**National University of Computer & Emerging Sciences**

**Karachi Campus**



**Project Report**

**Operating System CS2006**

**SCHEDULING ALGORITHMS (PREAMPTIVE AND NON-PREAMPTIVE) WITH REAL LIFE SCENARIOS**

**Group Members:**

**19k-0286 Maryam Hussain**

**19k-0287 Abdul Ghaffar Ansari**

**19k-1490 Rafay Akram**

**CS-5E**

**INTRODUCTION**

The basic functionality of the project is to implement the two different concepts studied in this course i.e scheduling algorithms and kernel threads.

**HOW PROJECT STARTS**

The project is Linux based and the first step of the project is to have the latest version of ubuntu so that there are no errors in the working of the programs. All the codes will be written and executed separately on the terminal or if ubuntu allows all of the approaches will be on the same c file.

The flow diagram of the project isn’t included because of there are not many parts of this given project all the work is done in a single. Swift manor.

**PROBLEMS FACED**

The problem faced were the codes that were going to be executed caused problems as it was the first time, we had studied about P-threads and Kernel programing But the scheduling algorithms and their logic was easily run and executed as it was a topic was a former one.

It took us at least one month for going through the links and manuals and books to get the full knowledge and grip over the kernel programming, that was in fact time taking it was worth it because we have learnt a lot about it.

**THE ACTUAL WORKING OF THE PROJECT (METHODOLOGY)**

Two scheduling algorithms are implemented in the source code that are preemptive and non-preemptive.

Burst time and priorities of the process are set randomly using random function.

**Non- preemptive**

**Shortest Job First Scheduling**

SJF is a preemptive and Non-Preemptive algorithm. It based on length of latter’s next CPU burst. If a process acquired CPU and execution is going on, a new process with small CPU burst entered. Then CPU is preempted from current process and will give to further process. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. Shortest Next CPU burst is also an appropriate term for SJF algorithm because scheduler examines the length of next CPU burst continuously.

**Preemptive**

### Round Robin Scheduling

Ready queue as a FIFO queue. The new process will add to the tail of the ready queue and process is removed from the head of queue. CPU select a process from ready queue, set a timer. After completion of time quantum, the process will be preempted doesn’t matter it is completed or not. IF the process not finished it will add in the tail of the queue and CPU select new process. In this way, the process is going on.

### Priority Scheduling

Priority Scheduling is a preemptive and Non-Preemptive algorithm. It based on the priority of process. If a process acquired CPU and execution is going on, a new process with more priority entered. Then CPU is preempted from current process and will give to other process. If two processes have the priority next process, FCFS scheduling is used to break the tie. Priority Scheduling Next CPU burst is also an appropriate term for Priority Scheduling algorithm because scheduler examines the priority of process continuously

**PROJECT CONFIGURATION AND CODE:**

The initial thing to run these programs are that one must have the latest version of ubuntu installed and ubuntu must have installed all the primary packages/libraries that runs the codes and some of the examples of these packages are obtained by

* *Sudo apt-get install build-essential*
* *Sudo apt-get update*

**REFERENCES:**

* + [www.geeksforgeeks.com](http://www.geeksforgeeks.com)
  + [www.github.com](http://www.github.com)
  + [www.youtube.com](http://www.youtube.com)
  + Os Labs
  + Os theory

**CODE SNIPPET:**

#include<linux/module.h>

#include<linux/kernel.h>

#include<linux/kthread.h>

#include<linux/sched.h>

#include<linux/time.h>

#include<linux/timer.h>

#include <linux/random.h>

static struct task\_struct \*thread1;

static struct task\_struct \*thread2;

static struct task\_struct \*thread3;

int a=21;

int Round\_Robin\_Scheduling(void\* t)

{

printk("ROUND ROBIN SCHEDULING: \n");

unsigned int rand = 0;

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

//float average\_wait\_time, average\_turnaround\_time;

limit = 5;

x = limit;

for(i = 0; i < limit; i++)

{

get\_random\_bytes(&rand,sizeof(rand)) ;

arrival\_time[i]= (rand %=8);

get\_random\_bytes(&rand,sizeof(rand)) ;

burst\_time[i]= (rand %=10) +1;

temp[i] = burst\_time[i];

}

//scanf("%d", &time\_quantum);

time\_quantum = 4;

printk("\nProcess ID\tBurst Time\t Turnaround Time Waiting Time\n");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printk("\nProcess[%d]\t\t%d\t\t %d\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

//average\_wait\_time = wait\_time \* 1.0 / limit;

//average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

//printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);

//printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

do\_exit(0);

return 0;

}

int shortest\_remaining\_time\_Scheduling(void\* t)

{

int a[10],b[10],x[10];

int waiting[10],turnaround[10],completion[10];

int i,j,smallest,count=0,time,n;

double avg=0,tt=0;

int end;

unsigned int rand = 0;

printk("SHORTEST REMAINING TIME SCHEDULING:\n");

n = 5;

for(i=0;i<n;i++)

{

get\_random\_bytes(&rand,sizeof(rand)) ;

a[i]= (rand %=8) +0;

get\_random\_bytes(&rand,sizeof(rand)) ;

b[i]= (rand %=10) +1;

}

for(i=0;i<n;i++)

x[i]=b[i];

b[9]=9999;

for(time=0;count!=n;time++)

{

smallest=9;

for(i=0;i<n;i++)

{

if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )

smallest=i;

}

b[smallest]--;

if(b[smallest]==0)

{

count++;

end=time+1;

completion[smallest] = end;

waiting[smallest] = end - a[smallest] ;

waiting[smallest] -= x[smallest];

turnaround[smallest] = end - a[smallest];

}

}

printk("pid burst arrival waiting turnaround completion");

for(i=0;i<n;i++)

{

printk("\n %d \t%d \t %d\t%d \t%d\t %d",i+1,x[i],a[i],waiting[i],turnaround[i],completion[i]);

avg = avg + waiting[i];

tt = tt + turnaround[i];

}

//printf("\n %lf %lf",avg,tt);

//printf("\n\nAverage waiting time = %lf\n",avg/n);

//printf("Average Turnaround time = %lf",tt/n);

do\_exit(0);

return 0;

}

int preampt(void\* t)

{

int x=\*(int \*)t;

int burst\_time[20], process[20], waiting\_time[20], turnaround\_time[20], priority[20];

int i,j, limit, sum = 0, position, temp,check=0;

//float average\_wait\_time, average\_turnaround\_time;

unsigned int rand = 0;

get\_random\_bytes(&rand,sizeof(rand));

printk("PREAMPTIVE PRIORITY SCHEDULING:\n");

limit = 5;

for(i = 0; i < limit; i++)

{

get\_random\_bytes(&rand,sizeof(rand)) ;

burst\_time[i]= (rand%=10) +1;

get\_random\_bytes(&rand,sizeof(rand)) ;

priority[i]= (rand%=limit) +2;

check= priority[i];

if(priority[i-1]==check)

{

priority[i] +=1;

}

process[i] = i + 1;

}

for(i = 0; i < limit; i++)

{

position = i;

for(j = i + 1; j < limit; j++)

{

if(priority[j] < priority[position])

{

position = j;

}

}

temp = priority[i];

priority[i] = priority[position];

priority[position] = temp;

temp = burst\_time[i];

burst\_time[i] = burst\_time[position];

burst\_time[position] = temp;

temp = process[i];

process[i] = process[position];

process[position] = temp;

}

waiting\_time[0] = 0;

for(i = 1; i < limit; i++)

{

waiting\_time[i] = 0;

for(j = 0; j < i; j++)

{

waiting\_time[i] = waiting\_time[i] + burst\_time[j];

}

sum = sum + waiting\_time[i];

}

sum = (sum / limit);

//average\_wait\_time = sum;

sum = 0;

printk("\nProcess ID Burst Time Waiting Time Turnaround Time Priority\n");

for(i = 0; i < limit; i++)

{

turnaround\_time[i] = burst\_time[i] + waiting\_time[i];

sum = sum + turnaround\_time[i];

printk("\nProcess[%d]\t%d\t\t%d\t\t%d\t\t%d\n", process[i], burst\_time[i], waiting\_time[i], turnaround\_time[i],priority[i]);

}

sum = (sum / limit);

do\_exit(0);

return 0;

}

int thread\_init(void)

{

char our\_thread1[8]="thread1";

char our\_thread2[8]="thread2";

char our\_thread3[8]="thread3";

printk(KERN\_INFO "in init");

thread1 = kthread\_create(Round\_Robin\_Scheduling,&a,our\_thread1);

thread2 = kthread\_create(shortest\_remaining\_time\_Scheduling,&a,our\_thread2);

thread3 = kthread\_create(preampt,&a,our\_thread3);

if((thread1))

{

printk(KERN\_INFO "in if");

wake\_up\_process(thread1);

}

if((thread2))

{

printk(KERN\_INFO "in if");

wake\_up\_process(thread2);

}

if((thread3))

{

printk(KERN\_INFO "in if");

wake\_up\_process(thread3);

}

return 0;

}

void threadCleanup(void)

{

printk(KERN\_INFO "cleaning up kthread");

printk(KERN\_INFO "");

kthread\_stop(thread1);

}

MODULE\_LICENSE("GPL");

module\_init(thread\_init);

module\_exit(threadCleanup);

**Output Screenshots**











