VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI - 590 018, KARNATAKA.

MINI PROJECT REPORT

ON

"ILLUSTRATION OF SEMAPHORES"

Submitted in the partial fulfillment of requirements

FOR

Computer Graphics and Visualization Lab (18CSL67)

Submitted by

AMRUTH R HEBBAR	**4BD20CS012**
JEEVAN H K	**4BD20CS040**

PROJECT GUIDES:

Prof. Maduri Deekshith S B.E, M. Tech.,
Asst. Professor,
Department of CS&E
B.I.E.T., Davanagere

Prof. B.E,M. Tech.,
Asst. Professor
Department of CS&E,
B.I.E.T., Davanagere

2022-2023

Department of Computer Science and Engineering.

Bapuji Institute of Engineering & Technology

Davangere- 577004

Bapuji Institute of Engineering and Technology

Davangere -577004

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that **AMRUTH R HEBBAR** and **JEEVAN H K** bearing **4BD20CS012** and **4BD20CS040** respectively of **Computer Science and Engineering** department have satisfactorily submitted the mini project report entitled **"ILLUSTRATION OF SEMAPHORES"**. The report of the project has been approved as it satisfies the academic requirements in respect of project work prescribed for the academic year 2022-2023.

Project Guide1

Prof. Madhuri Deekshith S B.E., M. Tech.,
Asst. Professor
Department of CS&E
B.I.E.T., Davangere.

Project Guide 2

Prof.Gangamma H B.E.,M.Tech.,
Asst. Professor
Department of CS&E
B.I.E.T., Davangere.

Head of Department

Dr. Nirmala C.R M. Tech., Ph.D.,
Prof & Head, Department of CS&E
B.I.E.T., Davangere

Date:

Place:

Signature of Examiners:

(1)

(2)

ACKNOWLEDGEMENT

Salutations to our beloved and highly esteemed institute, **"BAPUJI INSTITUTE OF ENGINEERING AND TECHNOLOGY"** for having well qualified staff and labs furnished with necessary equipments.

We express our sincere thanks to our guide **Prof. Madhuri Deekshith S & Prof.Gangamma H** for giving us constant encouragement, support and valuable guidance throughout the course of the project without whose stable guidance this project would not have been achieved.

We express whole hearted gratitude to **Dr. Nirmala C R** who is our respectable HOD of Computer Science & Engineering Department. We wish to acknowledge her help who made our task easy by providing with his valuable help and encouragement.

We also express our whole hearted gratitude to our principal, **Dr. Aravind H B,** for his moral support and encouragement.

We would like to extend my gratitude to all staff of **Department of Computer Science and Engineering** for the help and support rendered to me. I have benefited a lot from the feedback, suggestions given by them.

We would like to extend my gratitude to all my family members and friends especially for their advice and moral support.

AMRUTH R HEBBAR(BD20CS012)
JEEVAN H K(4BD20CS040)

Bapuji Educational Association (Regd.)
Bapuji Institute of Engineering and Technology, Davangere-577004

Vision and Mission of the Institute

Vision

"To be a centre of excellence recognized nationally internationally, in distinctive areas of engineering education and research, based on a culture of innovation and invention."

Mission

"BIET contributes to the growth and development of its students by imparting a broad based engineering education and empowering them to be successful in their chosen field by inculcating in them positive approach, leadership qualities and ethical values."

Vision and Mission of the Computer Science and Engineering Department

Vision

"To be a centre-of-excellence by imbibing state-of-the-art technology in the field of Computer Science and Engineering, thereby enabling students to excel professionally and be ethical."

Mission

1.	Adapting best teaching and learning techniques that cultivates Questioning and Reasoning culture among the students.
2.	Creating collaborative learning environment that ignites the critical thinking in students and leading to the innovation.
3.	Establishing Industry Institute relationship to bridge skill gap and make them industry ready and relevant.
4.	Mentoring students to be socially responsible by inculcating ethical and moral values.

Program Educational Objectives (PEOs):

PEO1	To apply skills acquired in the discipline of computer science and engineering for solving Societal and industrial problems with apt technology intervention.
PEO2	To continue their carrier ion industry /academia or pursue higher studies and research.
PEO3	To become successful entrepreneurs, innovators to design and develop software products and services that meets societal, technical and business challenges.
PEO4	To work in the diversified environment by acquiring leadership qualities with effective communication skills accompanied by professional and ethical values.

Program Specific Outcomes (PSOs):

PSO1	Analyse and develop solutions for problems that are complex in nature but applying the knowledge acquired from the core subjects of this program.
PSO2	To develop secure, scalable, resilient and distributed applications for industry and societal Requirements.
PSO3	To learn and apply the concepts and contract of emerging technologies like artificial intelligence, machine learning, deep learning, big-data analytics, IOT, cloud computing etc for any real time problems.

Course Outcome

CO1	Implement the concepts of computer graphics primitives like lines, polygons triangles, cubes and 3D gasket.
CO2	Applying the different lighting and shading properties and illumination models
CO3	Animate real world problems using OpenGL API's.
CO4	Design and implement computer graphics applications (mini project) using OpenGL.

.

CONTENTS

LIST OF FIGURES

ABSTRACT

The aim of this mini project is to simulate mutual exclusion using **SEMAPHORES** by exploiting graphical processing capabilities. This mini project demonstrates the working of semaphores.The operating system consists of all the resources that a process needs.Whenever more than one process request for the resources, the OS will allocate resource to the process one at a time based on first come first serve basis.The other processes requesting for the resources are kept in the waiting queue.Whenever the first process releases the resource,the other process waiting in the queue are allocated resources by the OS.With the curiosity about synchronization, we have made our mind to show how orderly execution of co-operating processes can be achieved using **SEMAPHORE** to avoid data inconsistency, with the knowledge of computer graphics.

CHAPTER 1

1. INTRODUCTION

Semaphores are a vital concept in computer graphics that facilitate efficient and synchronized communication between different processes or threads. In a computer graphics project, semaphores play a crucial role in managing access to shared resources such as the GPU, memory, or rendering buffers. By utilizing semaphores, developers can effectively control the order of execution, prevent data races, and synchronize the rendering pipeline. This project aims to explore the implementation and utilization of semaphores in computer graphics, demonstrating their significance in achieving smooth and coordinated visual output.

1.1 OPENGL

OpenGL is the abbreviation for Open Graphics Library. It is a software interface for graphics hardware. This interface consists of several hundred functions that allow you, a graphics programmer, to specify the objects and operations needed to produce high-quality color images of two-dimensional and three-dimensional objects. Many of these functions are actually simple variations of each other, so in reality there are about 120 substantially different functions.The main purpose of OpenGL is to render two-dimensional and three-dimensional objects into the frame buffer. These objects are defined as sequences of vertices (that define geometric objects) or pixels (that define images).OpenGL performs several processes on this data to convert it to pixels to form the final desired image in the frame buffer.

1.2 HISTORY

As a result, SGI released the **OpenGL** standard In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort.

By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their IRIS GL API was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered.

SGI considered that the Iris GL API itself wasn't suitable for opening due to licensing and patent issues. Also, the Iris GL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NEWS systems were developed.

In addition, SGI had a large number of software customers; by changing to the OpenGL API they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary Iris Inventor and Iris Performer programming APIs.

1.3 FEATURES OF OPENGL

- **Industry standard**

  An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

- **Stable**

  OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable**

  All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

- **Evolving**

  Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

- **Scalable**

  OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

- **Easy to use**

  OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

- **Well-documented**

  Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

1.4 BASIC OPENGL OPERATION

The following diagram illustrates how OpenGL processes data. As shown, commands enter from the left and proceed through a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during various processing stages.

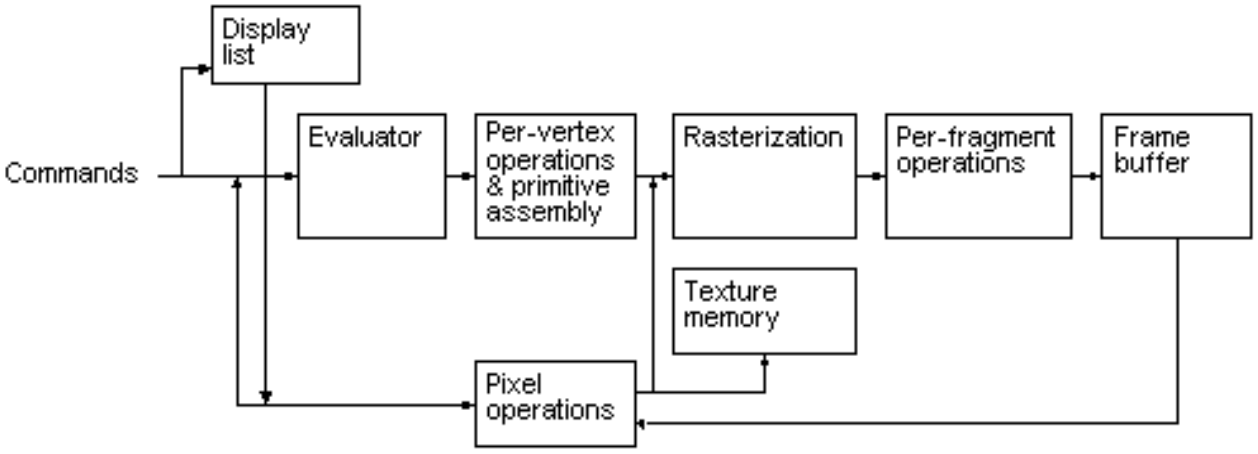

Fig:1.4 OpenGL Block Diagram

The processing stages in basic OpenGL operation are as follows:

- **Display list**

  Rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.

- **Evaluator**

  The evaluator stage of processing provides an efficient way to approximate curve and surface geometry by evaluating polynomial commands of input values.

- **Per-vertex operations and primitive assembly**

  OpenGL processes geometric primitives - points, line segments, and polygons all of which are described by vertices. Vertices are transformed, and primitives are clipped to the viewport in preparation for rasterization.

- **Rasterization**

  The rasterization stage produces a series of frame-buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each so produced is fed into the last stage, per-fragment operations.

- **Per-fragment operations**

  These are the final operations performed on the data before it is stored as pixels in the frame buffer Per-fragment operations include conditional updates to the frame buffer based on incoming and previously stored z values (for z buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

- **Pixel operation**

  Input data can be in the form of pixels rather than vertices. Such data which might describe an image for texture mapping skips the first stage of processing and instead processed as pixels in the pixel operation stage.

- **Texture memory**

  The result of pixel operation stage is either stored as texture memory for use in rasterization stage or rasterized and resulting fragment merged into the frame buffer just as they were generated from the geometric data.

1.5 THE OPENGL INTERFACE

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are

- **GL – Graphics Library**

  Functions in the main GL (or OpenGL in Windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in Windows).

- **GLU – Graphics Utility Library**

  This library uses only GL functions but contain code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begins with the letters glu.

- **GLUT – OpenGL Utility Toolkit**

  To interface with the window system and to get input from external devices into our programs we need at least one more library. For the X window System, this library is called GLX, for Windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT) , which provides minimum functionality that should be expected in any modern windowing system.

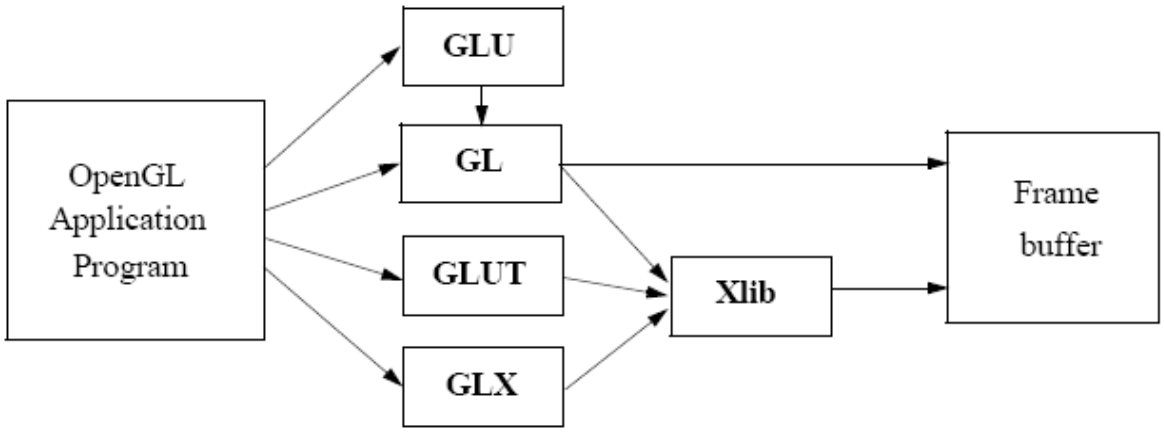

Fig: 1.5 Library Organization

The above figure shows the organization of the libraries for an X Window System environment.

In most implementations, one of the include lines

#include<GL/glut.h>

or

#include<GLUT/glut.h>

is sufficient to read in glut.h, gl.h and glu.h.

1.6 Graphics Functions

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs; we may know nothing about its internal workings.

OpenGL functions can be classified into seven major groups:

- **Primitive function:** The primitive functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, lines, polygons, pixels, text, and various types of curves and surfaces.
- **Attribute functions**

  If primitives are what of an API – the primitive objects that can be displayed- then attributes are the how. That is, the attributes govern the way the primitive appears on the display. Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill inside of a polygon.
- **Viewing functions**

  The viewing functions allow us to specify various views, although APIs differ in the degree of flexibility they provide in choosing a view.

- **Transformation functions**

  One of the characteristics of a good API is that it provides the user with a set of transformations functions such as rotation, translation and scaling.

- **Input functions**

  For interactive applications, an API must provide a set of input functions, to allow users to deal with the diverse forms of input that characterize modern graphics systems. We need functions to deal with devices such as keyboards, mice and data tablets.

- **Control functions**

  These functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.

- **Query functions**

  If we are to write device independent programs, we should expect the implementation of the API to take care of the differences between devices, such as how many colors are supported or the size of the display. Such information of the particular implementation should be provides through a set of query functions.

1.7 DATA TYPES:

OpenGL supports different data types. A list of data types supported by OpenGL is given in the following table.

Sl no.	Suffix	Data type	C type	OpenGL type
1.	B	8 bit int	signed int	GLbyte
2.	S	1 bit int	Short	GLshort
3.	I	32 bit int	Long	GLint , GLsizei
4.	F	32 bit float	Float	GLfloat ,GLclampf
5.	D	64 bit float	Double	GLdouble, GLclampd

6.	Ub	8 bit unsigned	unsigned char	GLubyte, GLboolean
7.	Us	16 bit unsigned	unsigned short	GLushort
8.	Ui	32 bit unsigned	unsigned int	GLuint, GLenum, GLbitfield

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 Software Requirements

1. Operating System : Microsoft Windows XP,Microsoft Windows 7
2. Compiler used: VC++ 6.0 compiler
3. Language used: C/C++

2.2 Hardware Requirements

1. Processor: Intel Core i5
2. Processor Speed: 2.20 GHz
3. RAM Size: 4 GB

CHAPTER 3

3. DESIGN

3.1 BLOCK DIAGRAM FOR SEMAPHORE:

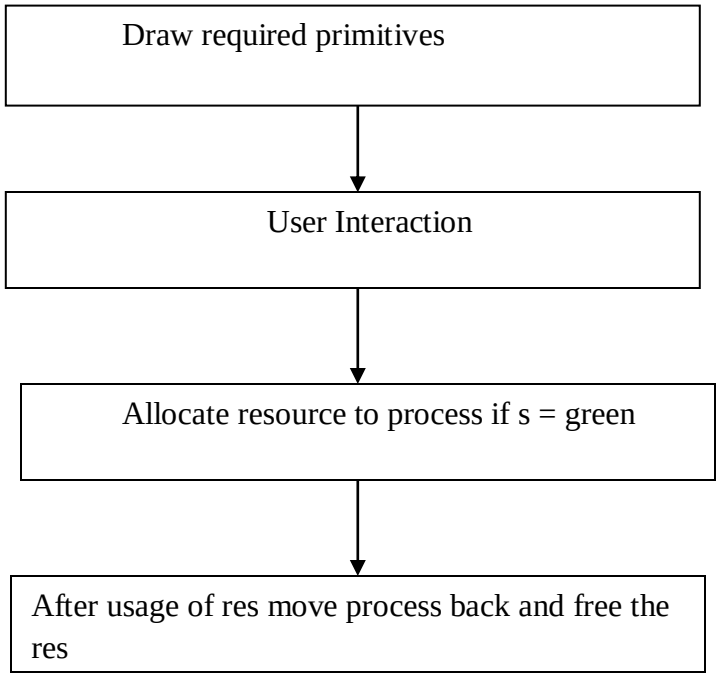

Fig 3.1 Block diagram of semaphore illustration

When process wishes to use the resource it requests for OS. Then depending upon the semaphore value OS decides whether resource should be allocated to the process or the process should be kept in queue.

3.2 FLOWCHART FOR SEMAPHORE:

The following diagram indicates the flow chart for the illustration of semaphores:

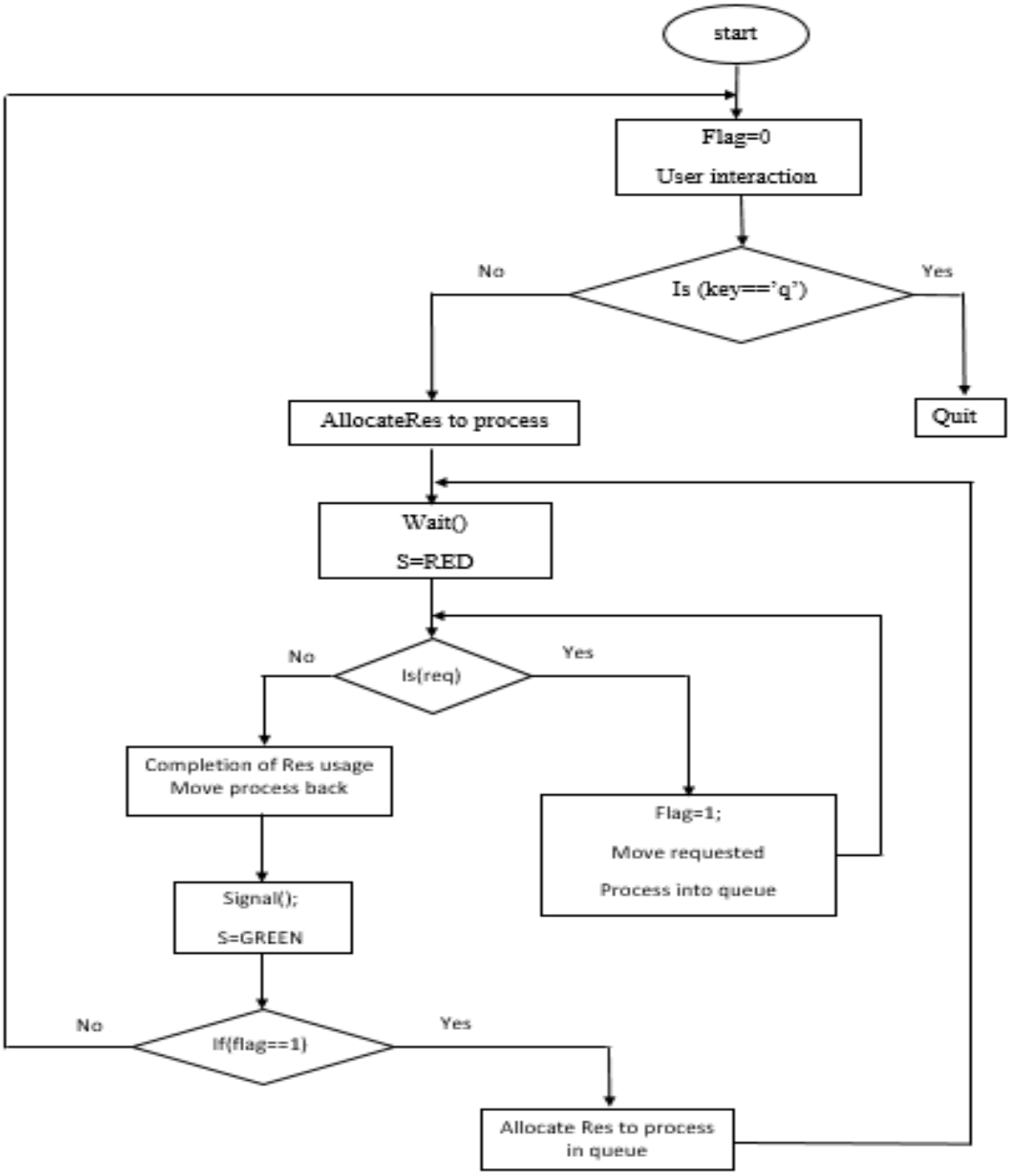

Fig 3.2 Flowchart for semaphore

Initially flag is set to zero. On user's interaction i.e. through keyboard interface if 'q' is given as input then the application is closed. Otherwise when continued and resources needs to be allocated to any process, signal is checked and if 'RED' signal is found the process which requests is sent to queue and is made to wait .When the process using the resource unlocks the resource it is using, the signal is changed to 'GREEN' and the first process waiting in the queue gets allocated to the resource and loop continues until aborted by keyboard interface.

3.3 SCENARIO IS IDENTIFIED AS FOLLOWS:-

- Man - process.
- Crown – Operating system
- 3-d cube- shared resource.
- Light board –Semaphore.
- Light board = GREEN->Semaphore Value = 1;
- Light board = RED->Semaphore Value = 0;
- Resource = WHITE->resource idle;
- Resource = GREEN->resource busy;

3.4 PROCESSES:-

There exist two types of processes. Independent processes and co-operating processes. A process is ***independent* if** it cannot affect or be affected by the other processes executing in the system. A process is ***co-operating*** if it can affect or be affected by the other processes executing in the system. In such a case it can cause to data inconsistency leading to ***race condition***.

With the curiosity about synchronization, we have made our mind to show how orderly execution of co-operating processes can be achieved using SEMAPHORE to avoid data inconsistency, with the knowledge of computer graphics.

- **Critical section** is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution.

- **Semaphore** is a synchronization tool which acts as a protected variable (abstract data type) which can only be accessed using two standard atomic operations: wait ( ) and signal ( ).

- **Mutual exclusion**: It is allowing only one process to access the resource at a time.

The problem with concurrent execution

- Concurrent processes often needs access to shared data or shared resource
- If there is no controlled access to shared data it is possible to obtain an inconsistent view of data.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of co-operating process.

The critical section problem

- More than one process is competing to use same shared data. Each process has a code segment called critical section.
- GOAL: When one process is executing in its critical section, no other process can execute in its critical section.
- Each process looks like this

REPEAT

ENTRY SECTION

CRITICAL SECTION-CS

EXIT SECTION

REMAINDER SECTION-RS

FOREVER

CHAPTER 4

4. IMPLEMENTATION

4.1 OVERVIEW

OpenGL is a state machine. You can put it into various states(modes) which then remain in effect until you change them. For example, the current color is a state variable. You can set the current color to white, red or any other color thereafter every object is drawn with that color until you set the current color to something else. The current color is only one of the many state variables that openGL maintains. Others controls such things as the current viewing and projection transformations, polygon drawing modes, pixel-packing convensions, positions and characteristicsof light and material properties of the objects being drawn.

This project demonstrates the working of semaphores.The operating system consists of all the resources that a process needs.Whenever more than one process request for the resources, the OS will allocate resource to the process one at time based on first come first serve basis.The other processes requesting for the resources are kept in the waiting queue.Whenever the first process releases the resource,the other process waiting in the queue are allocated resources by the OS.

4.2 USER INTERFACE

In this project we are demonstrating the illustration of semaphores,using both keyboard and mouse interactions.

- Pressing key 1 indicates porocess1 is requesting resource to the operating system.If the resource is idle,then the OS will allocate it to the process1 and semaphore becomes red indicating resource is busy.
- If at this time,when key 2 is pressed indicates that process2 also requests for the resource.The OS informs the process2 that resource is busy and process2 is made to enter the semaphore's waiting queue.
- By pressing key 'b' or 'B' the process1 releases its allocated resources and semaphore signal becomes green and process2 can now access the resource.
- By pressing key 'q' or 'Q' all the resources are released and the process terminates.

4.3 STRUCTURE

- Draw_resource - function defined to build 3d cube which represents the shared resource.
- Draw_crown - function defined to build operating system in the shape of crown.
- Draw_man - function defined to build process in the form of man.
- Draw_lightboard - function defined to build semaphore in the form of signal board.
- Draw_q - function defined to build semaphore queue.
- Mykey - function defined to get user interaction and it's a parameter for glutKeyboardFunc.
- Move_man - function defined to move process by varying parameters of glTranslatef using for loops.
- Wait – to have delay.

4.4 ANALYSIS

PURPOSE

1.Demonstrating the concept of semaphore.

2.Understanding various call back functions in OpenGL.

3.Demonstrating the use of user defined functions.

FUNCTIONS

4.4.1 FUNCTION TO DRAW RESOURCE

```
void draw_resource(float x, float y, float z)
{
Char*msg=”Resource”;
.
.
.
glPopMatrix();
}
```

4.4.2 FUNCTION TO DRAW CROWN

```
void draw_crown(char*msg)
{
int i;
char *label=”operating system”;
.
.
.
glLoadIdentity();
}
```

4.4.3 FUNCTION TO DRAW MAN

```
void draw_man(float x, float y, float z, intisprocess)
{
int i;
float theta;
.
.
.
glLoadIdentity();
}
```

4.4.4 FUNCTION TO DRAW LIGHT BOARD

```
void draw_light_board(float x, float y, float z, intisred, char*msg)
{
int i;
float theta;
char *label=”semaphore”;
.
.
.
glLoadIdentity();
}
```

4.4.5 FUNCTION TO DRAW QUEUE

```
void draw_q(void)
{
char *msg=”QUEUE(process waiting for resource”);
int i;
.
.
.
glFlush();
}
```

4.4.6 FUNCTION FOR KEYBOARD INTERFACE

```
void MyKey(unsigned char key, int x, int y)
{
if (!in_welcome)
{
if(key==’1’)
.
.
.
if(key==’q’ || key==’Q’)
exit (0);
}
```

4.4.7 FUNCTION TO MOVE MAN

```
void move_man(int man)
{
floatx,y=3,z=-10;
char *msg=”wait”;
```

.

.

.

glutPostRedisplay();

}

4.4.8 FUNCTION TO WAIT

```
void wait (int nTimes)
{
int i,j;
for(i=0;i<nTimes*100;i++)
for(j=0;j<=nTimes*100;j++);
}
```

CHAPTER 5

5. SNAPSHOT

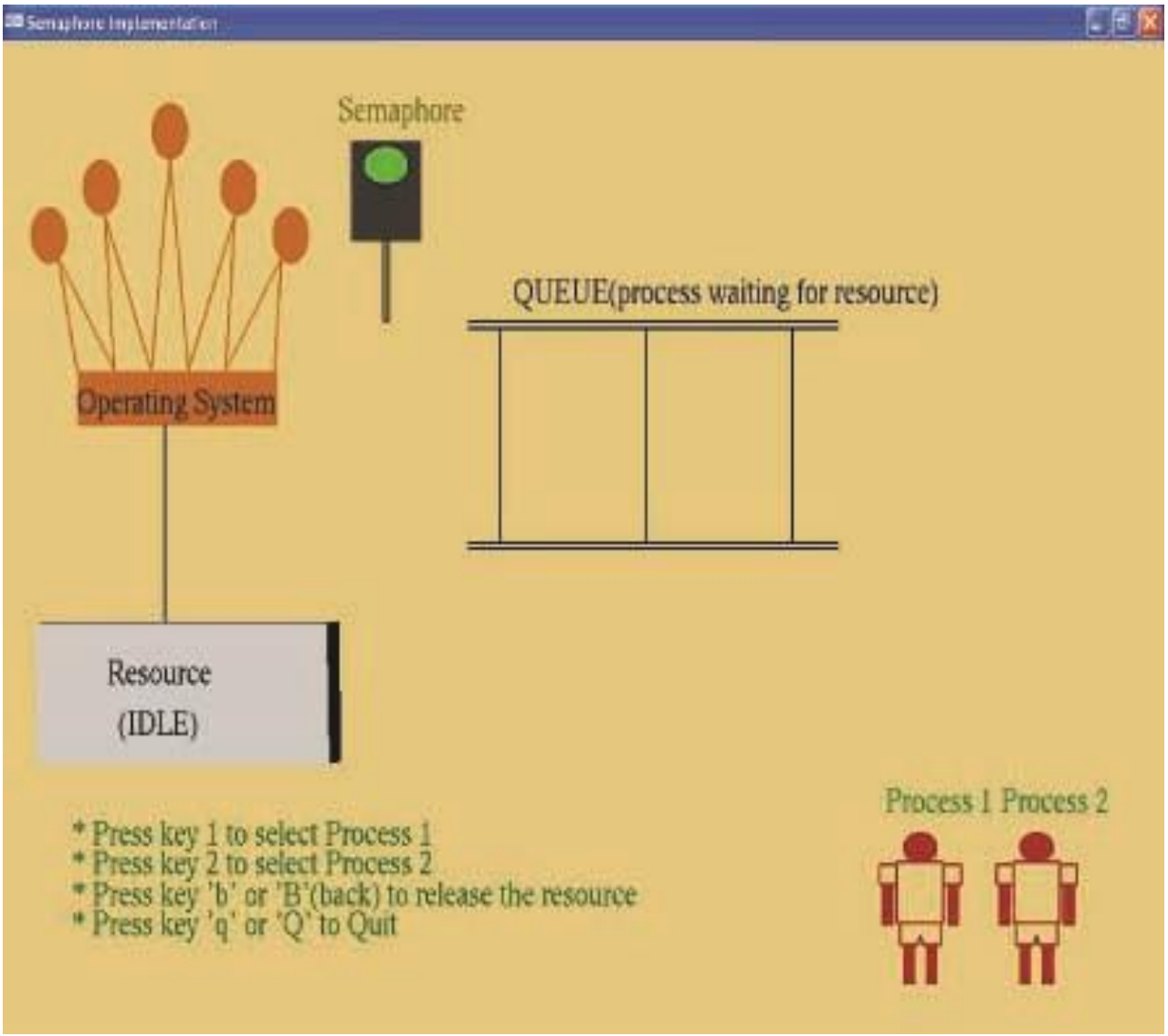

Fig 5.1 Initial stage of processes before requesting for resource.

In this figure,no process is requesting the operating system to allocate the resource ,hence the resource is idle.

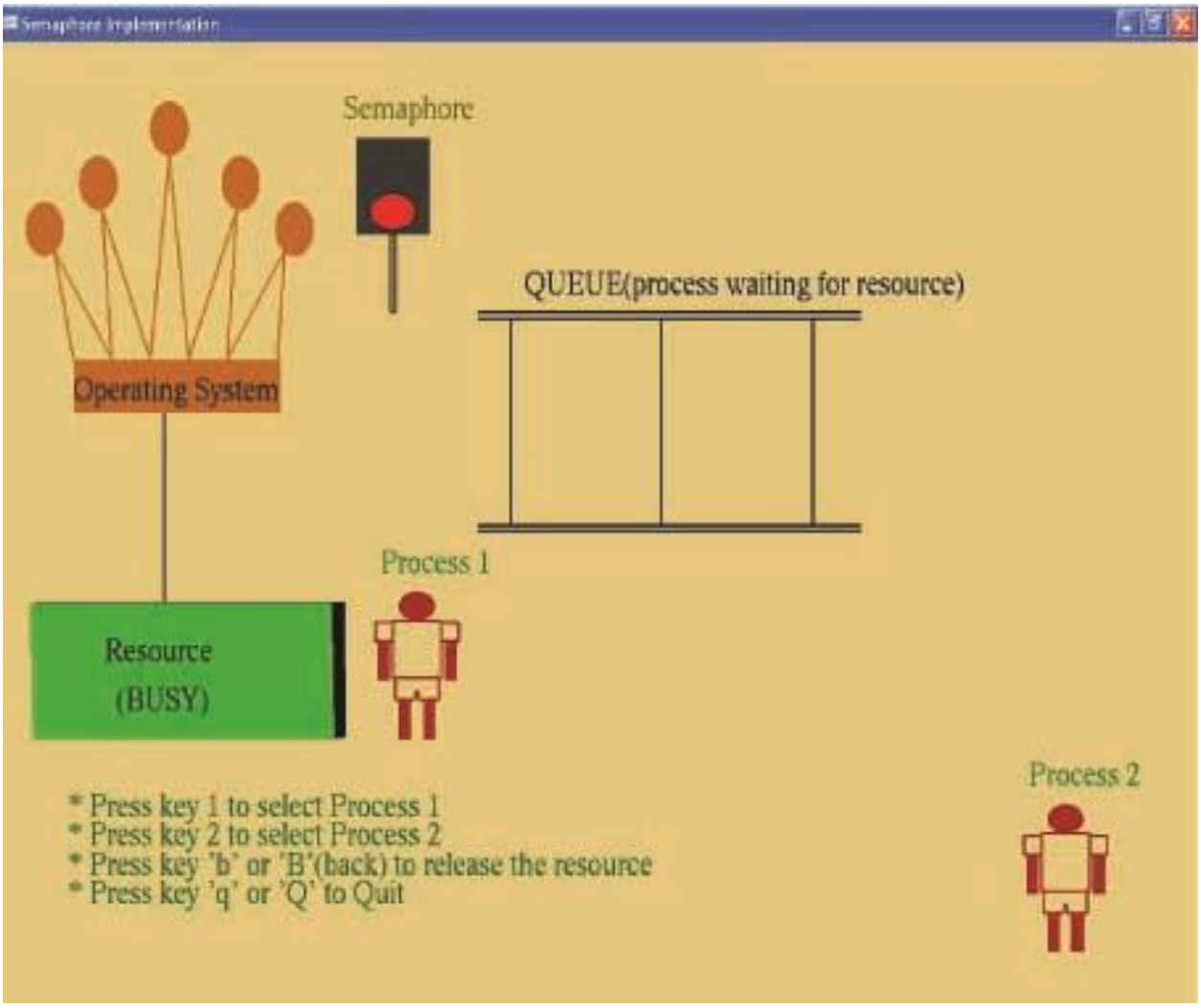

Fig5.2 process1 moves to use resource and semaphore turns red.

In this figure,process1 requests for the resource and hence the OS allocates it to the process1 for its use.

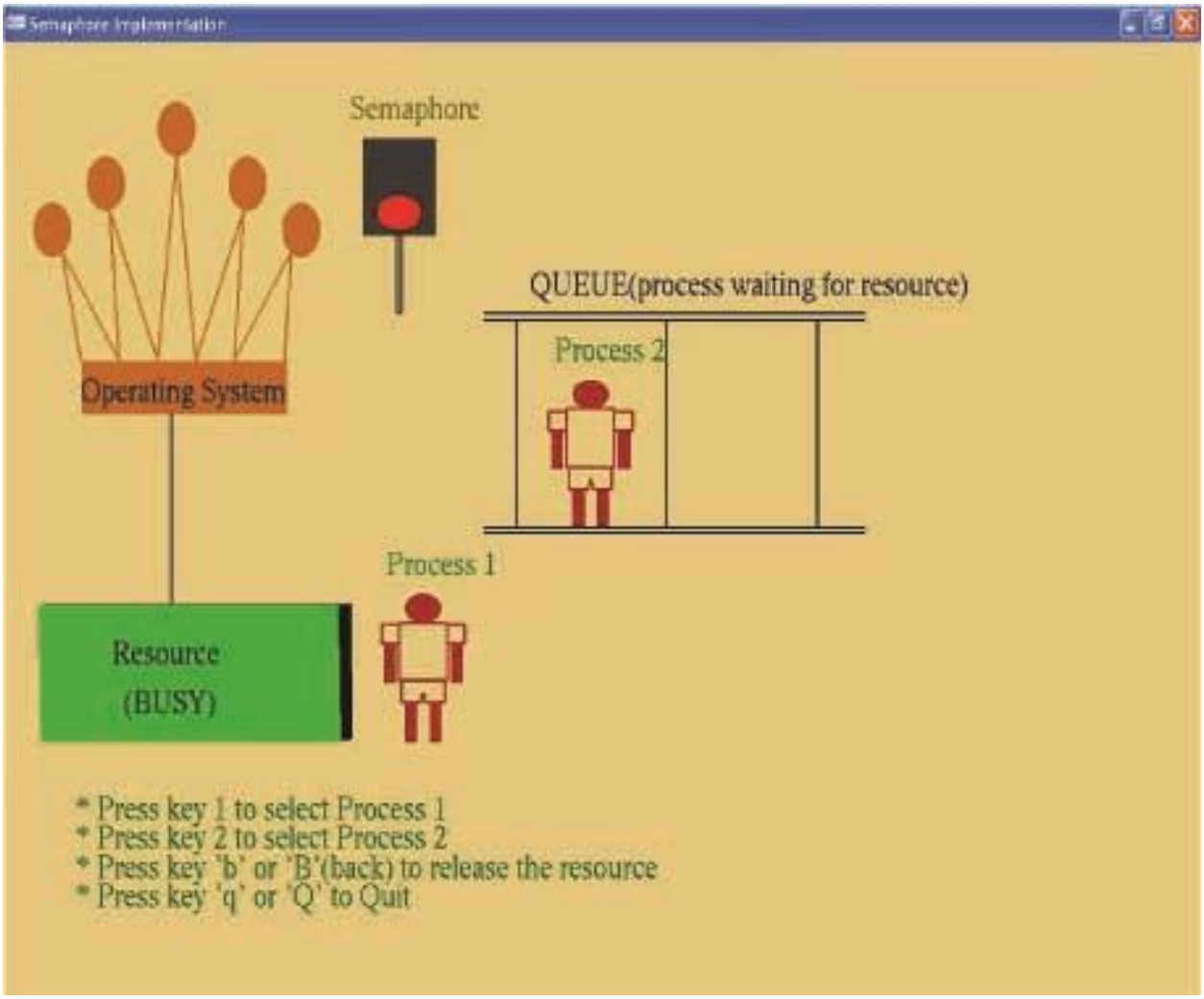

Fig 5.3 OS moves process2 into queue

In this figure,The process1 is already using the resource, the process2 also requests for the resource at the same time,so the OS sends it to the waiting queue.

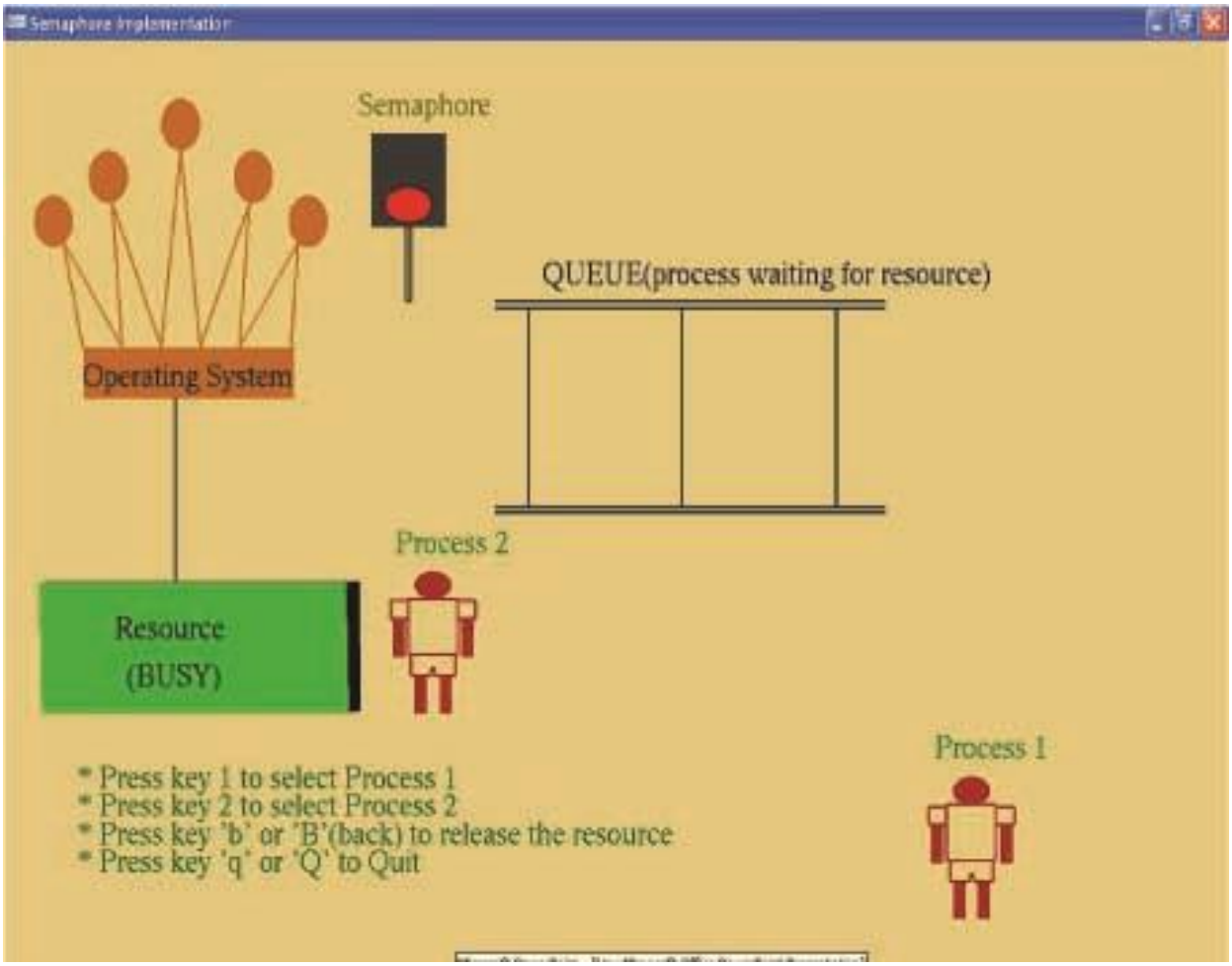

Fig 5.4 OS allocates resource to resource to process2

In this figure, the process1 released it resource and the OS allocates the resource to the process2 which was earlier in the waiting queue.

CONCLUSION

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact with the computer efficiently and easily. In this project we have demonstrated the usage of semaphores in allocating the resources to the processes by the OS. If only one process is allowed to use shared resource at a time then race condition can be easily avoided. **SEMAPHORE** can be used to solve various synchronization problems and can be implemented efficiently.

BIBLIOGRAPHY

Books:

1. Edward Angel: Ineractive Computer Graphics A Top-Down Approach Using OpenGL, Pearson Education, 5th Edition.
2. F. S. Hill, Jr: Computer Graphics Using OpenGL Pearson Education, 3rd Edition.
3. Foley Van Dam Feiner Hughes: Computer Graphics Principles & Practice, Pearson Education, 2nd Edition in C.
4. Donald Hearn, M.Pauline Baker: Computer Graphics 'C' version Pearson Education, 2nd Edition.

List of websites:

1. http://www.opengl.org/
2. http://www.academictutorials.com/graphics/graphics-flood-fill.asp
3. http://www.glprogramming.com/

APPENDIX

- **API:** Application Programming Interface.
- **GL:** Graphics Library.
- **GLU:** Graphics Utility Library.
- **GLUT:** OpenGL Utility Toolkit.
- **GLX:** OpenGL Extension to the X Window System.

.