NITTE EDUCATION TRUST
N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎ : 08258 - 281039 – 281263, Fax: 08258 – 281265

Department of Computer Science and Engineering
B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

Report on Mini-Project

"CPU SCHEDULING ALGORITHMS"

Course Name: Computer Graphics Lab

Course Code: 18CS607

Semester: 6 Section: C

Submitted to,

Mr. Puneeth R. P.

Asst Prof Gd. II,

Dept. of CSE, NMAMIT

Submitted by:

Name: Prathvik Shervegar **USN:** 4NM18CS119

Name: Pratik Chandra Chandragiri **USN:** 4NM18CS120

Date of submission:

28/05/2021

Signature of Course Instructor

ABSTRACT

The main theme behind the project is to use the basic concepts of computer graphics to visualize CPU scheduling algorithms along with Banker's algorithm. What we learnt from this project is how to build the project from scratch and the basics of computer graphics by programming in OpenGL.

We are developing this using OpenGL. OpenGL is an application program interface(API) offering various functions to implement models and images and movement of them. This offer Functions to create and manipulate render lighting colouring, viewing the models.

TABLE OF CONTENTS

INTRODUCTION

The Computer Graphics mini project "CPU SCHEDULING ALGORITHMS AND BANKER'S ALGORITHM" is a project that is developed using OpenGL in which we implement the basic 2D animation of polygons to show how process scheduling takes place using various CPU scheduling algorithms such as

i. FCFS (First Come First Serve)
ii. Round Robin
iii. SJF (Shortest Job First)

Also, we show Deadlock avoidance using Banker's algorithm.

1. FCFS (First Come First Serve) Scheduling:

   First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue.

2. Round Robin Scheduling:

   The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is mostly used for multitasking. In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

3. SJF (Shortest Job First) Scheduling:

   Shortest Job First (SJF)is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution. We are implementing non-preemptive method.

4. Banker's algorithm:

   This algorithm is used to test for safely simulating the allocation for determining the maximum amount available for all resources. It also checks for all the possible activities before determining whether allocation should be continued or not.

The animations made in OpenGL using C++ programming language. OpenGL is a cross-language, cross-platform application programming interface(API) offering various functions to produce 2D and 3D graphics.

IMPLEMENTATION DETAILS

The functions mentioned below are involved in creating the crucial components in the project.

A. void displaytext() used to display text characters.

```
void displayText( float x, float y, float r, float g, float b,int op,char str[] ) {
   int j = strlen (str);
   glColor3f( r, g, b );
   glRasterPos2f( x, y );
   for( int i = 0; i < j; i++ ){
      switch(op){
         case 1: glutBitmapCharacter( GLUT_BITMAP_TIMES_ROMAN_24, str[i] );
               break;
         case 2: glutBitmapCharacter( GLUT_BITMAP_9_BY_15, str[i] );
               break;
         case 3: glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, str[i] );
               break;
         case 4: glutBitmapCharacter( GLUT_BITMAP_8_BY_13, str[i] );
               break;
         default: break;
      }
   }
   glFlush();
}
```

B. void displayfirst() displays the first introductory window.

```
void displayfirst() {
      displayText(450,450,0,0,0.4,1,"NMAM Institute of Technology");
      displayText(370,410,0,0,0.4,1,"Department of Computer Science and
        Engineering");
      displayText(450,335,0.3,0.5,0,1,"Computer Graphics Mini Project");
      displayText(450,330,0.3,0.5,0,1,"_____");
      displayText(80,180,0,.5,1,1,"Topic: CPU Scheduling Algorithms and Banker's
        Algorithm");
      displayText(80,120,1,0,0,1,"Faculty In-charge: Mr. Puneeth R.P.");
      displayText(270,100,1,0,0,3,"(Assistant Professor Gd-II)");
      displayText(770,191,0.2,0,0.2,1,"Members:-");
      displayText(770,190,0.2,0,0.2,1,"_______");
      displayText(770,150,0.2,0,0.5,3,"Prathvik Shervegar: 4NM18CS119");
      displayText(770,120,0.2,0,0.5,3,"Pratik Chandra Chandragiri: 4NM18CS120");
      displayText(900,25,0.5,0.5,0.5,3,"Press 'ENTER' to continue");
}
```

C. void osprocess() displays the explaination about Process scheduling in OS.

```
void osprocess() {
  displayText(60,445,2,0.5,0,1,"Process Scheduling:");
  displayText(60,440,2,0.5,0,1,"_________________");
  glColor3f(0.9,0.9,0.9);
  glBegin(GL_LINE_LOOP);
  glVertex2f(50,380);
  glVertex2f(50,80);
  glVertex2f(550,80);
  glVertex2f(550,380);
  glEnd();
  glBegin(GL_LINE_LOOP);
  glVertex2f(50,80);
  glVertex2f(550,80);
  glVertex2f(550,30);
  glVertex2f(50,30);
  glEnd();
  displayText(270,445,1,1,1,3,"It is the activity of process manager in OS that handles
    removal of the running process from the CPU and ");
  displayText(270,415,1,1,1,3,"the selection of another process on the basis of a
    particular strategy.");
  displayText(60,355,1,1,1,3,"Types of Process Schedulers");
  displayText(60,325,1,1,1,2,"i) Long Term Scheduler:");
  displayText(110,305,1,1,1,4,"It is also known as Job Scheduler. This regulates the");
  displayText(80,290,1,1,1,4,"program and select process from queue and load into");
  displayText(80,275,1,1,1,4,"memory for execution. It also regulates the degree of ");
  displayText(80,260,1,1,1,4,"multi-programming.");
  displayText(60,230,1,1,1,2,"ii) Medium Term Scheduler:");
  displayText(110,210,1,1,1,4,"It plays an important part of Swapping. It enables us");
  displayText(80,195,1,1,1,4,"to handle the swapped-out processes.");
  displayText(60,165,1,1,1,2,"iii) Short Term Scheduler:");
  displayText(110,145,1,1,1,4,"It is also known as CPU Scheduler. The main goal is");
  displayText(80,130,1,1,1,4,"to boost the system performance according to set
    criteria.");
  displayText(80,115,1,1,1,4,"This helps us to select from group of processes that
    are");
  displayText(80,100,1,1,1,4,"ready to execute and allocates CPU to one of them.");
  displayText(70,50,1,1,1,3,"We are going to focus on CPU Scheduling algorithms.");
  glColor3f(1,0.9,0);
  glBegin(GL_POLYGON);
  glVertex2f(610,260);
  glVertex2f(610,200);
  glVertex2f(710,200);
  glVertex2f(710,260);
  glEnd();
  glBegin(GL_LINES);
  glVertex2f(710,230);
  glVertex2f(810,230);
 glEnd();
```

```
glBegin(GL_POLYGON);
glVertex2f(810,260);
glVertex2f(810,200);
glVertex2f(930,200);
glVertex2f(930,260);
glEnd();
glBegin(GL_LINES);
glVertex2f(930,230);
glVertex2f(1030,230);
glEnd();
draw_circle(1060,230,35);
glBegin(GL_LINES);
glVertex2f(1090,230);
glVertex2f(1160,230);
glEnd();
glPushAttrib(GL_ENABLE_BIT);
glLineStipple(1,0xBBBB);
glEnable(GL_LINE_STIPPLE);
glBegin(GL_LINES);
glVertex2f(1060,260);
glVertex2f(1060,360);
glEnd();
glBegin(GL_LINES);
glVertex2f(1060,360);
glVertex2f(760,360);
glEnd();
glBegin(GL_LINES);
glVertex2f(760,360);
glVertex2f(760,245);
glEnd();
glBegin(GL_LINES);
glVertex2f(760,245);
glVertex2f(810,245);
glEnd();
glPopAttrib();
glBegin(GL_LINES);
glVertex2f(1060,260);
glVertex2f(1060,110);
glEnd();
glBegin(GL_LINES);
glVertex2f(1060,110);
glVertex2f(1020,110);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(1020,140);
glVertex2f(1020,80);
glVertex2f(860,80);
glVertex2f(860,140);
glEnd();
glBegin(GL_LINES);
glVertex2f(860,110);
glVertex2f(810,110);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(810,140);
glVertex2f(810,80);
glVertex2f(730,80);
glVertex2f(730,140);
glEnd();
glBegin(GL_LINES);
glVertex2f(770,140);
glVertex2f(770,215);
glEnd();
glBegin(GL_LINES);
glVertex2f(770,215);
glVertex2f(810,215);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(810,230);
glVertex2f(800,225);
glVertex2f(800,235);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(810,245);
glVertex2f(800,240);
glVertex2f(800,250);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(810,215);
glVertex2f(800,210);
glVertex2f(800,220);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(1160,230);
glVertex2f(1150,225);
glVertex2f(1150,235);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(1020,110);
glVertex2f(1030,105);
glVertex2f(1030,115);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(810,110);
glVertex2f(820,105);
glVertex2f(820,115);
glEnd();
```

```
    glColor3f(1,0,0);
    strin("Job Queue",9,620,230);
    strin("Ready Queue",11,820,230);
    strin("CPU",3,1045,230);
    strin("I/O Waiting Queue",17,865,110);
    strin("I/O",3,755,110);
    glColor3f(0,1,0);
    strin("SWAP-OUT AND SWAP-IN",20,810,365);
    strin("END",3,1110,235);
    displayText(900,25,0.5,0.5,0.5,3,"Press 'ESC' to go back");
    glFlush();
}
```

Similarly osdeadlock() displays the explaination about Deadlock and methods to solve deadlock in OS which is implemented same as above.

D. struct process is used to store input values

```
struct process {
  int bt,at,st,ct,no;
  char name;
  int alloc[3],maxi[3];
};
```

Queue is used to operate on the obtained input values

E. Taking user input by rendering different screens and storing value of key pressed in variable.

```
void constr() // To show the constraints {
  int p=15,q=200;
  glColor3f(1,.0,.0);
  strin("Please note:",12,p,q);
  strin("_____________",12,p,q-4);
  strin("-> The number of processes is limited to 5.",43,p,q-(30*1));
  strin("-> All processes must arrive before 7sec.",41,p,q-(30*2));
  strin("-> Total burst time of all the processes must not exceed 30.",60,p,q-(30*3));
  glColor3f(.0,1.0,0.0);
  strin("PRESS E or e TO RUN PROGRAM FOR STATIC INPUT",44,15,320);
  strin("PRESS R or r TO RESET THE PROGRAM",33,15,300);
  glColor3f(0,0,0);
}
```

```
void backgrnp()      //to ask for number of process {
     stri("Enter the number of processes:",30,15,375);
    constr();
    glFlush();
}
void backgrat(int I)       //to get the arrival time {
    char a='0'+(i+1);
    stri("Enter the arrival time of process     :",39,15,375);
    stri(&a,1,340,375);
    constr();
    glFlush();
}//background for getting arrival time
void backgrbt(int I)     //to get the burst time {
    char a='0'+(i+1);
    stri("Enter the burst time of process     :",37,15,375);
    stri(&a,1,325,375);
    constr();
    glFlush();
}//background for getting brust time

void backgrtq()        //Request the time quantum {
    glColor3f(0,0,0);
    stri("Enter the time quantum: ",24,15,375);
    constr();
    glFlush();
}

void backgralom(int I) {
    char a='0'+(i+1);
    glColor3f(0,0,0);
    stri("Enter the values of allocation matrix of process     :",47,15,375);
    stri(&a,1,385,375);
    constr1();
    glFlush();
}
void backgrmaxm(int I) {
    char a='0'+(i+1);
    glColor3f(0,0,0);
    stri("Enter the values of max matrix of process     :",46,15,375);
    stri(&a,1,365,375);
    constr1();
    glFlush();
}
void backgravailm(){
    glColor3f(0,0,0);
    stri("Enter the values of available matrix     :",34,15,375);
    constr1();
    glFlush();
}
```

F. Storing keyboard key press value in variable (Reading value of no. of processes)

```
if(isdigit(a) && one==1) {
    one=0;
    numb=a-'0';
    if(np==1)
    {
       if(numb<=5)
       {
          stri(&a,1,360,375);
          backgrnp();
          np=0;
          att=numb-1;
          index1=0;
          n=numb;
          usleep(300000);
          glClear(GL_COLOR_BUFFER_BIT);
          backgrat(index1);
          one=1;
       }
       else
       {
          for(int k=0;k<3;k++) {
             glClear(GL_COLOR_BUFFER_BIT);
             backgrnp();
             glFlush();
             glColor3f(1, 1, 1);
             strin("ERROR", 5, 15, 95);
             strin("_____", 5, 15, 93);
             strin("Number of processes greater than 5 not allowed ", 47, 15, 80);
             glFlush();
             delay();
             glClear(GL_COLOR_BUFFER_BIT);
             glClearColor(0.1,0.1,.6,.803);
             backgrnp();
             delay();
          }
          one=1;
          glClearColor(0.1,0.1,.6,.803);
       }
    }
}
```

Similarly, arrival time of each process, burst time of each process, time quantum, allocation matrix, max matrix and available matrix inputs are read.

G. Options menu with user inputs represented in a table

I. void displayoptions()

```
void displayOptions() {
  glClear(GL_COLOR_BUFFER_BIT);
  displayText(600, 170, 1, 1, 1,1,op6);
  displayText(600, 200, 1, 1, 1,1,op5);
  displayText(600, 230, 1, 1, 1,1,op4);
  displayText(600, 260, 1, 1, 1,1,op3);
  displayText(600, 290, 1, 1, 1,1,op2);
  displayText(600, 320, 1, 1, 1,1,op1);
  displayText(600, 350, 1, 0, 0,1,op);
  table();
  table1();
  glBegin(GL_LINES);
  glVertex2f(500,500);
  glVertex2f(500,0);
  glEnd();
  glBegin(GL_LINES);
  glVertex2f(0,245);
  glVertex2f(500,245);
  glEnd();
  displayText(50,460,1,1,1,3,"Inputs for CPU scheduling algorithm");
  displayText(70,215,1,1,1,3,"Inputs for Banker's algorithm");
  displayText(100,260,1,1,0,2,"Time Quantum = ");
  prinum(240,260,tq);
  glLineWidth(1.0);
  glColor3f(0.0,0.87,.803);
}
```

II. void table() and void table1()

```
void table() {
   int p,q,r,s,i;
   char a;
   keypa=0;
   glColor3f(1,1,1);
   p=30;
   q=435-(25*(1+n));
   s=435;
   r=330;
   glLineWidth(2.0);
   glBegin(GL_LINE_LOOP);
   glVertex2i(p,q);
   glVertex2i(p,s);
   glVertex2i(r+30,s);
   glVertex2i(r+30,q);
   glEnd();
   glBegin(GL_LINES);
   glVertex2i(130,s);
   glVertex2i(130,q);
   glVertex2i(233+13,s);
   glVertex2i(233+13,q);
   glVertex2i(p,410);
   glVertex2i(r+30,410);
   glEnd();
   strin("Process",7,55,415);
   strin("Arrival time",12,135,415);
   strin("Burst time",10,253,415);
   for( i=1;i<=n;i++)
   {
      strin("P",1,75,438-(25*(i+1)));
      a='0'+i-1;
      strin(&a,1,83,438-(25*(i+1)));
   }
   pro();
   glFlush();
}
```

```
void table1() {
    int p,q,r,s,i;
    char a;
    keypa=0;
    glColor3f(1,1,1);
    p=30;
    q=180-(25*(1+n));
    s=200;
    r=450;
    glLineWidth(2.0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(p,q);
    glVertex2i(p,s);
    glVertex2i(r+30,s);
    glVertex2i(r+30,q);
    glEnd();
    glBegin(GL_LINES);
    glVertex2i(130,s);
    glVertex2i(130,q);
    glVertex2i(233+13,s);
    glVertex2i(233+13,q);
    glVertex2i(351+13,s);
    glVertex2i(351+13,q);
    glVertex2i(p,150);
    glVertex2i(r+30,150);
    glVertex2i(130,175);
    glVertex2i(r+30,175);
    glEnd();
    strin("Process",7,55,180);
    strin("Allocation",10,145,180);
    strin("Max",3,290,180);
    strin("Available",9,381,180);
    strin("A  B  C",7,154,155);
    strin("A  B  C",7,272,155);
    strin("A  B  C",7,387,155);
    for( i=1;i<=n;i++)
    {
       strin("P",1,75,182-(25*(i+1)));
       a='0'+i-1;
       strin(&a,1,83,182-(25*(i+1)));
    }
    pro1();
    glFlush();
}
```

III. Keyboard press function for options

```
switch(a) {
 case '1': execFCFS();
          break;
 case '2': execRR();
          break;
 case '3': execSJF();
          break;
 case '4': execBSA();
          break;
 case '5': glClear(GL_COLOR_BUFFER_BIT);
          tq=99;
          restart();
          doneInput=false;
          backgrnp();
          break;
 case '6': glutDestroyWindow(wid);
          exit(0);
 case 'm':
 case 'M': restart();
          doneInput=true;
          displayOptions();
          break;
}
```

H. Visualization of Round Robin scheduling

```
//display gantt chart
void gchart() {
    int i;
    char a;
    float me;
    // setcst();
    drawscale();
   // basic=34;
    strin("Gantt Chart",11,30,158+40);
    strin("_____________",11,30,156+40);
    for(i=0;i<=rear;i++) {
        glColor3fv(col[q[i]]);
        drawrec(30+p[i].st*36,100,30+p[i].ct*36,130);
        glLineWidth(2.0);
        glColor3f(0.0,0,0);
        drawloop(30+p[i].st*36,100,30+p[i].ct*36,130);
        glColor3fv(col[q[i]]);
        me=(30+p[i].st*36+30+p[i].ct*36)/2;
        strin("P",1,me-8,88);
        a='0'+q[i];
        strin(&a,1,me,88);
        arri(i);
        glFlush();
        delay();
    }
    glLineWidth(1.0);
    glColor3f(1,.84,.0);
}

//to create animation
void delay() {
    for(int i=0;i<35000;i++)
        for(int j=0;j<10000;j++);
}

//to set values for round robin method(logic part)
void setValuesRR() {
    int cp;
    sort1(); //Sort the processes according to their arrival time in increasing order
    while(front<=rear) {
        cp=rem();
```

```
            if(cp!=-1) {
                if(process[cp].rbt<=tq && process[cp].rbt>0)  {
                    pt=total;
                    total = total + process[cp].rbt;
                    process[cp].rbt=0;
                    process[cp].ct=pt;
                    process[cp].st=total;
                    display(cp);
                }
                else if(process[cp].rbt>tq) {
                    pt=total;
                    total=total + tq;
                    process[cp].rbt = process[cp].rbt - tq;
                    display(cp);
                    checkForArrivals(cp);
                }
            }
        }
    }

    void execRR() {
        int i;
        for(i=0;i<n;i++) {
            process[i].at=p[i].at;
            process[i].rbt=process[i].bt=p[i].bt;
        }
        for(i=0;i<n;i++) {
            p[i].name='0'+char(i);
        }
        glClear(GL_COLOR_BUFFER_BIT);
        table();
        displayName("Round Robin");
        setValuesRR();
        gchart();
        displayText(700, 300, 1, 1, 0,1,"Press 'm' to go back to the main menu");
        avgRR();
    }
```

Similarly, other algorithms are visualized considering their respective logic.

CONCLUSION

The pseudocode shows how the entire project has been implemented. With this project we have shown the visualization of CPU scheduling algorithms and Banker's Agorithm using various features of OpenGL.

OpenGL supports enormous flexibility in design and the use of OpenGL graphics programs. The Presence of many built in classes methods take care of much functionality and reduce the job of coding as well as makes the implementation simpler. We have implemented the project making it user-friendly and error free as possible.

REFERENCES

- https://docs.microsoft.com/en-us/windows/win32/opengl/gl-functions
- https://www.opengl.org/resources/libraries/glut/spec3/spec3.html
- https://lazyfoo.net/tutorials/OpenGL/index.php
- https://www.opengl.org/resources/libraries/

APPENDIX

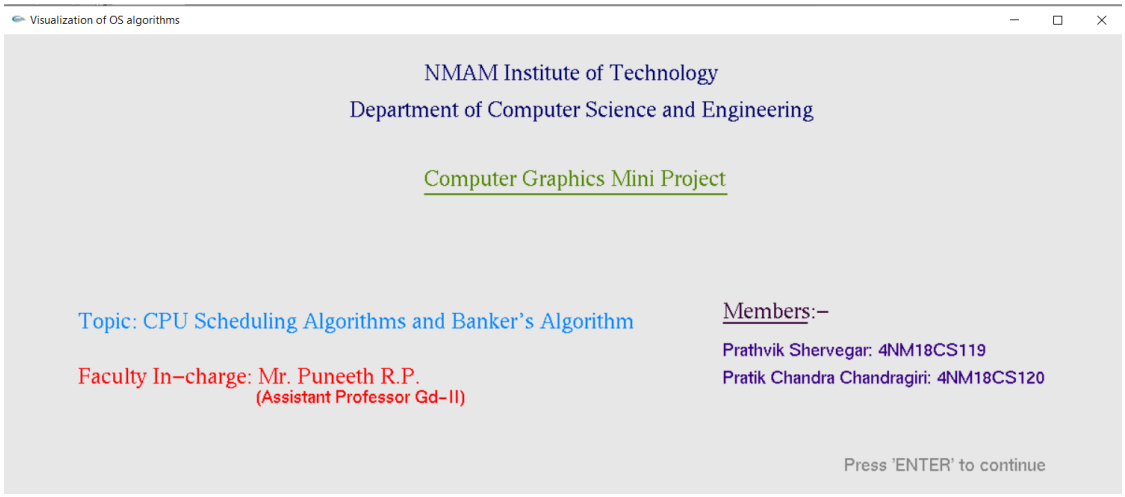

Figure 1: Introduction window

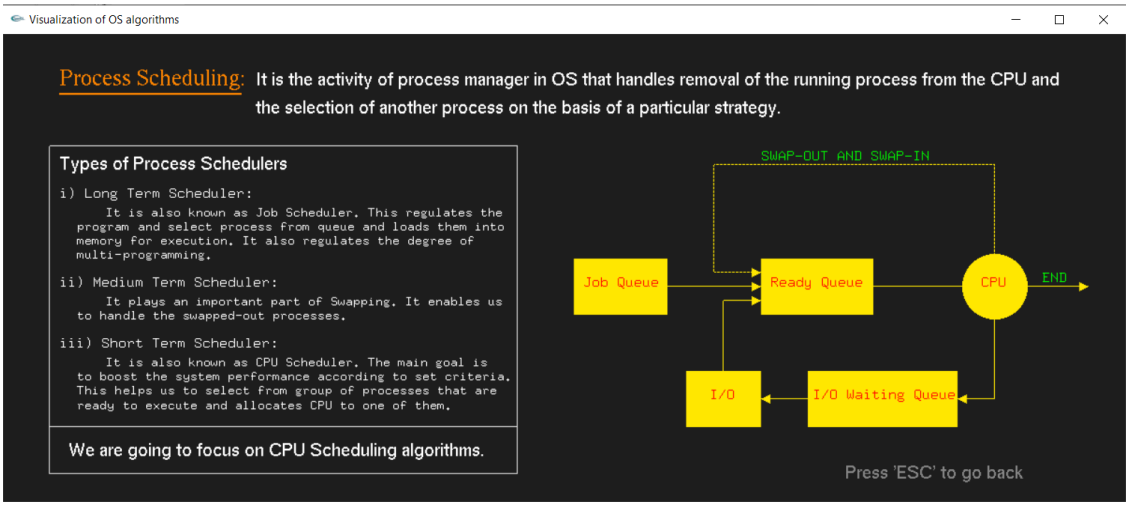

Figure 2: About Process scheduling

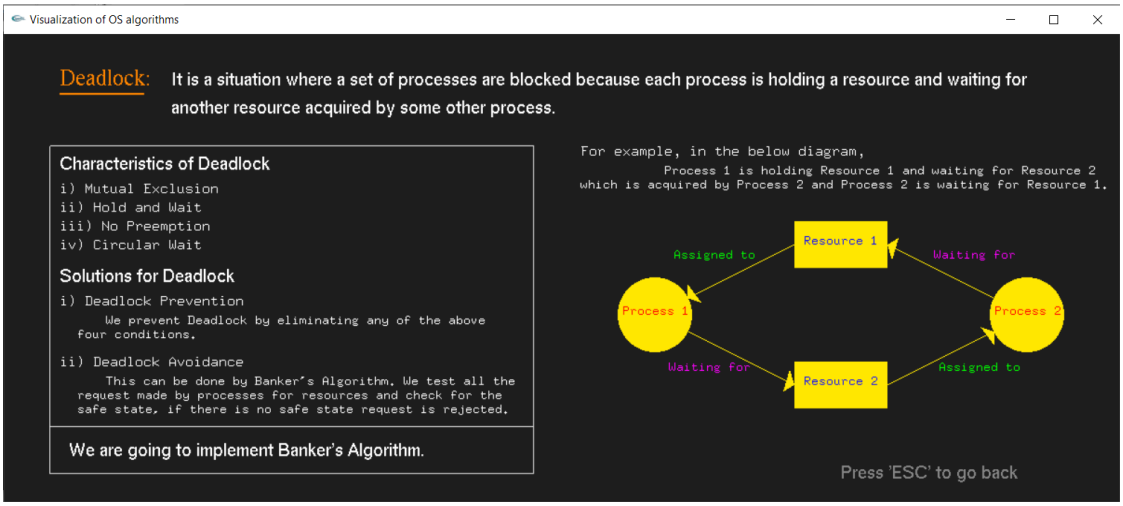

Figure 3: About Deadlock in OS

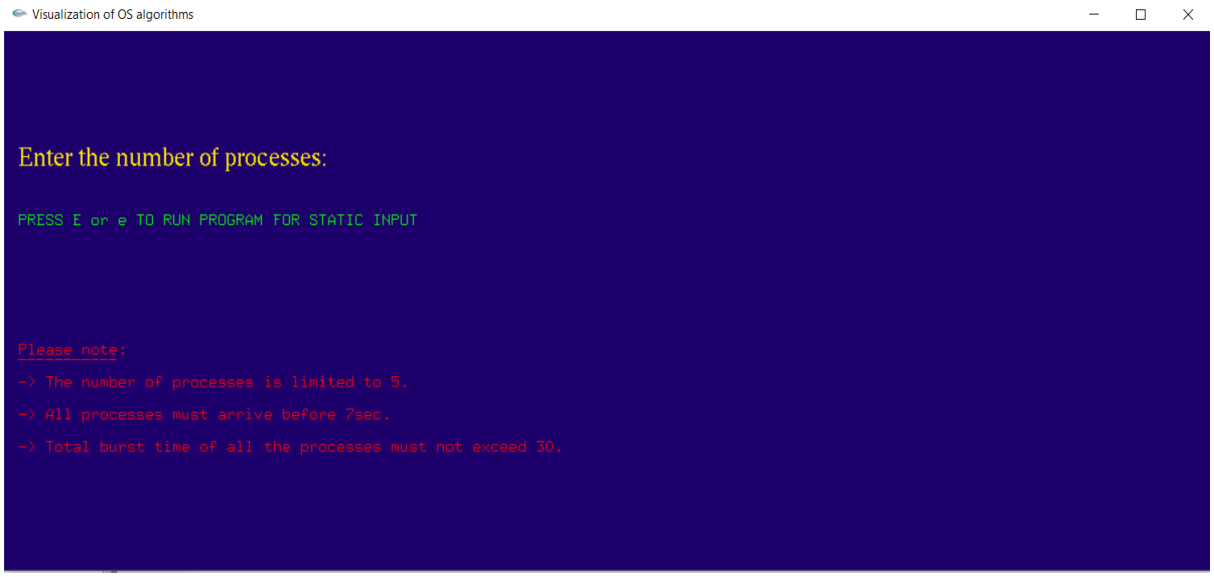

Figure 4: User input screen

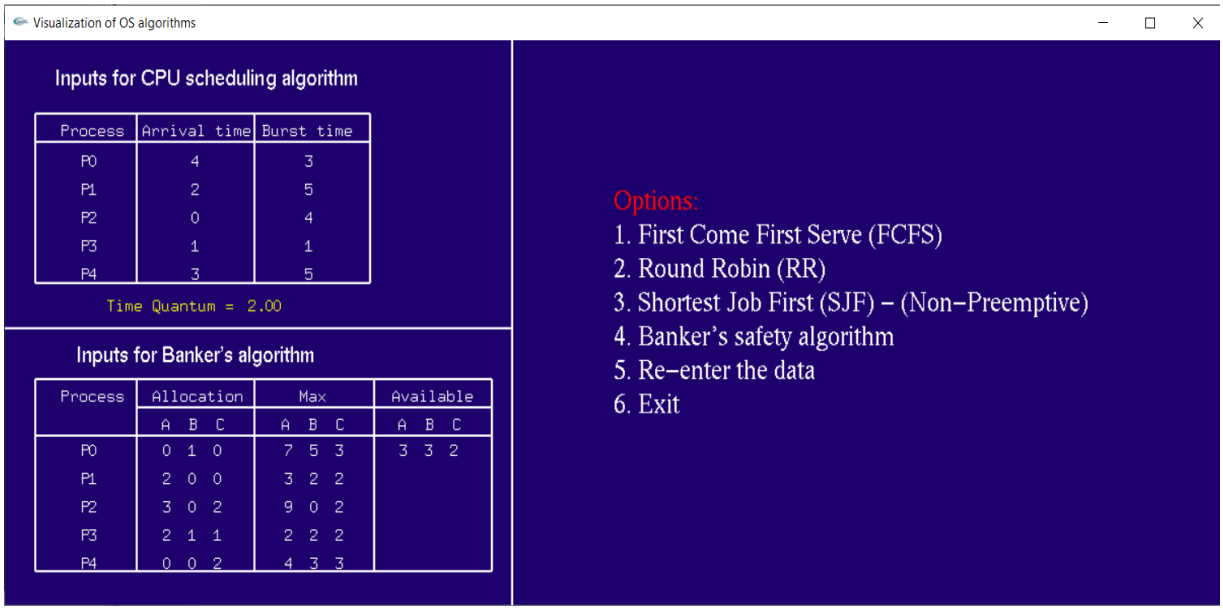

Figure 5: Option menu with table

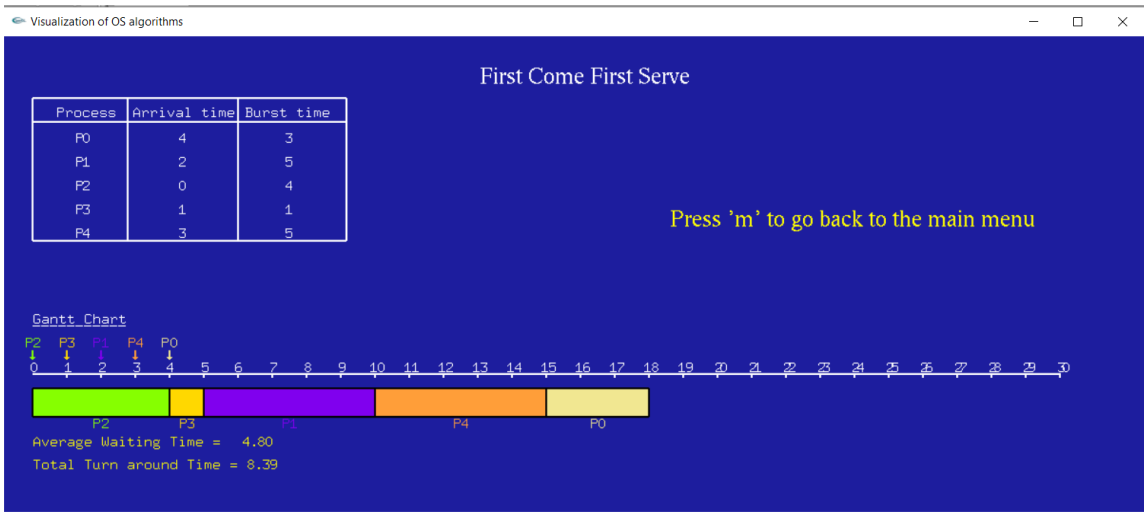

Figure 6: Visualization of FCFS scheduling

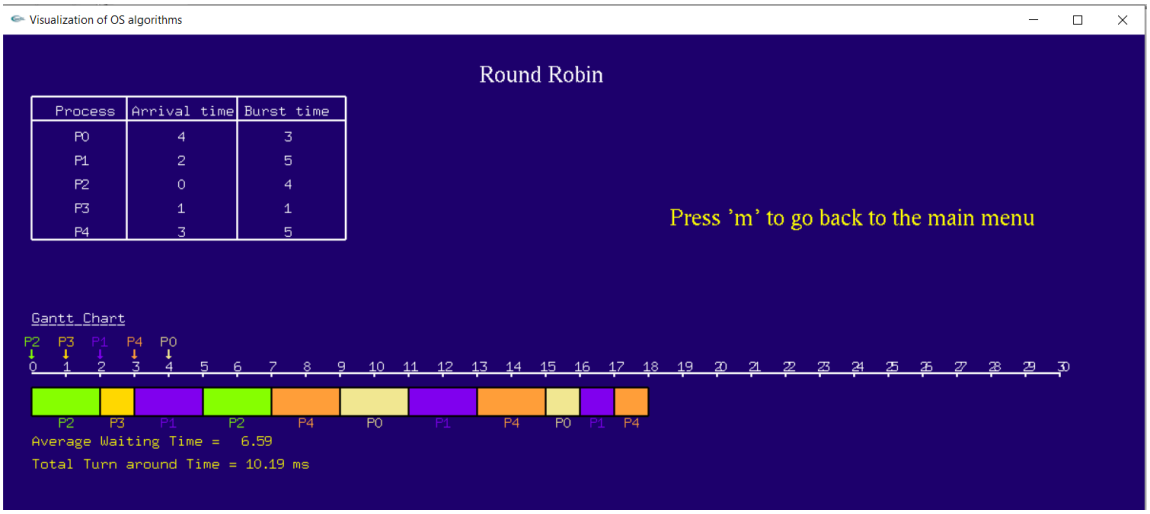

Figure 7: Visualization of Round Robin scheduling

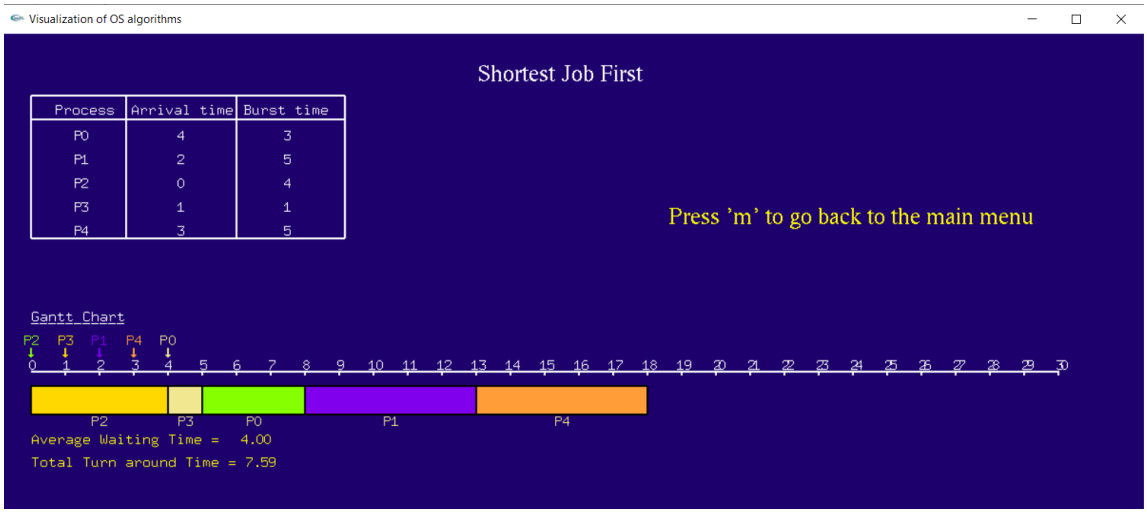

Figure 8: Visualization of SJF scheduling(Non-preemptive)

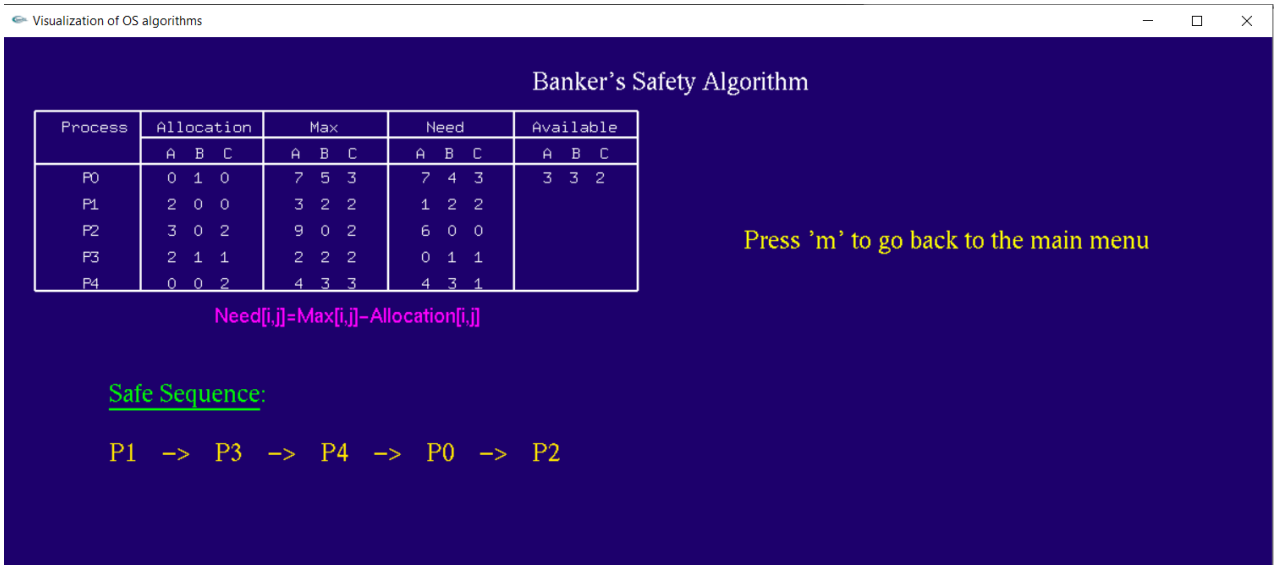

Figure 9: Banker's algorithm(when system is safe)

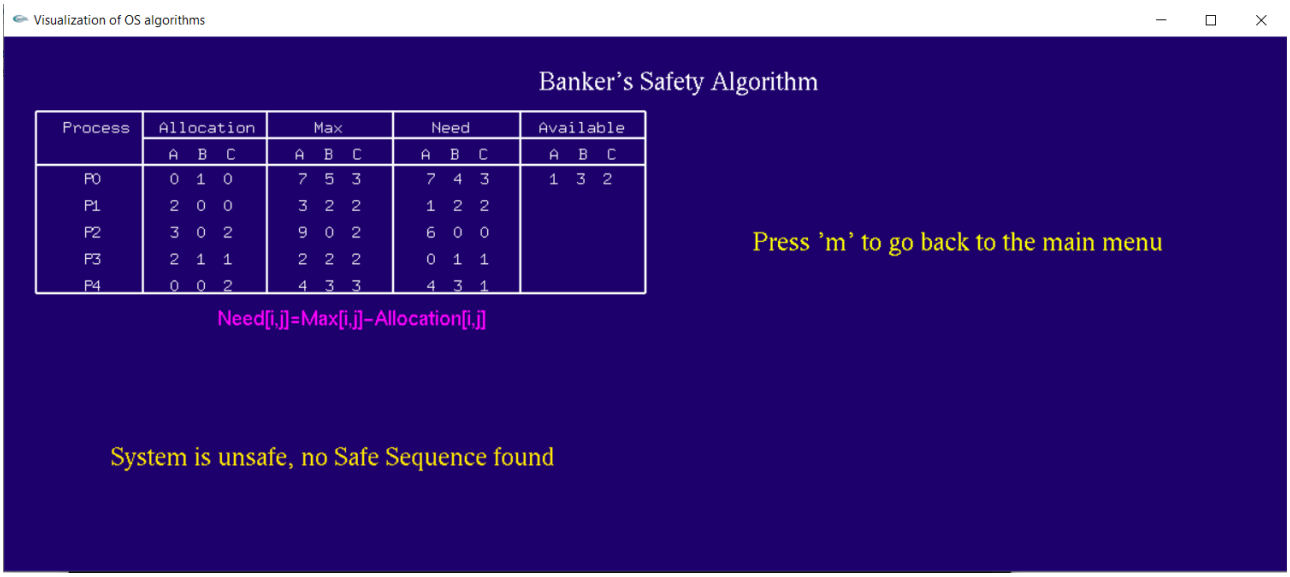

Figure 10: Banker's algorithm(when system is unsafe)