FINAL REPORT OS Project

OPERATING SYSTEM (CSE 316)

Name of Student: - N. Abhiram

Registration No: - 11801535

Roll no: - 15

Email Id: - abhiramnallamekala143@gmail.com

Github Link: https://github.com/abhiram143143/OS-project.git

Submitted To *SamreenFayaz madam*.

Lovely Professional University

Jalandhar, Punjab, India.

Description of the project:

Preemptive Scheduling is defined as the scheduling which is done when the process changes from running state to ready state or from waiting for the state to ready state. In this, the resources are allocated to execute the process for a certain period. After this, the process is taken away in the middle and is placed in the ready queue its bursts time is left and this process will stay in ready line until it gets its turn to execute.

Suppose if a process which has the highest priority arrives, then this process does not wait for the complete execution of the current process. Instead of it, the ongoing process is interrupted in between and is placed in the ready queue until the process which has the highest priority does its execution. Thus in this way, all the processes which are in the available line get some time to run.

Example of Preemptive scheduling are: Round robin scheduling, priority scheduling, and shortest job first(SJF) scheduling

Explanation Suppose there are four processes P1, P2, P3 and P4 whose arrival time and burst time are given in the table below:

Process	Arrival Time	Brust Time	Priority
P1	0	4	0
P2	1	1	1
P3	2	2	2
P4	3	1	3

When the process starts execution (i.e. CPU assigned), priority for that process changes at the rate of m=1.When the process waits for CPU in the ready queue (but not yet started execution), its priority changes at a rate n=2. All the processes are initially assigned priority value of 0 when they enter ready queue for the first time . The time slice for each process is q = 1. When two processes want to join ready queue simultaneously, the process which has not executed recently is given priority.

It’s Complexity:

We discuss the problem of scheduling af set of independent tasks T, each ti ϵ T of lenght ℓi ϵ Z+, on m identical processors. We allow preemption but assume a communication delay of time k ϵ N. Whenever a task is preempted from one processor to another, there must be a delay of at least k time units. We show that if k = 1, an optimal schedule can be found in polynomial time but if k ⩾ 2; the corresponding decision problem is NP-complete.

WHY Scheduling?:

To make your day more logical and efficient, you work on a schedule. An operating system operates in a similar manner: by scheduling tasks, improving efficiency, reducing delays and wait times (response times to the system), and managing CPU resources better.

This activity is called process scheduling. A process is like a job in computer systems that can be executed. Some processes are input/output (I/O) like graphics display process; others are CPUfocused and can be transparent to users. If your computer freezes, sometimes the underlying issue could be that a system process is trying to acquire CPU resources, but those resources are already occupied by other processes. Through process scheduling, the operating system tries to avoid these kinds of deadlocks and lockups.

Criteria for Scheduling:

There are several criteria for invoking the best scheduling policies for a system:

criteria	Description
Cpu utilization	Reduces the strain on the CPU and manages the percentage of time the CPU is busy

Throughput	Increases the number of processes completed in a given time frame
Wait Time	Reduces the waiting time of a process
Response Time	Minimizes the time a user has to wait for a process to run
Turn around Time	Total time a process takes to run, from start to finish (includes all waiting time)

Scheduling Policies:

To fulfil those criteria, a scheduler has to use various policies or strategies:

1. Fairness

Just as it isn't fair for someone to bring a loaded shopping cart to the 10-items-or-less checkout, the operating system shouldn't give an unfair advantage to a process that will interfere with the criteria we listed (CPU utilization, wait time, throughput). It's important to balance long-running jobs and ensure that the lighter jobs can be run quickly.

Let's take a look at the policies and then do a final comparison that addresses the fairness of each item.

2. FCFS - First Come First Served

Also called FIFO (first-in-first-out), first-come-first-served (FCFS) processes jobs in the order in which they are received. This is not a very fair policy, because a long-running job could be running, and other processes have to wait for it to finish. To the end-user, this could look like a system freeze or lock-up.

Consider the following example of a long-running process A that now holds other processes B, C and D as hostage.

3. Round Robin

This policy works like musical chairs but more methodical. A timer is used to determine when to move the current running process to the back of the line. For example, you set the timer to 10ms; if the printer process isn't done within that time frame, move it to the end of the line and move up the next process.

If the time (10ms) is too long, processes still have to wait longer, which basically puts you back into FCFS territory. However, if the time is too small, you spend more time context switching and hence, the throughput suffers.

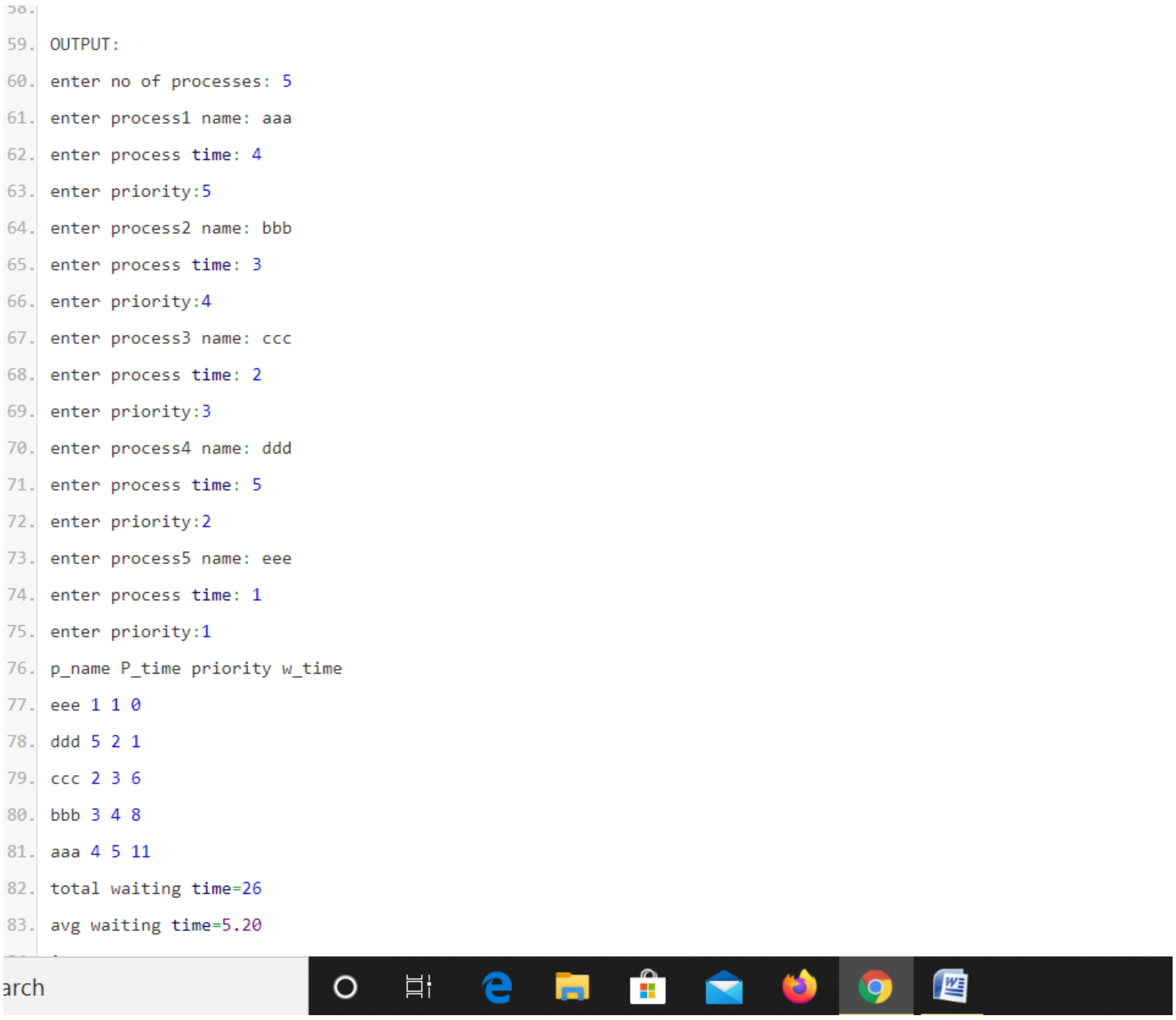
What are the advantages of preemptive scheduling? • More robust, one process cannot monopolize the CPU

• Fairness. The OS makes sure that CPU usage is the same by all running process.

CODE:

```
#include<stdio.h>

#include<conio.h>

void main()

{

  char p[10][5],temp[5];

  int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;

  float avgwt;

printf("enter no of processes:");

scanf("%d",&n);

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

  {

printf("enter process%d name:",i+1);
scanf("%s",&p[i]);
printf("enter process time:");
scanf("%d",&pt[i]);
printf("enter priority:");
scanf("%d",&pr[i]);
  }
 for(i=0;i<n-1;i++)
 {
 for(j=i+1;j<n;j++)
 {
   if(pr[i]>pr[j])
 {
   temp1=pr[i];
pr[i]=pr[j];
pr[j]=temp1;
  temp1=pt[i];
pt[i]=pt[j];
pt[j]=temp1;
strcpy(temp,p[i]);
strcpy(p[i],p[j]);
strcpy(p[j],temp);
  }
  }
  }
wt[0]=0;
  for(i=1;i<n;i++)
  {
wt[i]=wt[i-1]+et[i-1];
totwt=totwt+wt[i];
   }
avgwt=(float)totwt/n;
```

```
printf("p_name\t p_time\t priority\t w_time\n");
for(i=0;i<n;i++)
{
printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
  }
printf("total waiting time=%d\n avg waiting time=%f",tot,avg);
getch();
  }
```



```
OUTPUT:
enter no of processes: 5
enter process1 name: aaa
enter process time: 4
enter priority:5
enter process2 name: bbb
enter process time: 3
enter priority:4
enter process3 name: ccc
enter process time: 2
enter priority:3
enter process4 name: ddd
enter process time: 5
enter priority:2
enter process5 name: eee
enter process time: 1
enter priority:1
p_name P_time priority w_time
eee 1 1 0
ddd 5 2 1
ccc 2 3 6
bbb 3 4 8
aaa 4 5 11
total waiting time=26
avg waiting time=5.20
```