Prof Ram Meghe College of Engineering & Management

New Express Highway, Badnera-Amravati

INDEX

S.No.	Title	Date	Remarks
1	To study linux & installation of ubuntu on provided machine.	17/1/25	
2	To study & execute basic file commands & process related open source ubuntun commands	24/1/25	
3	To write the program to implement fork() system call.	31/1/25	
4	Write a program in c to implement FCFS Scheduling Algorithm	7/2/25	
5	Write a program in c to implement SJF scheduling Algorithm.	28/2/25	
6	Write a program in c to implemant PRIORITY scheduling Algorithm.	7/3/25	
7	Write a program in c to implement ROUNDROBIN Scheduling Algorithm	21/3/25	
8	Write a program in c to implemant Page Replacement Algorithm using FIFO.	28/3/25	
9	Write a program in c to implement page Replacement Algorithm using LRU	4/4/25	
10	Write a program in c to implement page Replacement Algorithm using OPTIMAL	11/4/25	
11			
12			
13			
14			

Practical 1

Aim: To study Linux and installation of Ubuntu on provided Machine.

Theory and Process:
One of the possible Installation steps:

1. Prepare for installation:

- Download the Ubuntu ISO image from the official Ubuntu website (https://ubuntu.com/download).
- Create a bootable USB drive with the Ubuntu ISO using a tool like Rufus (for Windows) or Etcher (for macOS and Linux).

2. Boot from the USB drive:

- Insert the bootable USB drive into your PC.
- Restart your computer and access the BIOS or UEFI settings. The method varies depending on your PC manufacturer (common keys include F2, F10, or Del).
- In the BIOS or UEFI settings, set the USB drive as the primary boot device.
- Save the changes and exit the BIOS or UEFI settings.
- Your computer should now boot from the USB drive.

3. Start the Ubuntu installation:

- Once the PC boots from the USB drive, you'll see the Ubuntu installation menu.
- Select "Install Ubuntu" from the menu and press Enter.

4. Configure language and keyboard layout:

- Choose your preferred language for the installation process.
- Select your keyboard layout.

5. Allocate disk space:

- You'll be presented with options for disk partitioning.
- If you're new to Ubuntu or want a simple installation, you can select the option to "Erase disk and install Ubuntu." This will use the entire disk for Ubuntu, removing any existing data.
- Alternatively, you can choose the "Something else" option to manually partition the disk.

6. Set up user and location:

- Enter your name, desired username, and a password for the user account.
- Choose your location for time and date settings.
-

7. Begin the installation:

- Click on the "Install Now" button to start the installation process.
- Ubuntu will begin installing system files and copying them to your hard drive.

8. Configure additional settings:

- During the installation, you may be prompted to configure additional settings like updates and third-party software installation. Choose the options according to your preference.

9. Restart and boot into Ubuntu:

- Once the installation completes, you'll be prompted to restart your computer.
- Remove the USB drive when prompted and allow your PC to restart.
- Your computer should now boot into Ubuntu.

10. Set up Ubuntu:

- On the first boot, you'll be greeted with a welcome screen where you can configure various settings, such as privacy options and online accounts.
- Follow the on-screen instructions to complete the initial setup.

Some of the Linux distributions

Linux Distribution	Name	Description
	Arch	This Linux Distro is popular amongst Developers. It is an independently developed system. It is designed for users who go for a do-it-yourself approach.
	CentOS	It is one of the most used Linux Distribution for enterprise and web servers. It is a free enterprise class Operating system and is based heavily on Red Hat enterprise Distro.
	Debian	Debian is a stable and popular non-commercial Linux distribution. It is widely used as a desktop Linux Distro and is user-oriented. It strictly acts within the Linux protocols.
	Fedora	Another Linux kernel based Distro, Fedora is supported by the Fedora project, an endeavor by Red Hat. It is popular among desktop users. Its versions are known for their short life cycle.
	Gentoo	It is a source based Distribution which means that you need to configure the code on your system before you can install it. It is not for Linux beginners, but it is sure fun for experienced users.
	LinuxMint	It is one of the most popular Desktop Distributions available out there. It launched in 2006 and is now considered to be the fourth most used Operating system in the computing world.
	OpenSUSE	It is an easy to use and a good alternative to MS Windows. It can be easily set up and can also run on small computers with obsolete configurations.

Linux Distribution	Name	Description
	RedHat enterprise	Another popular enterprise based Linux Distribution is Red Hat Enterprise. It has evolved from Red Hat Linux which was discontinued in 2004. It is a commercial Distro and very popular among its clientele.
	Slackware	Slackware is one of the oldest Linux kernel based OS's. It is another easy desktop Distribution. It aims at being a 'Unix like' OS with minimal changes to its kernel.
	Ubuntu	This is the third most popular desktop operating system after Microsoft Windows and Apple Mac OS. It is based on the Debian Linux Distribution. and it is known as its desktop environment.

Conclusion: Thus we have studied Linux & installation of Ubuntu on provided machine.

Practical No. 2

Aim: To Study and execute basic file commands and process related open source ubuntu commands

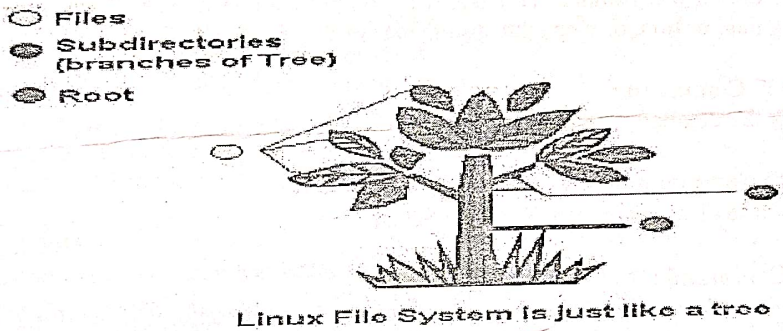
a. Commands to view all executing, block and suspended process.
b. Command to check and change the priority of process CPU utilization for executing processes.
c. Commands to check for child process, sub-processes, process tree, abort & end process and all other basics commands related to processes

Software Used: GCC Compiler

Theory:

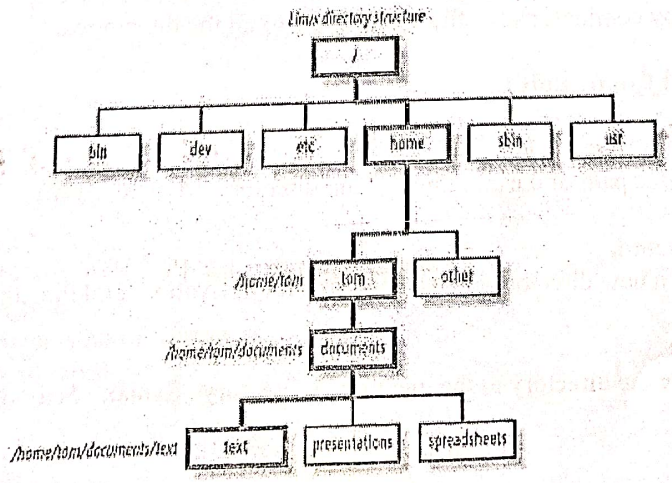
The Linux file system is the structure in which all the information on your computer is stored.

Files are organized within a hierarchy of directories. Each directory can contain files, as well as other directories.



Linux File System is just like a tree

To map out the files and directories in Linux, it would look like an upside-down tree. At the top is the root directory, which is represented by a single slash (/). Below that is a set of common directories in the Linux system, such as bin, dev, home, lib , and tmp , to name a few. Each of those directories, as well as directories added to the root, can contain subdirectories.



- **Date Command :**
  This command is used to display the current data and time. **Syntax :** $date $date +%ch To change the format : **Syntax :** $date „+%l1-%M-%S"
- **Calendar Command :**
  This command is used to display the calendar of the year or the particular month of calendar year. **Syntax :** a.$cal<year> b.$cal<month><year> Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.
- **Echo Command:**
  This command is used to print the arguments on the screen . **Syntax :** $echo <text>
- **Banner Command :**
  It is used to display the arguments in „#" symbol . **Syntax :** $banner <arguments>
- **"who" Command :**
  It is used to display who are the users connected to our computer currently. **Syntax :** $who – option"s **Options :** - H–Display the output with headers. b–Display the last booting date or time or when the system was lastely rebooted.
- **"who am i" Command :**
  Display the details of the current working directory. **Syntax :** $who am *i*
- **"CLEAR" Command :**
  It is used to clear the screen. **Syntax :** $clear
- **"MAN" Command :**
  It help us to know about the particular command and its options & working. It is like help" command in windows. **Syntax :** $man <command name>
- **LIST Command :**
  It is used to list all the contents in the current working directory. ***Syntax:*** *$ ls – options <arguments>* If the command does not contain any argument means it is working in the Current directory. ***Options:*** a– used to list all the files including the hidden files. c– list all the files column wise. d- list all the directories. m- list the files separated by commas. p- list files include „/" to all the directories.

Directory Related Commands :

1. Present Working Directory Command:
To print the complete path of the current working directory. **Syntax :** $pwd

2. MKDIR Command:
To create or make a new directory in a current directory . Syntax : $mkdir<directory name>

3. CD Command:
To change or move the directory to the mentioned directory . Syntax : $cd <directory name.

4. RMDIR Command:

To remove a directory in the current directory & not the current directory itself. Syntax : $rmdir<directory name>

File Related Commands:

1. Create A File :
To create a new file in the current directory we use CAT command.
Syntax : $cat ><filename.
The > symbol is redirectory we use cat command.

2. Display A File :
To display the content of file mentioned we use CAT command without „>" operator.
Syntax : $cat <filename.
Options –s = to neglect the warning /error message.

3. Copying Contents :
To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.
Syntax: $ cat <filename source>>><destination filename>
$ cat<source filename>>><destination filename> it is avoid overwriting.
Options: -n content of file with numbers included with blank lines.
Syntax : $cat –n <filename>

4. Sorting A File
To sort the contents in alphabetical order in reverse order.
Syntax: $sort <filename >
Option: $ sort –r <filename>

5. Copying Contents From One File To Another:
To copy the contents from source to destination file .so that both contents are same.
Syntax: $cp<source filename><destination filename>
$cp<source filename path ><destination filename path>

6. Move Command:
Syntax: $ mv<source filename><destination filename>

7. Remove Command:
To permanently remove the file we use this command .
Syntax : $rm<filename>

8. Word Command :
To list the content count of no of lines , words, characters .
Syntax : $wc<filename>
Options : -c – to display no of characters.
-l – to display only the lines.

-w – to display the no of words.

Process Related Commands

The ps command produces a list of the currently running processes on your computer. The ps command is commonly used in conjunction with the grep command and the more or less commands.
To invoke ps simply type the following:
ps
The output will show rows of data containing the following information:
PID, TTY, Time, Command
The PID is the process ID which identifies the running process. The TTY is the terminal type. On its own the ps command is quite limited. You probably want to see all the running processes.

➢ **To view all the running processes use either of the following commands:**
ps -A
ps -e

To show all of the processes except for session leaders run the following command:
ps -d
ps -d -N

Obviously the -N is not very sensible when used with the -e or -A switches as it will show nothing at all.
ps T
to see all the running processes using the following command:
ps r

Selecting Specific Processes Using The ps Command
For instance if you know the process id you can simply use the following command:
ps -p <pid>

select multiple processes by specifying multiple process IDs as follows:
ps -p "1234 9778"

specify them using a comma separated list:
ps -p 1234,9778

ps -C <command>
ps -C chrome
ps -G <groupname>
ps --Group <groupname>

For example to find out all the processes being run by the accounts group type the following:
ps -G "accounts"
ps --Group "accounts"
ps -g <groupid>
ps --group <groupid> ps
-s <sessionid>

Alternatively use the following to search by terminal type.
ps -t <sessionid>
to find all the processes run by a specific user try out the following command:
ps U <userlist>

For example to find all the processes ran by gary run the following:
ps U "gary"
ps -ef
You can get even more columns by using the following command:

ps -eF
This returns the following columns:
Command
to use the formats type the following:
ps -e --format="uidunamecmd time"
Sorting Output
To sort the output use the following notation:
ps -ef --sort <sortcolumns>

The choice of sort options are as follows:
An example sort command is as follows:
ps -ef --sort user,pid

Using ps With grep, less and more commands
As mentioned at the beginning it is common to use ps with the grep, less and more commands.
ps -ef | more ps
-ef | less

The grep command helps you filter the results from the ps command. For example: **ps -ef | grep chrome**

Output:

Conclusion: Basic Linux directory commands such as /bin, /boot /devetc with other important system commands such as echo, clear man, date , calendar etc were studied and executed successfully. Syntax and execution of process related –ps command was analyzed by passing different arguments to check process id, process priority, blocked process etc.

Practical No. 03

Aim :- To write the program to implement fork () system call.

Teacher's Signature

Practical No. 3

Page No :
Date : / /

Name of Practical

Aim :- To write the program to implement fork () system call.

Description :-

Fork () used to create new processes. The new process consits of a copy of the address space of the orginal process. The value of process id for the child process is zero whereas the value of process id for the parent is an integer value greater than zero.

Teacher's Signature

Page No :
Date : \ \

Name of Practical

Algorithm :-

step 1 :- Start the program.

step 2 :- Declare the variables pid & child id.

step 3 :- Get the child id value using system call fork()

step 4 :- If child id value is greater than zero then print as "i am in the parent process".

step 5 :- If child id != 0 then using system call fork().

step 6 :- Print " i am in the parent process" & print the process id.

step 7 :- If child id != 0 then using getppid() system call get the parent process id.

Teacher's Signature

Page No :
Date : / /

Step 8 :- Print " i am in the parent process " & print the parent process id.

Step 9 :- Else if child id value is less than zero then print as " i am in the child process ".

Step 10 :- If child id != 0 then using getpid() system call get the process id.

Step 11 :- Print " i am in the child process " & print the process id

Step 12 :- If child id != 0 then using ~~getchild~~ getppid() system call get the parent process id.

Step 13 :- Print " i am in the child process " & print the parent process id.

Step 14 :- Stop the program.

Teacher's Signature

Page No :
Date : \ \

Program :-

```
/* Fork system call */

# include <stdio.h>
# include <unistd.h>
# include <sys/types.h>
 int main ()
{
int id, childid;
 id = getpid ();
if ((childid = fork()) > 0)
    {
     printf ("\n i am in the parent process
                                %d", id);
Printf ("\n i am in the parent process
                                %d", getpid ());
Printf ("\n i am in the parent process
                                %d\n", getppid ());
```

Teacher's Signature

output :-

```
i am in the Parent Process : 22061
i am in the Parent Process : 22061
i am in the Parent Process : 22060

i am in child Process 22061
i am in the child Process : 22062
i am in the child Process : 1
```

Conclusion :- By this program to implement Fork() system.

Page No :
Date : / /

ame of Practical

```
}
else
{
Printf("\n i am in child Process %d", id);

Printf("\n I am in the child Process %d",
                                getpid());

Printf("\n i am in the child Process %d",
                                getPpid());

}}
```

Conclusion :- By this program to implement Fork() system.

Teacher's Signature

ame of Practical

Page No :
Date : \ \

Practical No. 04

Aim:- write a program in C to implement FCFS scheduling Algorithm

Theory :-

First come, First serve (FCFS) is one of the simplest types of CPU scheduling. algorithms. It is exactly what it sounds like: processes are attended to in the order in which they arrive in the ready queue, much like customers lining up at a grocery store.

FCFS scheduling is a non-preemptive algorithm, meaning once a process starts running, it cannot be stopped until it voluntarily relinquishes the CPU, typically when it terminates or performs I/O. This method schedules processes in the orders they arrive, without considering priority or other factors.

Teacher's Signature

Date :

Page No :
Date / /

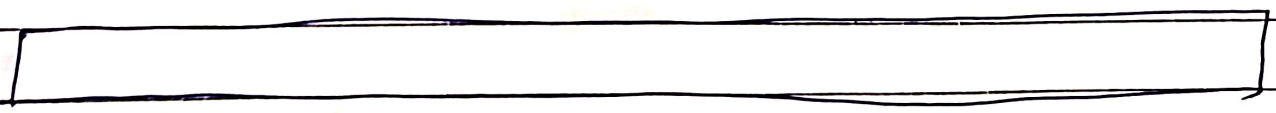
AT : Arrival Time
BT : Burst Time or CPU Time
TAT : Turn Around Time
WT : Waiting Time

Processes with Arrival Time:

Consider the following table of arrival time and burst time for processes P_1, P_2, P_3 and P_4.

Process No.	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P_1	0	2			
P_2	1	2			
P_3	5	3			
P_4	6	4			

Gantt Chart



Time →

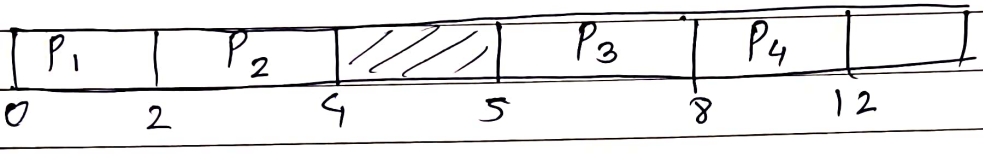
Teacher's Signature

EXPERIMENT : No.

Date :
Page :

Process No.	Arrival Time	Burst Time	completion time	Turn Around Time	waiting Time
P_1	0	2	2		
P_2	1	2	4		
P_3	5	3	8		
P_4	6	4	12		

Gentt chart



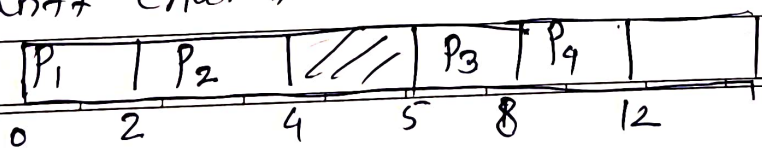
Time →

Now, let's calculate waiting & turn around Time.

TAT = CT − AT

Process No.	Arrival Time	Burst Time	completion time	Turn Around Time	waiting time
P_1	0	2	2	2	
P_2	1	2	4	3	
P_3	5	3	8	3	
P_4	6	4	12	6	

Gantt chart



Teacher's Sign : _____

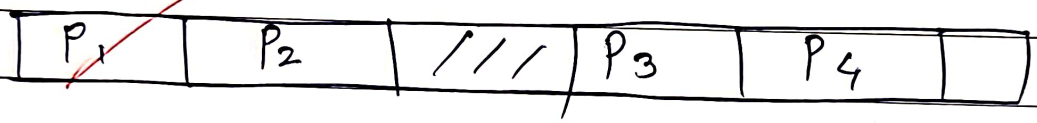
EXPERIMENT : No.

Date :
Page :

WT = TAT - BT

Process No.	Arrival Time	Burst Time	completion Time	Turn Around Time	waiting Time
P_1	0	2	2	2	
P_2	1	2	4	3	
P_3	5	3	8	3	
P_4	6	4	12	6	

Gantt Chart



Teacher's Sign : _____

EXPERIMENT : No. Date : Page :

```
sum = at[0];
for (j=0; j<n ; j++)
{
    tat[k] = ct[k] - at[k];
  totaltat = totaltat + tat [k];
}

 for(k=0; k<n; k++)
 {
     wt[k] = tat[k] - bt[k];

   totalwt = totalwt + wt [k];
 }

printf("\n Process \t AT \t BT \t CT \t TAT \t
                    WT \n \n \n");

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

printf("\n P%d \t %d \t %d \t %d \t
      %d \t %d \t \n", j+1, at[i],
  bt[i], ct[i], ta t[i] wt[i]);
}
```

Output:-

Enter the total no. of processes:- 2
Enter The process Arrival Time & Burst time
Enter Arrival time of process [1]: 0
Enter Burst time of process [1]: 2
Enter Arrival time of process [2]: 1
Enter Burst time of process [2]: 2

Process	AT	BT	CT	TAT	WT
P1	0	2	2	2	0
P2	1	2	4	3	1

Conclusion:-

Thus we have studies the write a program in c to implement FCFS Scheduling Algorithm.

EXPERIMENT : No.

Date :
Page :

```
printf("\n Average Turnaround Time: %f\n",
                totaltat/n);
printf("\n Average Waiting Time: %f",
                totalwt/n);
return 0;
}
```

conclusion :-

Thus we have studies the write a program in c to implement FCFS Scheduling Algorithm.

Teacher's Sign : _____

Page No :
Date : \ \

Practical No. 05

Aim :- Write a Program in C to implement SJF Scheduling Algorithm

Theory :-

The Shortest job first (SJF) or Shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN, also known as Shortest Job Next (SJN).

How to Compute below times in SJF using a Program?

~~Completion Time: Time at which Process completes~~ ~~time and arrival time.~~

Completion Time :- Time at which Process completes it's execution

Turon Around Time : Time Difference betn completion time & arrival time.

~~Turn Around Time = Turn Around Time - Burst Time~~

Teacher's Signature


, P2 ,
waiting
Time.

Page No :
Date : / /

ctical

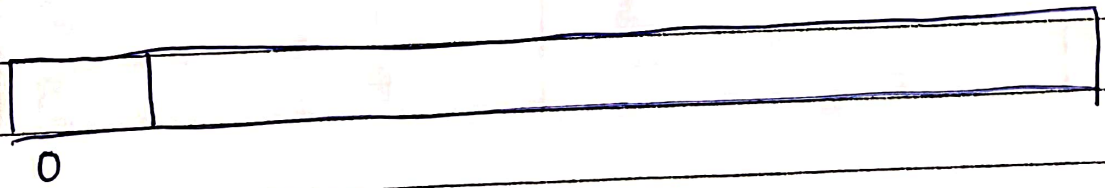
Turn Around Time = Completion Time − Arrival Time

- Waiting Time (W.T) is Time Difference betn turn around time and burst time.
  Waiting Time = Turn Around Time − Burst Time

- Consider the following table of arrival time and burst time for processes P_1, P_2, P_3 and P_4.

Process No	Arrival Time	Burst Time	Completion Time	Turn Around Time	waiting Time.
P_1	1	3			
P_2	2	4			
P_3	1	2			
P_4	4	4			

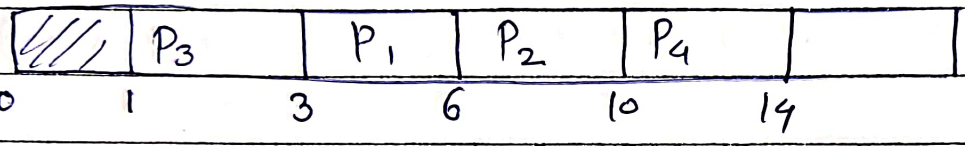
Gantt Chart



Teacher's Signature

Process No.	Arrival Time	Burst Time	completion Time	Turn Around Time	waiting Time
P_1	1	3	6		
P_2	2	4	10		
P_3	1	2	3		
P_4	4	4	14		

Gantt chart



Now, lets waiting time & Turn Around Time

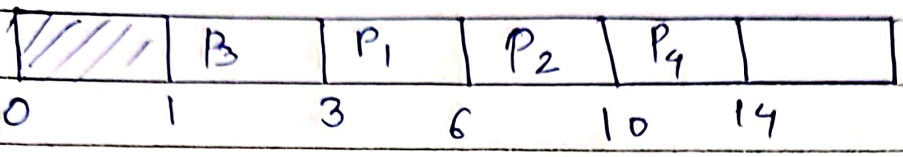
TAT = CT - AT

Process No.	Arrival Time	Burst Time	completion Time	Turn Around Time	Waiting time
P_1	1	3	6	5	
P_2	2	4	10	8	
P_3	1	2	3	2	
P_4	4	4	14	10	

Teacher's Signature _____

Page No :
Date : \ \

Gantt Chart

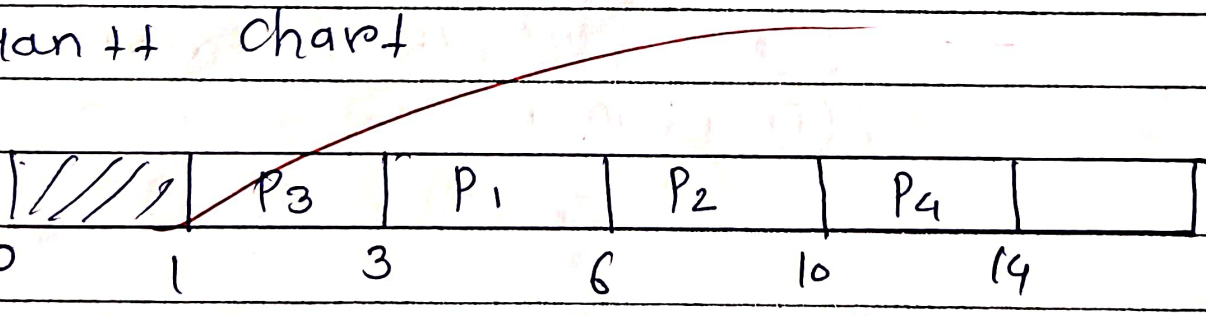


Time →

WT = TAT - BT

Process No	Arrival Time	Burst Time	Completion Time	Turn Around Time	waiting Time
P_1	1	3	6	5	2
P_2	2	4	10	8	4
P_3	1	2	3	2	0
P_4	4	4	14	10	6

Gantt Chart



Time →

Teacher's Signature

Page No.
Date : \ \

Program :-

```
#include<stdio.h>
int main()
{
// matrix for storing Process Id, Burst
// Time, Average Waiting Time & Average
// Turn Around Time.
 int A[100][4];
int i,j, n, total=0, index, temp;
float avg_wt, avg_tat;
printf("Enter number of process:");
scanf("%d", &n);
print("Enter Burst Time:\n");
// User Input Burst Time and alloting Process
for(i=0; i<n; i++) {
    printf("P%d:", i+1);
    scanf("%d", &A[i][1]);
    A[i][0] = i+1;
}
// sorting Process according to their Burst Time
for(i=0; i<n; i++){
    index = i;
```

Teacher's Signature

Output :-

```
Enter no. of process : 4
Enter Burst Time :-
P1 : 3
P2 : 4
P3 : 2
P4 : 4
```

P	BT	WT	TAT
P3	2	0	2
P1	3	2	5
P2	4	5	9
P4	4	9	13

```
Average Waiting Time := 4.000000
Average Turnaround Time : 7.250000.
```

Page No :
Date : / /

Name of Practical

```
for (j=i+1 ; j<n ; j++)
    if (A[j][1] < A[index][1])
        index = j;
temp = A[i][1];
A[i][1] = A[index][1];
A[index][1] = temp;

temp = A[i][0];
A[i][0] = A[index][0];
A[index][0] = temp;
}

A[0][2] = 0;

// Calculation of waiting Times
for (i=1 ; i<n ; i++) {
    A[i][2] = 0;
    for (j=0 ; j<i ; j++)
        A[i][2] += A[j][1];
    total += A[i][2];
}
avg_wt = (float) total / n;
total = 0;
printf ("P    BT    WT    TAT\n");
```

Teacher's Signature

Page No :
Date : \ \

Name of Practical

```
//Calculation of Turn Around Time and
        Printing Time
// data.
for (i=0; i<n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
Printf("P %d  %od  %od  %od\n", A[i][0],
        A[i][1], A[i][2], A[i][3]);
}
   avg_tat = (float) total/n;
Printf("Average Waiting Time = %f", avg_wt);
Printf("\n Average Turnaround Time = %f", avg_
                                        tat);
}
```

Conclusion :-

Thus we have studies the write a program in C to implement SJF scheduling Algorithm.

Teacher's Signature _____

Page No :
Date : \ \

Practical No. 6.

Name of Practical

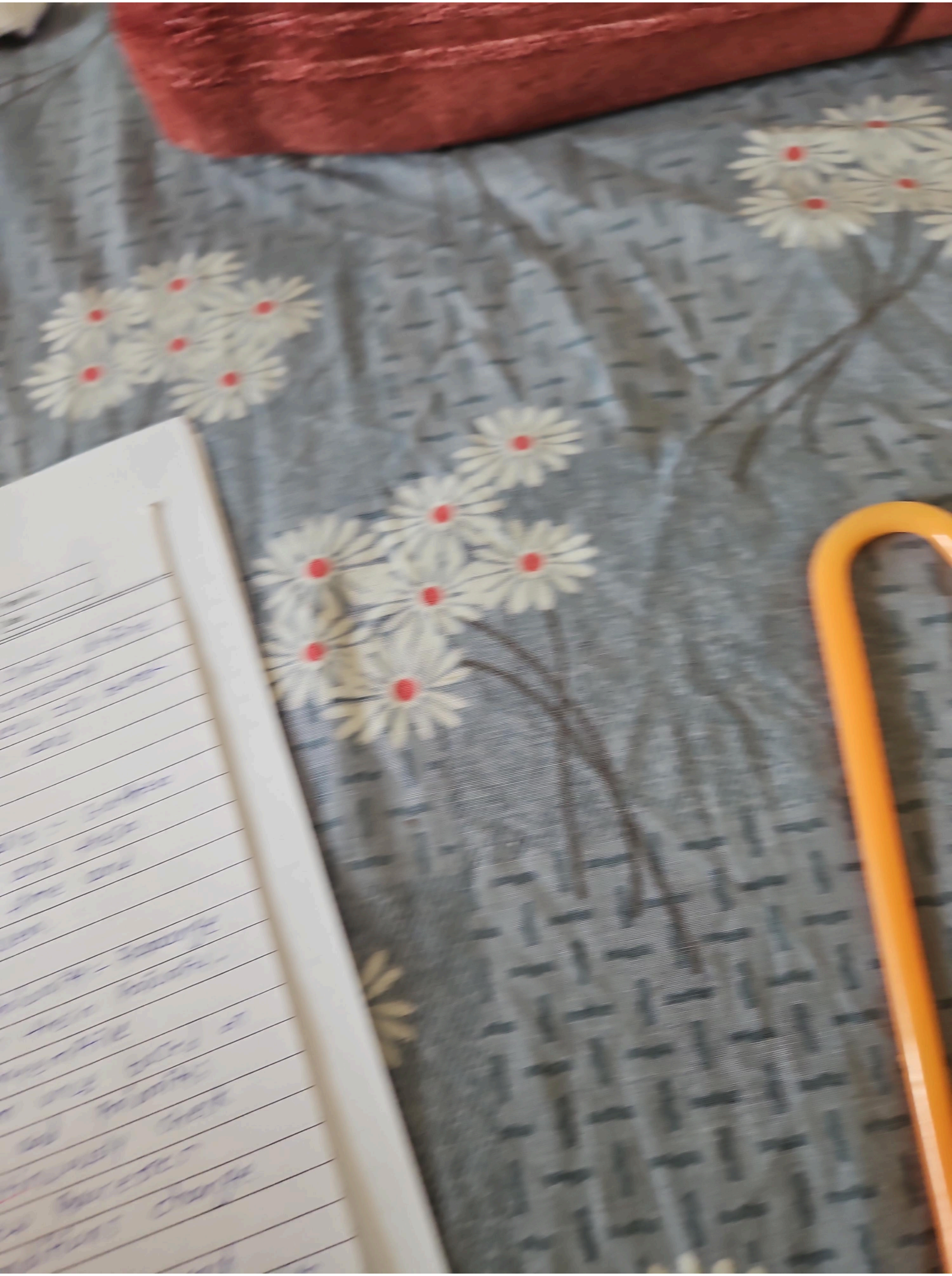
Aim :- write a program in C to implement PRIORITY scheduling Algorithm.

Theory :-

In preeptive priority scheduling, the process with the highest priority is always executed first, even if it means interrupting a currently running Process. If a higher priority process arrives while a lower priority process is executing, the lower priority process is preempted & the higher priority process is given the CPU.

Process	Arrival time	Burst Time	Priority
P_1	0	9	2
P_2	1	2	1
P_3	2	3	3
P_4	4	1	2

Teacher's Signature



Page No :
Date : \ \

Name of Practical

- Define process structure :- first, define a structure to hold the necessary process details such as process ID, burst time, priority, waiting time, and turnaround time.

- ~~Imput~~ Input process Details :- Gather the number of processes and their respective details (burst time and priority) from the user.

- sorting Processes by priority :- Arrange the processes based on their priority. If implementing non-Preemptive scheduling, sort them once based on their arrival time and priority; for preemptive, continuously check and reorder as new processes arrive or as conditions change.

- Calculate Waiting Time :- for each process, calculate the waiting time. This is the total time spent in the system before it gets the CPU.

Teacher's Signature

Page No :
Date : \ \

Name of Practical

- Calculate Turnaround Time :- for each process, ~~to~~ determine the turnaround time, which is the total time from when the process arrives to when it completes.

- Execute processes :- According to the scheduling (preemptive or non-preemptive), execute the processes. In non-preemptive, each process runs to completion once it starts. In preemptive, monitor if a higher priority process arrives and preempt accordingly.

- output Results : Display the results, showing the process ID along with its waiting time and turnaround time.

Teacher's Signature

Page No :
Date : \ \

Name of Practical

program :-

```
#include <stdio.h>

// Define the maximum number of processes
#define MAX 50

typedef struct {
int Process_id, arrival_time, burst_time,
Priority;
int waiting_time, turnaround_time;
} Process;

// function to sort Processes by priority
void sortProcessesByPriority(Process proc[],
                             int n) {
  Process temp;
  for(int i=0; i<n-1; i++){
    for(int j=0; j<n-i-1; j++){
if(proc[j].Priority > proc[j+1].priority){
      temp = proc[j];
      proc[j] = proc[j+1];
      proc[j+1] = temp;
    }
  }
}
```

Teacher's Signature _____

Page No :
Date : \ \

Name of Practical

```
// Function to calculate waiting time and
   turnaround time

void calculateTimer(Process proc[], int n) {
proc[0].waiting_time = 0; // first process has
                             no waiting time
proc[0].turnaround_time = proc[0].burst_time;

for(int i=1; i<n; i++) {
 proc[i].waiting_time = proc[i-1].waiting_time
                        + proc[i-1].burst_time;
 proc[i].turnaround_time = proc[i].waiting_
      time + proc[i].burst_time;
    }
}

// function to ~~display~~ the processes along
              with all details
~~void~~ displayProcesses(Process proc[], int n) {
printf("Process ID\t Priority \t Burst Time\t
waiting Time\t Turnaround Time\n");
for(int i=0; i<n; i++) {
```

Teacher's Signature _____

Output :-

```
Enter total no. of processes :- 4
Enter Process ID, Priority, Burst Time:
 4  2  5
 2  1  3
 3  3  8
 4  2  6
```

Process ID	Priority	Burst Time	Waiting time
2	1	3	0	3
1	2	5	3	8
4	2	6	8	14
3	3	8	14	22

Page No :
Date : / /

Name of Practical

```
        printf("%d\t\t%d\t\t%d\t\t%d\n",
        proc[i].process_id, proc[i].priority,
        proc[i].burst_time, proc[i].waiting_time,
        proc[i].turnaround_time);
    }
}

int main() {
    int n;
    Process proc[MAX];

    // Input number of processes
    printf("Enter total number of processes:");
    scanf("%d", &n);
    // Input process details
    for(int i=0; i<n; i++) {
        printf("Enter Process ID, Priority, Burst
            Time:");
        scanf("%d %d %d", &proc[i].process_id, &
        proc[i].priority, &proc[i].burst_time);
        proc[i].arrival_time = 0; // Assuming all
                    processes arrive at time 0.
    }
```

Teacher's Signature

Page No :
Date : \ \

Name of Practical

```
// Sort processes by priority
sortProcessesByPriority(proc, n);

// Calculate waiting time and turnaround time
calculateTimes(proc, n);

// Display processes along with all calculated times
displayProcesses(proc, n);

return 0;
}
```

conclusion :-

Thus we have studied the write a program in C to implement PRIORITY Scheduling Algorithm.

Teacher's Signature

EXPERIMENT : No.

Date : Page :

Program :-

```
#include <stdio.h>
int main()
{
int n, at[10], bt[10], wt[10], tat[10], ct[10],
                                    sum, i, j, k;
float totaltat = 0, totalwt = 0;
Printf("\n enter the total number of
                    Processes:");
Scanf ("%d", &n);
Printf("\n Enter The Process Arrival Time &
          Burst Time \n");
for (i=0; i<n; i++);
{
  Printf ("Enter Arrival time of Process
                                [%d]:", i+1);
Scanf ("%d", &at[i]);
Printf("Enter Burst time of Process
          [%d]:", i+1);
Scanf ("%d", &bt[i]);
}
```

Teacher's Sign : _____

Practical No. 7

Write a Program in C to implement ROUND ROBIN Scheduling Algorithm

Theory

The round-robin scheduling algorithm equally distributes each resource and processes each division in a circular sequence without giving any consideration to priority. The terms "time quantum" and "time slice" are used to distribute all resources equally. If a process is finished, it is removed from the ready queue in this round-robin method; otherwise, it will return to the ready queue for the remaining execution. The running queue's processes that have arrived and are awaiting execution are in the ready queue. The processes that originated from the ready queue are carried out using the running queue.

Round Robin CPU Scheduling

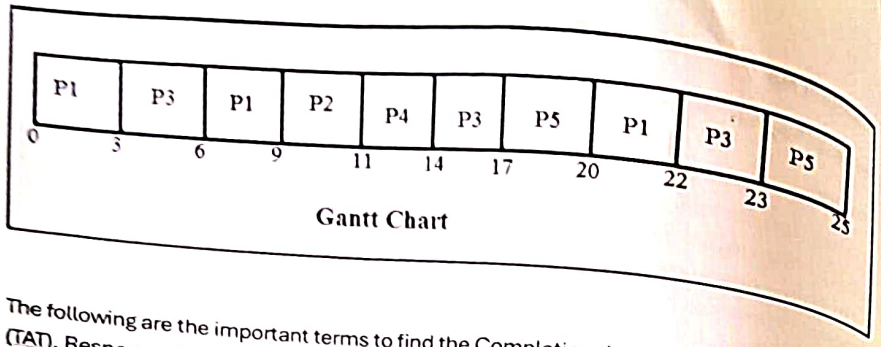
Let's understand the concepts of Round Robin with an example. Suppose we have five processes P1, P2, P3, P4 and P5

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	8
P2	5	2
P3	1	7
P4	6	3
P5	8	5

Now we have to create the **ready queue** and the **Gantt chart** for Round Robin CPU Scheduler.

Ready queue: P1, P3, P1, P2, P4, P3, P5, P1, P3, P5

Here is the Gantt chart:



Gantt Chart

The following are the important terms to find the Completion time, Turn Around Time (TAT), Response Time (RT) and Waiting Time (WT).

1. **Completion Time**: It defines the time when processes complete their execution.
2. **Turn Around Time**: It defines the time difference between the completion time (CT) and the arrival time (AT).
   Turn Around Time (TAT) = Completion Time (CT) - Arrival Time (AT)
3. **Waiting Time**: It defines the total time between requesting action and acquiring the resource.
   Waiting Time (WT) = Turn Around Time (TAT) - Burst Time (BT)
4. **Response Time**: It is the time that defines at which time the system response to a process.

Process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	0	8	22	22	14	0
P2	5	2	11	6	4	4
P3	1	7	23	22	15	2
P4	6	3	14	8	5	5
P5	8	5	25	17	12	9

Program

```
// round robin scheduling program in c

#include<stdio.h>


void main()

{

  int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

  float avg_wt, avg_tat;

  printf(" Total number of process in the system: ");

  scanf("%d", &NOP);

  y = NOP;


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

  {

    printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

    printf(" Enter Arrival time: \t");

    scanf("%d", &at[i]);

    printf(" \nEnter Burst time: \t");

    scanf("%d", &bt[i]);

    temp[i] = bt[i];

  }


  printf("Enter the Time Quantum for the process: \t");

  scanf("%d", &quant);

  printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

  for(sum=0, i = 0; y!=0; )

  {

  if(temp[i] <= quant && temp[i] > 0)
```

```
        {
          sum = sum + temp[i];
          temp[i] = 0;
          count=1;
        }
        else if(temp[i] > 0)
        {
          temp[i] = temp[i] - quant;
          sum = sum + quant;
        }
        if(temp[i]==0 && count==1)
        {
          y--;
          printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-
bt[i]);
          wt = wt+sum-at[i]-bt[i];
          tat = tat+sum-at[i];
          count =0;
        }
        if(i==NOP-1)
        {
          i=0;
        }
        else if(at[i+1]<=sum)
        {
          i++;
        }
      }
      else
        {
          i=0;
        }
      }
    }
```

Output

Conclusion

By this program c to implement Round Robin Scheduling Algorithm

Output:-

```
Total Number of process in the system: 2
Enter the Arival & Burst time of the process[1]
Enter Arrival time: 0
Enter Burst time: 5

Enter the Arrival & Burst time of the
process [2]
Enter Arrival time: 1
Enter Burst time: 3

Enter the time Quantum for the process: 2
```

Process No	Burst Time	TAT	WT
Process No[1]	5	8	3
Process No[2]	3	7	4

```
Average Waiting Time: 3.50
Average Turnaround Time: 7.50.
```

Practical No. 8

Write a Program in C to implement Page Replacement Algorithm Using FIFO

Theory

FIFO Page Replacement Algorithm in C

A page replacement algorithm is required in an operating system that uses paging to manage memory in order to determine which page needs to be replaced when a new page is received.

A page fault occurs when an active application tries to access a memory page that is loaded in virtual memory but not physically in the system. Page errors occur because actual physical memory is substantially less than virtual memory. In the event of a page fault, the operating system might have to swap out one of the current pages for the one that is now required. Different page replacement algorithms offer various suggestions for selecting the appropriate replacement page. All algorithms strive to lower the number of page faults.

Program

```
#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

      printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

      printf("\n ENTER THE PAGE NUMBER :\n");

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

      scanf("%d",&a[i]);

      printf("\n ENTER THE NUMBER OF FRAMES :");

      scanf("%d",&no);

for(i=0;i<no;i++)

      frame[i]= -1;

            j=0;

            printf("\tref string\t page frames\n");
```

```
for(i=1;i<=n;i++)
        {
            printf("%d\t\t",a[i]);
            avail=0;
            for(k=0;k<no;k++)
if(frame[k]==a[i])
                avail=1;
            if (avail==0)
            {
                frame[j]=a[i];
                j=(j+1)%no;
                count++;
                for(k=0;k<no;k++)
                printf("%d\t",frame[k]);
}
            printf("\n");
}
        printf("Page Fault Is %d",count);
        return 0;
}
```

Output

Conclusion

By this program C to implement Page Replacement Algorithm using Fifo

Output:-

Number of Pages: 12

Page Reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Number of Frames: 3.

Practical No. 9

Write a Program in C to Implement Page Replacement Algorithm Using LRU

Theory

LRU Page Replacement Algorithm C

In this article, we will discuss the ***LRU Page replacement*** in C with its pseudocode.

Any page replacement algorithm's main goal is to decrease the amount of ***page faults.*** The LRU page replacement algorithm substitutes a new page for the least recently used page when one is needed. This approach is predicated on the idea that, among all pages, the least recently used page won't be utilized for a very long time. It is a well-liked and successful page replacement method.

Program

```
#include <stdio.h>

// Function to find the index of the least recently used page in frames
int findLRU(int time[], int n) {
  int i, minimum = time[0], pos = 0;
  for (i = 1; i < n; ++i) {
    if (time[i] < minimum) {
      minimum = time[i];
      pos = i;
    }
  }
  return pos;
}

int main() {
  int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], i, j, pos,
faults = 0;
```

```
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter reference string: ");
    for (i = 0; i < no_of_pages; ++i) {
      scanf("%d", &pages[i]);
    }

    for (i = 0; i < no_of_frames; ++i) {
      frames[i] = -1;
      time[i] = 0; // Initialize the time array to 0
    }

    for (i = 0; i < no_of_pages; ++i) {
      int page = pages[i];
      int page_found = 0;

      // Check if the page is already in frames
      for (j = 0; j < no_of_frames; ++j) {
        if (frames[j] == page) {
          time[j] = counter++; // Update the time of use
          page_found = 1;
          break;
        }
      }

      // If the page is not in frames, find the LRU page to replace
      if (!page_found) {
        pos = findLRU(time, no_of_frames);
        frames[pos] = page; // Replace the LRU page
        time[pos] = counter++; // Update the time of use
        faults++;
      }

      // Print the current state of frames
      printf("Current frames: ");
      for (j = 0; j < no_of_frames; ++j) {
        printf("%d\t", frames[j]);
      }
      printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", faults);

    return 0;
}
```

Output

Conclusion By this program c to implement Page Replacement Algorithm using LRU

Output:-

```
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 1 3 0 3 5 3
Current Frames: 1  -1  -1
Current Frames: 1   3  -1
Current Frames: 1   3   0
Current Frames: 1   3   0
Current Frames: 1   3   5
Current Frames: 1   3   5
Total Pages Faults = 4
```

Practical No. 10

Write a Program in C to implement Page Replacement Algorithm Using OPTIMAL

Theory

Optimal Page Replacement Algorithm in C

The ***Optimal Page Replacement*** Algorithm, often known as ***Belady's Algorithm***, is a page replacement algorithm. It is used in operating systems to determine which page to evict from memory when a page fault occurs. It is different from some other algorithms like ***FIFO*** or ***LRU***, the Optimal Algorithm is not practical for implementation in a real operating system because it requires knowledge of future page references, which is impossible in practice. However, it serves as an ideal benchmark to measure the performance of other page replacement algorithms.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define NUM_FRAMES 3
#define NUM_PAGES 10

// Function to find the page that will be referenced furthest in the future
int findOptimalPage(int page[], int pageFrames[], int index, int numFrames) {
  int farthest = -1;
  int farthestIndex = -1;

  for (int i = 0; i < numFrames; i++) {
    int j;
    for (j = index; j < NUM_PAGES; j++) {
      if (pageFrames[i] == page[j]) {
```

```
if (j > farthest) {
farthest = j;
farthestIndex = i;
}
break;
        }
      }
      if (j == NUM_PAGES) {
return i;
      }
    }
    if (farthestIndex == -1) {
      return 0;
    }
    return farthestIndex;
}
int main() {
    int pageReferences[NUM_PAGES] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3};
    int pageFrames[NUM_FRAMES];
    bool isPageInFrame[NUM_FRAMES];
    int pageFaults = 0;
    for (int i = 0; i < NUM_FRAMES; i++) {
pageFrames[i] = -1;
isPageInFrame[i] = false;
    }
printf("Page Reference String: ");
    for (int i = 0; i < NUM_PAGES; i++) {
printf("%d ", pageReferences[i]);
    }
    printf("\n");
    for (int i = 0; i < NUM_PAGES; i++) {
      int page = pageReferences[i];
      if (!isPageInFrame[page]) {
        int pageToReplace = findOptimalPage(pageReferences, pageFrames, i + 1,
NUM_FRAMES);
pageFrames[pageToReplace] = page;
isPageInFrame[pageToReplace] = true;
```

```
            pageFaults++;

            printf("Page %d loaded into frame %d\n", page, pageToReplace);
        }
    }

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;
}
```

Output

Conclusion By this program c to implement page replacement Algorithm using OPTIMAL

Output :-

Page reference string :- 7 0 1 2 0 3 0 4 2 3

Page 7 loaded into frame 0
Page 0 loaded into frame 1
Page 1 loaded into frame 2
Page 2 loaded into frame 1
Page 3 loaded into frame 2
Page 4 loaded into frame 0
Total page faults : 6.