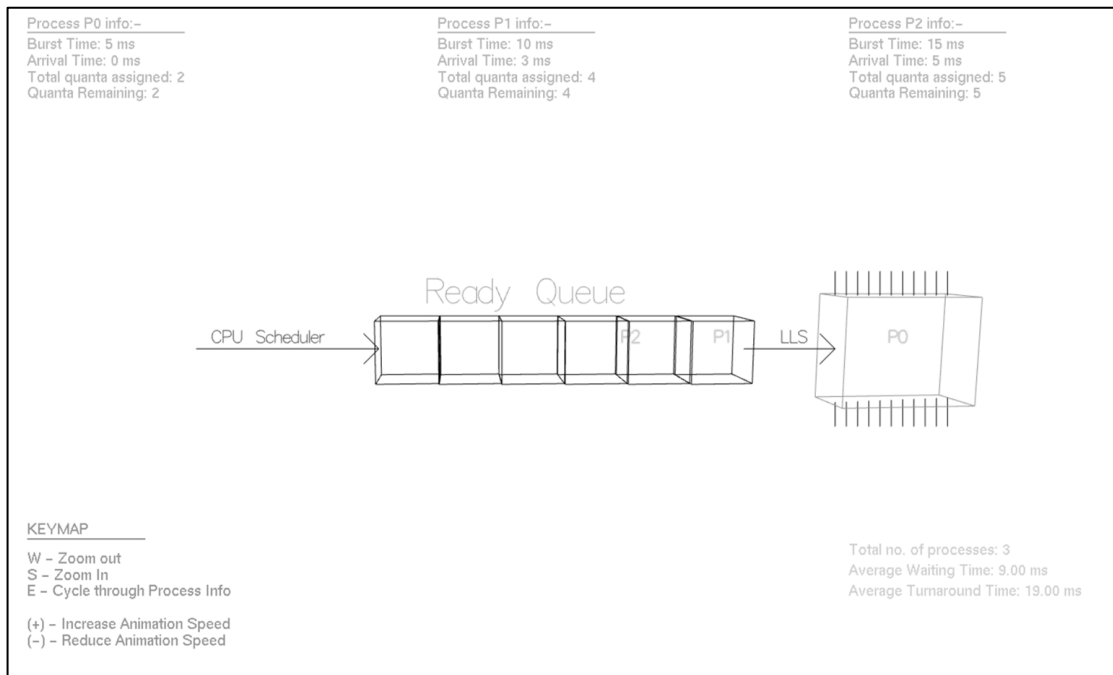


Figure 5.4 Animation Loop

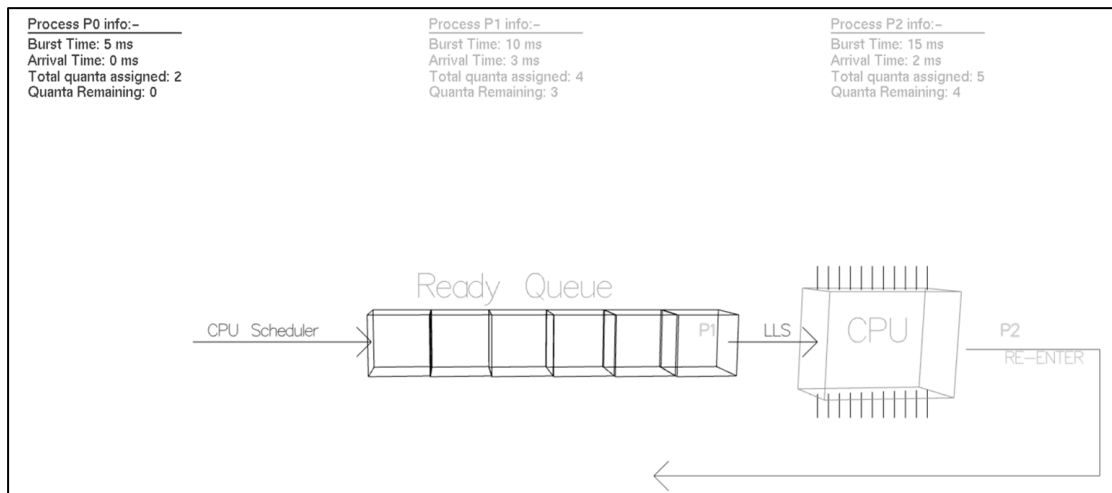


Figure 5.5 Animation Loop

One by one, the processes move through the ready queue, and into the CPU when it becomes free. The duration of time a process spends with the CPU is proportional to the burst time. When the process has executed for the duration of its burst, it leaves the CPU. If the process has run to completion, it simply exits. If not, it means that the process has more time slices assigned to it. Consequently, the process re-enters the ready queue at its tail.

The animation proceeds in this way until all processes have been executed to completion. Selecting the “Play / Pause’ Animation again from the menu will reinitialize all required variables and restart the animation from the beginning.

**CHAPTER 6**  
**CONCLUSIONS AND FUTURE**  
**ENHANCEMENT**

## CHAPTER 6

### CONCLUSIONS AND FUTURE ENHACEMENT

The project demonstration is but an abstract simulation of how a pre-emptive scheduling algorithm, namely round robin works on a limited set of processes, taking their burst times and arrival times into consideration. Nonetheless, it does a good job of clarifying all concepts with regards to round robin scheduling. Aided with some background knowledge, anyone viewing the simulation will be able to get a concrete idea about how round robin plays a vital role as a CPU scheduling algorithm in most of today's operating systems.

While the size of the project and the features available are enough to meet the course objective and project guidelines, given extra time, there is scope for improvement and room for incorporating new ideas and features. Some possible future enhancements include: -

- Allow users to add a process while the animation is running. The reason this was not included in the original project was that, if the user entered undesirable values for arrival time and burst time for the new process, it would mess up the entire animation. Nonetheless, a solution could be worked out.
- Display a Gantt chart at the end of the animation. A Gantt chart gives a good overview of how long and in what order the processes have run.
- Add music to the animation. For instance, when a process enters the CPU, a beep could be triggered. This would make the application more interactive.
- CPU scheduling is a lot more complex than the animation shows. In addition to a ready queue, there is a waiting queue where processes waiting for I/O or some other resource reside. Only upon moving to ready state can they enter the ready queue. After finishing its burst execution, a process can move back to the waiting state. This part of the CPU scheduling could be shown in the animation.

## **BIBLIOGRAPHY**

- [1] Edward Angel, Dave Shreiner. Interactive Computer Graphics: A Top-Down Approach. Pearson Publications. 6<sup>th</sup> edition; 2011
- [2] Mark Segal, Kurt Akeley. The OpenGL R Graphics System: A Specification; The Khronos Group. Pages 1-2. - March 11, 2010
- [3] The Khronos Group. Rendering Pipeline Overview. Retrieved 10<sup>th</sup> May, 2018.  
[https://www.khronos.org/opengl/wiki/Rendering\\_Pipeline\\_Overview](https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview)