# B.M.S College of Engineering



**P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019**

### DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

**Course –Operating System**

**Course Code –19IS3ESOPS**

**AY 2020-21**

**Title:**

**CONTEXT SWITCHING**

Submitted to – B S Mahalaxmi

Submitted by –

Srujan Kumar(1BM20IS043)

Gauravesh Sharma(1BM20IS040)

Adarsh Singh(1BM20IS008)

Harsh Parihar(1BM20IS045)

# B.M.S College of Engineering

**P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sl. no.** | **Contents** | **Page no.** |
| **1** | **INTRODUCTION** | **3** |
| **2** | **PROBLEM STATEMENT** | **4** |
| **3** | **CONCEPTS USED** | **5** |
| **4** | **MODEL & FLOWCHART** | **6** |
| **5** | **IMPLEMENTATION** | **7** |
| **6** | **CODE** | **8-15** |
| **7** | **OUTPUT** | **16-19** |
| **8** | **CONCLUSION** | **20** |
| **9** | **REFERENCE** | **21** |

**INTRODUCTION**

The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system. When switching perform in the system, it stores the old running process's status in the form of registers and assigns the CPU to new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue. The execution of the old process starts at that point where another process stopped it. A context switching helps to share a single CPU across all processes to complete its execution and store the system's tasks status. When the process reloads in the system, the execution of the process starts at the same point where there is conflicting.

PROBLEM STATEMENT

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

CONCEPTS USED

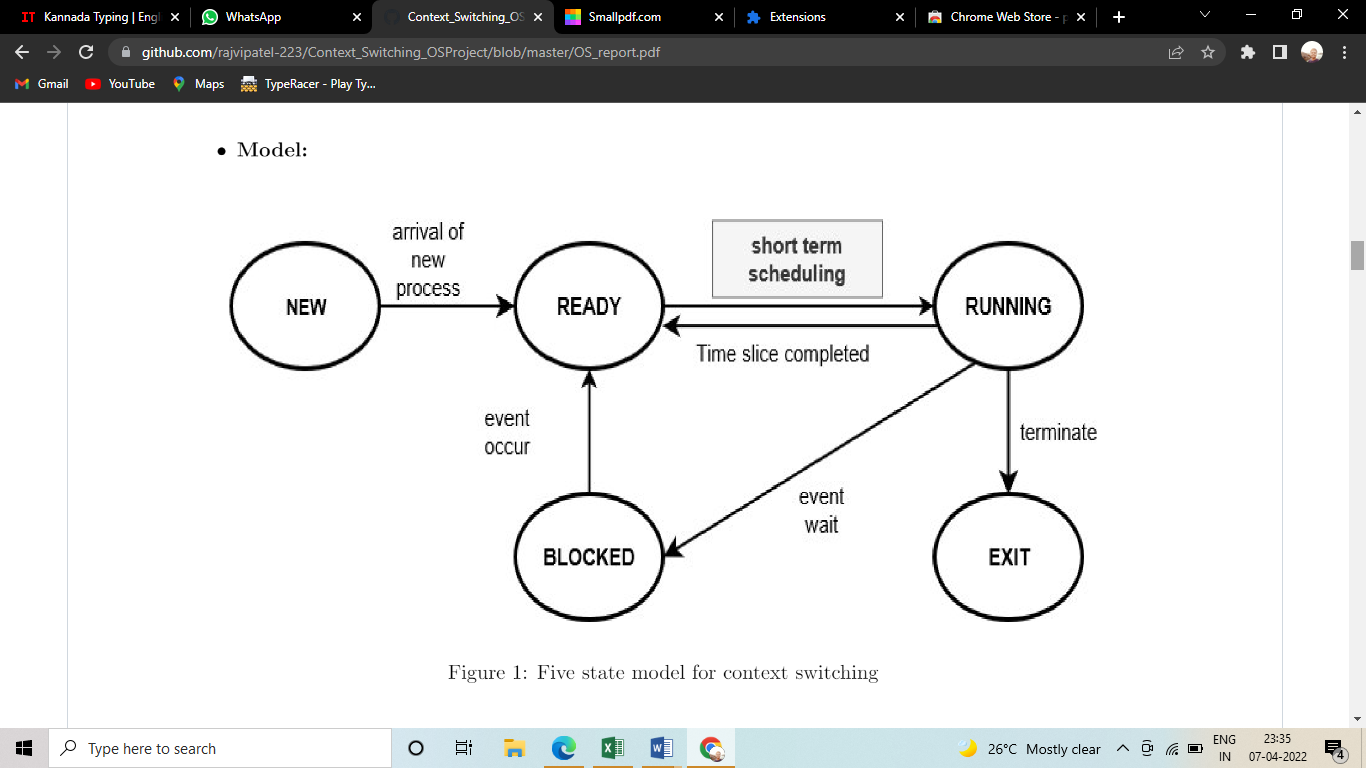
Five-state model: For handling processes we have considered five-state model. States are Blocked, Running and Ready. Blocked and Ready States are implemented through Queue. Ready Queue is implemented using circular Queue because process should remain in ready until it terminates or its execution gets 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 starvation. Also if process is short than RR provides good response time. It gives fair treatment to all processes.

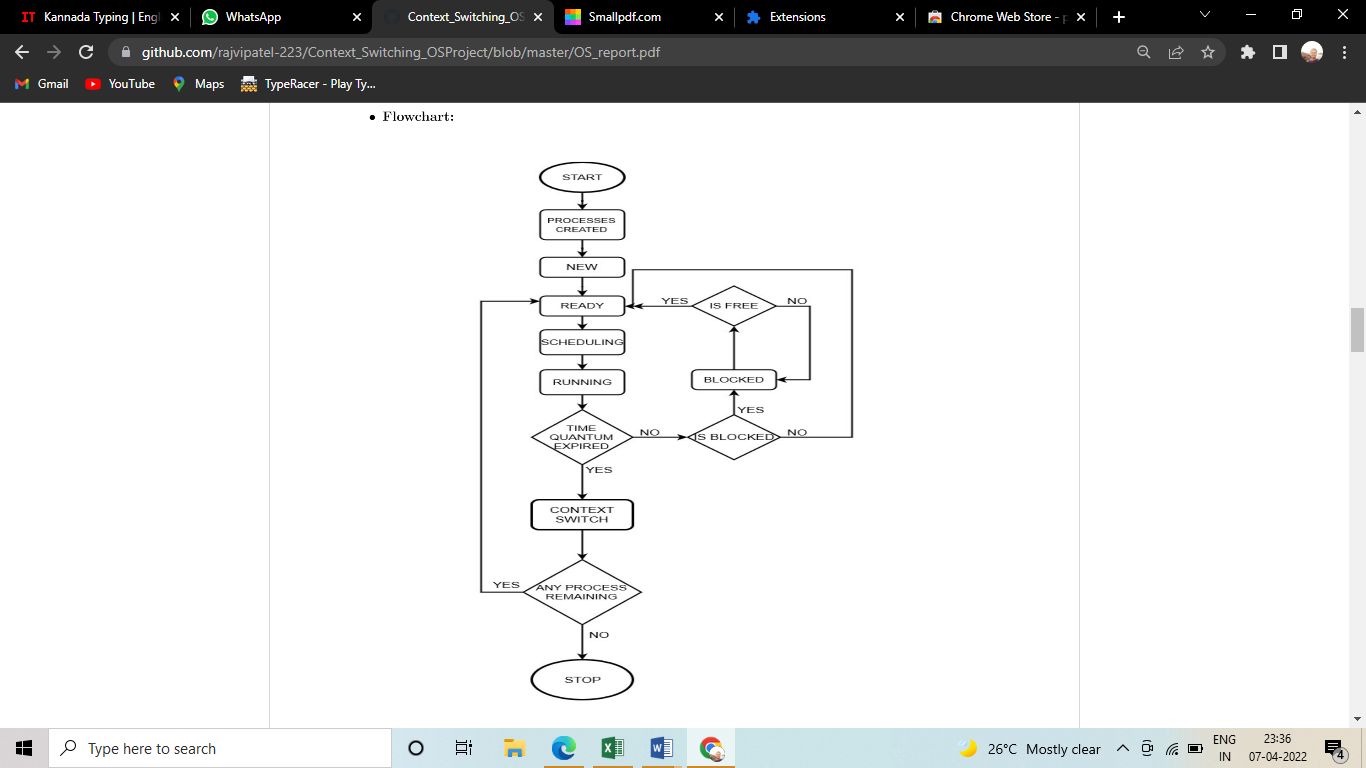
Context Switching: When context switch occurs, for example if process runs for one time slice, but its execution is not completed, then whenever next time processor is allocated to it process must 1 start from where it has left. For this purpose PCB is used. This stored data is the context of process.

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



FLOWCHART



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 pushed 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 o ready queue. As a result we are showing before and updated PCB of each process. Resources are blocked and released through GUI.

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}; /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\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)

{

if (count == t[running])

{

break;

}

else

{ //to read single word

for(int p=0;p<=(strlen(line));p++)

{

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(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\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 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 ();

OUTPUT

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

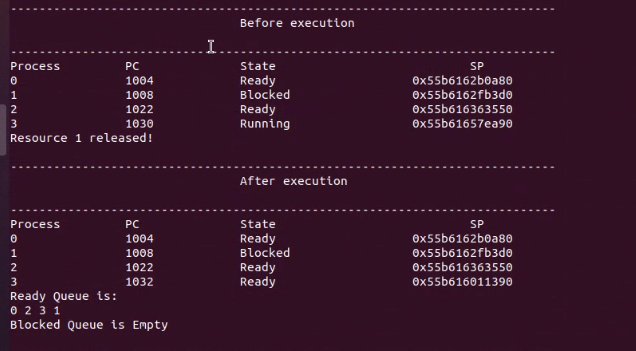
Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**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.

REEFERENCES

1. www.geeksforgeeks.com

2. www.wikipedia.com

3. www.javatpoint.com

4. www.tutorialspoint.com