SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
SCHOOL OF COMPUTING
DEPARTMENT OF DATASCIENCE AND BUSINESS SYSTEMS
21CSC202J OPERATING SYSTEMS

MINI PROJECT REPORT

CONTEXT SWITCHING

Name: Vamshi Gadde

Register Number: RA2112704010017

Mail ID: vd5323@srmist.edu.in

Department: M Tech integrated computer science

Specialization: Data science

Semester: 3

Team Members

Name: Gaurav Saha **Registration Number: RA2112704010004**

Table of contents

ABSTRACT

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

A diagram that demonstrates context switching is as follows −

In the above diagram, initially Process 1 is running. Process 1 is switched out and Process 2 is switched in because of an interrupt or a system call. Context switching involves saving the state of Process 1 into PCB1 and loading the state of process 2 from PCB2. After some time again a context switch occurs and Process 2 is switched out and Process 1 is switched in again. This involves saving the state of Process 2 into PCB2 and loading the state of process 1 from PCB1.

CHAPTER 1

INTRODUCTION

We have implemented Context Switching and RR scheduling. We need to do Context Switching whenever a Process switch takes place. Here Scheduling is done, so whenever the time slice of one process is completed, the processor is allocated to the next process, for this we need to save the state of the process so that next time it should run from where it was left. We have implemented a context switch for Scheduling and I/O interrupts.

Context switching refers to a technique/method used by the OS to switch processes from a given state to another one for the execution of its function using the CPUs present in the system. When switching is performed in the system, the old running process's status is stored as registers, and the CPU is assigned to a new process for the execution of its tasks. While new processes are running in a system, the previous ones must wait in the ready queue. The old process's execution begins at that particular point at which another process happened to stop it. It describes the features of a multitasking OS where multiple processes share a similar CPU to perform various tasks without the requirement for further processors in the given system.

Context switching helps in sharing a single CPU among all processes so as to complete the execution and store the status of the system's tasks. Whenever the process reloads in a system, its execution begins at the very point in which there is conflict.

- The switching of a given process to another one isn't directly in the system. Context switching helps an OS switch between multiple processes to use the resources of the CPU for accomplishing its tasks and storing its context. The service of a process can be resumed at the same point later on. In case we don't store the data or context of the currently running process, this stored info may be lost when switching between the given processes.
- In case a high-priority process is falling in a ready queue, the process running currently would be shut down/stopped with the help of a high-priority process for completing its tasks in a system.
- In case a running process needs various I/O resources in a system, another process will switch the current process if it wants to use the CPU. And when it meets the I/O requirements, the previous process would go into a ready state so that it can wait for the CPU execution. Context switching helps in storing the process's state to resume

the tasks in an OS. Else, the process has to restart the execution from the very initial levels.

Examples of Context Switching:

Suppose that numerous processes get stored in a PCB (Process Control Block), and a process is in its running state for the execution of its task using the CPUs. As this process runs, another one arrives in the ready queue, which has a comparatively higher priority of completing its assigned task using the system's CPU. In this case, we used context switching such that it switches the currently running process with the new one that requires the CPU to finish its assigned tasks. When a process is switching, the context switch saves the old process's status in registers. Whenever any process reloads into a CPU, it initiates the process execution in which the new process intercepts the old process. In case we don't save the process's state, we have to begin its execution at its initial level. This way, context switching helps an OS to switch between the given processes and reload or store the process.

Context Switching Triggers:

Here are the triggers that lead to context switches in a system:

Interrupts: The CPU requests the data to be read from a disk. In case there are interrupts, the context switching would automatically switch a part of the hardware that needs less time to handle the interrupts.

Multitasking: Context switching is the characteristic of multitasking. They allow a process to switch from the CPU to allow another process to run. When switching a given process, the old state gets saved so as to resume the execution of the process at the very same point in a system.

Kernel/User Switch: It's used in the OS when it is switching between the kernel mode and the user mode.

CHAPTER 2

REVIEW OF EXISTING METHODS AND THEIR LIMITATIONS

Five-state model:

For handling processes we have considered five-state model. States are Blocked, Running and Ready. 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. We have considered that 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).

CHAPTER 3

PROPOSED METHOD WITH SYSTEM ARCHITECTURE / FLOW DIAGRAM

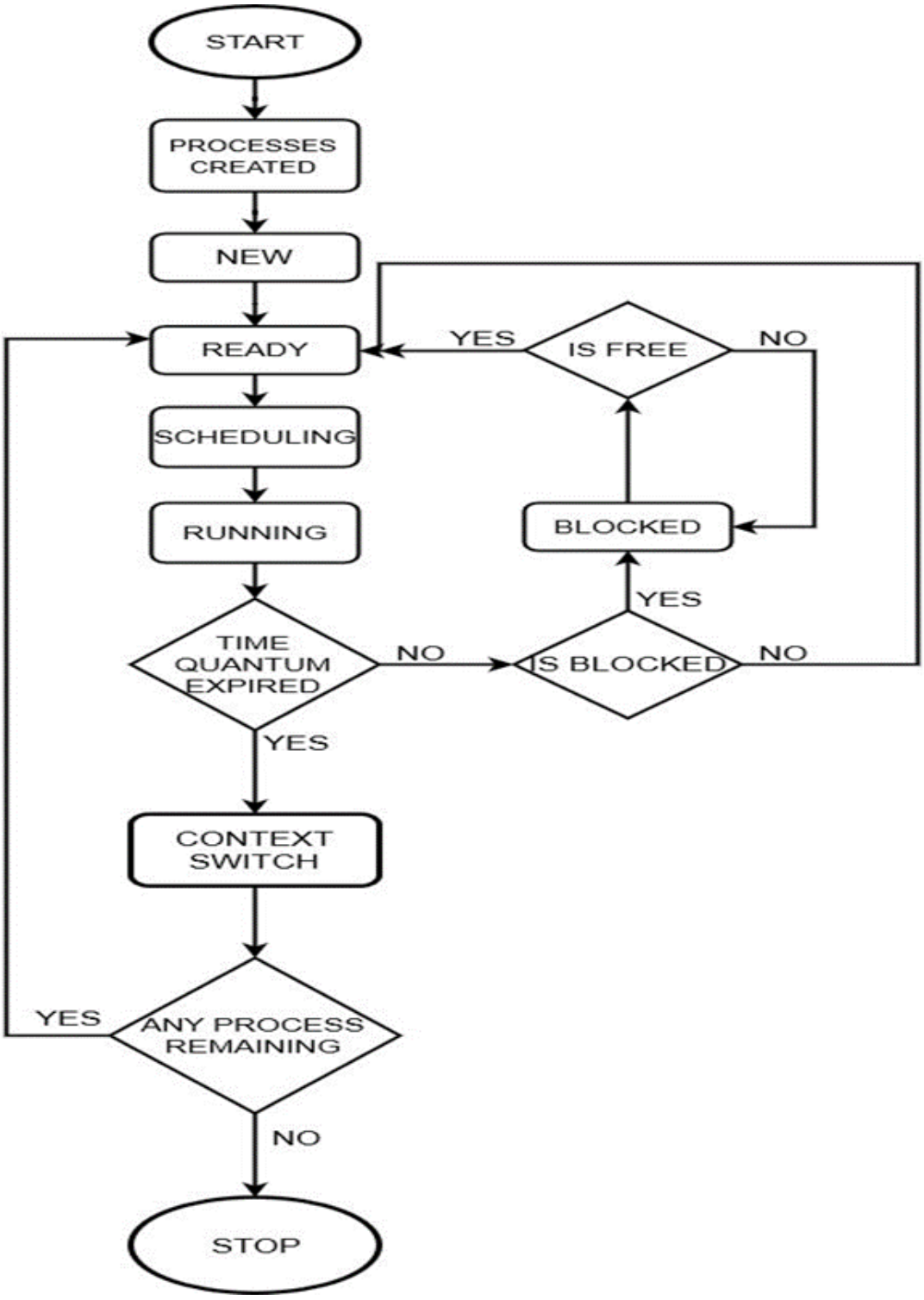

Fig 2: FLOW CHART

Model:

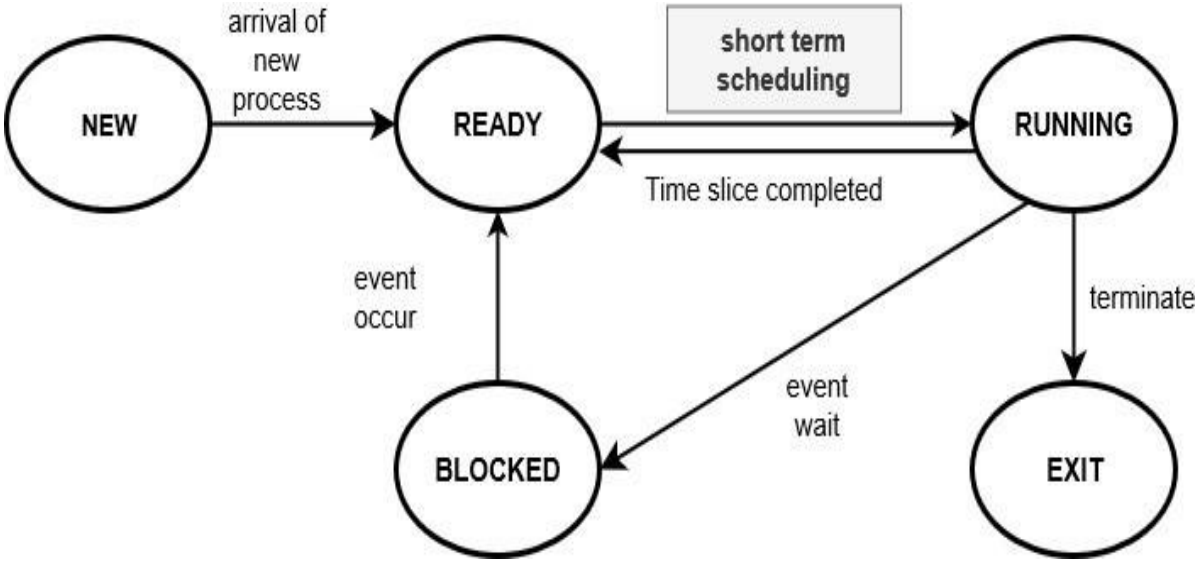

Fig 2: Five state model for context switching

CHAPTER 4

MODULES DESCRIPTION

Main.c It contains all operations such as process creation, scheduling ,context switch,updating PCB and GUI,which is implemented using GTK.

To Run program:

Compile:

gcc `pkg-config --cflags gtk+-3.0` -o gui gui.c `pkg-config --libs gtk+-3.0`

Execute:

./gui

This code can run in linux. Only main.c needs to be runned.

You need to **install GTK3.0** to run this.

CHAPTER 5

IMPLEMENTATION REQUIREMENTS

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 pro- cess. 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 o ready queue.

As a result, we are showing before and updated PCB of each process. Resources are blocked and released through GUI.

CHAPTER 6

OUTPUT SCREENSHOTS

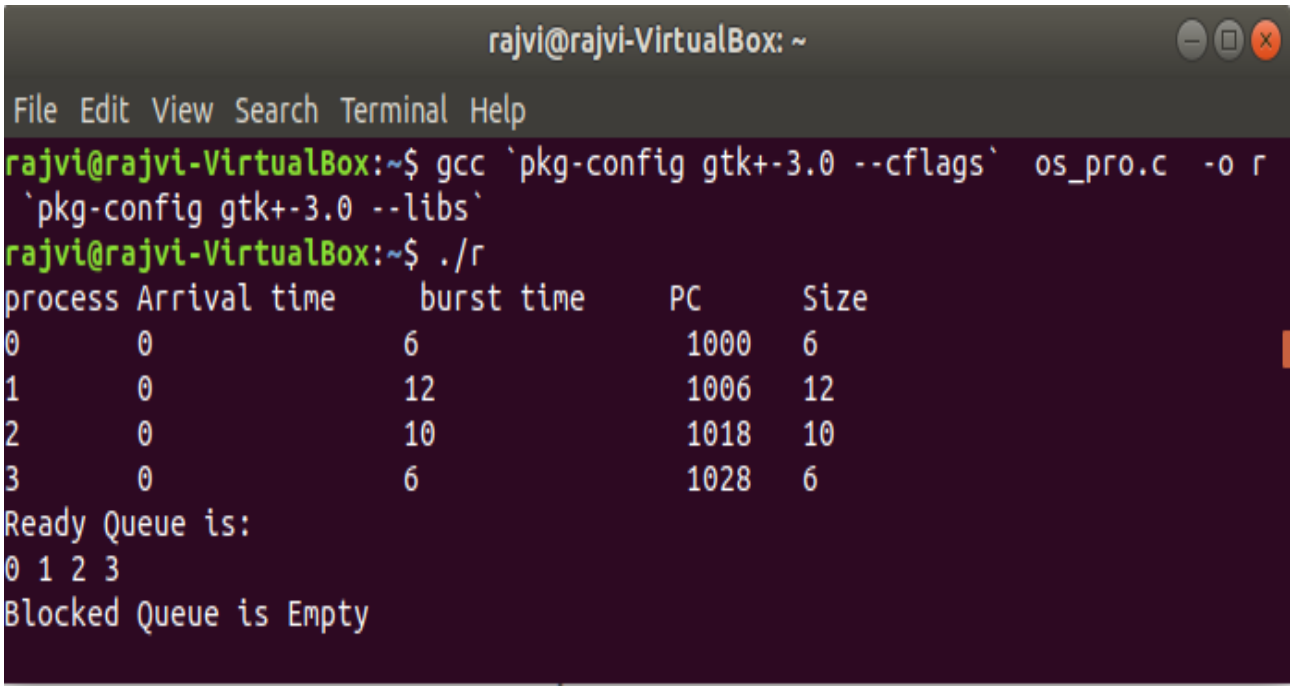

Fig 3: process block

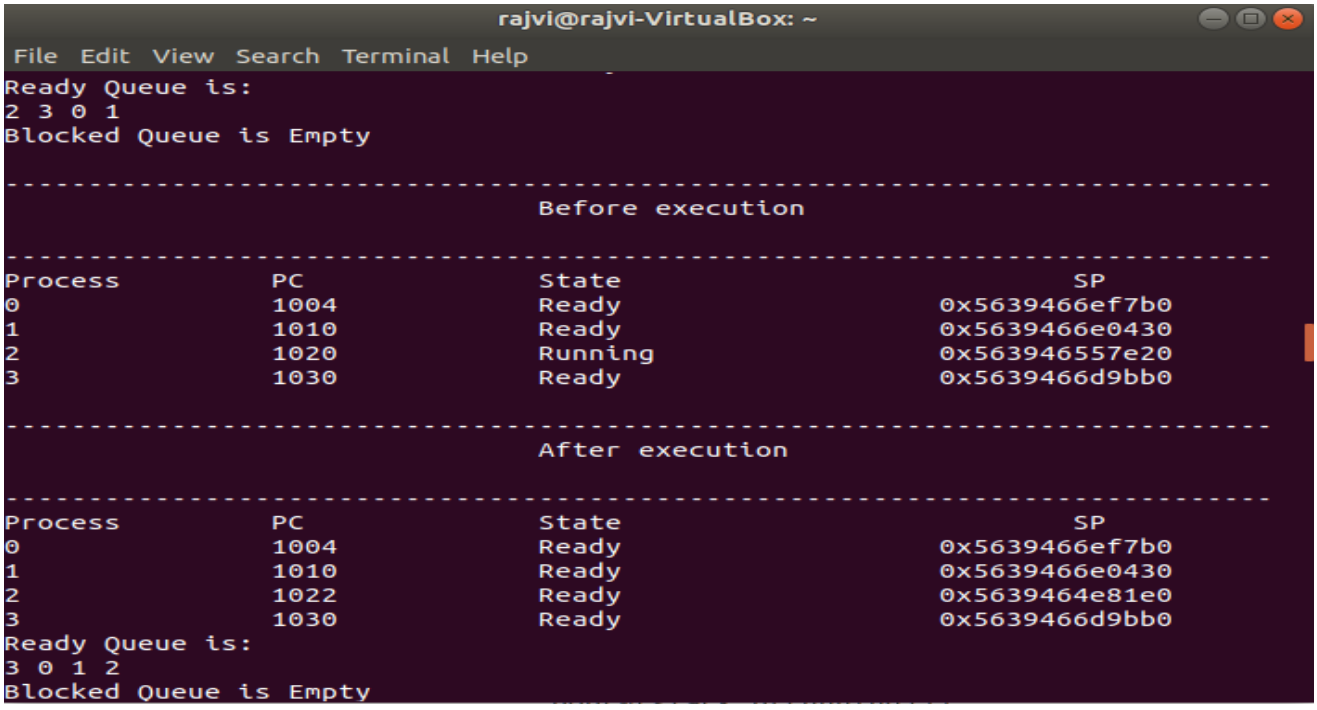

Fig 4: Normal execution without any process blocked

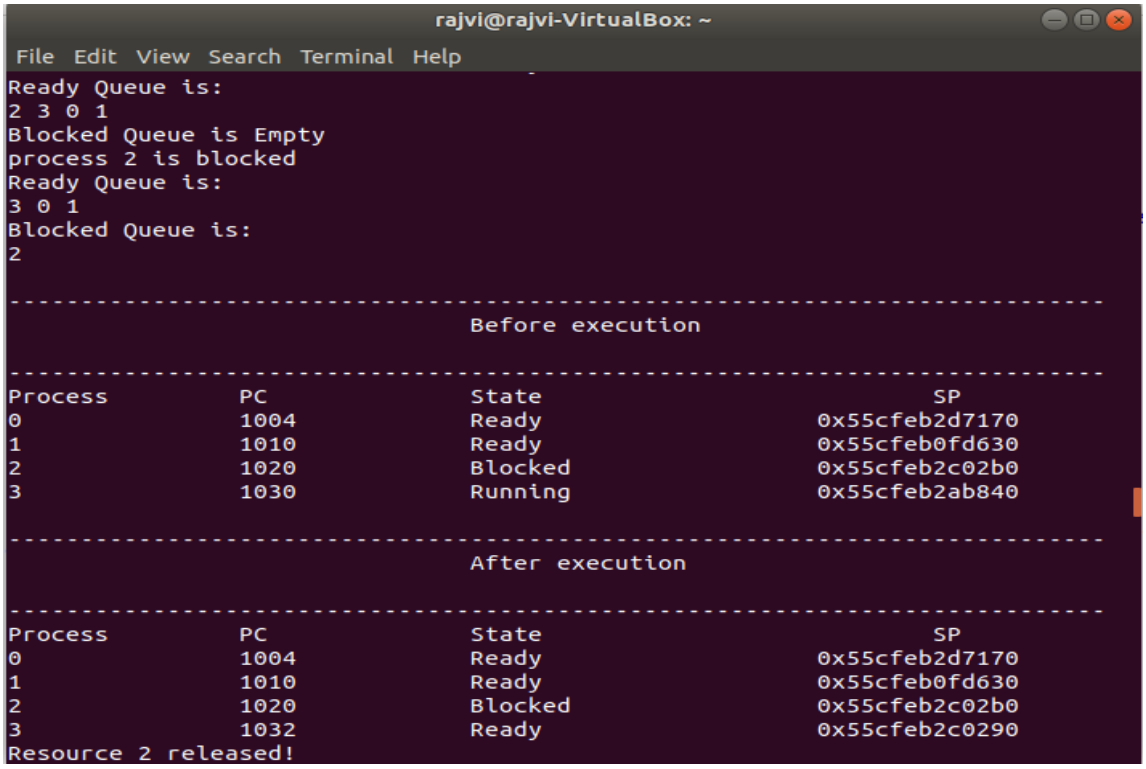

Figure 5: when process is blocked

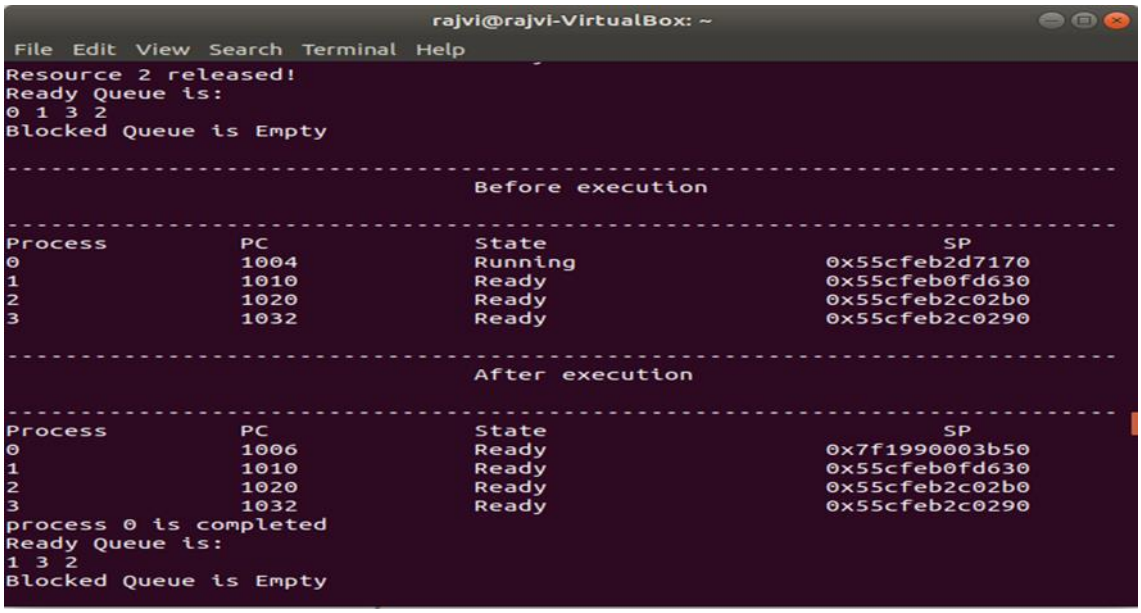

Figure 6: when process is released

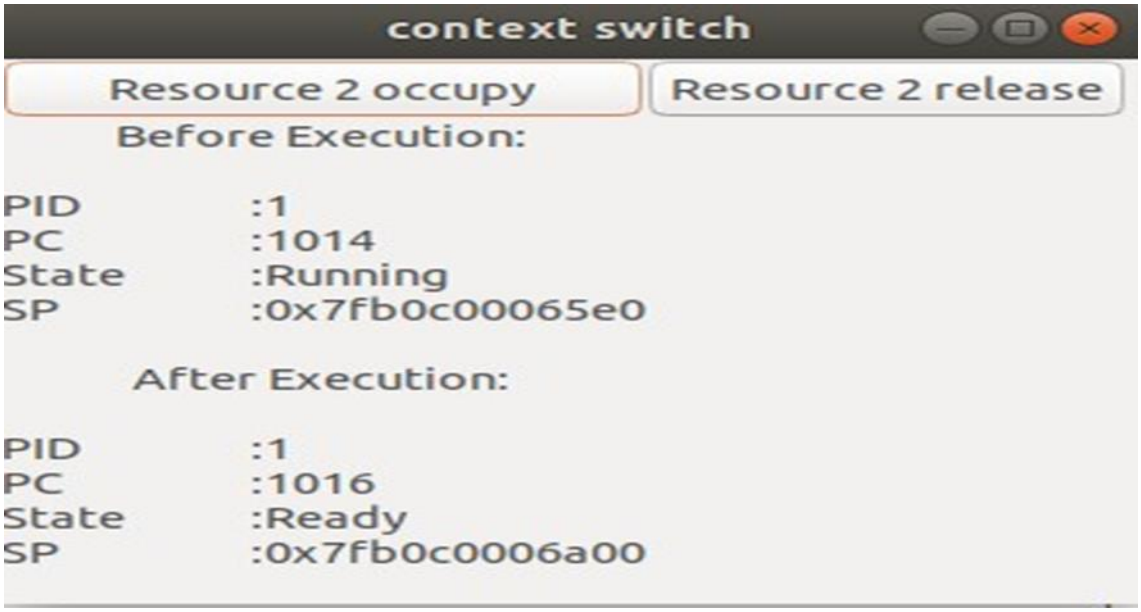
context switch
Resource 2 occupy
Resource 2 release
Before Execution:
PID :1
PC :1014
State :Running
SP :0x7fb0c00065e0
After Execution:
PID :1
PC :1016
State :Ready
SP :0x7fb0c0006a00

Fig 7: GUI

CONCLUSION

Context switching is important part of OS. As without context switching there is no use of different scheduling algorithms. If concept of context switch is not implemented then forcibly, we have to use FCFS (first come first server) scheduling. RR (round robin) and other scheduling algorithms are not possible to implement without switching. And SRT (shortest remaining time) and SPN (shortest process next) and HRRN (highest response ratio next) are not applicable in real life as we do not know service time. In FCFS no need of context switching as once process enters it gets executed. For large value of quantum RR will behave like FCFS. So, to show context switching in better way RR with small quantum value is preferred.

REFERENCES

[1] https://www.tutorialspoint.com/operating_system/os_processes.html

[2] https://www.tutorialspoint.com/operating_system/os_process_scheduling.html

[3] https://www.sciencedirect.com/topics/engineering/process-stack-pointer

Appendix A – Source Code

```
#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};

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 "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);
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],state[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++;
                }
            }
    if(strcmp(string1[0],"add")!=0||strcmp(string1[0],"sub")!=0||strcmp(string
1[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\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]);

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


                                if(t[running]==burst_time[running]){
                        printf("process %d is completed\n",process[running]);
                                        state[running]="Ended";}
                                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 ();
}
```

Appendix B – GitHub Profile and Link for the Project:

https://github.com/Vamshi1009

https://github.com/Vamshi1009/Context_Switching_OSProject