MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Operating System Project

To show the working of context switching in cpu scheduling

Team Members (Batch 1 Section B)

G. Bhavesh Reddy - 190905177 (Roll No. 1)

AppalaReddy Tarun Kumar Reddy - 200905028 (Roll No. 8)

Mayala Laxman - 200905064 (Roll No. 14)

Jampula Sai Rahul Chowdhary - 200905088 (Roll No. 19)

Pranai Rao Marru - 200905091 (Roll No. 21)

INTRODUCTION

This project is created to show the implementation of context switching in CPU scheduling. The process are allocated to CPU based on the arrival time and Round Robin algorithm to implement the same with a constant quantum time. When a process is switched out its context is saved in Process Control Block (PCB) and when the process turn arrives again the process is resumed with the last saved state in PCB. To show the working for context switching a set of custom instruction are implemented in a file which is considered as a process for a hypothetical hardware. A GUI is created using GTK Framework to show the PCB and allow user to allocate resources. This project is related to the CPU scheduling chapter in Operating System Concepts Ninth Edition by Avi Silberschatz, Peter Baer Galvin, Greg Gagne with emphasis on Context Switching and Process Control Block.

OS CONCEPTS USED

Five-state model

For handling processes five-state model is considered . States are New, Ready, Running, Blocked and Exit . Blocked and Ready States are Implemented through Queue. Ready Queue is implemented using circular Queue because process should remain in ready until it terminates or its execution gets completed. If process suffers from I/O interrupt then that Process is dequeued from Ready queue and enqueued to Blocked Queue. Running is not queue because we have considered that one process can run at a time. Whenever Resource is available It is removed from Blocked queue and enqueued to ready queue.

Scheduling

For managing Ready Queue, scheduling is done.We have considered short term scheduling. For this, Round Robin is chosen as scheduling algorithm and quantum=2 .Scheduling is done for the process present in ready queue. Quantum is chosen to be 2 to minimise the risk of starvation. Also if process is short than RR provides good response time.It gives fair treatment to all processes.

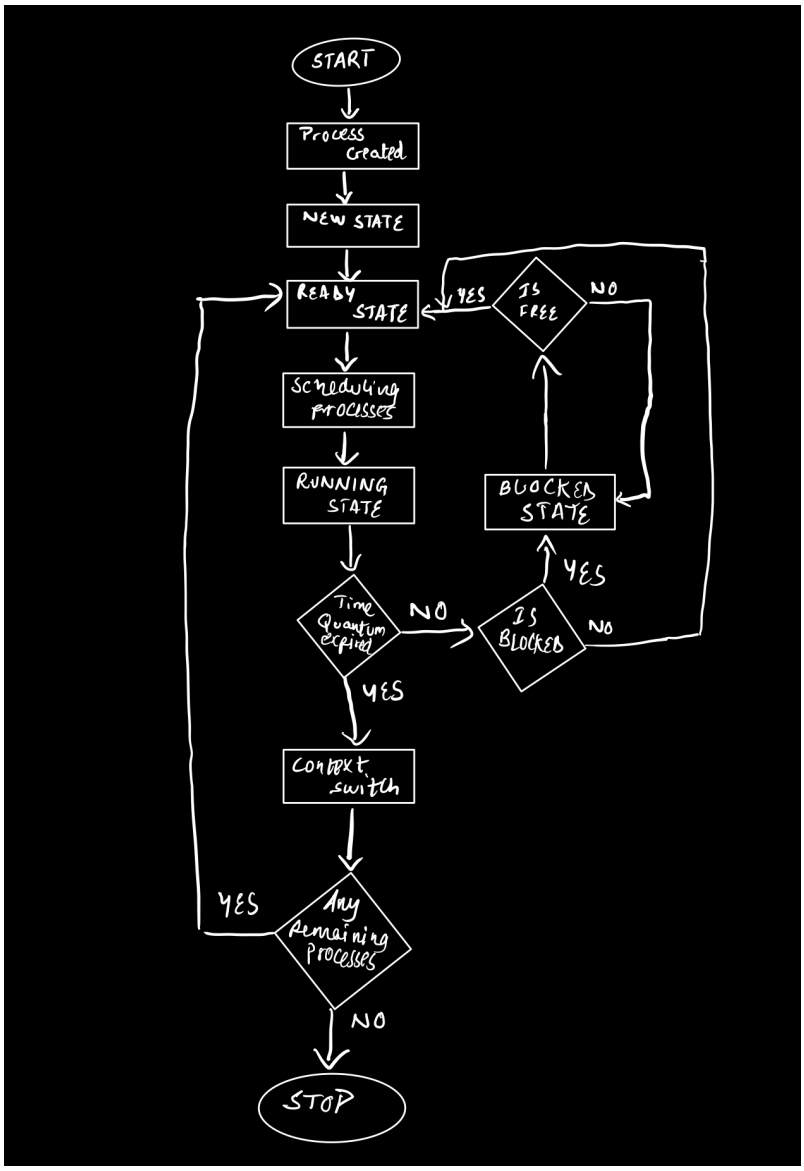
Context Switching:

When context switch occurs, for example if process runs for one time slice, but its execution is not completed, then whenever next time processor is allocated to it process must start from where it has left. For this purpose PCB is used. This stored data is the context of process.

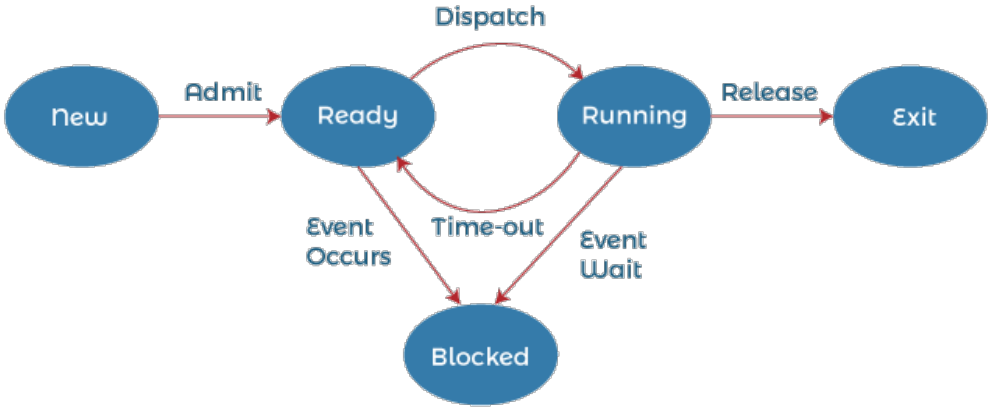
Process Control Block

Process has many elements. Out of which Program and code are essential. PCB contains crucial information needed for a process to execute.The Implemented PCB contains PID (process identifier), State (Describes in which state the process is), PC(Program Counter: it contains address of the next instruction which will be executed), SP(Stack Pointer : it is small register that stores the address of the last program request in a stack).

FLOWCHART



FIVE STATE MODEL ON WHICH PROJECT IS BASED



Fire State Process Model

CODE

MAIN FILE (PROJECT.C)

```c
#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>
#include <string.h>
#include "stack_implementation.h"
#include "queue_implementation.h"


bool res[4]={false,false,false,false};                  /*For
buttons in GTK*/
int count=0;
GtkWidget *pcb1;                                        /*To
display in GUI PCB state before execution*/
GtkWidget *pcb2;                                        /*PCB
state after execution*/

/*GUI button to lock resource*/
int occ_1(GtkWidget *widget,gpointer   data)
{
    g_print ("Resource 1 is occupied!\n");
    res[1]=true;
    return 0;
}

/*GUI button to release resource*/
int free_1(GtkWidget *widget,gpointer   data)
{
    g_print ("Resource 1 released!\n");
    res[1]=false;
    return 0;
}

int occ_2(GtkWidget *widget,gpointer   data)
{
    g_print ("Resource 2 is occupied!\n");
    res[2]=true;
    return 0;
}

/*GUI button to release resource*/
int free_2(GtkWidget *widget,gpointer   data)
{
    g_print ("Resource 2 released!\n");
```

```
    res[2]=false;
    return 0;
}

void updateLabel(GtkLabel *disp,int x,int y,char* z,void* w)
{
    gchar *display;
    /*Updates PCB*/
    display = g_strdup_printf("PID      :%d\nPC         :
%d\nState      :%s\nSP         :%p\n",x,y,z,w);         //
concate data to display
    gtk_label_set_text (GTK_LABEL(disp), display);      //
set label to "display"
    g_free(display);                                    //
free display
}

void updateL(GtkLabel *disp,int x,int y,char* z,void * w)
{
    gchar *display;
    display = g_strdup_printf("PID      :%d\nPC         :
%d\nState      :%s\nSP         :%p\n",x,y,z,w);         //
concate data to display
    gtk_label_set_text (GTK_LABEL(disp), display);      //
set label to "display"
    g_free(display);                                    //
free display
}
void* threadFunction(void* args)
{
    /*files of processes containing instructions*/
    static const char* filename[4];
    filename[0] = "process1.txt";
    filename[1] = "process2.txt";
    filename[2] = "process3.txt";
    filename[3] = "process4.txt";
    int size=4;

    /*Ready queue*/
    Queue *ready_queue = createQueue(size);

    /*Blocked queue*/
    Queue *blocked_queue = createQueue(size);

    /*Stack register*/
    struct stack_t stack_p[4];

    /*PID*/
    int process[] = { 0,1,2,3 };
```

```
    /*arrivaltime*/
    int arrivaltime[] = {0,0,0,0};
    int burst_time[4];
    int pc[]={0,0,0,0};
    int len[size];
    /*Stack pointer*/
    void*
sp[]={stackpointer(&(stack_p[0])),stackpointer(&(stack_p[1]))
),stackpointer(&(stack_p[2])),stackpointer(&(stack_p[3]))};
    int t[]={0,0,0,0};
    int l = 2, u = 7;
    /*State of process*/
    char* state[size];
    state[0] = "ready";
    state[1] = "ready";
    state[2] = "ready";
    state[3] = "ready";
    int tot_time=0;

    /*initial pc*/
    pc[0]=1000;
    for (int i = 0; i < size; i++)
    {
        burst_time[i] =2*( (rand() %(u - l + 1)) + l);
        len[i]=burst_time[i];

    }
    for (int i = 1; i < size; i++)
    {
        pc[i] = pc[i-1]+len[i-1];

    }


    /*total time (for RR)*/
    for (int i = 0; i < size; i++)
    {
        tot_time =tot_time+ burst_time[i];
    }

    /*print process description*/
    printf("process\tArrival time\t burst
time\tPC\tSize\n");
    for (int i = 0; i < size; i++)
    {
          printf("%d\t%d\t\t%d\t\t %d \t%d
\n",process[i],arrivaltime[i],burst_time[i],pc[i],len[i] );
```

```
    }

    /*Initially Add all processes to ready queue*/
    for (int i= 0; i<size; i++)
    {
        Enqueue(ready_queue,process[i]);
    }

    /*Implementation of RR scheduling and context switch*/

    for(int i= 0; i<(tot_time/2); i++)
    {
            int blocked=front(blocked_queue);
            if(res[blocked]==false && blocked!
=-1)          //check resource is released for blocked
process
            {
                Enqueue(ready_queue,process[blocked]);
                Dequeue(blocked_queue);
                state[blocked]="Ready";
            }
            else
            {
                if(blocked!=-1)
                {
                    Dequeue(blocked_queue);
                    Enqueue(blocked_queue,process[blocked]);
                }

            }
        int running;
        /*display ready and blocked queue*/
        printf("Ready ");
        display(ready_queue);
        printf("Blocked ");
        display(blocked_queue);
        running=Dequeue(ready_queue);           //Add
process for running

        /*For I/O process check if resource is available*/
        if(res[running]==false)
        {
            if(t[running]<burst_time[running])
            {
                state[running]="Running";
                /*print state before execution*/
```

```
printf("\n-----------------------------------------------------
-------------------------\n");
                printf("                  Before
execution\n");

printf("\n-----------------------------------------------------
-------------------------\n");

printf("Process\t\tPC\t\tState\t\t\tSP\n");
                for (int j = 0; j < size; j++)
                {

printf("%d\t\t%d\t\t%s\t\t\t%p\n",process[j],pc[j],state[j],
sp[j]);

                }
                /* update pcb value of process before
running */

updateLabel(GTK_LABEL(pcb1),process[running],pc[running],sta
te[running],sp[running]);

                    /* run process for quantum 2 */
                    for(int k=0;k<2;k++)
                    {
                        t[running]++;
                        pc[running]=pc[running]+1;
                        sleep(1);
                    }

                    /* open file of process to execute */
                    FILE *file = fopen(filename[running],
"r");
                    int count = 0;
                    if ( file != NULL )
                    {
                        char string1[1000][1000];
                        int ctr=0;
                        int q=0;
                        char line[256];

                        while (fgets(line, sizeof line,
file) != NULL) /* read a line */
                        {
                            if (count == t[running])
                            {
                                break;
                            }
```

```
                                else
                                {   //to read single word
                                    for(int
p=0;p<=(strlen(line));p++)
                                    {
                                        // if space or NULL
found, assign NULL into newString[ctr]
                                        if(line[p]==' '||
line[p]=='\0')
                                        {
                                            string1[ctr]
[q]='\0';
                                            ctr++;  //for next
word
                                            q=0;
                                        }
                                        else
                                        {
                                            string1[ctr]
[q]=line[p];
                                            q++;
                                        }
                                    }
                                    //check whether variable is
declared or not
                                    if(strcmp(string1[0],"add")!
=0||strcmp(string1[0],"sub")!=0||strcmp(string1[0],"div")!
=0||strcmp(string1[0],"mult")!=0)
                                    {

push(&(stack_p[running]),string1[1]);
                                    }
                                    else
                                    {

pop(&(stack_p[running]));
                                    }
                                    count++;
                                }
                            }
                            fclose(file);
                        }
                        state[running]="Ready";

sp[running]=stackpointer(&(stack_p[running]));
                        /*print state after execution*/

printf("\n-----------------------------------------------
------------------------\n");
```

```
                    printf("                After
execution\n");

printf("\n-----------------------------------------------------
-------------------------\n");

printf("Process\t\tPC\t\tState\t\t\tSP\n");
                    for (int i = 0; i < size; i++)
                    {

printf("%d\t\t%d\t\t%s\t\t\t%p\n",process[i],pc[i],state[i],
sp[i]);
                    }

                    /* update pcb value of process after
running */

updateL(GTK_LABEL(pcb2),process[running],pc[running],state[r
unning],sp[running]);
                    sleep(2);

                    /*check if Process is completed then
don't add it to ready queue*/
                    if(t[running]==burst_time[running]){
                        printf("process %d is
completed\n",process[running]);
                        state[running]="Ended";}

                    /*if Process is not completed then add
it to ready queue*/
                    else

Enqueue(ready_queue,process[running]);

            }
        }
        /*if resource is unavailable, add it to blocked
queue*/
        else{
            state[running]="Blocked";
            Enqueue(blocked_queue,process[running]);
            printf("process %d is
blocked\n",process[running]);
            i--;
        }
```

```
    }
}

/*main function*/
int main(int argc, char * argv[])
{
    /*declaration of variables For GUI*/
    pthread_t id;
    pthread_create(&id,NULL,&threadFunction,NULL);
    gtk_init (&argc, &argv);
    GtkWidget *window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);
    GtkWidget *grid;
    GtkWidget *button;
    GtkWidget *label;
    GtkWidget *l1;
    GtkWidget *l2;
    GtkWidget *l3;
    GtkWidget *l4;

    /*to show pcb data on gui screen*/
    l1 = gtk_label_new ("Before Execution:\n");
    l2 = gtk_label_new ("After Execution:\n");
    pcb1 = gtk_label_new ("PID      :-\nPC          :-
\nState       :-\nSP          :-\n");
    pcb2 = gtk_label_new ("PID      :-\nPC          :-
\nState       :-\nSP          :-\n");
    gtk_window_set_title (GTK_WINDOW (window), "context
switch");
    gtk_window_set_default_size (GTK_WINDOW (window), 200,
200);
    g_signal_connect (window, "destroy", G_CALLBACK
(gtk_main_quit), NULL);


    //create grid for alignment
    grid = gtk_grid_new ();

    //add grid to window
    gtk_container_add (GTK_CONTAINER (window), grid);

    button = gtk_button_new_with_label ("Resource 1
occupy");
    g_signal_connect (button, "clicked", G_CALLBACK (occ_1),
NULL);

    //attach buttons to grid
    gtk_grid_attach (GTK_GRID (grid), button, 1, 2, 1, 1);
```

```
    button = gtk_button_new_with_label ("Resource 1 
release");
    g_signal_connect (button, "clicked", G_CALLBACK 
(free_1), NULL);

    gtk_grid_attach (GTK_GRID (grid), button, 2, 2, 1, 1);

    button = gtk_button_new_with_label ("Resource 2 
occupy");
    g_signal_connect (button, "clicked", G_CALLBACK (occ_2), 
NULL);

    //attach buttons to grid
    gtk_grid_attach (GTK_GRID (grid), button, 1, 3, 1, 1);

    button = gtk_button_new_with_label ("Resource 2 
release");
    g_signal_connect (button, "clicked", G_CALLBACK 
(free_2), NULL);

    gtk_grid_attach (GTK_GRID (grid), button, 2, 3, 1, 1);

    gtk_grid_attach (GTK_GRID(grid),l1,1, 5, 1, 1);
    gtk_grid_attach (GTK_GRID(grid),pcb1,1, 6, 1, 1);
    gtk_grid_attach (GTK_GRID(grid),l2,1, 7, 1, 1);
    gtk_grid_attach (GTK_GRID(grid),pcb2,1, 8, 1, 1);

    gtk_widget_show_all (window);
    gtk_main ();


}
```

FUNCTIONS FOR STACK & QUEUE

stack.c

```
#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>
#include <string.h>


struct stack_entry {
```

```
  char *data;
  struct stack_entry *next;
};

struct stack_t{
  struct stack_entry *head;
  size_t stackSize;
};

/*Create a new stack instance*/


char *copyString(char *str);

void push(struct stack_t *theStack, char *value);

char *top(struct stack_t *theStack);

void * stackpointer(struct stack_t *theStack);

char* pop(struct stack_t *theStack);

struct stack_t *newStack(void)
{
  struct stack_t *stack = malloc(sizeof *stack);
  if (stack)
  {
    stack->head = NULL;
    stack->stackSize = 0;
  }
  return stack;
};
char *copyString(char *str)
{
  char *tmp = malloc(strlen(str) + 1);
  if (tmp)
    strcpy(tmp, str);
  return tmp;
}

/* Push a value onto the stack */
void push(struct stack_t *theStack, char *value)
{
  struct stack_entry *entry = malloc(sizeof *entry);
  if (entry)
  {
    entry->data = copyString(value);
    entry->next = theStack->head;
    theStack->head = entry;
```

```
    theStack->stackSize++;
  }
  else
  {
    // handle error here
    printf("stack full\n");
  }
}

/* Get the value at the top of the stack*/
char *top(struct stack_t *theStack)
{
  if (theStack && theStack->head)
    return theStack->head->data;
  else
    return NULL;
}
/*Returns pointer to the top of element*/
void * stackpointer(struct stack_t *theStack)
{
  if (theStack && theStack->head)
    return theStack->head;
  else
    return NULL;
}

/* Pop the top element from the stack*/
char* pop(struct stack_t *theStack)
{
  if (theStack->head != NULL)
  {
    struct stack_entry *tmp = theStack->head;
    theStack->head = theStack->head->next;
  return theStack->head->data;
    free(tmp->data);
    free(tmp);
    theStack->stackSize--;
  }
}
```

queue.c

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Queue
{
        int capacity;
```

```
        int size;
        int front;
        int rear;
        int *elements;
        int* data;
        int s;
}Queue;

Queue * createQueue(int maxElements);
void Enqueue(Queue *Q,int element);
int Dequeue(Queue *Q);
int front(Queue *Q);
int display(Queue *Q);
int search(Queue *Q,int element);
int arr[4];

Queue * createQueue(int maxElements)
{
        /* Create a Queue */
        Queue *Q;
        Q = (Queue *)malloc(sizeof(Queue));
        /* Initialise its properties */
        Q->elements = (int
*)malloc(sizeof(int)*maxElements);
        Q->size = maxElements;
        Q->s=0;
        Q->front = -1;
        Q->rear = -1;
        /* Return the pointer */
        return Q;
}


void Enqueue(Queue *Q,int element)
{
    if ((Q->front == 0 && Q->rear == Q->size-1) ||
            (Q->rear == (Q->front-1)%(Q->size-1)))
    {
        printf("Queue is Full");
        return;
    }

    else if (Q->front == -1) /* Insert First Element */
    {
        Q->front = 0;
        Q->rear = 0;
        Q->elements[Q->rear] = element;
        Q->s++;
    }
```

```
    else if (Q->rear == Q->size-1 && Q->front != 0)
    {
        Q->rear = 0;
        Q->elements[Q->rear] = element;
        Q->s++;
    }

    else
    {
        Q->rear++;
        Q->elements[Q->rear] = element;
        Q->s++;
    }
    return;
}
```

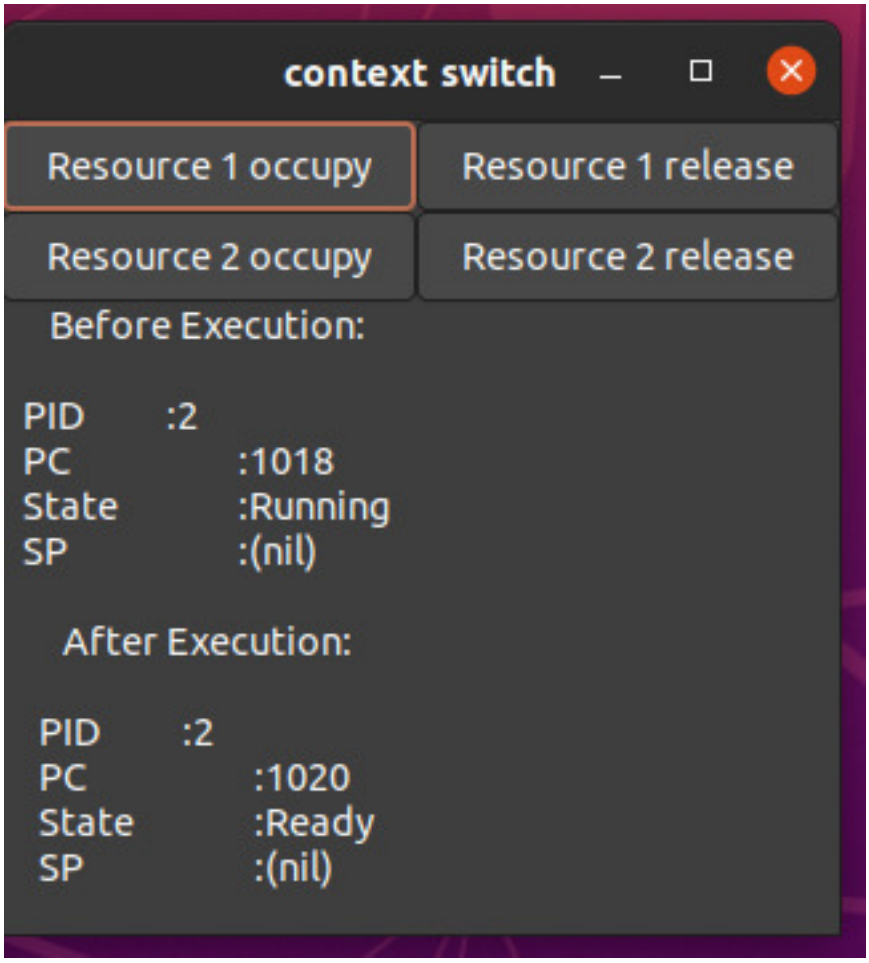
```
int Dequeue(Queue *Q)
{
    if (Q->front == -1)
    {
        //printf("\nQueue is Empty");
        return -1;
    }
    int data=Q->elements[Q->front];
    Q->elements[Q->front] = -1;
    if (Q->front == Q->rear)
    {
        Q->front = -1;
        Q->rear = -1;
        Q->s--;
    }
    else if (Q->front == Q->size-1){
        Q->front = 0;
        Q->s--;
    }
    else{
        Q->front++;
        Q->s--;
    }
    return data;
}

int front(Queue *Q)
{
        if(Q->front==-1)
```

```
        {
                //printf("Queue is Empty\n");
                return -1;//exit(0);
        }
        /* Return the element which is at the front*/
        return Q->elements[Q->front];
}
int display(Queue *Q)
{
    int arr[4];
    int r=0;
        if(Q->front==-1)
        {
                printf("Queue is Empty\n");

                //exit(0);
        }
        else
        {

                if(Q->rear>=Q->front)
                {
                    printf("Queue is:\n");
                    for(int y=(Q->front);y<=(Q->rear);y++)
                    {
                    printf("%d ",Q->elements[y]);
                    arr[r]=Q->elements[y];
                     //set label to "display"
                    //g_free(display);
                    }
                }
                else
                {
                    printf("Queue is:\n");
                    for(int y=(Q->front);y<(Q->size);y++)
                    {
                    printf("%d ",Q->elements[y]);
                    arr[r]=Q->elements[y];
                    r++;
                    }
                    for(int y=0;y<= (Q->rear);y++)
                    {
                    printf("%d ",Q->elements[y]);
                    arr[r]=Q->elements[y];
                    r++;
                    }
                }

                for(int y=0;y< 4;y++)
```

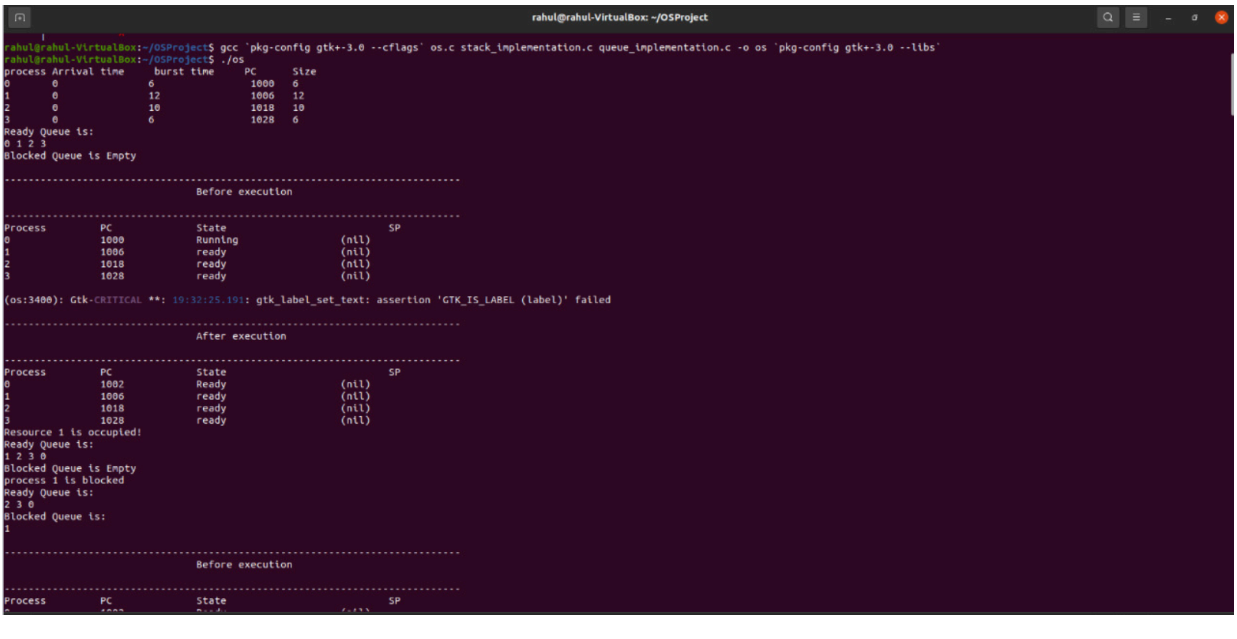
```
                    {
                    if(arr[y]==0 || arr[y]==1 ||arr[y]==2 ||
arr[y]==3){}
                    else
                        arr[y]= -1 ;
                    }
                printf("\n");
        }
}

int search(Queue *Q,int element)
{
        if(Q->size==0)
        {
                printf("Queue is Empty\n");

        }
        else
        {
                int j;
                for(j=(Q->front);j<= (Q->rear);j++)
                {
                    if(Q->elements[j]==element)
                        return 1;
                }
        }
}
```

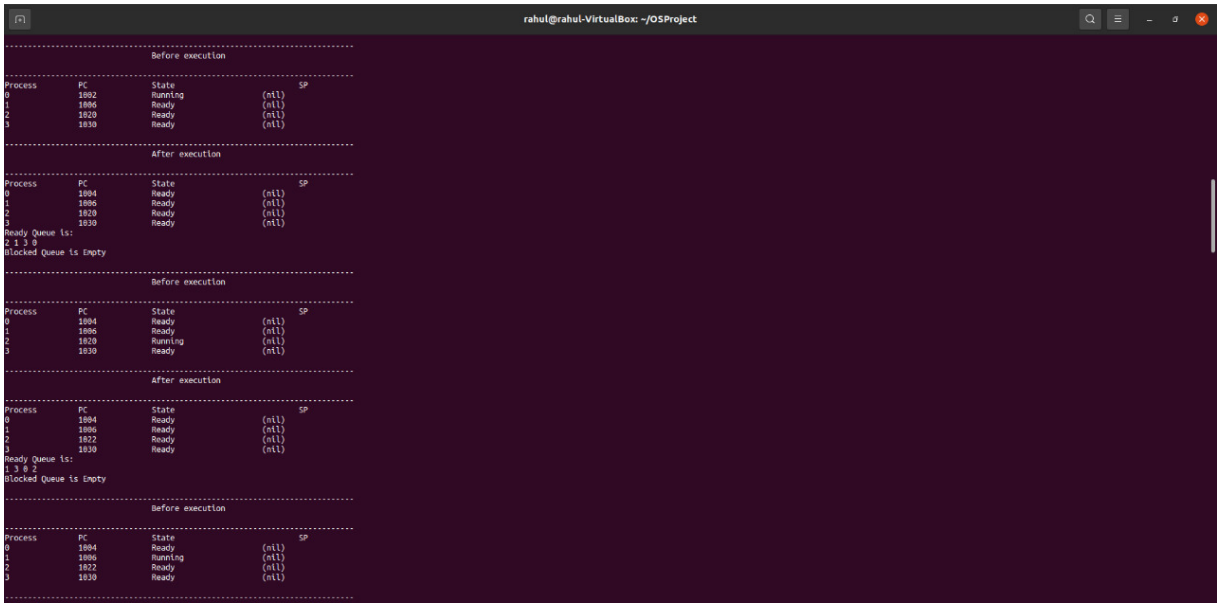
OUTPUT

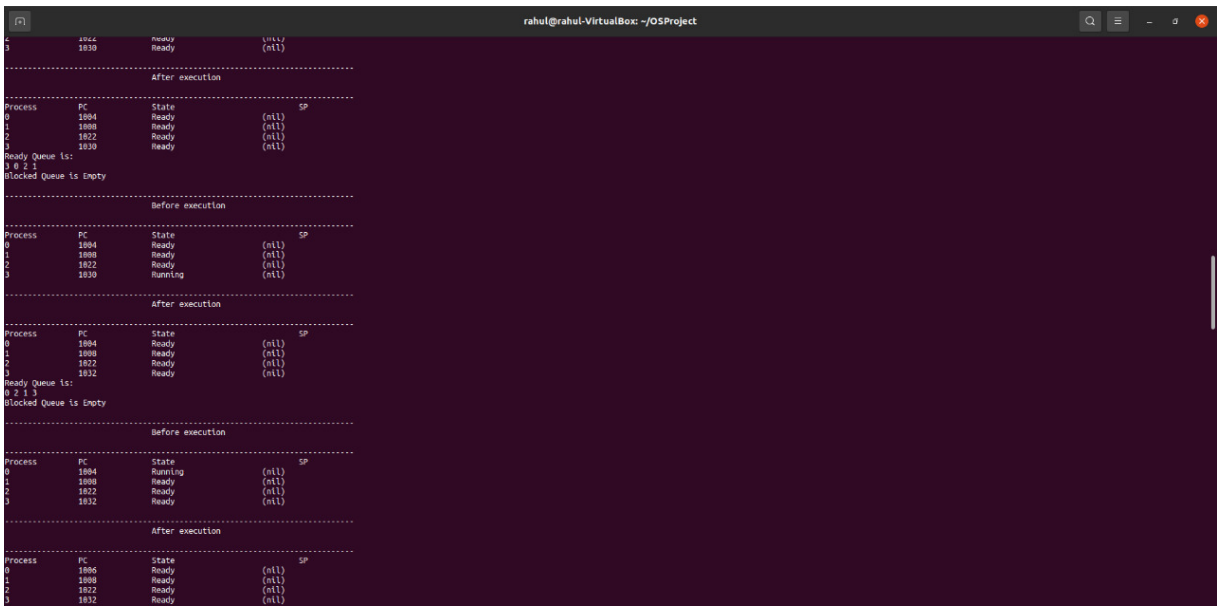
GUI-graphical user interface

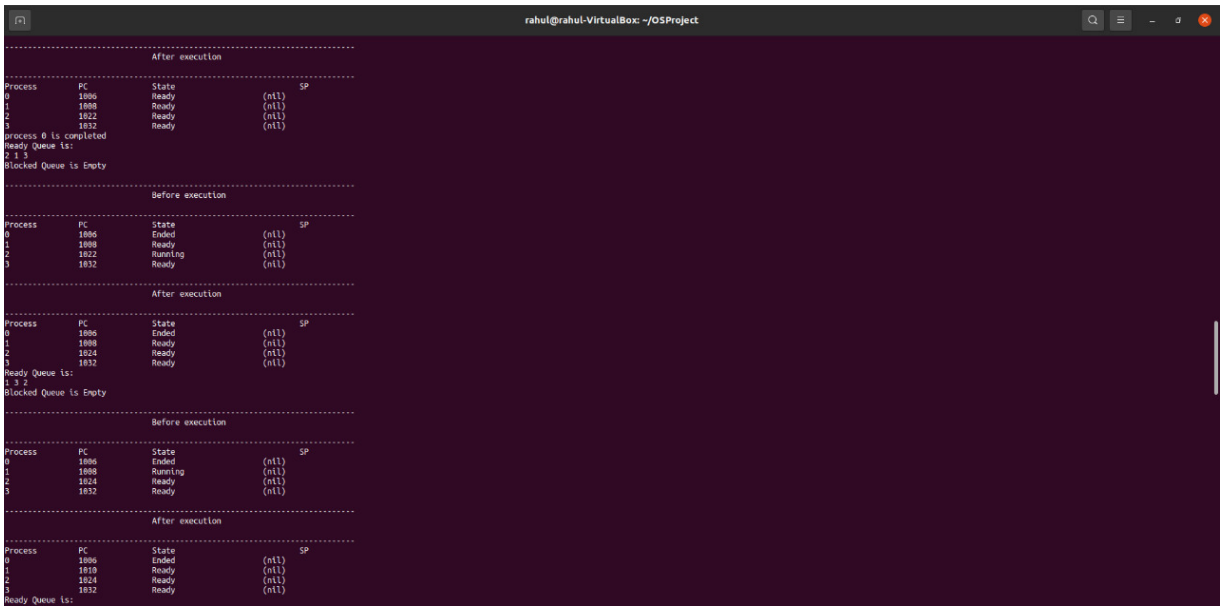


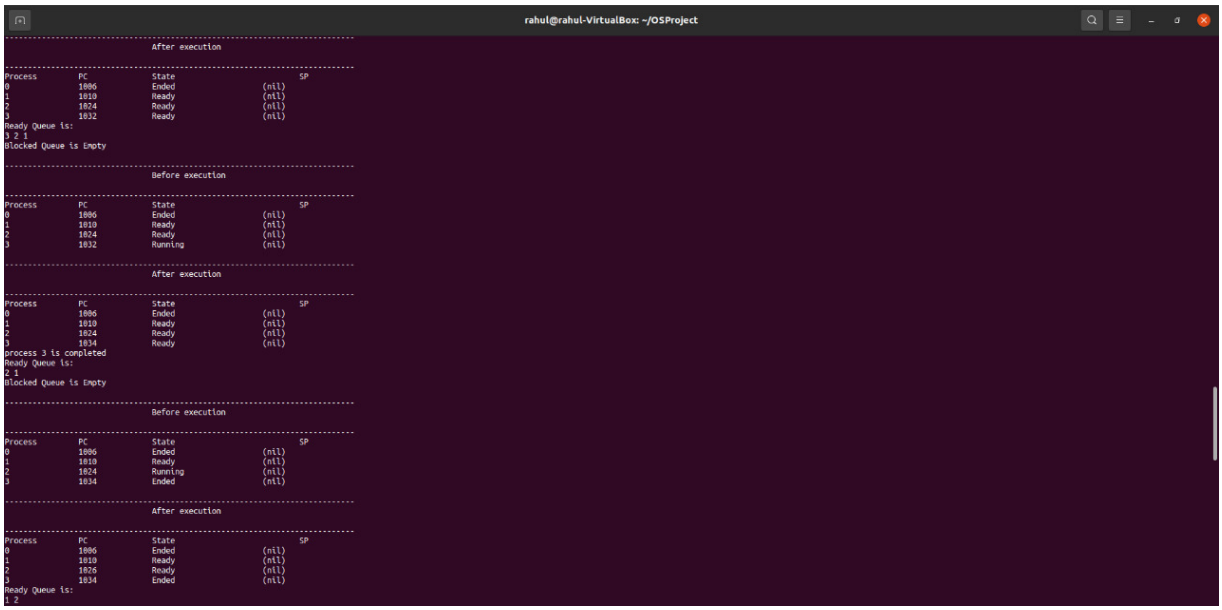
Scheduler Output

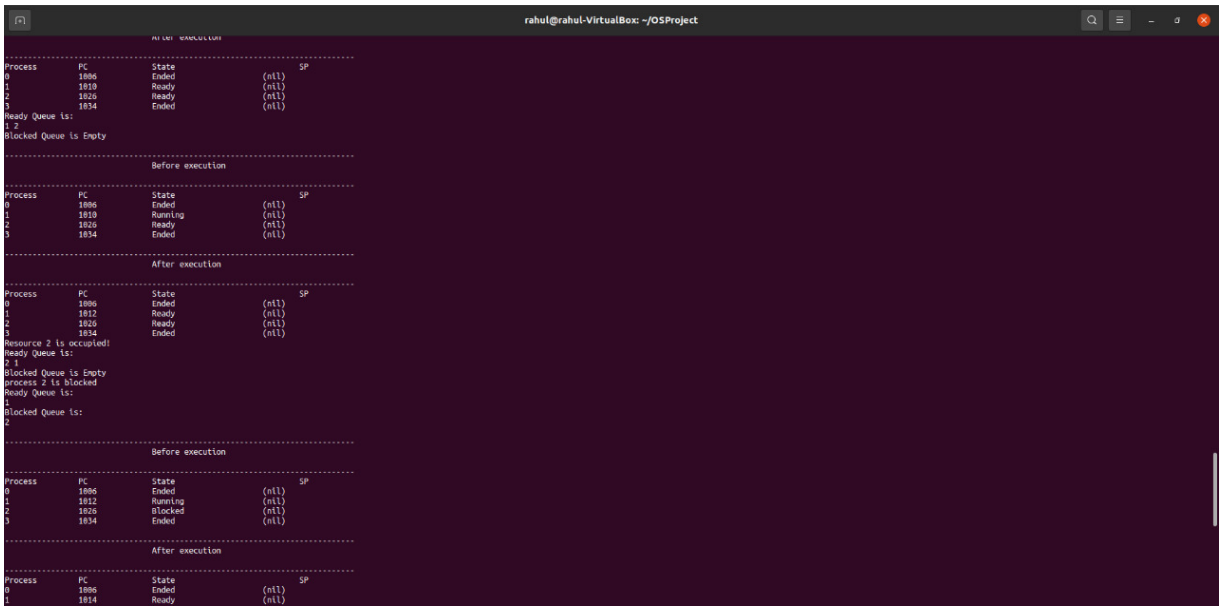


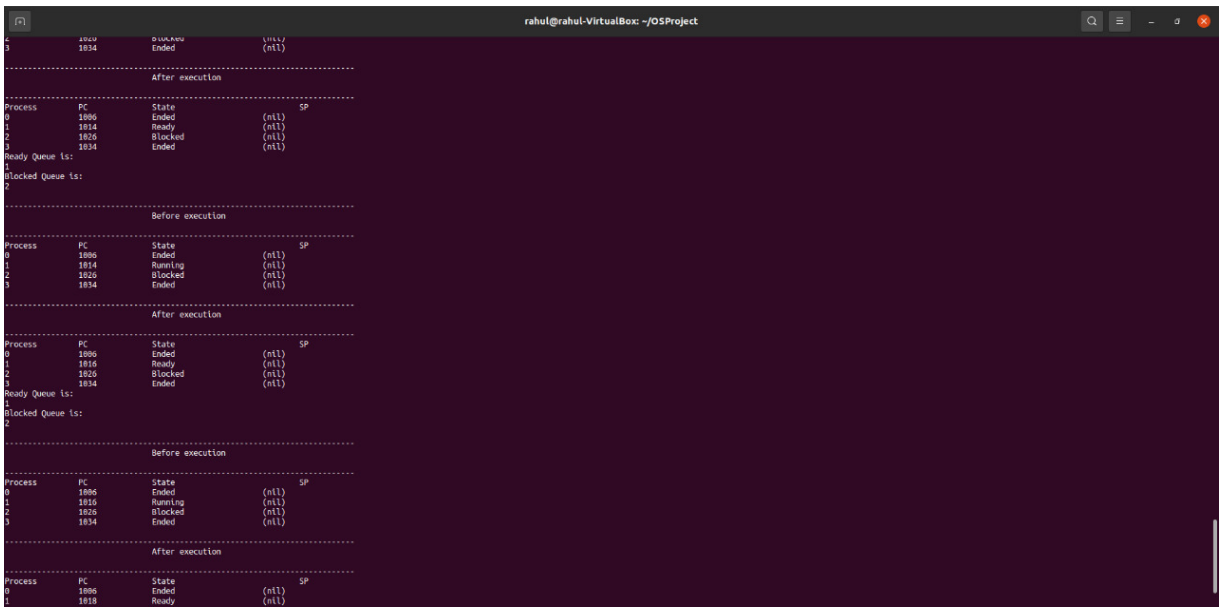

rahul@rahul-VirtualBox: ~/OSProject
Before execution
Process PC State SP
0 1002 Running (nil)
1 1006 Ready (nil)
2 1020 Ready (nil)
3 1030 Ready (nil)
After execution
Process PC State SP
0 1004 Ready (nil)
1 1006 Ready (nil)
2 1020 Ready (nil)
3 1030 Ready (nil)
Ready Queue is:
2 1 3 0
Blocked Queue is Empty
Before execution
Process PC State SP
0 1004 Ready (nil)
1 1006 Ready (nil)
2 1020 Running (nil)
3 1030 Ready (nil)
After execution
Process PC State SP
0 1004 Ready (nil)
1 1006 Ready (nil)
2 1022 Ready (nil)
3 1030 Ready (nil)
Ready Queue is:
1 3 0 2
Blocked Queue is Empty
Before execution
Process PC State SP
0 1004 Ready (nil)
1 1006 Running (nil)
2 1022 Ready (nil)
3 1030 Ready (nil)

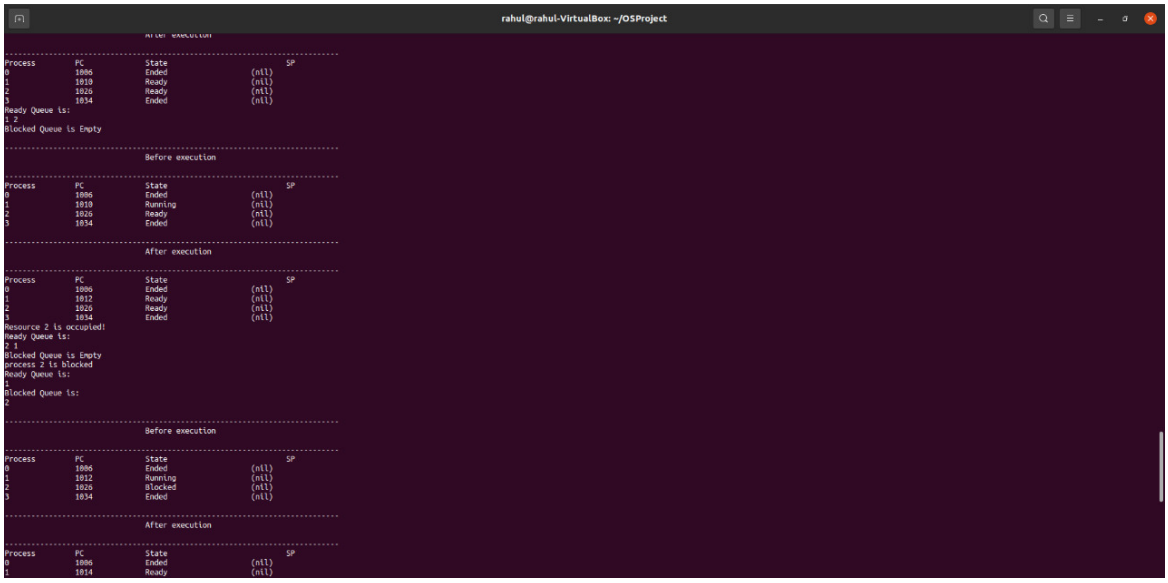

rahul@rahul-VirtualBox: ~/OSProject
3 1030 Ready (nil)
After execution
Process PC State SP
0 1004 Ready (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1030 Ready (nil)
Ready Queue is:
3 0 2 1
Blocked Queue is Empty
Before execution
Process PC State SP
0 1004 Ready (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1030 Running (nil)
After execution
Process PC State SP
0 1004 Ready (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1032 Ready (nil)
Ready Queue is:
0 2 1 3
Blocked Queue is Empty
Before execution
Process PC State SP
0 1004 Running (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1032 Ready (nil)
After execution
Process PC State SP
0 1006 Ready (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1032 Ready (nil)

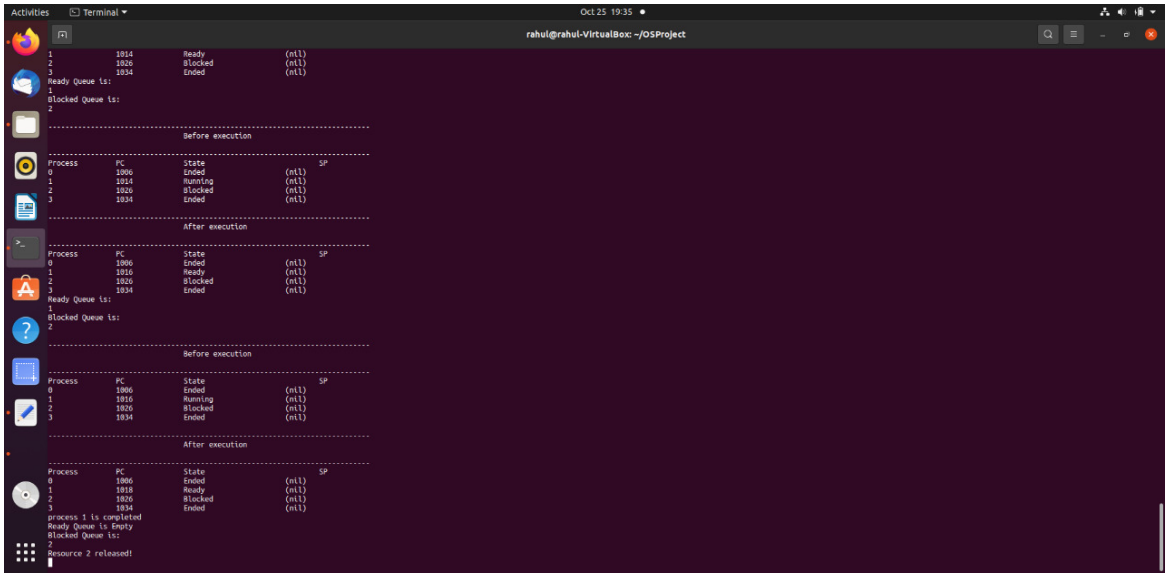

rahul@rahul-VirtualBox: ~/OSProject
After execution
Process PC State SP
0 1006 Ready (nil)
1 1008 Ready (nil)
2 1022 Ready (nil)
3 1032 Ready (nil)
process 0 is completed
Ready Queue is:
2 1 3
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1008 Ready (nil)
2 1022 Running (nil)
3 1032 Ready (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1008 Ready (nil)
2 1024 Ready (nil)
3 1032 Ready (nil)
Ready Queue is:
1 3 2
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1008 Running (nil)
2 1024 Ready (nil)
3 1032 Ready (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1024 Ready (nil)
3 1032 Ready (nil)
Ready Queue is:

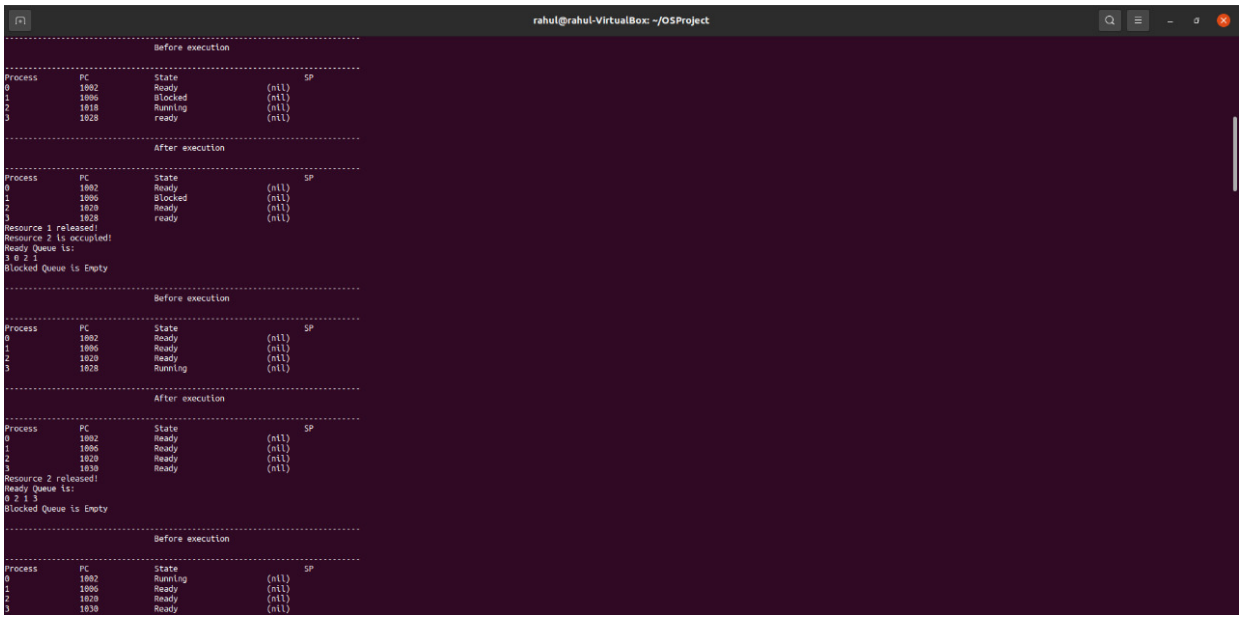

rahul@rahul-VirtualBox: ~/OSProject
After execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1024 Ready (nil)
3 1032 Ready (nil)
Ready Queue is:
3 2 1
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1024 Ready (nil)
3 1032 Running (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1024 Ready (nil)
3 1034 Ready (nil)
process 3 is completed
Ready Queue is:
2 1
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1024 Running (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
Ready Queue is:
1 2


rahul@rahul-VirtualBox: ~/OSProject
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
Ready Queue is:
1 2
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Running (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1012 Ready (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
Resource 2 is occupied!
Ready Queue is:
2 1
Blocked Queue is Empty
process 2 is blocked
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1012 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1014 Ready (nil)


rahul@rahul-VirtualBox: ~/OSProject
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1014 Ready (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1014 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1016 Ready (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1016 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1018 Ready (nil)


rahul@rahul-VirtualBox: ~/OSProject
After execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Ready (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
Ready Queue is:
1 2
Blocked Queue is Empty
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1010 Running (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1012 Ready (nil)
2 1026 Ready (nil)
3 1034 Ended (nil)
Resource 2 is occupied!
Ready Queue is:
2 1
Blocked Queue is Empty
process 2 is blocked
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1012 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1014 Ready (nil)


Activities
Terminal
Oct 25 19:35
rahul@rahul-VirtualBox: ~/OSProject
1 1014 Ready (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1014 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1016 Ready (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
Ready Queue is:
1
Blocked Queue is:
2
Before execution
Process PC State SP
0 1006 Ended (nil)
1 1016 Running (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
After execution
Process PC State SP
0 1006 Ended (nil)
1 1018 Ready (nil)
2 1026 Blocked (nil)
3 1034 Ended (nil)
process 1 is completed
Ready Queue is Empty
Blocked Queue is:
2
Resource 2 released!


rahul@rahul-VirtualBox: ~/OSProject
Before execution
Process PC State SP
0 1002 Ready (nil)
1 1006 Blocked (nil)
2 1018 Running (nil)
3 1028 ready (nil)
After execution
Process PC State SP
0 1002 Ready (nil)
1 1006 Blocked (nil)
2 1020 Ready (nil)
3 1028 ready (nil)
Resource 1 released!
Resource 2 is occupied!
Ready Queue is:
3 0 2 1
Blocked Queue is Empty
Before execution
Process PC State SP
0 1002 Ready (nil)
1 1006 Ready (nil)
2 1020 Ready (nil)
3 1028 Running (nil)
After execution
Process PC State SP
0 1002 Ready (nil)
1 1006 Ready (nil)
2 1020 Ready (nil)
3 1030 Ready (nil)
Resource 2 released!
Ready Queue is:
0 2 1 3
Blocked Queue is Empty
Before execution
Process PC State SP
0 1002 Running (nil)
1 1006 Ready (nil)
2 1020 Ready (nil)
3 1030 Ready (nil)

PROCESSES

Each process file has custom instruction used by a hypothetical processor

PROCESS 1

```
a 2
b 3
add x a b
a 5
b 6
add y a b
```

PROCESS 2

```
a 7
b 3
div x a b
a 9
b 6
div y a b
a 6
b 4
div x a b
a 5
b 6
div y a b
```

PROCESS 3

```
a 2
b 3
read file1
a 5
b 6
read file2
x 2
y 3
read file3
y 0
```

PROCESS 4

```
a 2
```

```
b 1
mult x a b
a 1
b 6
mult y a b
```

EXPLANATION

Four processes are added to four different text files and in which instruction for process is given.Ready queue is formed using circular queue. Now scheduling is done, for every quantum when process runs, its PC is incremented, completed time for particular running process increases by quantum, 2 instructions are executed in one quantum and value of variables are PUSH-ed in stack of that process. Whenever that process again gets processor to execute, value of this registers is used. After this, next process which is in ready queue gets turn and execute instructions in similar way. This will continue until any of the process gets blocked. Whenever any process gets blocked, it is added to block queue and processor is given to next process. When needed resource for blocked process is free/available, it is again added to ready queue. As a result we are showing before and updated PCB of each process. Resources are blocked and released through GUI.

PROGRAM EXECUTION

To run this program GTK library has to be installed which can be installed in linux by the following command -

```
sudo apt install libgtk-3-dev
```

To compile the program the following command is used -

```
gcc `pkg-config gtk+-3.0 --cflags` os.c stack_implementation.c queue_implementation.c -o os `pkg-config gtk+-3.0 —libs`
```

To execute the command is used -

```
./os
```

The main outcome of the project was to show the internal working of the dispatcher and process scheduler with emphasis on Context Switching and PCB (Process Control Block).