OPERATING SYSTEM AND SYSTEM PROGRAMMING LAB

PROJECT TITLE: CONTEXT SWITCHING

SUBMITTED TO:

DR. CHETNA DABAS

GROUP MEMBERS:

KUSH KAPOOR

PROBLEM STATEMENT

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 context switch for Scheduling and I/O interrupts.

OS CONCEPTS USED

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 minimize the risk of star-vation. 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).

MODEL

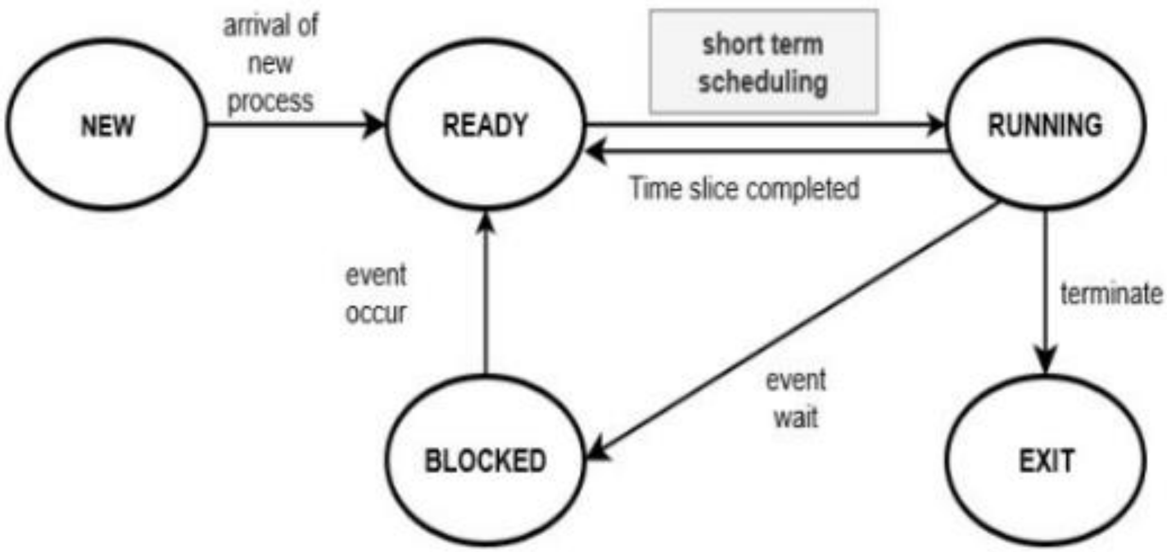


Figure 1: Five state model for context switching

FLOWCHART

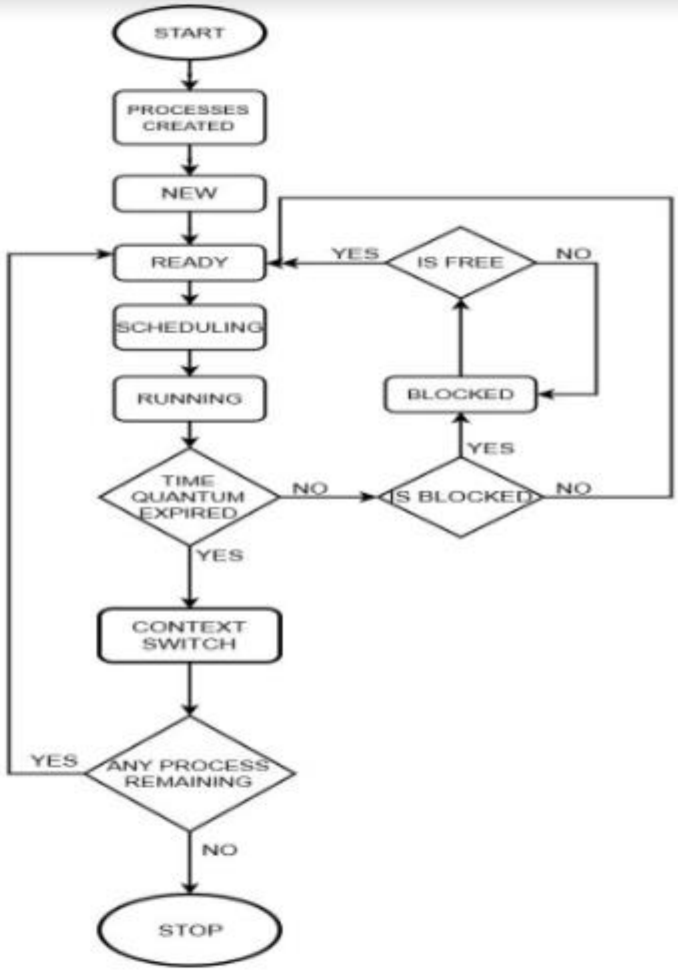


Figure 2: FLOW CHART

PROCESS MODULES AND FILES

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

To run : /**gcc ‘pkg-config gtk+-3.0 --cflags‘ main.c stack implementation.c queue implementation.c-o os ‘pkg-config gtk+-3.0 --libs**‘

./os

stack implementation.c: It contains stack implementation for push,pop operations

queue implementation.c: It contains queue implementation for enqueue, dequeue operations.

This code can run in linux. Only main.c needs to be runned.You need to install GTK3.0 to run this.

IMPLEMENTATION

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.

CODE

QUEUE IMPLEMENTATION

```
#include <stdio.h>
#include <stdlib.h>
#include "queue_implementation.h"
int arr[4];

Queue * createQueue(int maxElements)
{
	Queue *Q;
	Q = (Queue *)malloc(sizeof(Queue));
	Q->elements = (int *)malloc(sizeof(int)*maxElements);
	Q->size = maxElements;
	Q->s=0;
	Q->front = -1;
```

```
    Q->rear = -1;
    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)
  {
    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 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;
```

```
            }
        }

}
```

STACK IMPLEMENTATION

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

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
  {

    printf("stack full\n");
  }
}

char *top(struct stack_t *theStack)
{
  if (theStack && theStack->head)
    return theStack->head->data;
  else
    return NULL;
}

void * stackpointer(struct stack_t *theStack)
{
  if (theStack && theStack->head)
    return theStack->head;
  else
    return NULL;
}


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--;
  }
}
```

MAIN FUNCTION

```
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])),stackp
ointer(&(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\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++;

}

}

//check whether variable is declared or not


if(strcmp(string1[0],"add")!=0||strcmp(string1[0],"sub")!=0||strcmp(string1[0],"div")!=0||strcmp(stri
ng1[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]);



}
```

```
/* update pcb value of process after running */

updateL(GTK_LABEL(pcb2),process[running],pc[running],state[running],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{
```

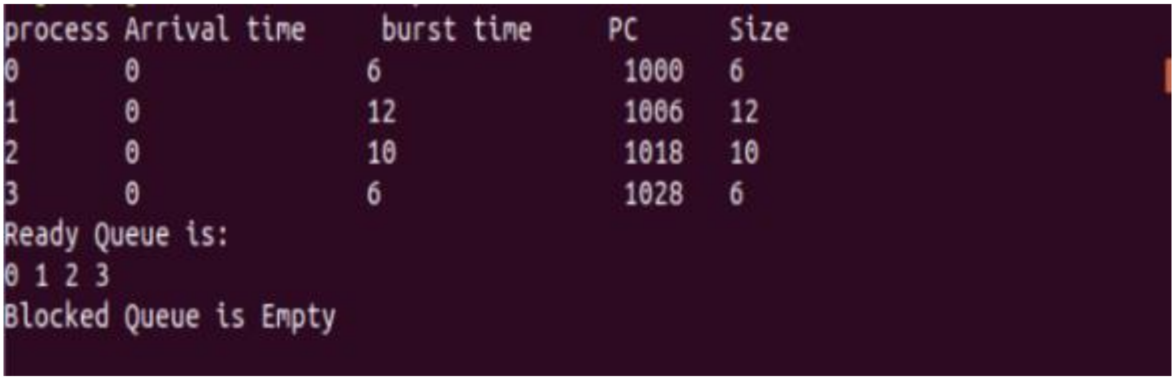
```
state[running]="Blocked";

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 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);




    // 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
```

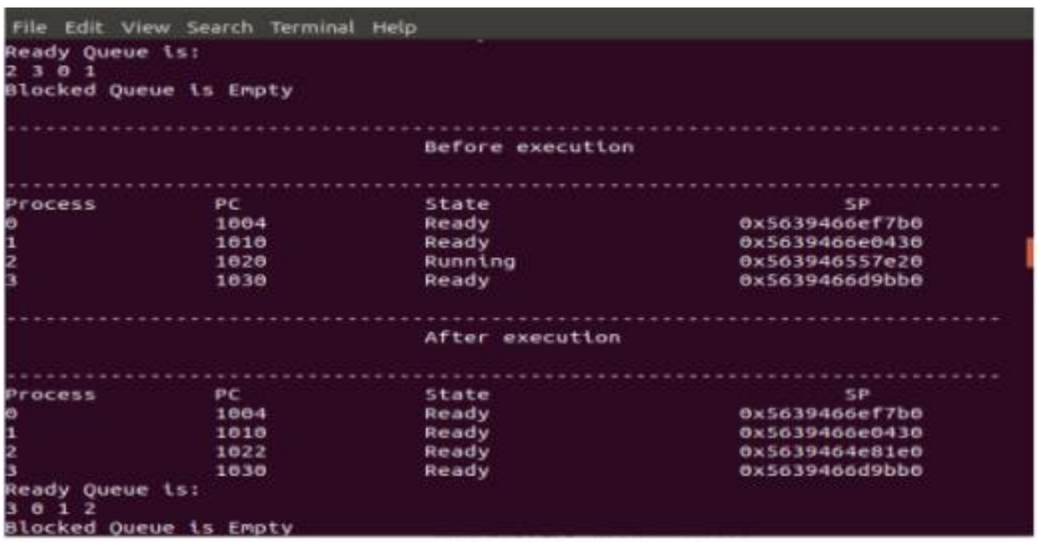
```
    gtk_grid_attach (GTK_GRID (grid), button, 1, 3, 1, 1);



    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 ();

}
```

OUTCOMES



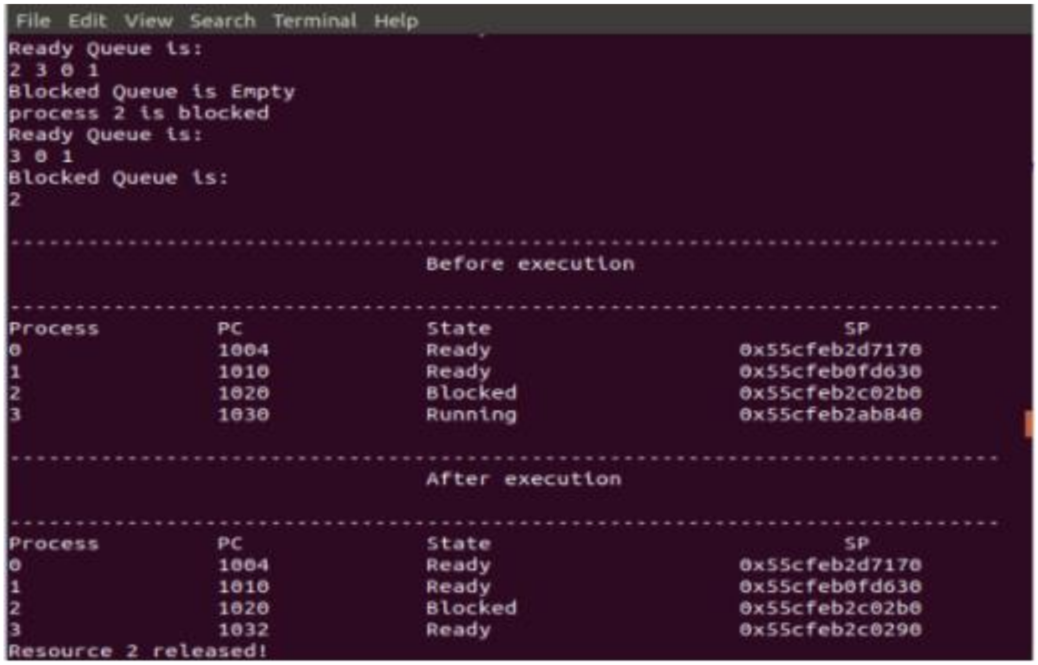
```
process Arrival time     burst time     PC      Size
0       0                6               1000   6
1       0                12              1006   12
2       0                10              1018   10
3       0                6               1028   6
Ready Queue is:
0 1 2 3
Blocked Queue is Empty
```

Figure3: Process block



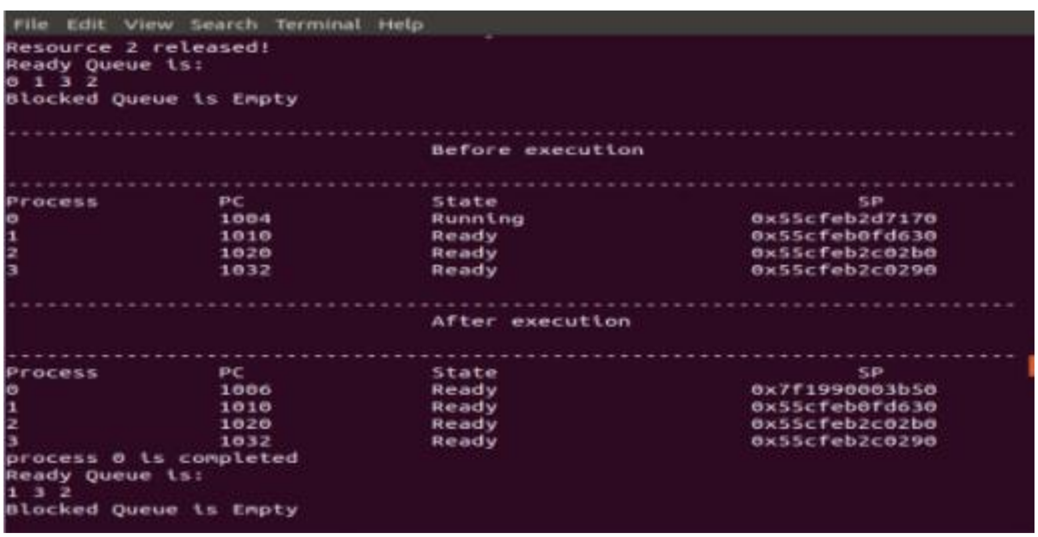
```
File  Edit  View  Search  Terminal  Help
Ready Queue is:
2 3 0 1
Blocked Queue is Empty
------------------------------------------------------------------------------
                          Before execution
------------------------------------------------------------------------------
Process          PC              State                 SP
0                1004            Ready                 0x5639466ef7b0
1                1010            Ready                 0x5639466e0430
2                1020            Running               0x563946557e20
3                1030            Ready                 0x5639466d9bb0
------------------------------------------------------------------------------
                          After execution
------------------------------------------------------------------------------
Process          PC              State                 SP
0                1004            Ready                 0x5639466ef7b0
1                1010            Ready                 0x5639466e0430
2                1022            Ready                 0x5639464e81e0
3                1030            Ready                 0x5639466d9bb0
Ready Queue is:
3 0 1 2
Blocked Queue is Empty
```

Figure4: Normal execution without any process blocked



```
File  Edit  View  Search  Terminal  Help
Ready Queue is:
2 3 0 1
Blocked Queue is Empty
process 2 is blocked
Ready Queue is:
3 0 1
Blocked Queue is:
2

---------------------------------------------------------------------------
                                Before execution

---------------------------------------------------------------------------
Process         PC              State                   SP
0               1004            Ready                   0x55cfeb2d7170
1               1010            Ready                   0x55cfeb0fd630
2               1020            Blocked                 0x55cfeb2c02b0
3               1030            Running                 0x55cfeb2ab840

---------------------------------------------------------------------------
                                After execution

---------------------------------------------------------------------------
Process         PC              State                   SP
0               1004            Ready                   0x55cfeb2d7170
1               1010            Ready                   0x55cfeb0fd630
2               1020            Blocked                 0x55cfeb2c02b0
3               1032            Ready                   0x55cfeb2c0290
Resource 2 released!
```

Figure5: When process is blocked



```
File  Edit  View  Search  Terminal  Help
Resource 2 released!
Ready Queue is:
0 1 3 2
Blocked Queue is Empty

---------------------------------------------------------------------------
                                Before execution

---------------------------------------------------------------------------
Process         PC              State                   SP
0               1004            Running                 0x55cfeb2d7170
1               1010            Ready                   0x55cfeb0fd630
2               1020            Ready                   0x55cfeb2c02b0
3               1032            Ready                   0x55cfeb2c0290

---------------------------------------------------------------------------
                                After execution

---------------------------------------------------------------------------
Process         PC              State                   SP
0               1006            Ready                   0x7f1990003b50
1               1010            Ready                   0x55cfeb0fd630
2               1020            Ready                   0x55cfeb2c02b0
3               1032            Ready                   0x55cfeb2c0290
process 0 is completed
Ready Queue is:
1 3 2
Blocked Queue is Empty
```

Figure6: When process is released

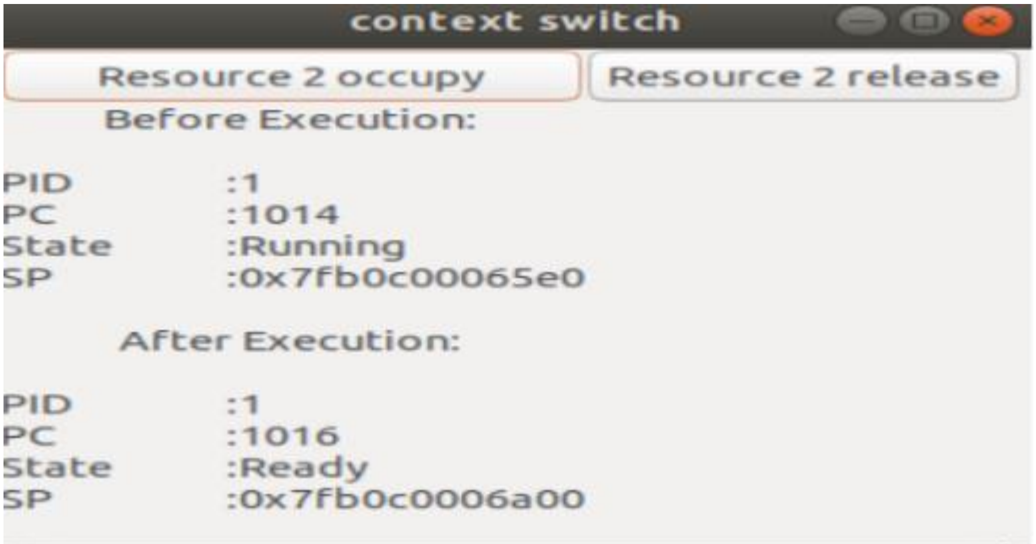


Figure7: GUI

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] Operating System :Internal and System Design, 9th edition, William Stallings, Pearson Publication.
[2] https://www.tutorialspoint.com/operating system/os processes.html
[3] https://www.tutorialspoint.com/operating system/os process scheduling.html
[4] https://www.sciencedirect.com/topics/engineering/process-stack-pointer