**SRI SIVA SUBRAMANIYA NADAR COLLEGE OF ENGINEERING**

(An Autonomous Institution, Affiliated to Anna University, Chennai)

RAJIV GANDHI SALAI (OMR), KALAVAKKAM – 603110.

LABORATORY RECORD

Name : …………Kanyalakshmi G…………………..

Reg. No : …………195002057………………………..

Dept. : …………IT……… Sem : ……5……… Sec. : ……A…….



**ANNA UNIVERSITY**

**BONAFIDE CERTIFICATE**

Certified that this is the bonafide record of

the practical work done for the

OPERATING SYSTEMS (UIT1512) LAB

By

Name …………………...Kanyalakshmi G……………Register Number ……..…195002057………………

Of Department of …INFORMATION TECHNOLOGY…………….

Sri Sivasubramaniya Nadar College of Engineering,

Kalavakkam, Chennai

During the academic year ………2019 - 2023……..

Staff In-charge Head

Department of …………………………

*Submitted for the University Examination held at Sri Sivasubramaniya Nadar college of Engineering on ………………………………………………………*

**Internal Examiner External Examiner**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX**  Name : \_Kanyalakshmi G\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Reg. No. : \_\_195002057\_\_\_\_\_\_\_\_\_  Sem : \_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Sec. : \_\_\_\_\_\_\_\_\_\_\_A\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |
| Ex. No. | Date of Expt. | Title of the Experiment | Page No. | Signature of the Faculty |
| 1.a | 15-7-2021 | Study of Basic Linux Commands | 5-14 |  |
| 1.b | 22-07-2021 | Shell Programming | 15-23 |  |
| 2 | 5-8-2021 | Simulation of Linux Commands | 24-29 |  |
| 3 | 19-8-2021 | System Calls Programming | 30-33 |  |
| 4 | 26-8-2021 | Implementation of Inter Process Communication(IPC) using a)Shared memory b)Pipes | 34-36 |  |
| 5.a | 2-9-2021 | Implementation of CPU Scheduling Algorithms  i)FCFS  ii)SJF | 37-52 |  |
| 5.b | 9-9-2021 | Implementation of CPU Scheduling Algorithms  i)Priority  ii)Round Robin | 53-68 |  |
| 6 | 16-9-2021 | Creation of Threads and Synchronization Applications | 69-78 |  |
| 7 | 18-9-2021 | Implementation of process Synchronization, Deadlock Avoidance and Detection Mechanisms. | 79-82 |  |
| 8 | 23-9-2021 | Implementation of Dynamic Storage Allocation Schemes. | 83-93 |  |
| 9 | 30-9-2021 | Implementation of Page Replacement Algorithm. | 94-103 |  |
| 10 | 7-10-2021 | Designing a Virtual Memory Manager | 104-106 |  |
| 11 | 21-10-2021 | Implementation of Disk Scheduling Algorithm | 107-111 |  |
| 12 | 23-10-2021 | Implementation of File Allocation and File Organization Methods | 112-127 |  |
| 13 | 2810-2021 | Mini-Project | 128-136 |  |

OS LAB EX 1a

Study of Basic Linux Commands

**Aim:** To give a small description of what Linux is, compare Linux with windows and write description of a list of Linux commands

**WHAT IS LINUX:**

Linux is an Operating system just like windows or mac OS. Linux is the most widely used operating system and can be found almost everywhere, an interesting fact is that Android is powered by Linux OS. It is considered one of the most user friendly, secure, customisable and worry free operating systems, these are the properties that makes Linux a very widely used OS. The most important thing about Linux that sets it apart is the fact that it is completely free unlike any other OS’s and is also an Open source software, therefore enabling the user to run any kinds of programs and tinker with the internal structures and also distribute copies of the customised version to others if they wish to do so. On the whole Linux is an Operating system that enable the users to work freely according to their liking while still being able to support all the operations that are fundamental to an operating system.

**LINUX VS WINDOWS:**

|  |  |
| --- | --- |
| **Linux** | **Windows** |
| is open source | is not open source |
| Is not paid | Is paid |
| Can have access to the source code | Cannot access the source code |
| Is more secure | Is less secure |
| Very highly customisable | Not customisable |
| Command line useful for everyday tasks | Command line not as useful for all tasks |
| Complicated to install | Not complicated to install |
| Full control of all applications even including system apps | Does have certain degree of control but not total autonomy over all apps |
| Very developer friendly | Is also developer friendly but with certain restrictions |

## Aim:

## To study the most frequently used linux commands with their examples given in their following categories:

## • Directory Commands

## • File Commands

## • File Content Commands

## • User Commands

## • Filter Commands

## • Utility Commands

## • Networking Commands

## The commands are:

## 1.pwd: Present working directory/Print working directory

## Used to display location of current working directory.

## Syntax: pwd

## 2. mkdir: make directory

## Used to create a new directory.

## Syntax :mkdir〈Directory Name〉

## 3. ls: list

## List all the files and all sub directories in the current directory.

## If highlighted it is the subdirectory

## Syntax: ls

## 4. cd: change directory

## Change to a particular directory

## Syntax: cd <directory name>

## 5. vi: visual

## Editor to create a new file.

## Syntax : vi 〈Filename.txt〉

## We will get an editor screen. To start writing into the file press the i key and start

## typing the contents of the file.

## To save the file we need to press the Escape key followed by :wq.

## To append to a file we need to type a.

## To exit without saving we need to enter the Escape key followed by :q.

## 6. rmdir: remove directory

## To remove a directory.

## Syntax :rmdir〈Directory Name〉

## Rule 1 We cannot be in the same directory and remove that directory

## Rule 2 The directory to be deleted must be empty

## To delete a directory we need to move to the previous directory

## To move to the previous directory we need to type cd ..

## 7. rm: remove a file

## To remove a file

## Syntax : rm 〈filename.txt〉

## 8. mv: move

## To move a file to a particular directory or rename a file

## Application 1 : Renaming a file.

## Syntax : mv 〈filename1.txt〉〈filename2.txt〉

## Application 2 : Moving a file to a particular directory.

## Syntax : mv 〈filename.txt〉〈Directory path〉

## 9. cat: concatenate

## It reads data from the file and gives their content as output.

## Application 1 : To display contents of file in command line if the file already exists

## Syntax : cat 〈filename.txt〉

## Application 2 : To create a file in the command line if the file does not exist

## Syntax : cat 〉〈filename.txt〉

## Application 3 : To concatenate two files.

## This command will work only if both the files exist

## Syntax : cat 〈filename1.txt〉〈filename2.txt〉

## Application 4 : To concatenate two files and save them in a new file.

## Syntax : cat 〈filename1.txt〉〈filename2.txt〉〉〈filename3.txt〉

## 10. tac:

## Displays the contents of the file in a reversed order.

## Concatenate and print files in reverse.

## Syntax : tac 〈Filename.t3xt〉

## 11. clear:

## Will clear the command line screen.

## Syntax: clear

## 12. cp: copy

## Copy contents of a file to a new file.

## If the destination file doesn’t exist, then first it creates one and

## content is copied to it.

## But if it existed then it is simply overwritten without any warning

## Syntax : cp 〈Existing Filename.txt〉〈New Filename.txt〉

## 13. man: manual

## Prints the manual page of a particular command.

## Syntax : man 〈Command Name〉

## 14. head:

## Displays first 10 lines of the given file

## Syntax : head 〈Filename.txt〉

## To display the first n lines of a file : head -n 〈Filename.txt〉

## 15. tail:

## Displays last 10 lines of the given file

## Syntax : tail 〈Filename.txt〉

## To display the last n lines of a file : tail -n 〈Filename.txt〉

## 16. more:

## Displays the content of the file according to the size of the laptop screen.

## To move to the next page of the output we need to press the space bar key.

## Syntax : more 〈Filename.txt〉

## 17. less:

## To read contents of the file one page at a time.

## Syntax : less 〈Filename.txt〉

## 18. touch:

## To create an empty file using the command line.

## To create a single empty file : touch 〈Filename.txt〉

## To create multiple empty files : touch

## 〈Filename1.txt〉〈Filename2.txt〉〈Filename3.txt〉

## 19. id: Used to display the user id and the group id.

## 20. cut:

## To display parts of a specific column of a particular file.

## Syntax : cut -d 〈delimiter〉 -f〈Column Number〉〈Filename.txt〉

## 21. comm:

## Compare the given files.

## Syntax : comm 〈Filename1.txt〉〈Filename2.txt〉

## 22. who:

## The who command is used to get information about currently logged in user on to

## system.

## who command is used to find out the following information :

## Time of last system boot

## Current run level of the system

## List of logged in users and more.

## Syntax: who <options><filename>

## 23. whoami: Who am i

## It is basically the concatenation of the strings “who”,”am”,”i” as whoami.

## It displays the username of the current user when this command is invoked.

## Syntax: whoami

## 24. chmod: change mode

## The chmod command is used to change the access mode of a file.

## Syntax: chmod<reference><operator><mode> file

## 25. date:

## To display the system date and time.

## Date command is also used to set date and time of the system.

## By default the date command displays the date in the time zone on which unix/linux

## operating system is configured.

## Syntax: date (or) date -u (or) –date=”string”

## Output:

**Pwd:**

****

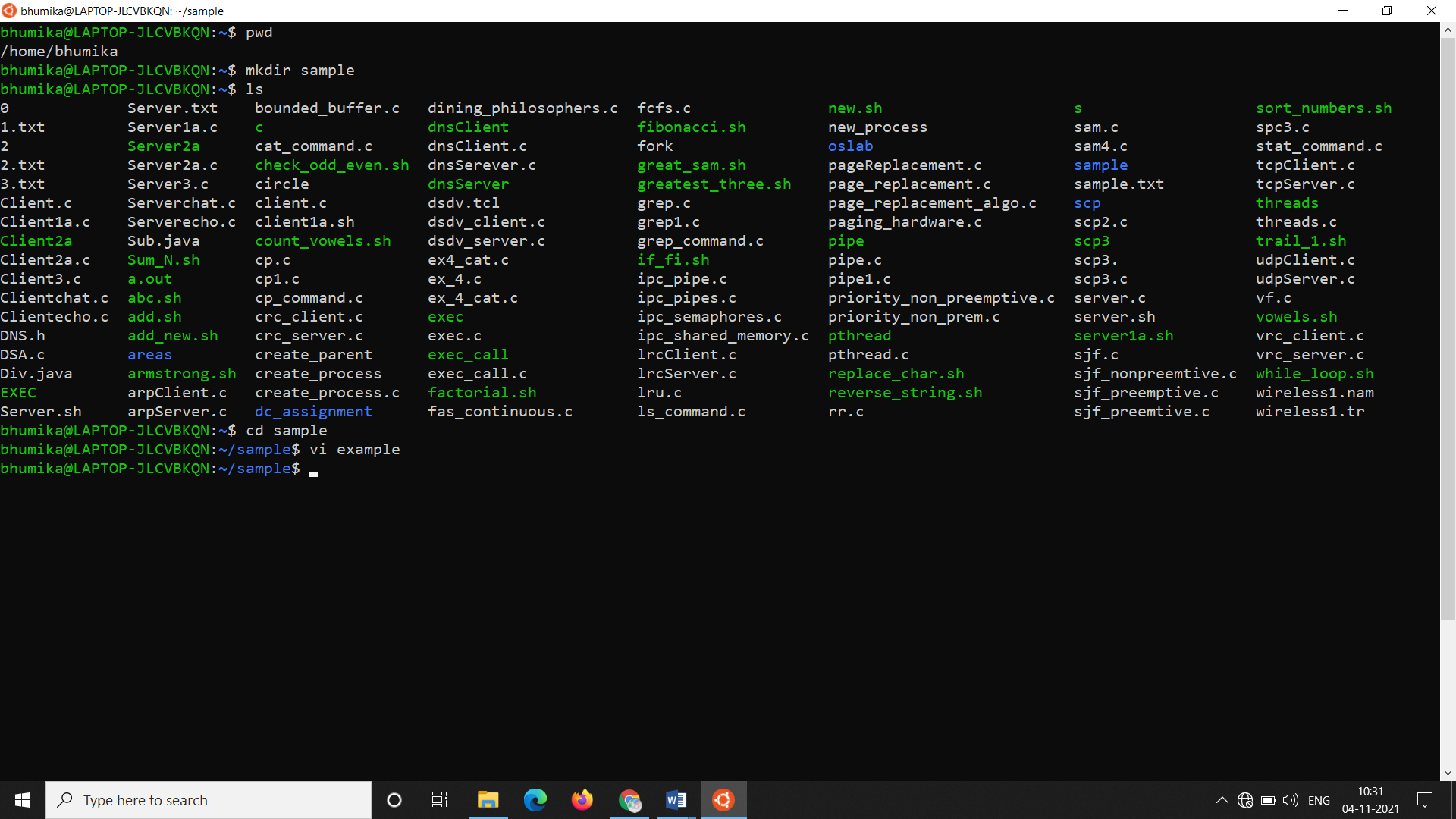
**Mkdir**

****

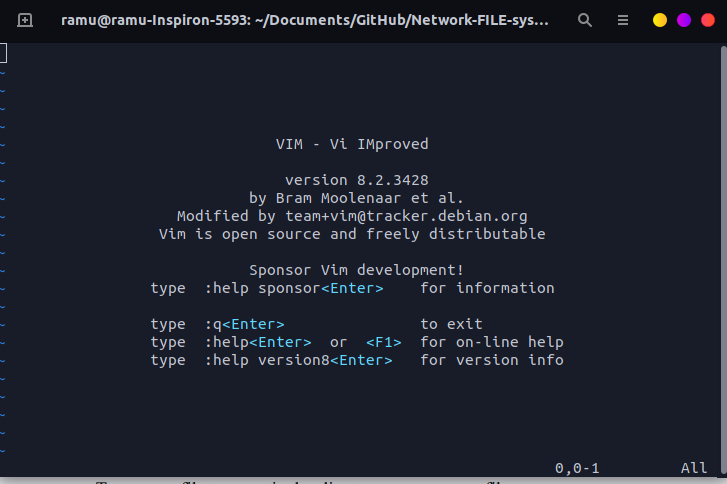
**Ls**

****

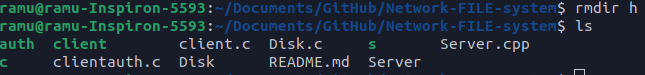
**Cd**



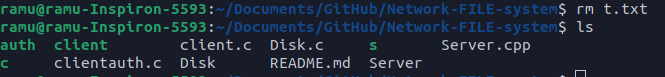
**Vi**

****

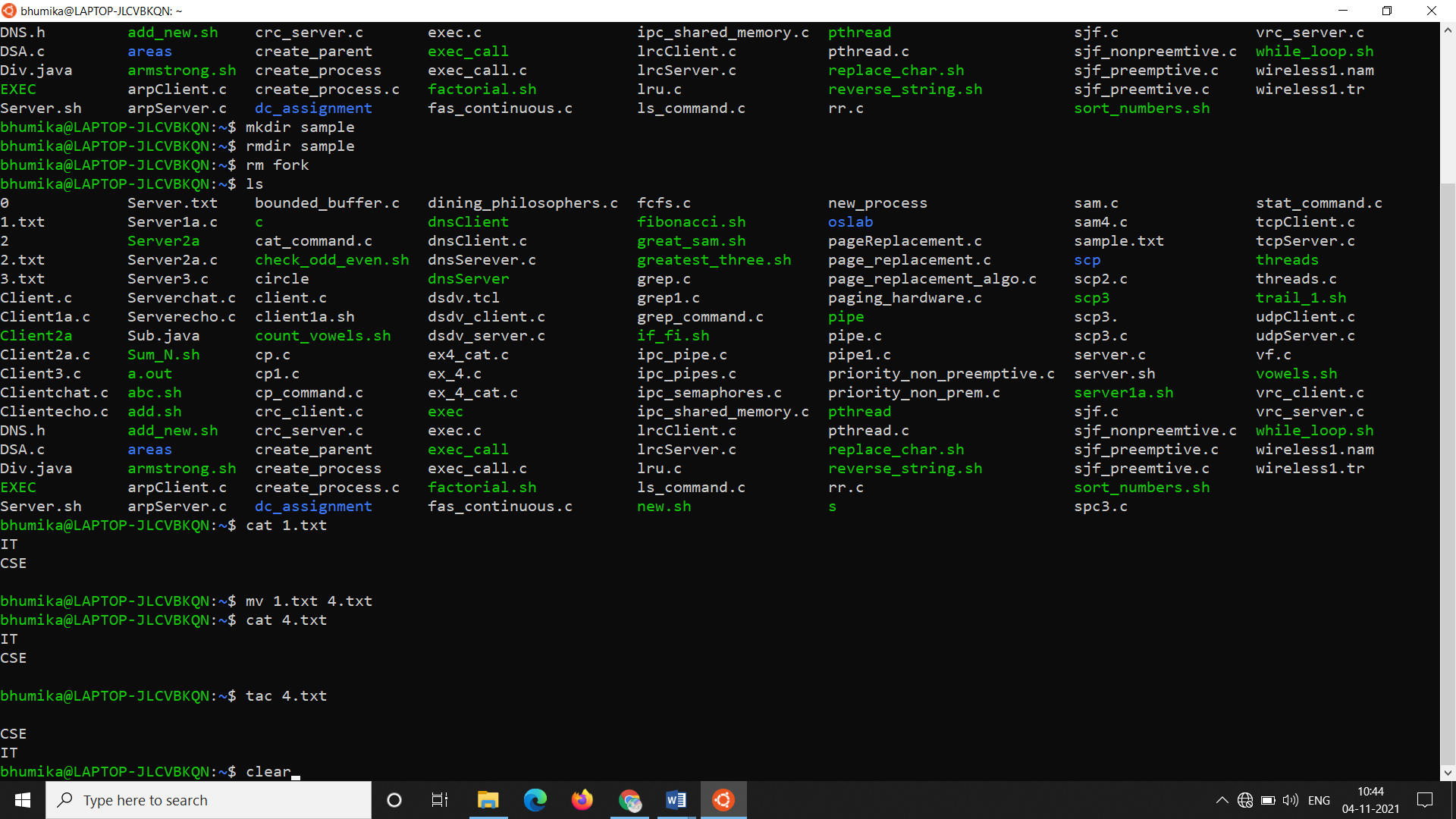
**Rmdir**

****

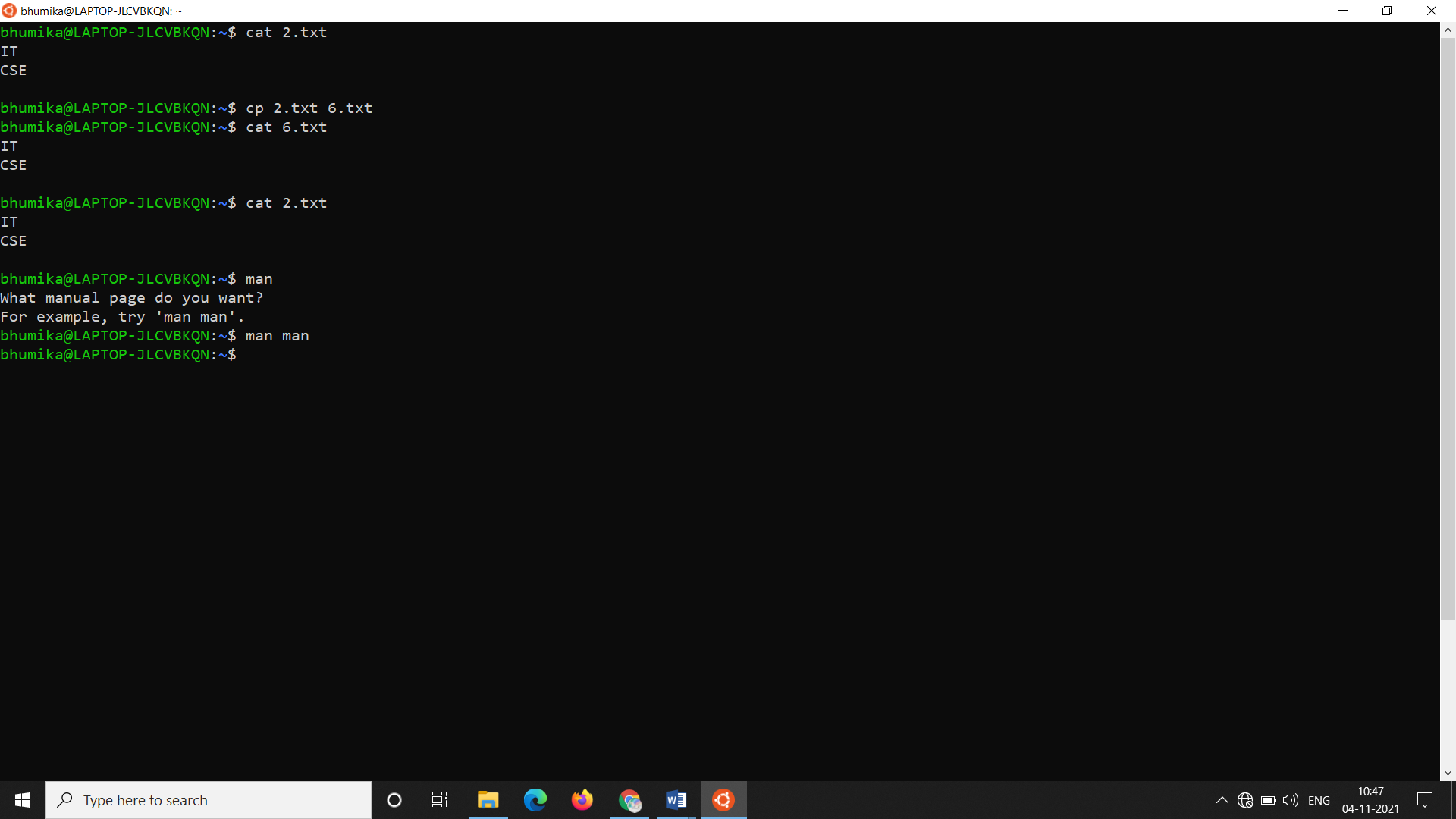
**Rm**

****

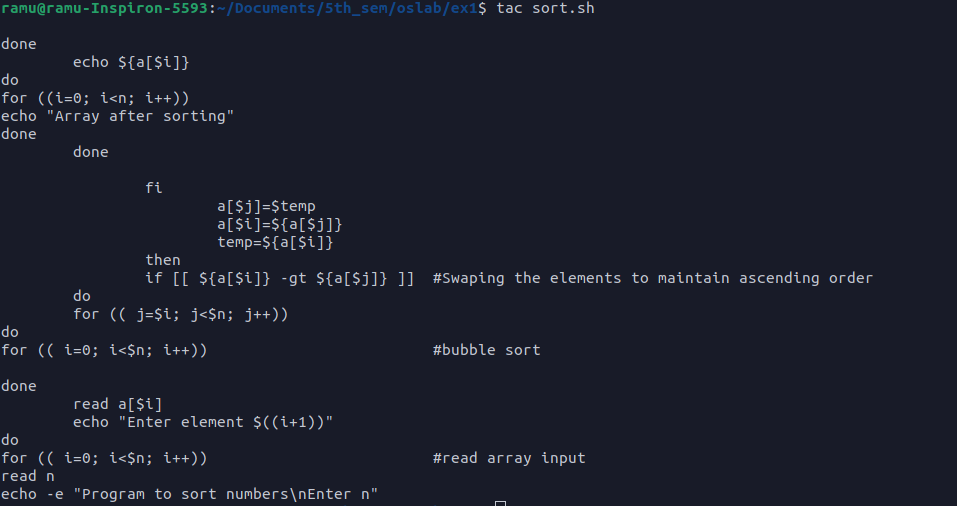
**Mv**

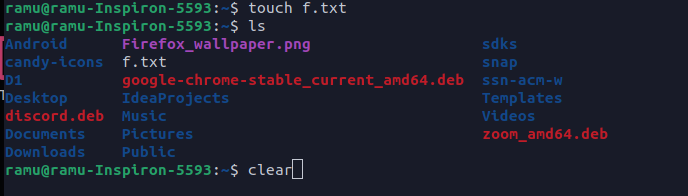


**Cat**

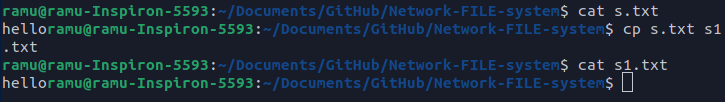


**Tac**

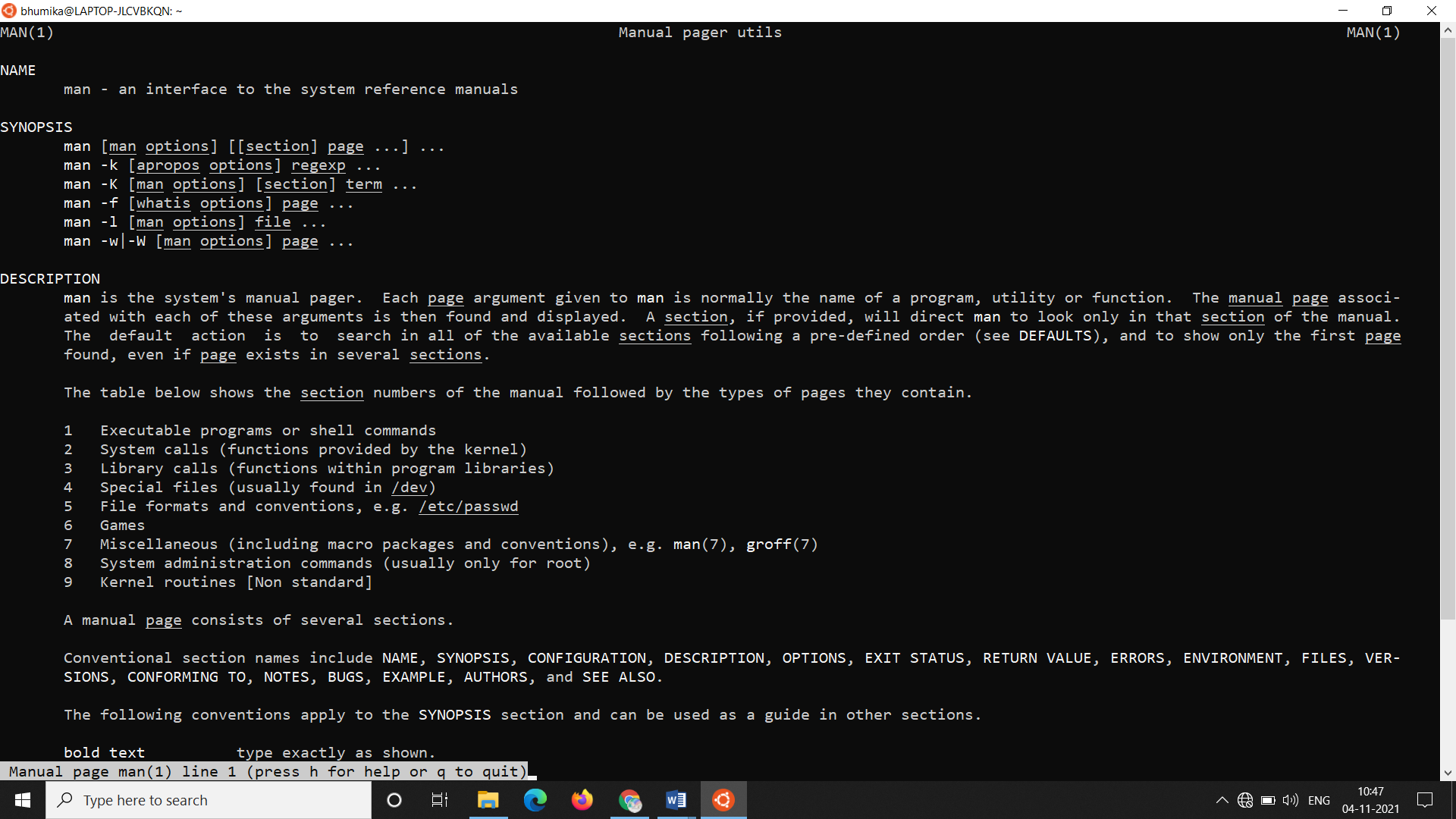
****

**Clear**

**Cp**

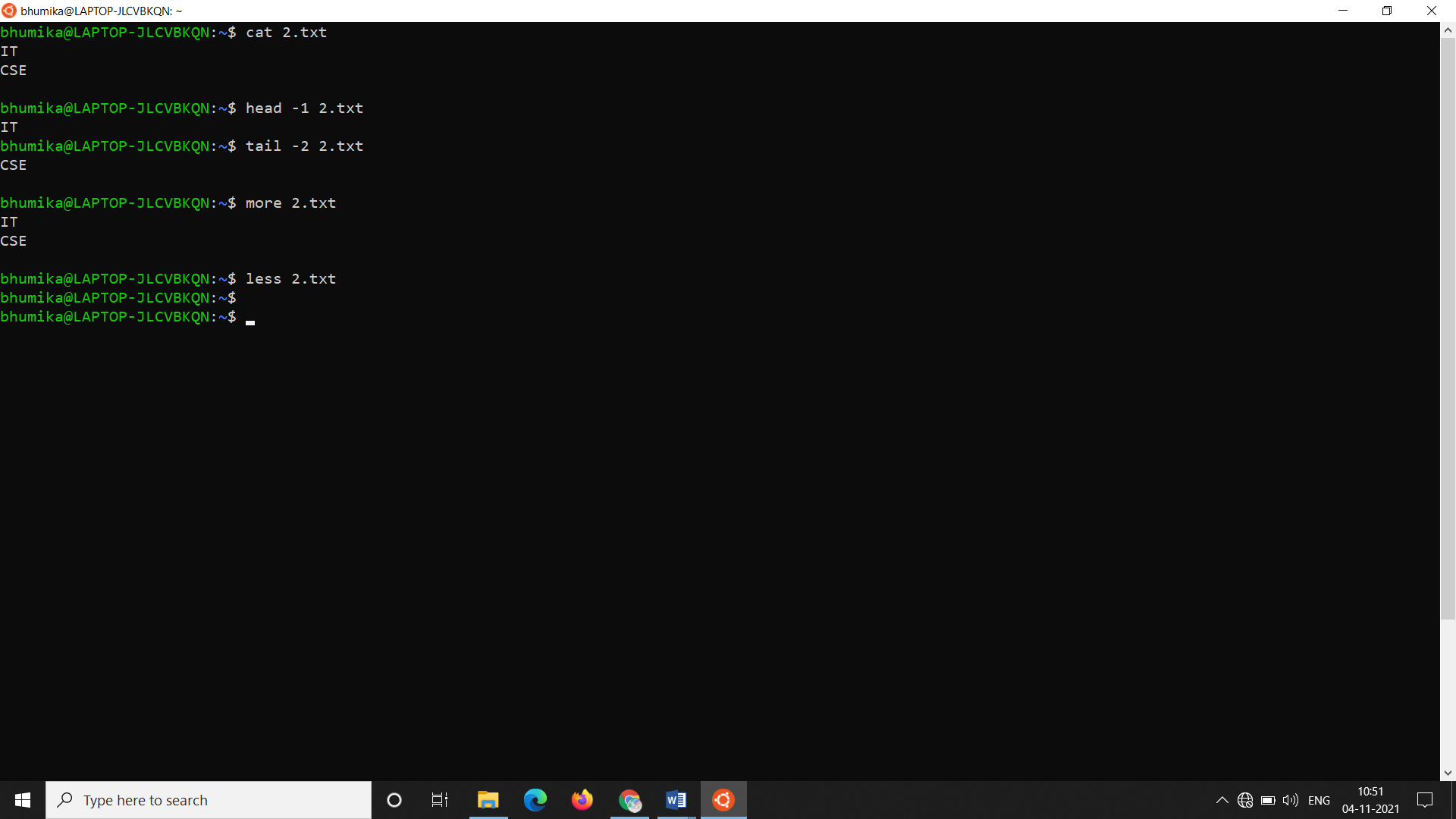
****

**Man**

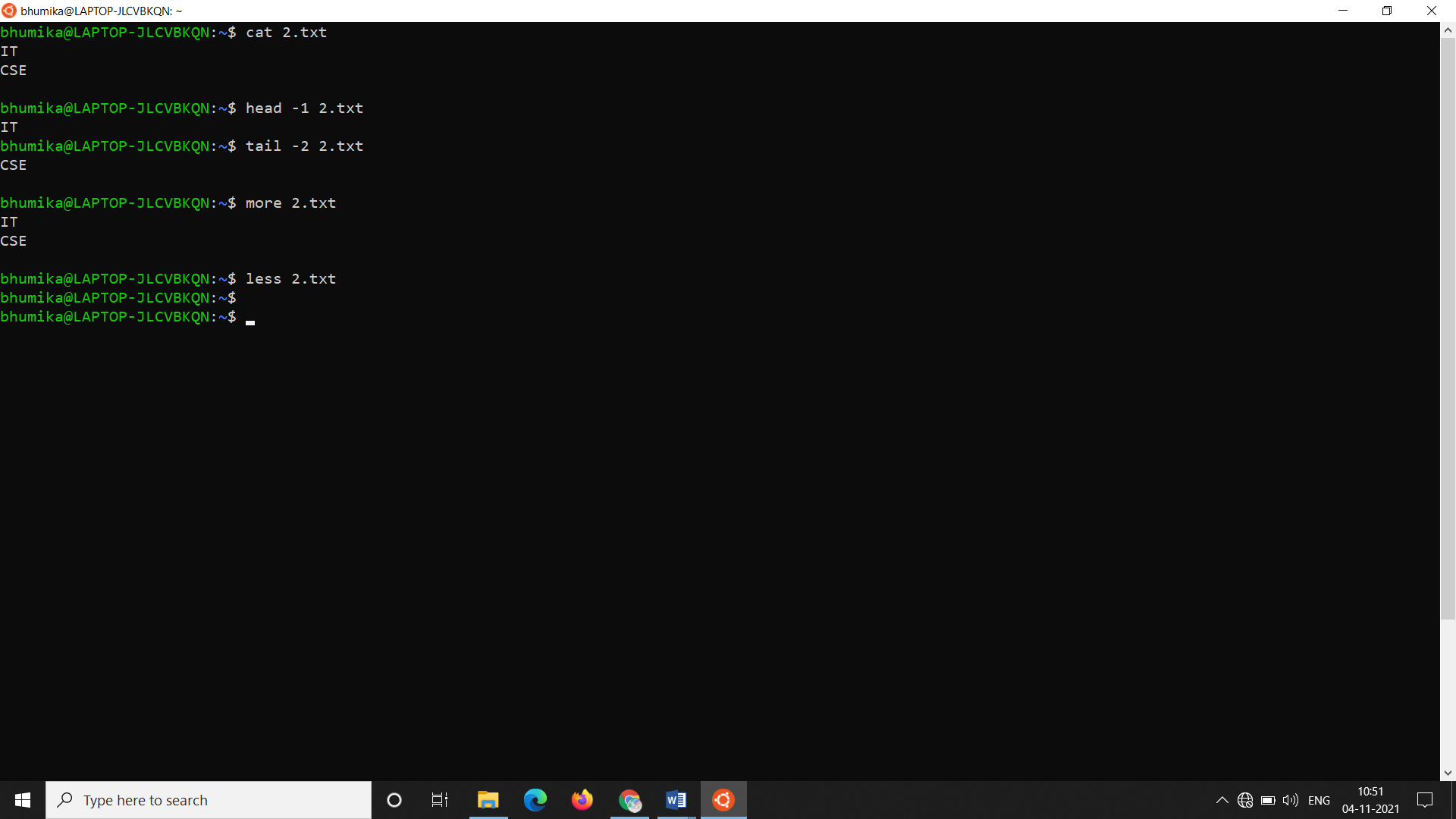
s

**Head**

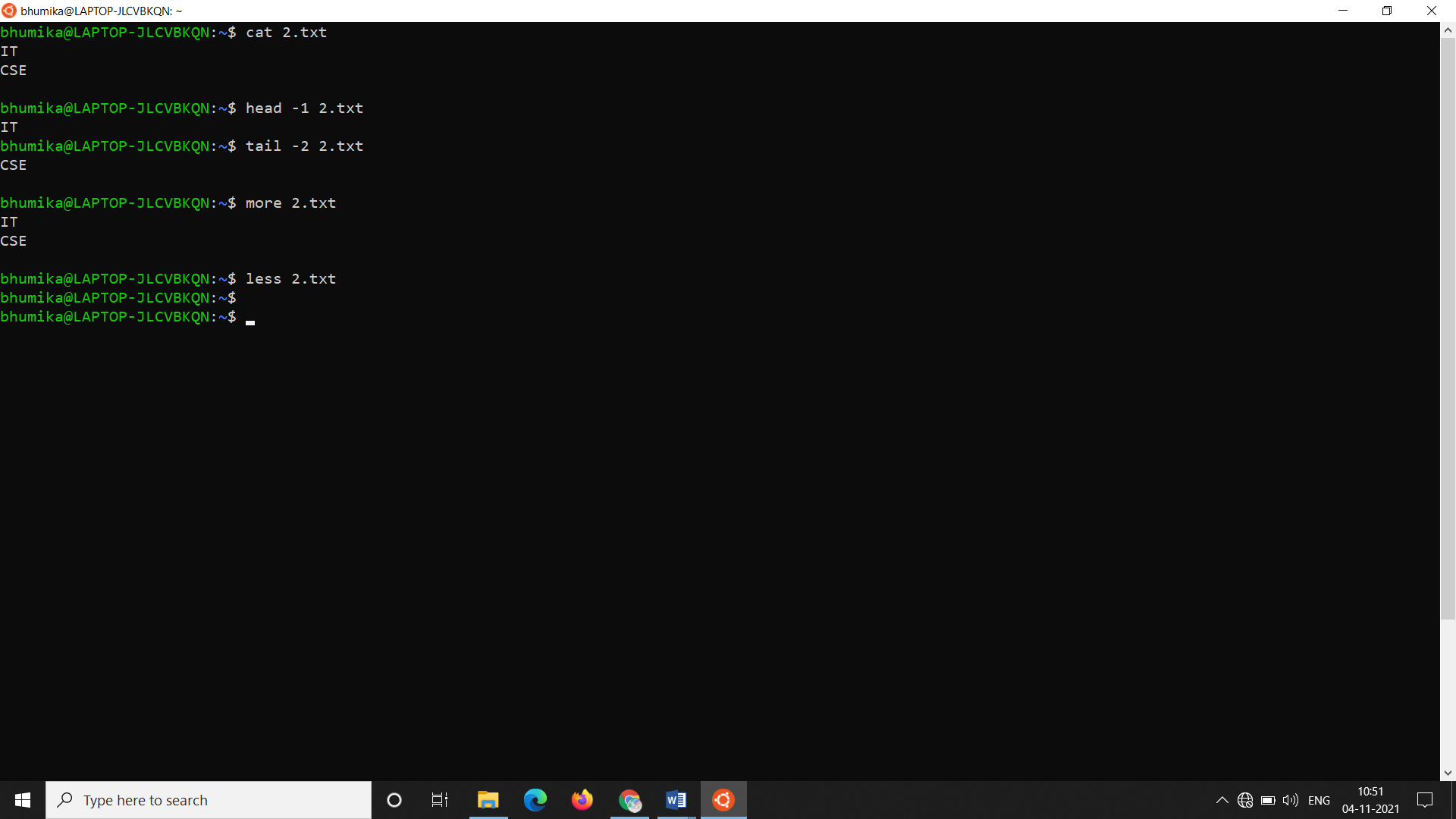
‘



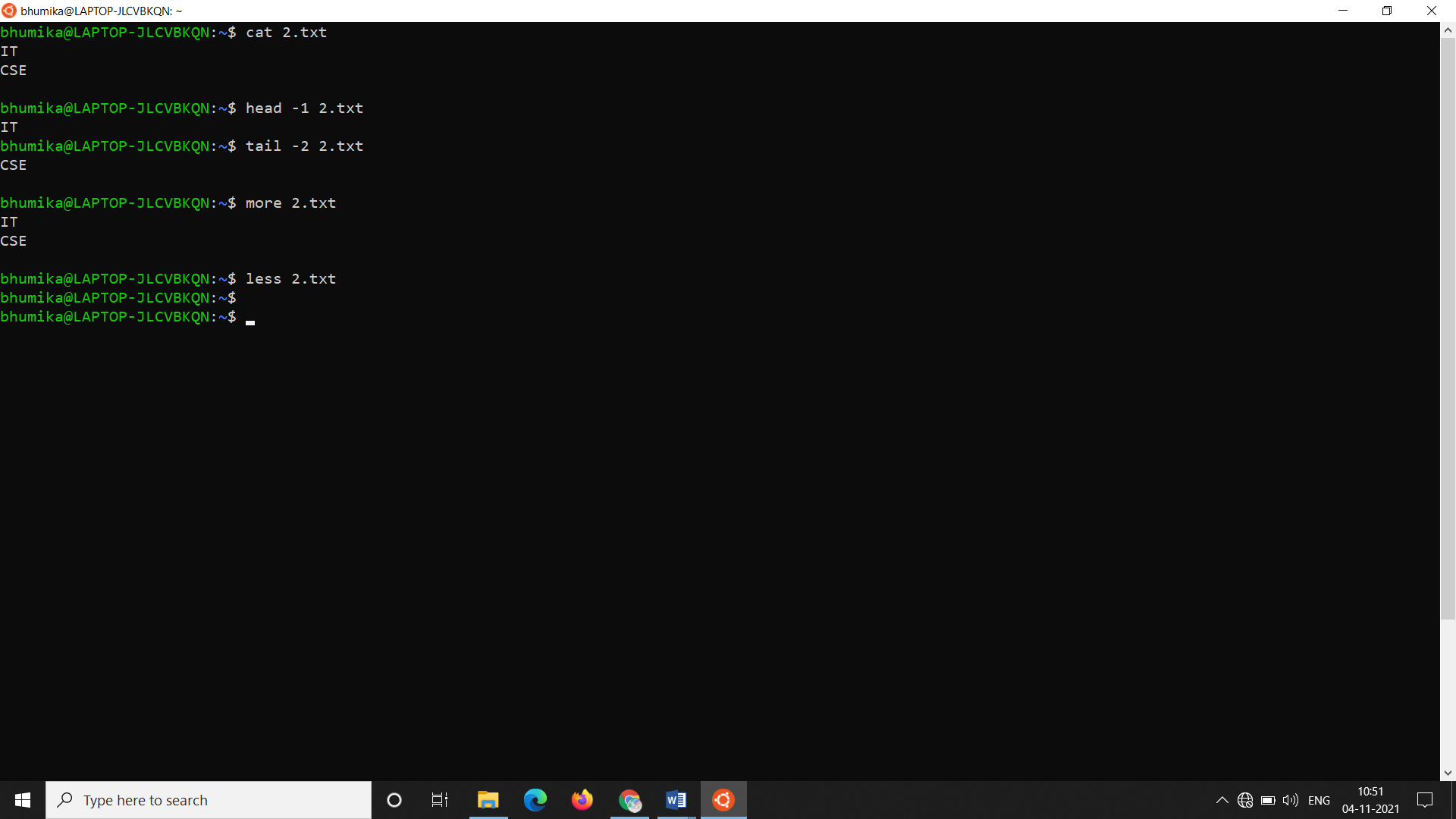
**Tail**



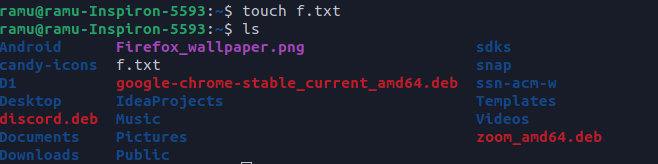
**More**



**Less**



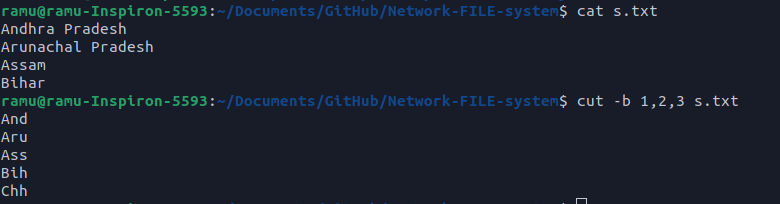
**Touch**

****

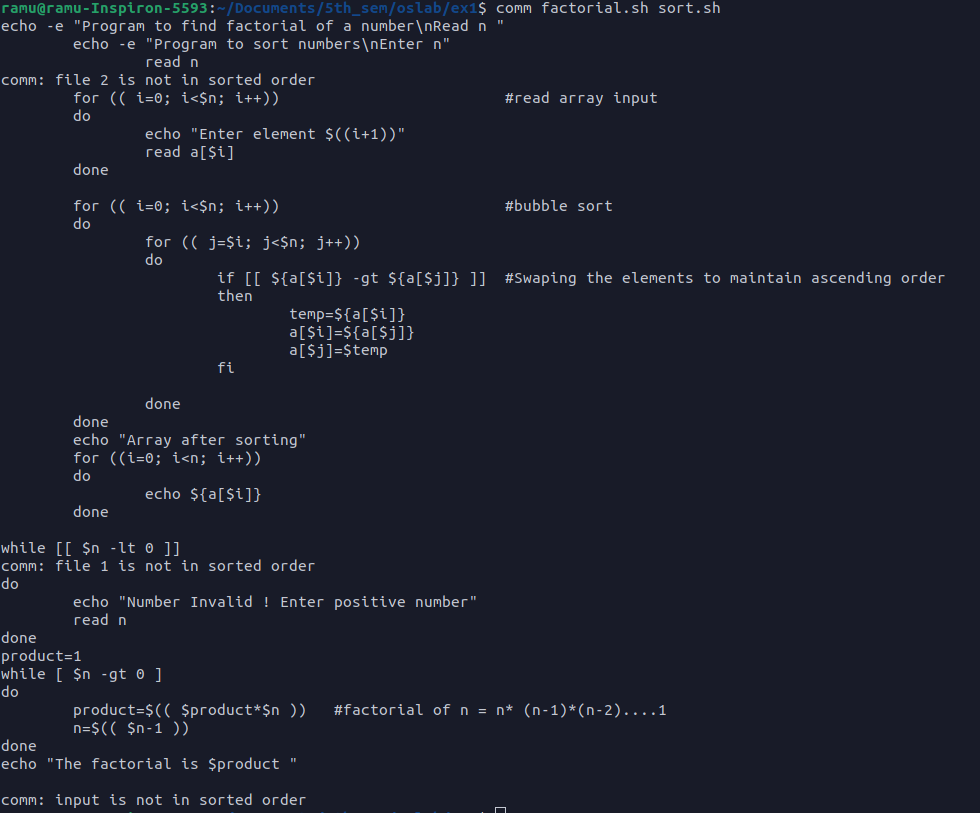
**Id**

****

**Cut**

****

**Comm**

****

**Who**

**Whoami**

****

**Chmod**

**Date**

****

**Result:**

Thus the Linux commands have been tested successfully.

OS LAB EX 1b

Shell Programming

**Q1**

**Aim:** to find the maximum of the given 3 numbers

**Algorithm:**

1. get the three numbers

2. check if no. 1 > no. 2

3. if no. 1 > no. 2, check if no. 1 >no. 2, if true then no.1 is the largest

4. if 2 is false, repeat the above steps 2, 3 with no. 2 and no. 3

5. if no result, print all numbers are equal

**Code:**

#!/bin/bash

echo "give 1st no.: "

read no1

echo "give 2nd no.: "

read no2

echo "give 3rd no.: "

read no3

echo "$no1, $no2, $no3"

if [[ $no1 -gt $no2 ]];

then

if [[ $no1 -gt $no3 ]];

then

echo "$no1 is the greatest"

fi

elif [[ $no3 -gt $no2 ]];

then

if [[ $no3 -gt $no1 ]];

then

echo "$no3 is the greatest"

fi

elif [[ $no2 -gt $no1 ]];

then

if [[ $no2 -gt $no3 ]];

then

echo "$no2 is the greatest"

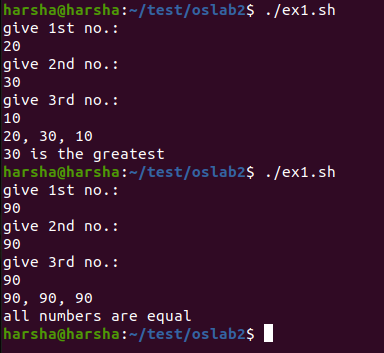
fi

else

echo "all numbers are equal"

fi

**Output:**



**Q2**

**Aim:** to find the sum of the first n natural numbers

**Algorithm:**

1. get the number till what the sum is required

2. allocate a dummy variable to maintain the sum

3. enter a for loop with start as 1 to the number which has been taken from the user, in each

Loop, add the value of loop variable to the dummy variable

4. print the dummy variable

**Code:**

#!/bin/bash

echo "enter the size : "

read n

sum=0

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

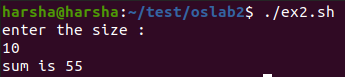
do

sum=$((sum+i))

done

echo "sum is $sum"

**Output:**



**Q3**

**Aim:** to find the factorial of a number

**Algorithm:**

1. get the number for which the factorial is to be found

2. assign a variable to store the end result

3. enter a for loop starting with the number taken and decrementing to 1

4. in each loop multiply the result with the previous loop result and store it

5. print the stored result

6. if input is 1 or 0, print 1 as the result

**Code:**

#!/bin/bash

echo "number: "

read n

if [[ $n -eq 0 ]]

then

echo "factorial is 1"

elif [[ $n -eq 1 ]]

then

echo "factorial is 1"

else

for (( i=$n-1; i>=1; i--))

do

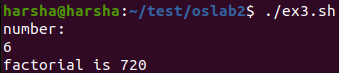
n=$((n\*i))

done

echo "factorial is $n"

fi

**Output:**



**Q4**

**Aim:** to print the Fibonacci series

**Algorithm:**

1. get the value till which the user wants to print the series for

2. declare two variable one with value 0(sum) and another with value 1(val)

3. enter a for loop starting with 0 and going to the input number

4. in each loop print sum, initialize dummy variable = sum+val, sum = val, val = dummy

**Code:**

#!/bin/bash

echo "number: "

read n

sum=0

val=1

echo "fibonachi series: "

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

do

echo $sum

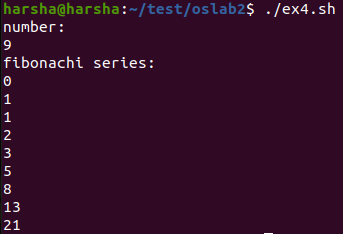
dummy=$((sum+val))

sum=$val

val=$dummy

done

**Output:**



**Q5**

**Aim:** to find whether a given number is Armstrong or not

**Algorithm:**

1. Input the number.
2. Initialize sum=0 and temp=number.
3. Find the total number of digits in the number.
4. Repeat until (temp != 0)
5. remainder = temp % 10
6. result = resut + pow(remainder,n)
7. temp = temp/10
8. if (result == number)
9. Display "Armstrong"
10. Else
11. Display "Not Armstrong"

**Code:**

#!/bin/bash

echo "enter number : "

read n

copy=$n

sum=0

while [ $n -gt 0 ]

do

dummy=$(($n%10))

dummy=$(($dummy\*\*4))

sum=$((sum+dummy))

n=$((n/10))

done

if [ $sum -eq $copy ]

then

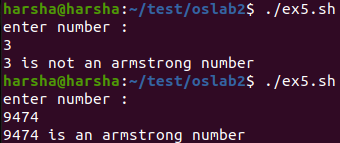
echo "$copy is an armstrong number"

else

echo "$copy is not an armstrong number"

fi

**Output:**



Q6

Aim: to reverse a given string

Code:

#!/bin/bash

echo "enter word: "

read word

len=${#word}

rev=""

for ((i=$len; i>=0; i--))

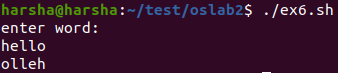
do

rev="$rev${word:i:1}"

done

echo "$rev"

Output:



**Q7**

**Aim:** to replace a character in a string

**Algorithm:**

1. get the character to be replaced

2. get the character to replace it with

3. calculate the length of the sentence

4. enter for loop spanning the length of the sentence and reads the sentence character by

character

5. if the character to replace matched, replace the character

6. print the edited sentence

**Code:**

#!/bin/bash

echo "enter word : "

read word

echo "enter character to replace : "

read rep

echo "enter replacement character : "

read char

reptext=""

len=${#word}

for ((i=0; i<$len; i++))

do

if [[ ${word:i:1} == $rep ]]

then

reptext="$reptext$char"

else

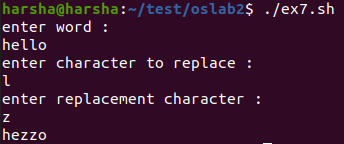
reptext="$reptext${word:i:1}"

fi

done

echo "$reptext"

**Output:**



**Q8**

**Aim:** to count the number of vowels in a string

**Algorithm:**

1. get the string from the user

2. calculate the length of the word

3. have a variable counter assigned with 0

4. enter for loop spanning the length of the sentence and reads the sentence character by

character

5. if the character matches any of the vowels, increment the counter

6. print the value of the counter

**Code:**

#!/bin/bash

echo "enter word : "

read word

count=0

len=${#word}

for ((i=0; i<$len; i++))

do

if [[ ${word:i:1} == "a" || ${word:i:1} == "e" || ${word:i:1} == "i" || ${word:i:1} == "o" || ${word:i:1} == "u" ]]

then

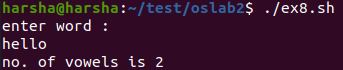
count=$((count+1))

fi

done

echo "no. of vowels is $count"

**Output:**



**Q9**

**Aim:** to check whether a given number is odd or even

**Algorithm:**

1. get the number from the user

2. check if the number gives a reminder when divided with 0

3. if it gives a remainder, print its an odd number, if not print the number is even

4. if the given number id 0, quit

**Code:**

#!/bin/bash

echo "enter number : "

read n

echo "enter 0 to quit"

while [ $n != 0 ]

do

if [ $((n%2)) -eq 0 ]

then

echo "$n is even"

else

echo "$n is odd"

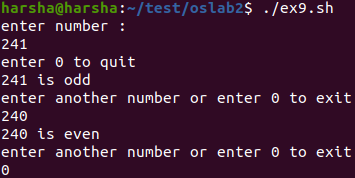
fi

echo "enter another number or enter 0 to exit"

read n

done

**Output:**

s

**Q10**

**Aim:** To sort n numbers

**Algorithm:**

1. get the number from the user

2.

3. if it gives a remainder, print its an odd number, if not print the number is even

4. if the given number id 0, quit

**Code:**

echo -e "Program to sort numbers\nEnter n"

read n

for (( i=0; i<$n; i++)) #read array input

do

echo "Enter element $((i+1))"

read a[$i]

done

for (( i=0; i<$n; i++)) #bubble sort

do

for (( j=$i; j<$n; j++))

do

if [[ ${a[$i]} -gt ${a[$j]} ]] #Swaping the elements to maintain ascending order

then

temp=${a[$i]}

a[$i]=${a[$j]}

a[$j]=$temp

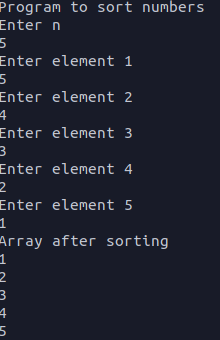
fi

done

done

echo "Array after sorting"

**Output:**



**Result:**

Thus the Shell programs have been executed .

OS LAB EX 2

Simulation of Linux Commands

**Aim:** to simulate the commands: ls, grep, cp, cat and stst

**Problem Description:**

In this experiment we are simulating the existing Linux commands available by making use

of C. In this program

ls: It will display the files available.

Cp: copies the content from one text file into another.

Grep: If the given word is present, then the contents will be displayed, else nothing will be

displayed.

Cat: Displays the contents of the file

Stat: Displays information regarding the file, such as access specification, etc.

**1. simulating ls command**

**Algorithm:**

1. get the folder name to be read

2. open the folder if present, if not, print folder not found

3. get the names of the file and print

**Code:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

char ch, filename[20];

FILE \*fptr;

printf("Enter the Filename: ");

scanf("%s", &filename);

fptr = fopen(filename, "r");

if (fptr == NULL) //exits if file not found

{

printf("File not found!");

exit(0);

}

while ((ch=fgetc(fptr))!=EOF)

{

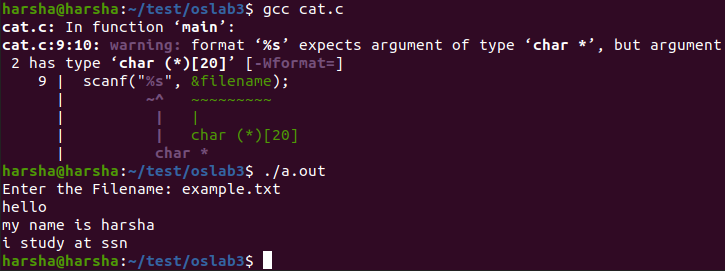
putchar(ch);

}

fclose(fptr);

}

**Output:**



**2. Simulating grep command**

**Algorithm:**

1: Start the program

2: Declare the variables fline[max], count=0, occurrences=0 and pointers \*fp,

\*newline.

3: Open the file in read mode.

4: In while loop check fgets(fline,max,fp)!=NULL

5: Increment count value.

6: Check newline=strchr(fline, „\n‟)

7: print the count,fline value and increment the occurrence value.

8: Stop the program

**Code:**

#include<stdio.h>

#include<string.h>

void main()

{

char file[10],word[10],dummy[200];

FILE \*fp;

printf("Enter file name: ");

scanf("%s",file);

printf("Enter letters to be searched: ");

scanf("%s",word);

fp=fopen(file,"r");

printf("matches found:- \n");

while(!feof(fp))

{

fgets(dummy,1000,fp);

if(strstr(dummy,word))

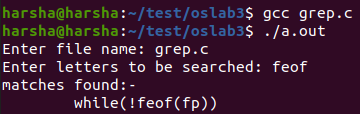
printf("%s\n",dummy);

}

fclose(fp);

}

**Output:**



**3. Simulating cp command**

**Algorithm:**

1: Start the program

2:Declare the variables ch, \*fp, sc=0

3: Open the file in read mode

4: Get the character

5: If ch== “ “ then increment sc value by one

6: Print no of spaces

7:Close the file

**Code:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*f1, \*f2;

char file1[20], file2[20], dummy;

printf("enter the filename to read: ");

scanf("%s", file1);

f1 = fopen(file1, "r");

if(f1==NULL)

{

printf("File Not found!");

exit(0);

}

printf("Enter the filename to write: ");

scanf("%s", file2);

f2 = fopen(file2, "w");

if (f2==NULL)

{

printf("File Not Found!");

exit(0);

}

dummy = fgetc(f1);

while (dummy != EOF )

{

fputc(dummy, f2);

dummy = fgetc(f1);

}

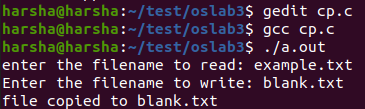
printf("file copied to %s\n", file2);

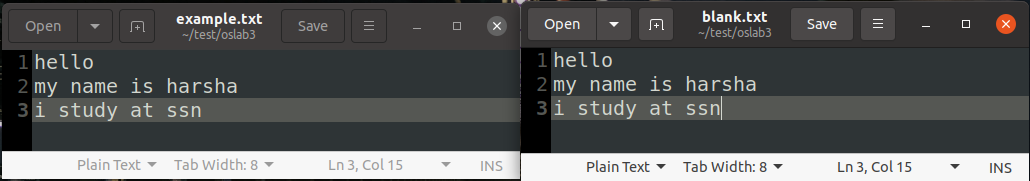
fclose(f1);

fclose(f2);

}

**Output:**





**4. Simulating cat command**

**Algorithm:**

1. get the file name to be read

2. open the file if present, if not, print file not found

3. get the content of the file and print

**Code:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

char ch, filename[20];

FILE \*fptr;

printf("Enter the Filename: ");

scanf("%s", &filename);

fptr = fopen(filename, "r");

if (fptr == NULL) //exits if file not found

{

printf("File not found!");

exit(0);

}

while ((ch=fgetc(fptr))!=EOF)

{

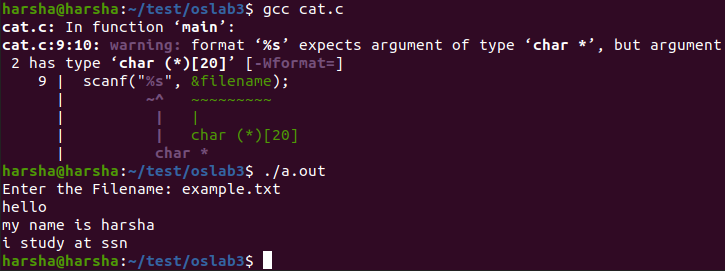
putchar(ch);

}

fclose(fptr);

}

**Output:**



**5. Simulating stat command**

**Algorithm:**

1. get the file name from the user

2. print the required status values using spl functions

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <time.h>

#include <string.h>

void sfile(char const fname[]);

void main()

{

struct stat buf;

char ntime[100], filename[20];

FILE \*fptr;

printf("Enter the filename: ");

scanf("%s",filename);

fptr = fopen(filename, "r");

if (fptr == NULL)

{

printf("File Not Found!");

exit(0);

}

else

{

sfile(filename);

}

stat(filename, &buf);

printf("Mode = %o\n", buf.st\_mode);

strcpy(ntime, ctime(&buf.st\_mtime));

printf("Created = %s\n", ntime);

}

void sfile(char const fname[]){

struct stat sfile;

if(stat(fname,&sfile)==-1){

printf("Error Occurred\n");

}

printf("Uid: %d \n",sfile.st\_uid);

printf("Group id: %d \n",sfile.st\_gid);

printf("Size: %ld bytes \n",sfile.st\_size);

printf("Links: %u \n",(unsigned int)sfile.st\_nlink);

printf("File Permissions User (");

printf((sfile.st\_mode & S\_IRUSR)? "r":"-");

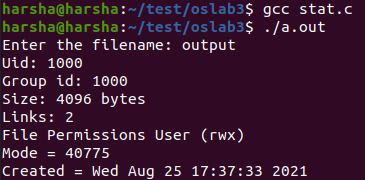
printf((sfile.st\_mode & S\_IWUSR)? "w":"-");

printf((sfile.st\_mode & S\_IXUSR)? "x":"-");

printf(")\n");

}

Output:



**Result:**

Thus the simulation of Linux commands have been executed successfully.

OS LAB EX 3

System Calls Programming

**Aim:**

To create a new process with appropriate system calls.

**Problem Description:**

In this experiment we are going to simulate system call commands like fork, getpid, getppid, exit, open, close, read, write, wait and exec.

**1. System calls (fork, getpid, getppid, exit)**

**Algorithm:**

* Create a new process by invoking the system call fork (the return code for the fork system call is zero for the new (child) process, whereas the nonzero process identifier of the child is returned to the parent.
* Get the process identifier of the currently running process and its respective parent using the system calls: getpid and getppid.
* Display the same.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

void main()

{

int pid=fork();

if(pid==0)

{

printf("Child process\n");

printf("Pid of child in child= %d\n", getpid());

printf("pid of parent in child= %d\n",getppid());

exit(0);

}

else

{

printf("parent process\n");

printf("pid of child in parent= %d\n", getpid());

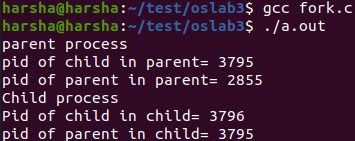
printf("pid of parent in parent= %d\n", getppid());

exit(0);

}

}

**Output:**



**2. System calls (open, close, read, write)**

**Aim:**

To copy the content of one file to another using system calls.

**Algorithm:**

* Get the source and destination file names.
* Open the source file in read only mode and destination file in write mode using the system call open.
* Read the content of the source file using the system call read and write the same in the destination file using the system call write.
* Close the source and destination file using the system call close.

**Code:**

#include<unistd.h>

#include<sys/types.h>

#include<stdio.h>

#include<fcntl.h>

void main(int argc,char \*argv[])

{

int f1,f2,c;

char buf[1024];

f1=open(argv[1],O\_RDONLY);

f2=open(argv[2],O\_WRONLY|O\_CREAT,S\_IRWXU|S\_IRGRP| S\_IROTH);

while((c=read(f1,buf,1024))>0)

write(f2,buf,c);

printf("Copied\n");

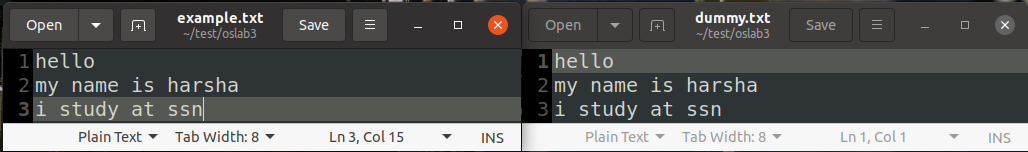
close(f1);

close(f2);

}

**Output:**

C:\Users\trhar\Downloads\open_output.png



**3. System calls (wait & exec)**

**Aim:**

To write a program to execute the basic Linux commands using the system call execlp.

**Algorithm:**

* Get the choice of the command to be executed from the user.
* The ‘execlp’ system call is used to execute the command.

**Code:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

void main()

{

int option;

do

{

printf("select the option\n");

printf("0.exit\n");

printf("1.pwd command \n");

printf("2.date command \n");

printf("3.who command \n");

printf("your choice: ");

scanf("%d",&option);

switch(option)

{

case 1:

if (fork())

wait(0);

else

execlp("pwd","pwd",(char \*)NULL);

break;

case 2:

if (fork())

wait(0);

else

execlp ("date","date",(char \*)NULL);

case 3:

if (fork())

wait(0);

else

execlp ("who","who",(char \*)NULL);

break;

case 0:

break;

default:

printf("enter the choice 0->3 only\n");

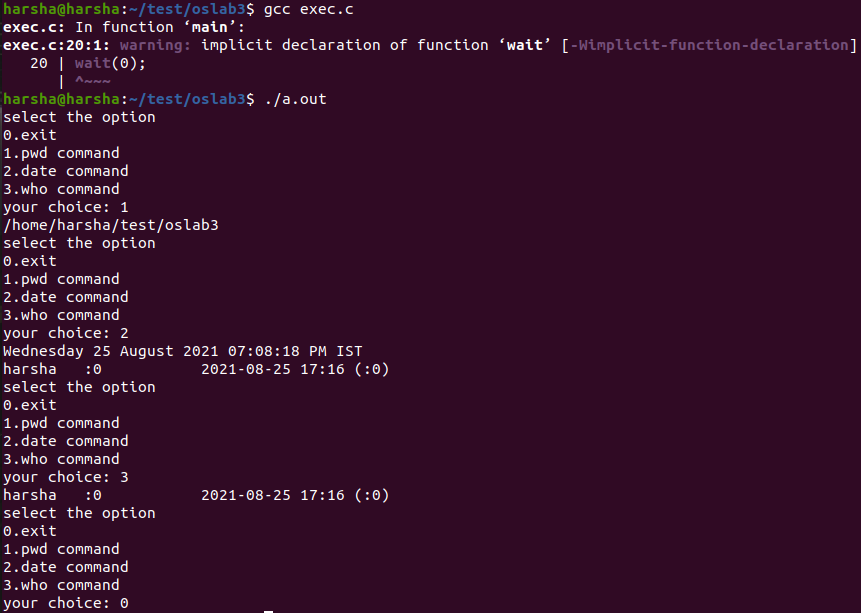
}

}

while (option != 0);

}

**Output:**



**Result:**

Thus the system calls have been tested and executed successfully.

OS LAB EX 4

Implementation of Inter Process Communication

**Aim:** To implement interprocess communication using pipes, shared memory.

**Problem Description:**

in this exercise we are going to simulate IPC using shared memory and pipes

**1. Shared Memory**

**Algorithm:**

1: Start the process

2: Declare the segment size

3: Create the shared memory

4: Read the data from the shared memory

5: Write the data to the shared memory

6: Edit the data

7: Stop the process

**Code:**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main(){

int r=fork();

if(r!=0){

wait(0);

printf("CLIENT PROCESS:\n");

key\_t key=ftok("shmfile",65);

int shmid= shmget(key,1024,0666|IPC\_CREAT);

char \*str1=(char\*) shmat(shmid,(void\*)0,0);

printf("data read from memory:%s\n",str1);

shmdt(str1);

}

else{

printf("SERVER PROCESS:\n");

key\_t key=ftok("shmfile",65);

int shmid =shmget(key,1024,0666|IPC\_CREAT);

char \*str= (char\*)shmat(shmid,(void\*)0,0);

printf("enter data t be written into shared memory:");

scanf("%[^\n]%\*c",str);

printf("the string is: %s\n",str);

shmdt(str);

}

}

**Output:**



**2. Pipes**

**Algorithm:**

1: Start the process

2: Declare the segment size

3: Create the pipe

4: Read the data from the pipe

5: Write the data to the pipe

6: Edit the data

7: Stop the process

**Code:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main()

{

int fd[2],size=32;

if(pipe(fd)<0)

{

printf("\nPipe creation failed.");

exit(0);

}

printf("\nPipe creation successful. In Parent Process.");

//close(fd[0]);

write(fd[1],"Hello from the Parent Process",size);

close(fd[1]);

printf("\nString written into pipe.");

int pid=fork();

if(pid==0)

{

printf("\nIn Child Process.");

for(int i=0;i<1;i++)

{

char reader[size];

read(fd[0],reader,size);

close(fd[0]);

printf("\n% s",reader);

}

exit(0);

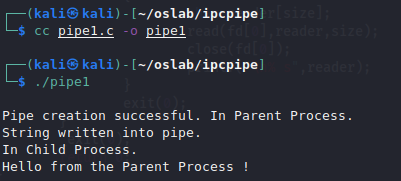
}

\_exit(0);

return 0;

}

Output:



**Result:**

Thus the interprocess communication simulation has been executed successfully.

OS LAB EX 5a

Implementation of CPU Scheduling Algorithms – FCFS & SJF

1. **FCFS**

**Aim:** to simulate FCFS Scheduling Algorithm

**Problem Description:**

The First Come First Served (FCFS) algorithm is the simplest scheduling algorithm that

schedules according to arrival times of processes.

First come first serve scheduling algorithm states that the process that requests the CPU first

is allocated the CPU first.

FCFS algorithm is implemented by using the FIFO queue.

**Algorithm:**

1: Inside the structure declare the variables.

2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

5: Calculate total wait time and total turnaround time by dividing by total number of

process.

6: Print total wait time and total turnaround time.

7: Stop the program

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time);

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], rr\_time = 1;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

// printf("enter the quantity of time quantum : ");

// scanf ("%d", &rr\_time);

// printf("\n\n");

  res(name, arrival\_time, burst\_time, number, rr\_time);

}

// function to sort

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (atc[i] > atc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0;float av\_time;

int buffer;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

sort(name, arrival\_time, burst\_time, atc, btc, exit\_time, number);

loop:

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

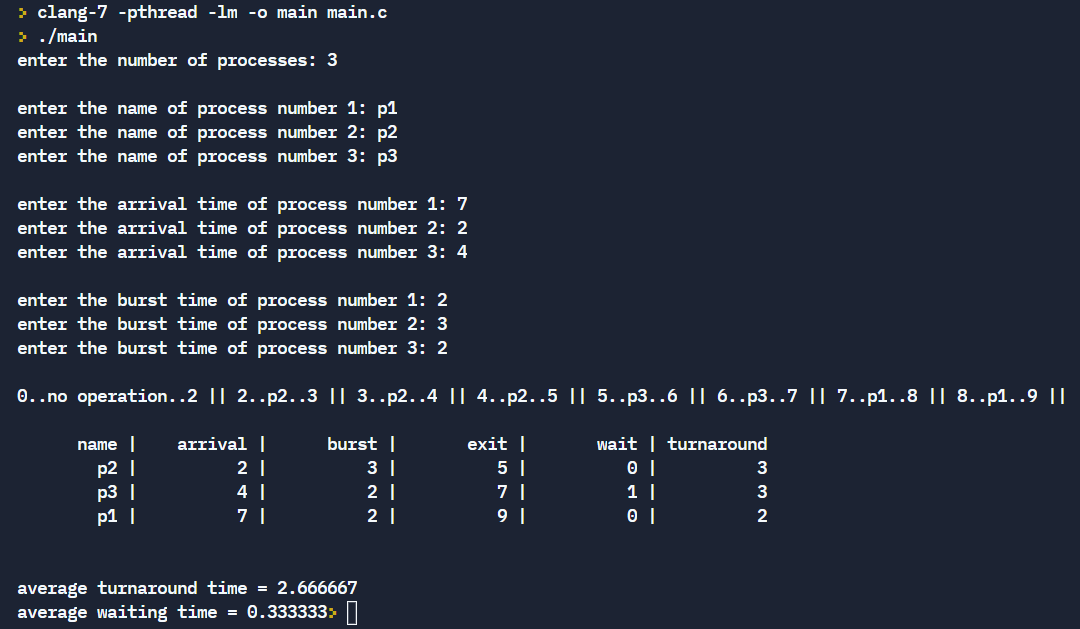
av\_tat/=number;

av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

**Output:**



**2. SJF(non- pre-emptive)**

**Problem Description:**

Shortest Job First (SJF) is an algorithm in which the process having the smallest

execution time is chosen for the next execution. This scheduling method can be pre-emptive

or non-pre-emptive. It significantly reduces the average waiting time for other processes

awaiting execution.

In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process

with shortest burst time begins execution. If a process with even a shorter burst time arrives,

the current process is removed or preempted from execution, and the shorter job is allocated

CPU cycle.

In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds

it till it reaches a waiting state or terminated.

**Aim:** To simulate SJF cpu scheduling

**Algorithm:**

1: Inside the structure declare the variables.

2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

5: Calculate total wait time and total turnaround time by dividing by total number of

process.

6: Print total wait time and total turnaround time.

7: Stop the program.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time);

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], rr\_time = 1;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

// printf("enter the quantity of time quantum : ");

// scanf ("%d", &rr\_time);

// printf("\n\n");

  res(name, arrival\_time, burst\_time, number, rr\_time);

}

// function to sort

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (btc[i] > btc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0;float av\_time;

int buffer;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

sort(name, arrival\_time, burst\_time, atc, btc, exit\_time, number);

loop:

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

// atc[i] = atc[i] + counter\_aft;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

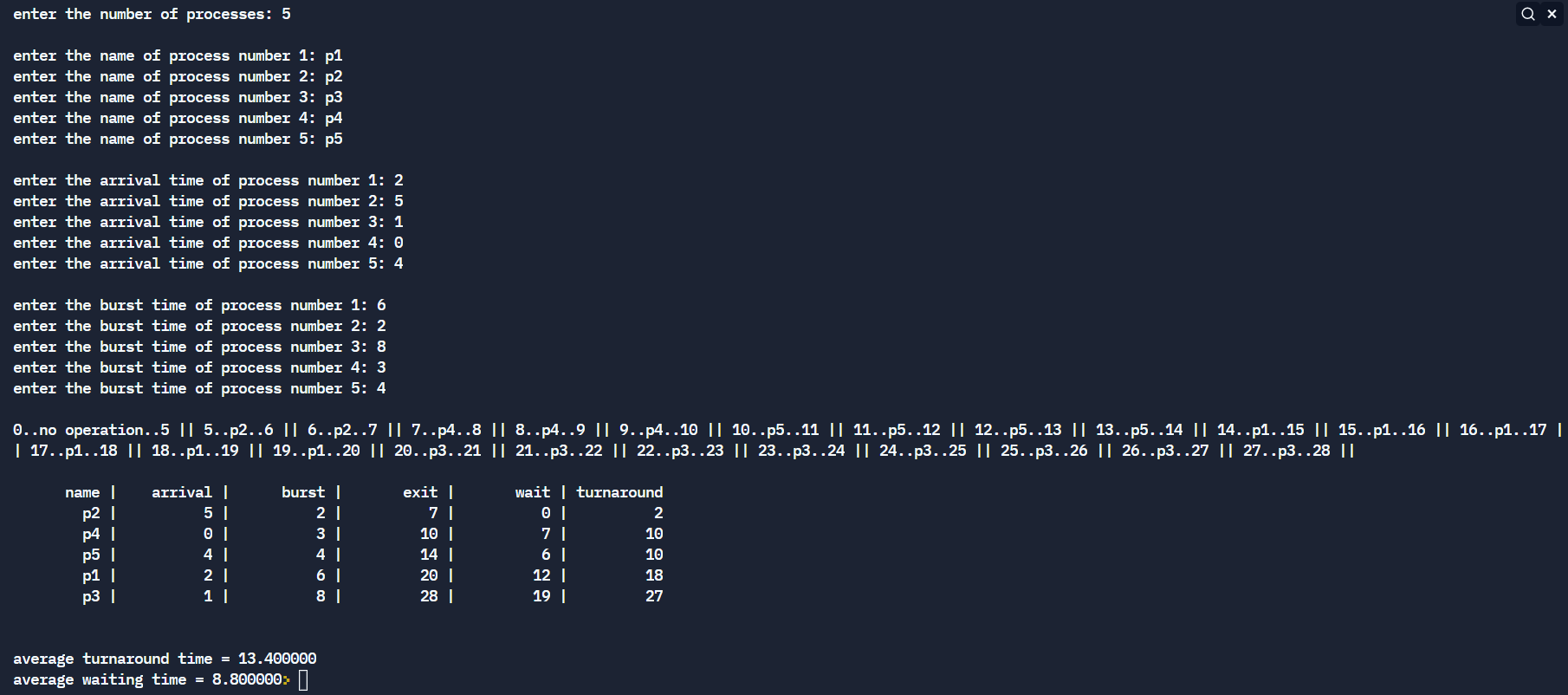
av\_tat/=number;

av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

**Output:**



**3. SJF(pre-emptive)**

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time);

void sort\_arrival(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

void sort\_burst(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int point);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], rr\_time = 1;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

// printf("enter the quantity of time quantum : ");

// scanf ("%d", &rr\_time);

// printf("\n\n");

  res(name, arrival\_time, burst\_time, number, rr\_time);

}

// function to sort arrival times

void sort\_arrival(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (atc[i] > atc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to sort burst time

void sort\_burst(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int point){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<point; i++){

for (int j = i+1; j<point; j++){

if (btc[i] > btc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0;float av\_time;

int buffer, cutoff = 0;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

sort\_arrival(name, arrival\_time, burst\_time, atc, btc, exit\_time, number);

loop:

for(int i = 0; i<number; i++){

if (atc[i] <= counter\_aft){

cutoff+=1;

}

}

if (cutoff >=2 ){

sort\_burst(name, arrival\_time, burst\_time, atc, btc, exit\_time, cutoff);

}

cutoff = 0;

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

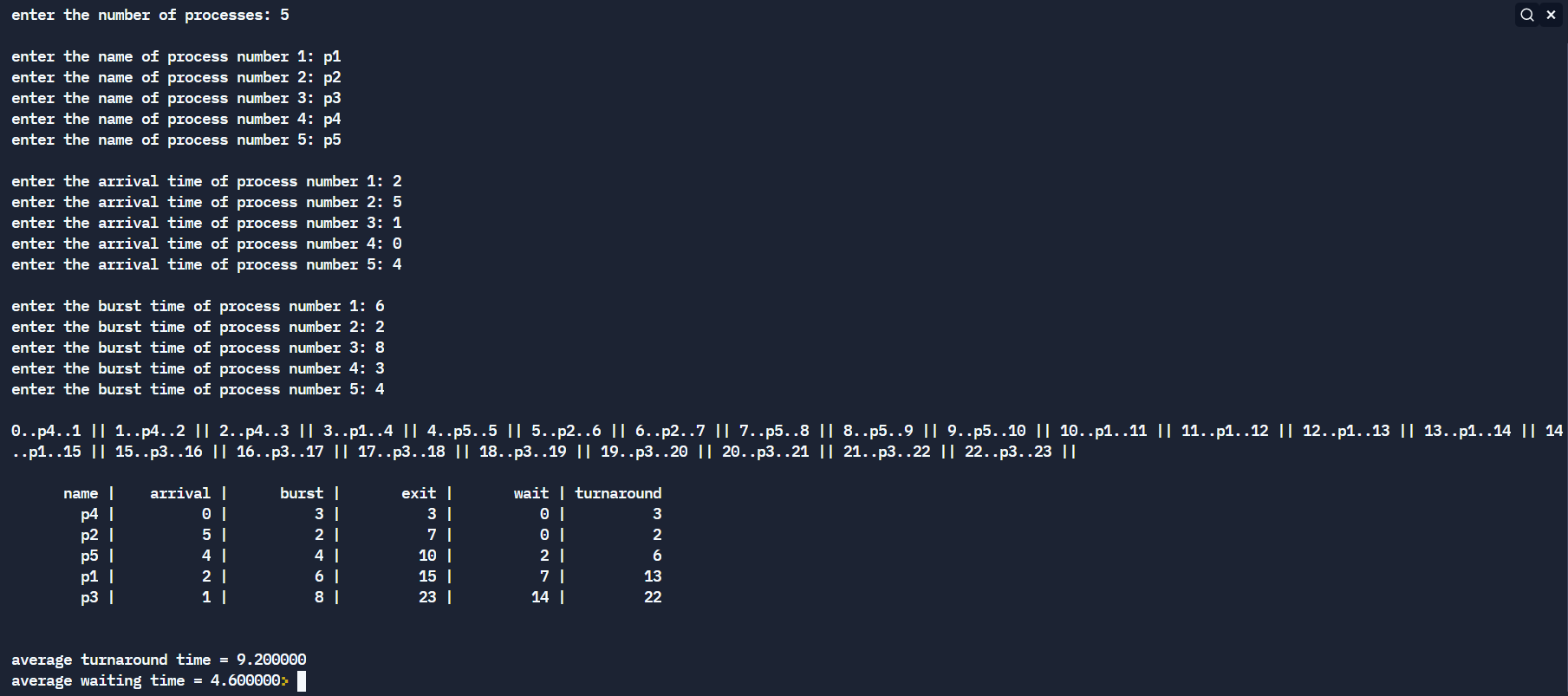
av\_tat/=number;

av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

**Output:**



**Result:**

Thus the scheduling algorithms FCFS and SJF have been compiled and executed successfully.

OS LAB EX 5b

Implementation of CPU Scheduling Algorithms – Priority & RR

**1. RR**

**Problem Description:**

Round robin is a pre-emptive algorithm. The CPU is shifted to the next process after fixed

interval time, which is called time quantum/time slice. The process that is pre-empted is

added to the end of the queue.

This method spends more time on context switching.

It gives the best performance in terms of average response time.

**Aim:** to simulate round robin scheduling

**Algorithm:**

1: Inside the structure declare the variables.

2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

3: Get the value of „n‟ assign p and allocate the memory.

4: Inside the for loop get the value of burst time and priority and read the time quantum.

5: Assign wtime as zero.

6: Check p[i].pri is greater than p[j].pri .

7: Calculate the total of burst time and waiting time and assign as turnaround time.

8: Stop the program.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time);

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], rr\_time;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

printf("enter the quantity of time quantum : ");

scanf ("%d", &rr\_time);

printf("\n\n");

  res(name, arrival\_time, burst\_time, number, rr\_time);

}

// function to sort

void sort(char name[][20], int arrival\_time[], int burst\_time[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (atc[i] > atc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0;float av\_time;

int buffer;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

loop:

sort(name, arrival\_time, burst\_time, atc, btc, exit\_time, number);

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

// atc[i] = atc[i] + counter\_aft;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

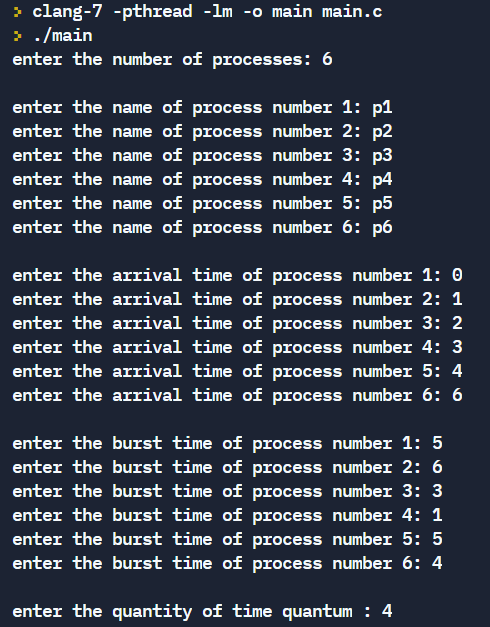
av\_tat/=number;

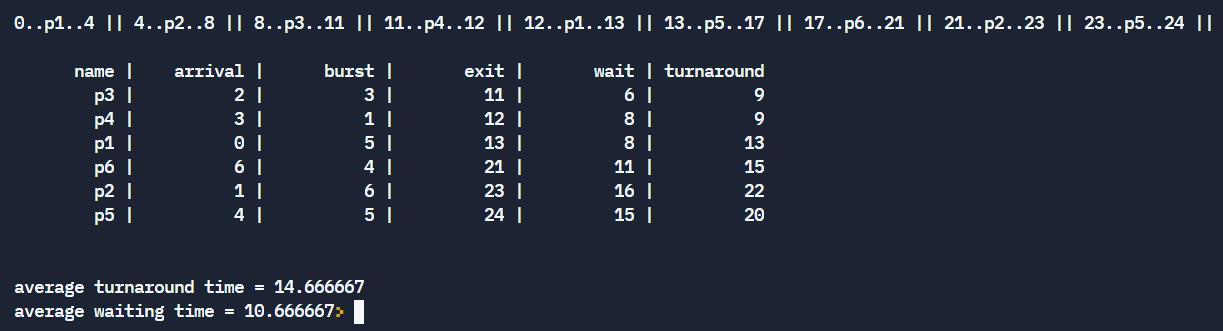
av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

**Output:**





**2. Priority (Non-pre-emptive)**

**Problem Description:**

Priority Scheduling is a method of scheduling processes that is based on priority. In this

algorithm, the scheduler selects the tasks to work as per the priority. The processes with

higher priority should be carried out first, whereas jobs with equal priorities are carried out on

a round robin or FCFS basis. Priority depends upon memory requirements, time

requirements, etc.

In Preemptive Scheduling, the tasks are mostly assigned with their priorities.

In Non-PreemptiveScheduling , the CPU has been allocated to a specific process.

**Aim:** to simulate Priority scheduling

**Algorithm:**

1: Inside the structure declare the variables.

2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

3: Get the value of „n‟ assign p and allocate the memory.

4: Inside the for loop get the value of burst time and priority.

5: Assign wtime as zero .

6: Check p[i].pri is greater than p[j].pri .

7: Calculate the total of burst time and waiting time and assign as turnaround time.

8: Stop the program.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int number, int rr\_time);

void sort(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], priority[number], rr\_time = 1;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

for (int i = 0; i<number; i++){

    printf("enter the priority of process number %d: ", i+1);

    scanf ("%d", &priority[i]);

  }

  printf("\n");

// printf("enter the quantity of time quantum : ");

// scanf ("%d", &rr\_time);

// printf("\n\n");

  res(name, arrival\_time, burst\_time, priority, number, rr\_time);

}

// function to sort

void sort(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (priority[i] > priority[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = priority[i];

priority[i] = priority[j];

priority[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0;float av\_time;

int buffer;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

sort(name, arrival\_time, burst\_time, priority, atc, btc, exit\_time, number);

loop:

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

// atc[i] = atc[i] + counter\_aft;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "priority", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], priority[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

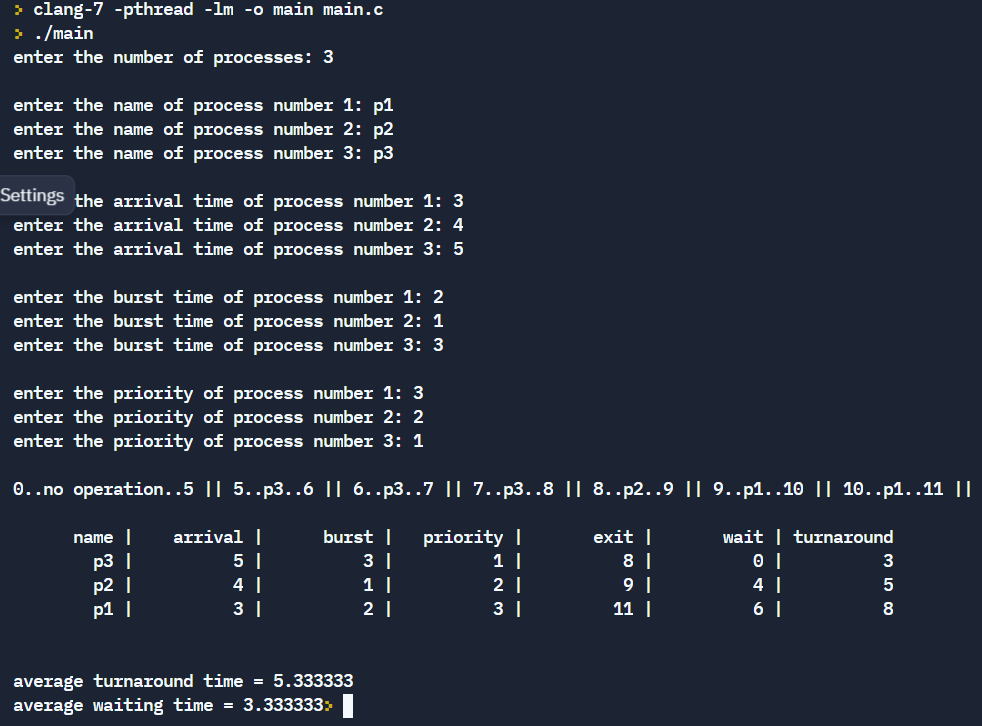
av\_tat/=number;

av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

**Output:**



**3. Priority (Pre-emptive)**

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void res(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int number, int rr\_time);

void sort\_arrival(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int number);

float avg(int burst\_time[], int number);

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number);

void sort\_priority(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int point);

int main(){

  // get the number of processes

  int number;

  printf("enter the number of processes: ");

  scanf("%d", &number);

  if (number == 0){

    printf("no processes given -> program terminated");

    exit(0);

  }

printf("\n");

  // assign variables to get the inputs from the user

  char name[number][20];

  int arrival\_time[number], burst\_time[number], priority[number], rr\_time = 1;

  // get the inputs from the user

  for (int i = 0; i<number; i++){

    printf("enter the name of process number %d: ", i+1);

    scanf("%s", name[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the arrival time of process number %d: ", i+1);

    scanf ("%d", &arrival\_time[i]);

  }

printf("\n");

  for (int i = 0; i<number; i++){

    printf("enter the burst time of process number %d: ", i+1);

    scanf ("%d", &burst\_time[i]);

  }

  printf("\n");

for (int i = 0; i<number; i++){

    printf("enter the priority of process number %d: ", i+1);

    scanf ("%d", &priority[i]);

  }

  printf("\n");

// printf("enter the quantity of time quantum : ");

// scanf ("%d", &rr\_time);

// printf("\n\n");

  res(name, arrival\_time, burst\_time, priority, number, rr\_time);

}

// function to sort

void sort\_arrival(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int number){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<number; i++){

for (int j = i+1; j<number; j++){

if (atc[i] > atc[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = priority[i];

priority[i] = priority[j];

priority[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to sort priority

void sort\_priority(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int atc[], int btc[], int exit\_time[], int point){

char dummy1[20];

int dummy2, dummy3;

  for (int i = 0; i<point; i++){

for (int j = i+1; j<point; j++){

if (priority[i] > priority[j]){

dummy2 = arrival\_time[i];

arrival\_time[i] = arrival\_time[j];

arrival\_time[j] = dummy2;

dummy2 = burst\_time[i];

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

burst\_time[j] = dummy2;

dummy2 = priority[i];

priority[i] = priority[j];

priority[j] = dummy2;

dummy2 = atc[i];

atc[i] = atc[j];

atc[j] = dummy2;

dummy2 = btc[i];

btc[i] = btc[j];

btc[j] = dummy2;

dummy2 = exit\_time[i];

exit\_time[i] = exit\_time[j];

exit\_time[j] = dummy2;

strcpy(dummy1, name[i]);

strcpy(name[i], name[j]);

strcpy(name[j], dummy1);

}

}

}

}

// function to calculate the avg of burst times

float avg(int burst\_time[], int number){

float tot = 0;

for (int i = 0; i<number; i++){

tot+=burst\_time[i];

}

float avg = tot/number;

return(avg);

}

//function to copy arrays

void cpy(int arrival\_time[], int burst\_time[], int atc[], int btc[],int number){

for (int i = 0; i<number; i++){

atc[i] = arrival\_time[i];

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

}

}

// function to calculate the result

void res(char name[][20], int arrival\_time[], int burst\_time[], int priority[], int number, int rr\_time){

// variables to store exit time, counter values and avg value

int exit\_time[number], tat[number], wait\_time[number], counter\_bef = 0, counter\_aft = 0, cutoff = 0;

float av\_time;

int buffer;

// making copy of arrival and burst times

int atc[number], btc[number];

cpy(arrival\_time, burst\_time, atc, btc, number);

sort\_arrival(name, arrival\_time, burst\_time, priority, atc, btc, exit\_time, number);

loop:

for(int i = 0; i<number; i++){

if (atc[i] <= counter\_aft){

cutoff+=1;

}

}

if (cutoff >=2 ){

sort\_priority(name, arrival\_time, burst\_time, priority, atc, btc, exit\_time, cutoff);

}

cutoff = 0;

av\_time = avg(btc, number);

while (av\_time != 0.0){

for (int i = 0; i<number; i++){

if ((atc[i] <= counter\_bef) && (btc[i] != 0)){

if ((btc[i] - rr\_time) > 0){

btc[i]-=rr\_time;

// atc[i] = atc[i] + counter\_aft;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

atc[i] = counter\_aft;

counter\_bef = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) < 0){

buffer = btc[i];

btc[i] = 0;

counter\_aft = counter\_bef + buffer;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

else if ((btc[i] - rr\_time) == 0){

btc[i]-=rr\_time;

counter\_aft = counter\_bef + rr\_time;

printf("%d..%s..%d || ", counter\_bef, name[i], counter\_aft);

counter\_bef = counter\_aft;

exit\_time[i] = counter\_aft;

goto loop;

}

}

else if ((atc[i] > counter\_bef) && (btc[i] != 0)){

buffer = atc[i] - counter\_bef;

counter\_aft = counter\_bef + buffer;

printf("%d..no operation..%d || ", counter\_bef, counter\_aft);

counter\_bef = counter\_aft;

goto loop;

}

}

}

printf("\n\n");

for (int i = 0; i<number; i++){

tat[i] = exit\_time[i] - arrival\_time[i];

wait\_time[i] = tat[i] - burst\_time[i];

}

//printing the table

printf("%10s | %10s | %10s | %10s | %10s | %10s | %10s\n", "name", "arrival", "burst", "priority", "exit", "wait", "turnaround");

for (int i=0; i<number; i++){

printf("%10s | %10d | %10d | %10d | %10d | %10d | %10d\n", name[i], arrival\_time[i], burst\_time[i], priority[i], exit\_time[i], wait\_time[i], tat[i]);

}

printf("\n\n");

float av\_tat = 0, av\_wait = 0;

for (int i = 0; i<number; i++){

av\_tat+=tat[i];

av\_wait+=wait\_time[i];

}

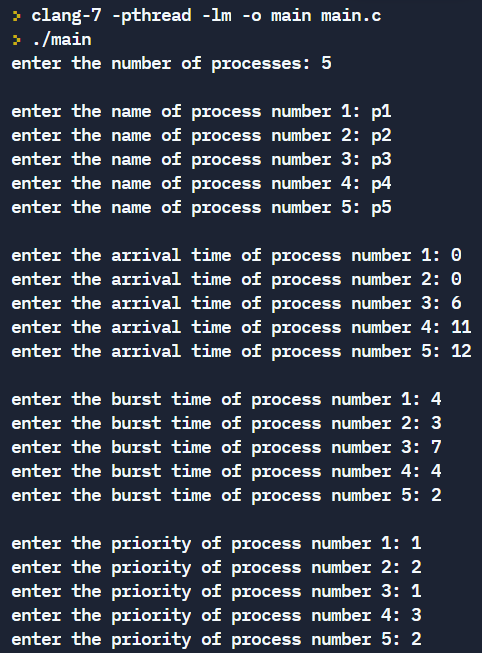
av\_tat/=number;

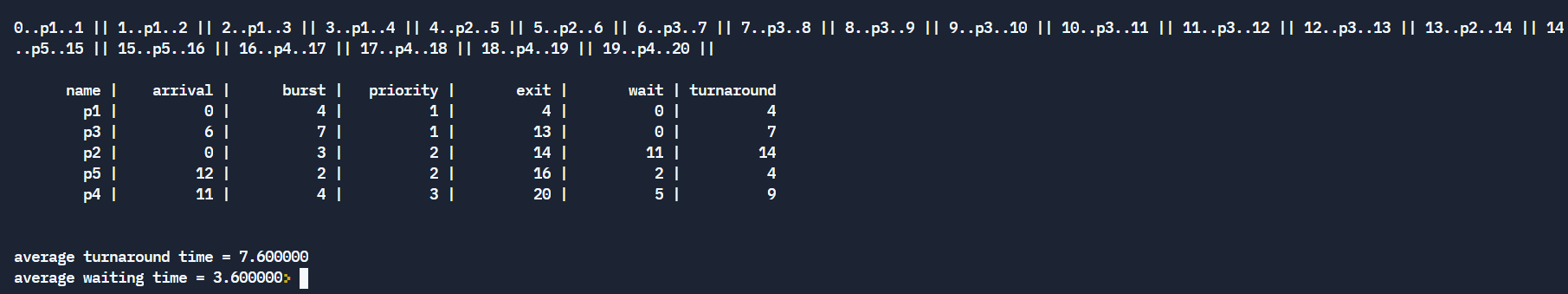
av\_wait/=number;

printf("average turnaround time = %f\naverage waiting time = %f", av\_tat, av\_wait);

}

Output:





**Result:**

Thus the priority and Round robin scheduling has been compiled and executed successfully.

OS LAB EX 6

Creation of Threads And Synchronization Applications

**1.Thread creation**

**Aim:**To simulate threading operation in C.

**Problem Description:** mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section.

**Code:**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

pthread\_t tid[2];

int counter;

pthread\_mutex\_t lock;

void\* trythis(void\* arg)

{

    pthread\_mutex\_lock(&lock);

    unsigned long i = 0;

    counter += 1;

    printf("\n Job %d has started\n", counter);

    for (i = 0; i < (0xF); i++)

    {

        printf("%ld count of job %d \n",i+1,counter);

    }

    printf("\n Job %d has finished\n", counter);

    pthread\_mutex\_unlock(&lock);

    return NULL;

}

int main(void)

{

    int i = 0;

    int error;

    if (pthread\_mutex\_init(&lock, NULL) != 0) {

        printf("\n mutex init has failed\n");

        return 1;

    }

    while (i < 2) {

        error = pthread\_create(&(tid[i]),

                            NULL,

                            &trythis, NULL);

        if (error != 0)

            printf("\nThread can't be created :[%s]",

                strerror(error));

        i++;

    }

    pthread\_join(tid[0], NULL);

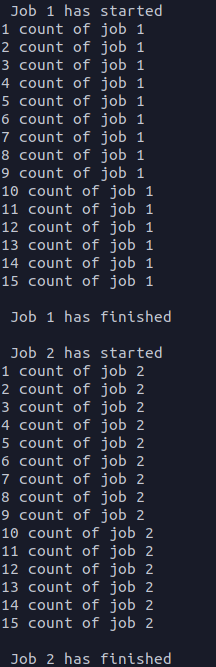
    pthread\_join(tid[1], NULL);

    pthread\_mutex\_destroy(&lock);

    return 0;

}

**Output:**

****

**2. Producer Consumer Problem**

**Problem Description:**

The bounded-buffer problem is a classic example of concurrent access to a shared

resource. A bounded buffer lets multiple producers and multiple consumers share a single

buffer. Producers write data to the buffer and consumers read data from the buffer.

**Aim:** to simulate producer consumer problem

**Algorithm:**

1: Start the program.

2: Declare the required variables.

3: Initialize the buffer size and get maximum item you want to produce.

4: Get the option, which you want to do either producer, consumer or exit from the

operation.

5: If you select the producer, check the buffer size if it is full the producer should not

produce the item or otherwise produce the item and increase the value buffer size.

6: If you select the consumer, check the buffer size if it is empty the consumer should not

consume the item or otherwise consume the item and decrease the value of buffer size.

7: If you select exit come out of the program.

8: Stop the program.

**Code:**

#include<stdio.h>

#include<stdlib.h>

static int s = 1, full = 0, empty = 10, buffer[10] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,};

static int common\_res[10] = {1,2,3,4,5,6,7,8,9,10};

static int count\_prod = 0, count\_cons = 0;

int wait(int \*count){

if (\*count <= 0){

printf("process stalled");

printf("\n\n");

return(0);

}

else{

\*count = \*count - 1;

return(1);

}

};

void signal(int\*count){

\*count = \*count + 1;

};

void producer(){

wait(&empty);

wait(&s);

buffer[count\_prod%10] = common\_res[count\_prod%10];

printf("%d : written into buffer from the common resource", buffer[count\_prod%10]);

printf("\n\n");

count\_cons = count\_prod%10;

count\_prod+=1;

signal(&s);

signal(&full);

};

void consumer(){

int chk\_full = wait(&full);

if (chk\_full == 0){

;

}

else{

wait(&s);

printf("%d : read from buffer by consumer", buffer[count\_cons]);

printf("\n\n");

buffer[count\_cons] = -1;

count\_prod = (count\_prod-1)%10;

count\_cons = (count\_cons-1)%10 ;

signal(&s);

signal(&empty);

}

};

int main(){

for (int i = 0; i<10; i++){

int random = rand()%100+1;

if(random%2 == 0){

printf("producer process: ");

producer();

}

else{

printf("consumer process: ");

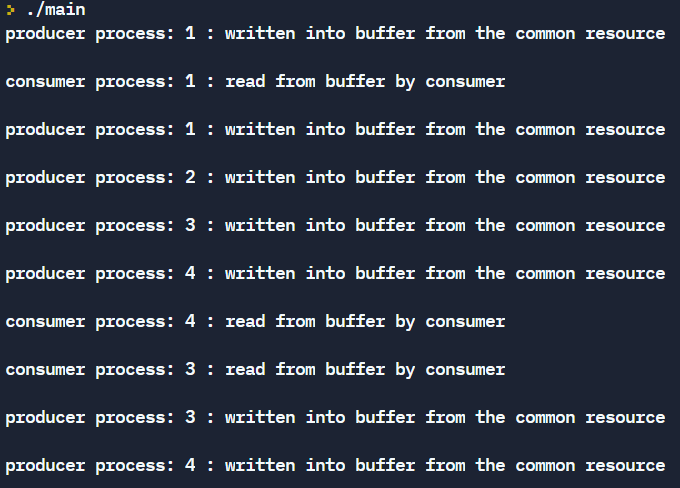
consumer();

}

}

}

**Output:**



**3. Dining Philosophers Problem**

**Problem Description:**

The dining philosopher’s problem states that there are 5 philosophers sharing a circular table

and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5

chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry

philosopher may only eat if there are both chopsticks available. Otherwise a philosopher puts

down their chopstick and begin thinking again.

**Aim:** to simulate dining philosophers problem

**Algorithm:**

1. start

2. initiate semaphore for each of the chopstick

3. when philosopher hungry, see if left and right chopstick free

4. if free, start eating, lock chopsticks

5. if not free, wait

6. release lock on chopstick after eating, start thinking

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int ph[6];

int semaphore[6];

/\* 5 forks for 4 philosophers

\*/

void state(){

int i;

for(i=0;i<5;i++){

switch(ph[i]){

case 0:printf("\nphilosopher %d is thinking",i);

break;

case 1:printf("\nphilosopher %d is feeling hungry",i);

break;

case 2:printf("\nphilosopher %d is eating",i);

break;

}

}

}

void wait(int \*S){

while((\*S)<=0){;}

(\*S)--;

}

void signal(int \*S){

(\*S)++;

}

void philosopher(int p){

ph[p]=1;

state();

printf("\n\n");

wait(&semaphore[p]);

wait(&semaphore[(p+1)%5]);

ph[p]=2;

state();

printf("\n\n");

signal(&semaphore[p]);

signal(&semaphore[(p+1)%5]);

ph[p]=0;

state();

printf("\n\n");

}

int random\_number(){

int x = rand();

return(x);

}

int main(){

int i, no\_runs, upper = 10, lower = 5;

no\_runs = (rand() % (upper - lower + 1)) + lower;

printf("no. of runs = %d \n\n", no\_runs);

for(i=0;i<6;i++){

semaphore[i]=1;

}

for(i=0;i<6;i++){

ph[i]=0;

}

i=1;

printf("starting state: \n");

state();

printf("\n\n\n");

printf("start of execution: \n");

while(i<=no\_runs){

int x=random\_number();

int ch=x%5;

switch(ch){

case 0:philosopher(0);

printf("\n\n");

break;

case 1:philosopher(1);

printf("\n\n");

break;

case 2:philosopher(2);

printf("\n\n");

break;

case 3:philosopher(3);

printf("\n\n");

break;

case 4:philosopher(4);

printf("\n\n");

break;

}

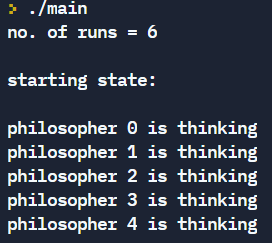
i++;

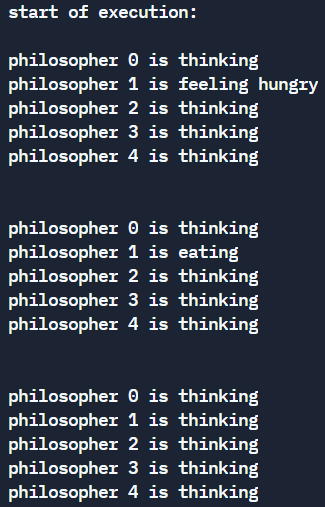
}

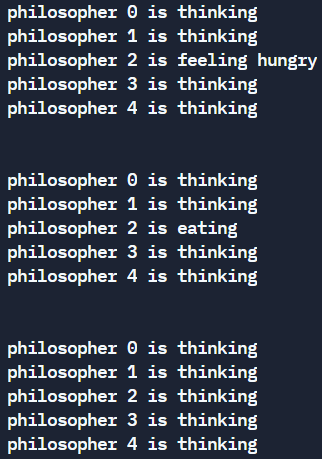
return 0;

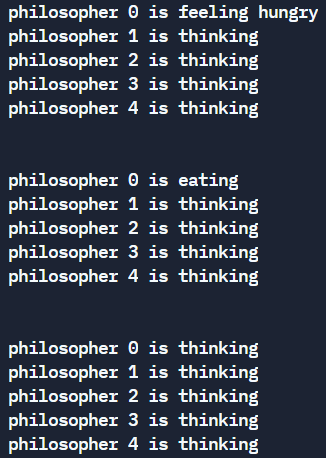
}

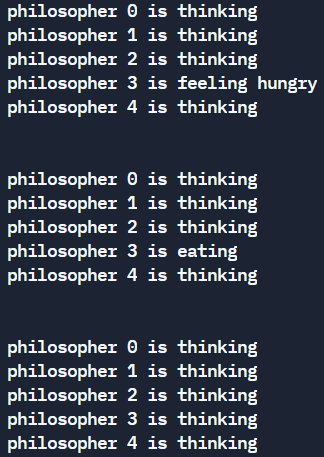
**Output:**

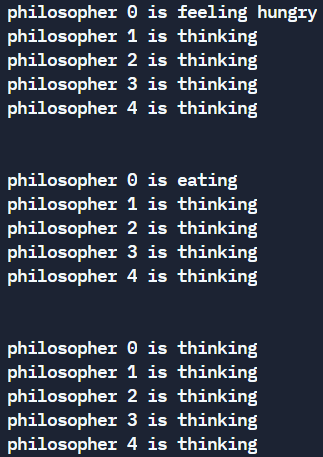


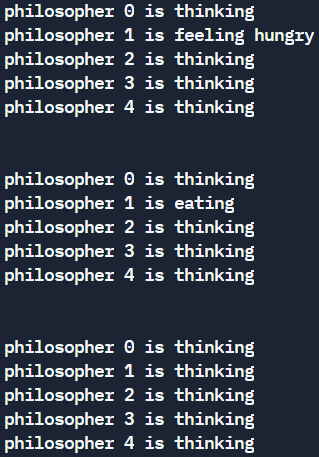












**Result:**

Thus the thread synchronisation algorithms have been executed successfully.

OS LAB EX 7

Implementation of Process Synchronization, Deadlock Avoidance and Detection Mechanism

**Aim:** to simulate bankers algorithm for deadlock avoidance

**Problem Description:**

Banker’s Algorithm is a deadlock avoidance algorithm. It is also used for deadlock detection. This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.

**Algorithm:**

1: Start the program.

2: Declare the memory for the process.

3: Read the number of process, resources, allocation matrix and available matrix.

4: Compare each and every process using the banker’s algorithm.

5: If the process is in safe state then it is a not a deadlock process otherwise it is a

deadlock process

6: produce the result of state of process

7: Stop the program

**Code:**

#include <stdio.h>

#include <stdlib.h>

int avg(int finish[], int process){

int dummy = 0;

for (int i = 0; i<process; i++){

dummy+=finish[i];

}

dummy = dummy/process;

return(dummy);

}

int main(){

//block for getting the number of processes and resources

int process, resource;

printf("enter the number of processes: ");

scanf("%d", &process);

printf("enter the number of resources: ");

scanf("%d", &resource);

printf("\n");

// block to allocate the other vars

int allo[process][resource], max[process][resource], avail[resource], need[process][resource];

//block to get the allocation values

for (int i = 0; i<process; i++){

printf("enter the allocation for process %d : ", i);

printf("\n");

for (int j = 0; j<resource; j++){

printf("resource %d : ", j);

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

}

printf("\n");

}

//block to get the max values

for (int i = 0; i<process; i++){

printf("enter the max for process %d : ", i);

printf("\n");

for (int j = 0; j<resource; j++){

printf("max %d : ", j);

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

}

printf("\n");

}

//block to get the available resources

for (int i = 0; i<resource; i++){

printf("enter available resource for resource %d : ", i);

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

}

printf("\n");

//block to find the need for the process

for (int i = 0; i<process; i++){

for (int j = 0; j<resource; j++){

need[i][j] = max[i][j] - allo[i][j];

}

}

int finish[process];

for (int i = 0; i<process; i++){

finish[i] = 0;

}

while (avg(finish, process) != 1){

static int loop\_counter = 0;

static int counter = 0;

for (int i = 0; i<process; i++){

if (finish[i] == 1){

continue;

}

else{

for (int j = 0; j<resource; j++){

if (need[i][j] <= avail[j]){

counter++;

}

}

if (counter == resource){

finish[i] = 1;

printf ("process %d -> ", i);

for (int k = 0; k<resource; k++){

avail[k]+=allo[i][k];

}

}

}

counter = 0;

}

loop\_counter++;

if (loop\_counter > process+1){

if (avg(finish, process) != 1){

printf("\n");

printf("Deadlock has occured, all processes terminating");

break;

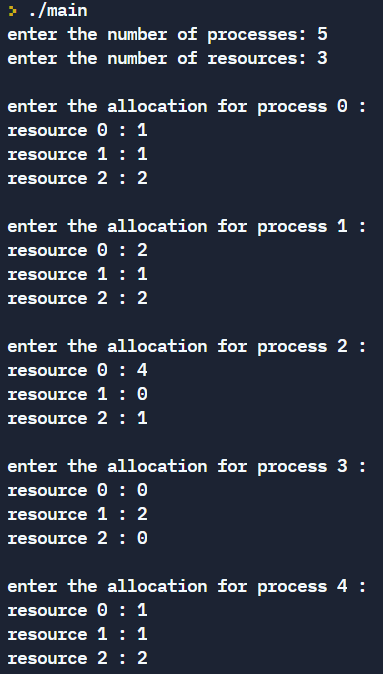
}

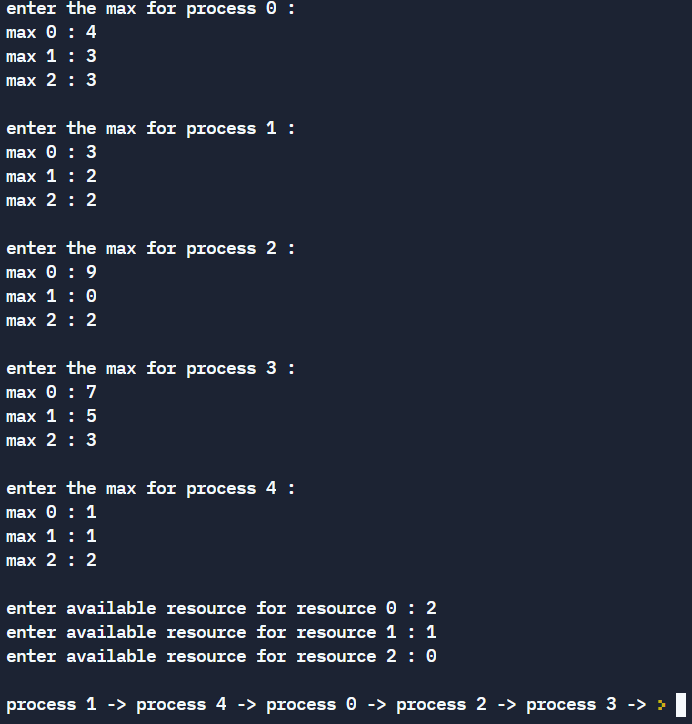
}

}

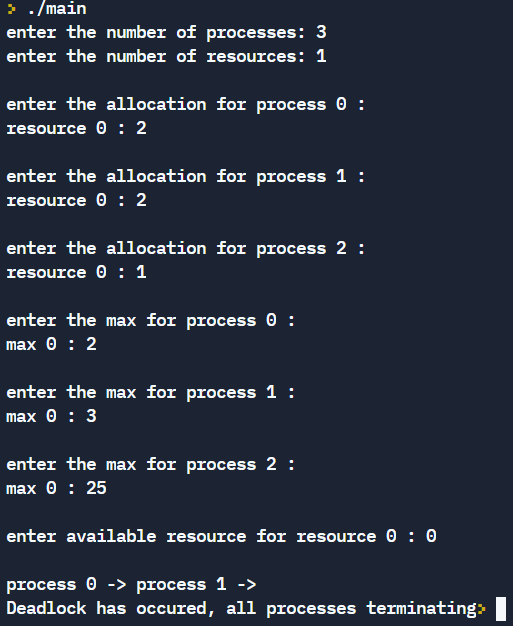
}

**Output 1:**





**Output 2:**



**Result:**

Thus the deadlock avoidance algorithm has been executed successfully.

OS LAB EX 8

Implementation of Dynamic Storage Allocation Schemes

**Aim:** to simulate first/best and worst fit allocation algorithms

**Problem Description:**

**First Fit:** In the first fit, the partition is allocated which is first sufficient from the top of Main

Memory.

Its advantage is that it is the fastest search as it searches only the first block i.e. enough to

assign a process.

**Best Fit:** Best Fit is a memory management algorithm; it deals with allocating smallest free

partition which meets the requirement of the requesting process.

**Worst Fit:** Worst Fit allocates a process to the partition which is largest sufficient among the

freely available partitions available in the main memory. If a large process comes at a later

stage, then memory will not have space to accommodate it.

**First Fit:**

**Algorithm:**

1:Define the max as 25.

2: Declare the variable frag[max],b[max],f[max],i, j, nb, nf, temp, highest=0, bf[max],ff[max].

3: Get the number of blocks, files, size of the blocks using for loop.

4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]

5: Check highest<temp, If so assign ff[i]=j, highest=temp

6: Assign frag[i]=highest, bf[ff[i]]=1,highest=0

7: Repeat step 4 to step 6.

8: Print file no, size, block no, size and fragment.

9: Stop the program.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void tostring(char str[], int num)

{

int i, rem, len = 0, n;

n = num;

while (n != 0)

{

len++;

n /= 10;

}

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

{

rem = num % 10;

num = num / 10;

str[len - (i + 1)] = rem + '0';

}

str[len] = '\0';

}

void calc(int pros, int hol, int p[], int h[]){

printf("\n\n Allocation: Process:\n");

int fin[hol][pros];

int hol\_buf[hol];

int es[hol];

for (int i = 0; i<hol; i++){

hol\_buf[i] = h[i];

}

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

fin[i][j] = -1;

}

}

int flag = 0;

int na\_count = 0;

int na[pros];

for (int i = 0; i<pros; i++){

for (int j = 0; j<hol; j++){

if (h[j]>=p[i]){

es[j] = h[j] - p[i];

h[j]-=p[i];

fin[j][i] = i;

flag = 1;

break;

}

else{

es[j] = h[j];

}

}

if (flag == 0){

na[na\_count] = i+1;

na\_count++;

}

flag = 0;

}

char buf[100] = "", actual[100] = "", dummy[100] = "";

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

if (fin[i][j] != -1){

tostring(buf, fin[i][j]+1);

strcat(actual,"p(");

strcat(actual,buf);

strcat(actual,"), ");

}

}

printf("+----------+\n");

printf("|%4d(%4d)| %s\n", hol\_buf[i], es[i], actual);

strcpy(buf, dummy);

strcpy(actual, dummy);

}

printf("+----------+\n");

printf("processes not allocated : ");

if (na\_count == 0){

printf("NONE");

}

else{

for (int i = 0; i<na\_count; i++){

printf("p(%d), ",na[i]);

}x

}

}

int main(void) {

int pros, hol;

printf("Enter the number of processes: ");

scanf("%d", &pros);

printf("\n");

printf("enter the number of holes: ");

scanf("%d", &hol);

int p[pros], h[hol];

printf("\n");

for (int i = 0; i<pros; i++){

printf("enter the size of process %d: ", i+1);

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

}

printf("\n");

for (int i = 0; i<hol; i++){

printf("enter the size of hole %d: ", i+1);

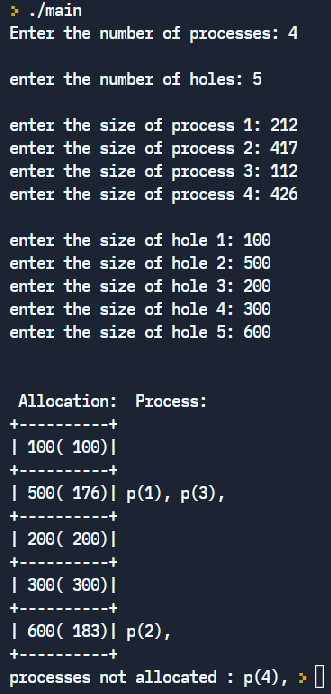
scanf("%d", &h[i]);

}

calc(pros, hol, p, h);

}

**Output:**



**Best Fit:**

**Algorithm:**

1:Define the max as 25.

2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max].

3: Get the number of blocks,files,size of the blocks using for loop.

4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]

5: Check lowest>temp,if so assign ff[i]=j,highest=temp

6: Assign frag[i]=lowest, bf[ff[i]]=1,lowest=10000

7: Repeat step 4 to step 6.

8: Print file no,size,block no,size and fragment.

9: Stop the program.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void tostring(char str[], int num)

{

int i, rem, len = 0, n;

n = num;

while (n != 0)

{

len++;

n /= 10;

}

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

{

rem = num % 10;

num = num / 10;

str[len - (i + 1)] = rem + '0';

}

str[len] = '\0';

}

void sort(int h[], int hol){

for (int i = 0; i < hol; ++i){

for (int j = i + 1; j < hol; ++j){

if (h[i] > h[j]){

int a = h[i];

h[i] = h[j];

h[j] = a;

}

}

}

}

void calc(int pros, int hol, int p[], int h[]){

printf("\n\n Allocation: Process:\n");

int fin[hol][pros];

int hol\_buf[hol];

int es[hol];

for (int i = 0; i<hol; i++){

hol\_buf[i] = h[i];

}

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

fin[i][j] = -1;

}

}

int flag = 0;

int na\_count = 0;

int na[pros];

for (int i = 0; i<pros; i++){

for (int j = 0; j<hol; j++){

if (h[j]>=p[i]){

es[j] = h[j] - p[i];

h[j]-=p[i];

fin[j][i] = i;

flag = 1;

break;

}

else{

es[j] = h[j];

}

}

if (flag == 0){

na[na\_count] = i+1;

na\_count++;

}

flag = 0;

}

char buf[100] = "", actual[100] = "", dummy[100] = "";

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

if (fin[i][j] != -1){

tostring(buf, fin[i][j]+1);

strcat(actual,"p(");

strcat(actual,buf);

strcat(actual,"), ");

}

}

printf("+----------+\n");

printf("|%4d(%4d)| %s\n", hol\_buf[i], es[i], actual);

strcpy(buf, dummy);

strcpy(actual, dummy);

}

printf("+----------+\n");

printf("processes not allocated : ");

if (na\_count == 0){

printf("NONE");

}

else{

for (int i = 0; i<na\_count; i++){

printf("p(%d), ",na[i]); } }}

int main(void) {

int pros, hol;

printf("Enter the number of processes: ");

scanf("%d", &pros);

printf("\n");

printf("enter the number of holes: ");

scanf("%d", &hol);

int p[pros], h[hol];

printf("\n");

for (int i = 0; i<pros; i++){

printf("enter the size of process %d: ", i+1);

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

}

printf("\n");

for (int i = 0; i<hol; i++){

printf("enter the size of hole %d: ", i+1);

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

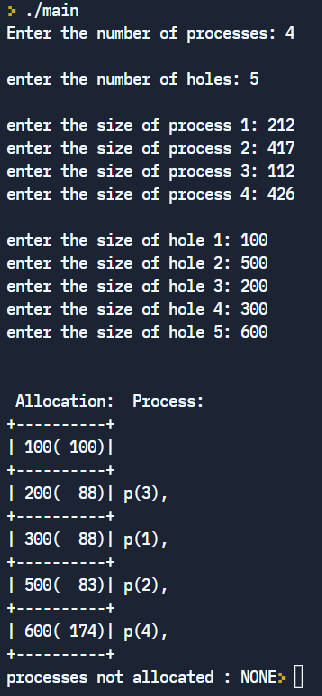
}

sort(h, hol);

calc(pros, hol, p, h);

}

**Output:**



**Worst Fit:**

**Algorithm:**

1:Define the max as 25.

2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max].

3: Get the number of blocks,files,size of the blocks using for loop.

4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]

5: Check temp>=0,if so assign ff[i]=j break the for loop.

6: Assign frag[i]=temp,bf[ff[i]]=1;

7: Repeat step 4 to step 6.

8: Print file no,size,block no,size and fragment.

9: Stop the program.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void tostring(char str[], int num)

{

int i, rem, len = 0, n;

n = num;

while (n != 0)

{

len++;

n /= 10;

}

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

{

rem = num % 10;

num = num / 10;

str[len - (i + 1)] = rem + '0';

}

str[len] = '\0';

}

void sort(int h[], int hol){

for (int i = 0; i < hol; ++i){

for (int j = i + 1; j < hol; ++j){

if (h[i] < h[j]){

int a = h[i];

h[i] = h[j];

h[j] = a;

}

}

}

}

void calc(int pros, int hol, int p[], int h[]){

printf("\n\n Allocation: Process:\n");

int fin[hol][pros];

int hol\_buf[hol];

int es[hol];

for (int i = 0; i<hol; i++){

hol\_buf[i] = h[i];

}

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

fin[i][j] = -1;

}

}

int flag = 0;

int na\_count = 0;

int na[pros];

for (int i = 0; i<pros; i++){

for (int j = 0; j<hol; j++){

if (h[j]>=p[i]){

es[j] = h[j] - p[i];

h[j]-=p[i];

fin[j][i] = i;

flag = 1;

break;

}

else{

es[j] = h[j];

}

}

if (flag == 0){

na[na\_count] = i+1;

na\_count++;

}

flag = 0;

}

char buf[100] = "", actual[100] = "", dummy[100] = "";

for (int i = 0; i<hol; i++){

for (int j = 0; j<pros; j++){

if (fin[i][j] != -1){

tostring(buf, (fin[i][j]+1));

strcat(actual,"p(");

strcat(actual,buf);

strcat(actual,"), ");

}

}

printf("+----------+\n");

printf("|%4d(%4d)| %s\n", hol\_buf[i], es[i], actual);

strcpy(buf, dummy);

strcpy(actual, dummy);

}

printf("+----------+\n");

printf("processes not allocated : ");

if (na\_count == 0){

printf("NONE");

}

else{

for (int i = 0; i<na\_count; i++){

printf("p(%d), ",na[i]);

}

}

}

int main(void) {

int pros, hol;

printf("Enter the number of processes: ");

scanf("%d", &pros);

printf("\n");

printf("enter the number of holes: ");

scanf("%d", &hol);

int p[pros], h[hol];

printf("\n");

for (int i = 0; i<pros; i++){

printf("enter the size of process %d: ", i+1);

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

}

printf("\n");

for (int i = 0; i<hol; i++){

printf("enter the size of hole %d: ", i+1);

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

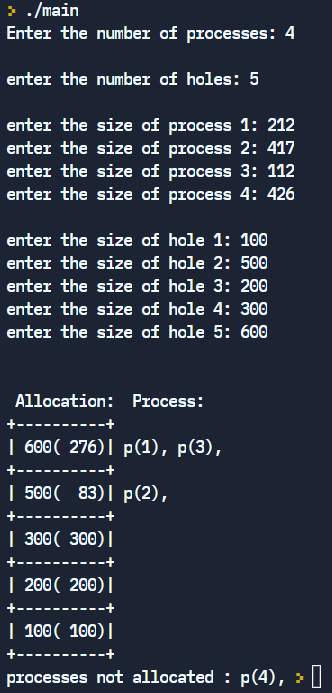
}

sort(h, hol);

calc(pros, hol, p, h);

}

**Output:**



**Result:**

Thus the dynamic allocation program has been executed successfully.

OS LAB EX 9

Implementation Of Page Replacement Algorithm

**Aim:** to simulate FIFO, LRU and optimal page replacement algorithms.

**Problem Description:**

**FIFO replacement:** In First In First Out replacement, When a page needs to be replaced page in the front of the queue is selected for removal.

**LRU replacement**: In Least Recently Used, the page which is not used for a long time is replaced.

**Optimal Page replacement:** In optimal the page that will be used very minimal in future will be replaced.

**FIFO:**

**Algorithm:**

1:Get the size of frames and no of requests from the user. Initialise page\_hit variable to 0.

2: Create an array to store the requests, array for the page.

3: For every request search through the page array for its occurrence

If it is present display “Page Hit” and increment the page\_hit variable.

Else: Display “Page miss”

If all pages in page array is consumed replacement the first element in the page array with the request.

Else place the request in the first not used page in page array.

4.Do step 3 till all the requests have been consumed.

5.Diplay the number of hits and number of misses.

**LRU**

**Algorithm:**

1:Ceate a LRU 2-D array to store the request and the number of occurrence.

2: Create a 1-D recently\_used array to store the last occurrence of each request

3. For every request look into LRU array to find its occurrence.

If it has already occurred display “Page hit” and increment the 1st index of LRU array and change its recently\_used value as the current iterator value.

Else: Display “Page miss”.

If there is space in LRU array create a new entry for it and have its occurrence as 1.In the recently\_used array store the current iterator value.

Else: Search in the recently\_used array for the index which was least used, replace that with the current request both in LRU array as well as recently\_used array.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<limits.h>

int find(int val,int frame[],int n){

        for(int i=0;i<n;i++){

            if(frame[i]==val){

                return 1;

            }

        }

        return -1;

}

int find\_index(int x,int nf,int lru[][nf]){

    for(int i=0;i<nf;i++){

        if(lru[i][0]==x && lru[i][1]>0){

            return i;

            break;

        }

    }

    return -1;

}

int least\_used(int nf,int lru[nf][nf],int recently\_used[]){

    int min\_index=0,final\_index=INT\_MAX;

    for(int i=0;i<nf;i++){

       if(recently\_used[min\_index]>recently\_used[i] && recently\_used[i]!=-1){

           min\_index=i;

       }

    }

    return min\_index;

}

void LRU(int request[],int n,int nf){

    int lru [nf][nf];

    int recently\_used[nf];

    memset(lru,0,sizeof(lru));

    memset(recently\_used,-1,sizeof(recently\_used));

    int k=0;

    for(int i=0;i<n;i++){

        int x=find\_index(request[i],nf,lru);

        if(x!=-1){

            printf("Page hit\n");

            lru[x][1]++;

            recently\_used[x]=i;

        }else{

            printf("Page miss\n");

            if(k<nf){

                lru[k][0]=request[i];

                lru[k][1]++;

                recently\_used[k]=i;

                k++;

            }else{

                int y=least\_used(nf,lru,recently\_used);

                lru[y][0]=request[i];

                lru[y][1]=1;

                recently\_used[y]=i;

            }

        }

        printf("Page no\t\t Occurance\t\t Last\_used\_index\t\n");

        for(int i=0;i<nf;i++){

            printf("%d\t\t %d\t  \t\t\t%d\n",lru[i][0],lru[i][1],recently\_used[i]);

            printf("\n");

        }

    }

}

int main(){

    int nf,n;

    printf("Enter the number of frames\n");

    scanf("%d",&nf);

    printf("Enter the size of request\n");

    scanf("%d",&n);

    int request[n];

    for(int i=0;i<n;i++){

        scanf("%d",&request[i]);

    }

    int frame[nf];

    memset(frame,-1,sizeof(frame));

    //FIFO

    printf("\n\n");

    printf("FIFO\n");

     printf("\n\n");

    int page\_hit=0;

    int j=0;

    for(int i=0;i<n;i++){

        int x=find(request[i],frame,nf);

        if(x==1){

            printf("Page Hit\n");

            page\_hit++;

        }else{

            printf("Page miss\n");

            if(j<nf){

                frame[j]=request[i];

                j++;

            } else{

                j=0;

                frame[j]=request[i];

                j++;

            }

        }

        printf("Current contents in table\n");

        for(int i=0;i<nf;i++){

            printf("%d ",frame[i]);

        }

       printf("\n");

    }

    printf("The number of page fault is %d ",n-page\_hit);

   //LRU

   printf("\n\n");

    printf("LRU\n");

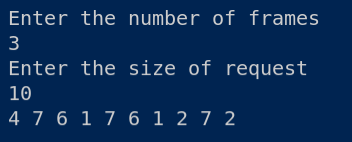
     printf("\n\n");

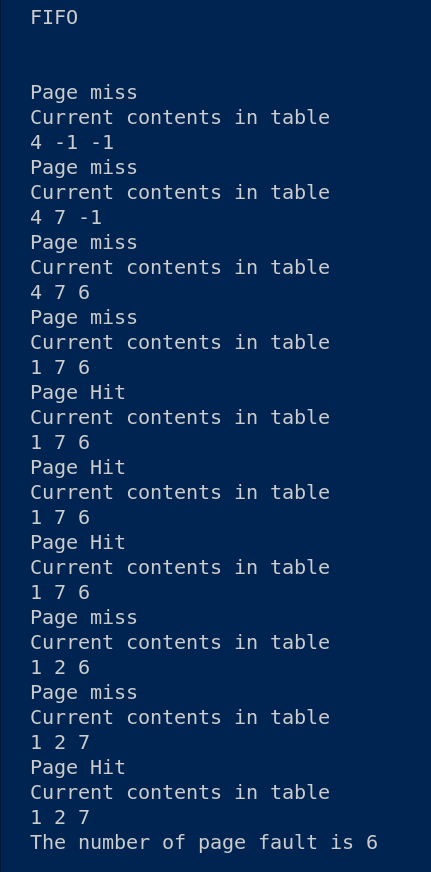
    LRU(request,n,nf);

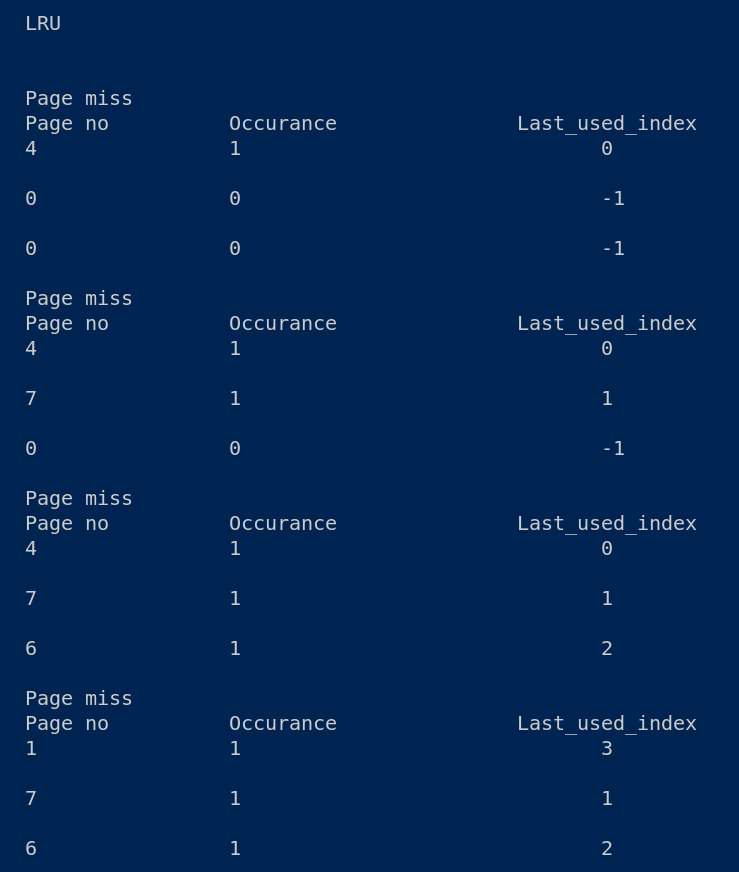
    return 0;

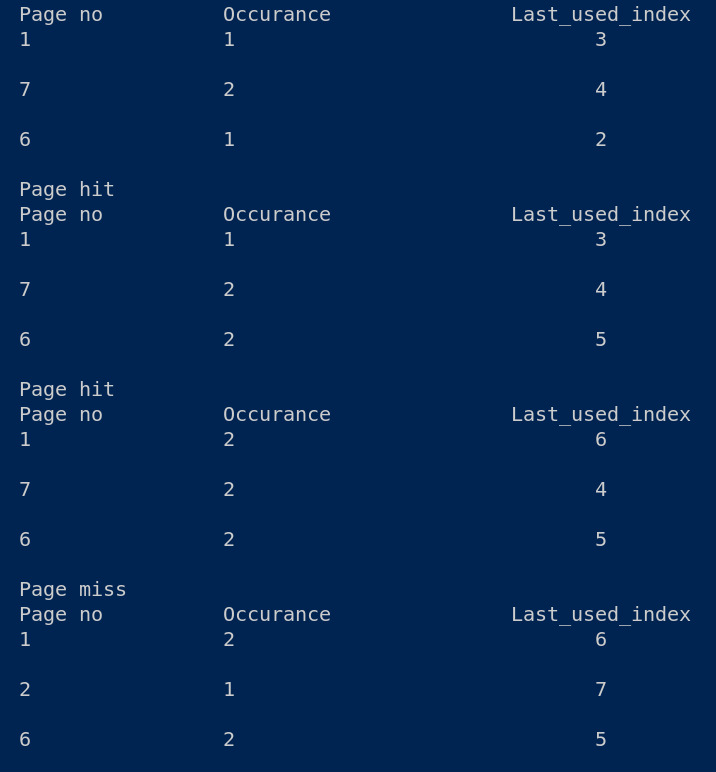
}

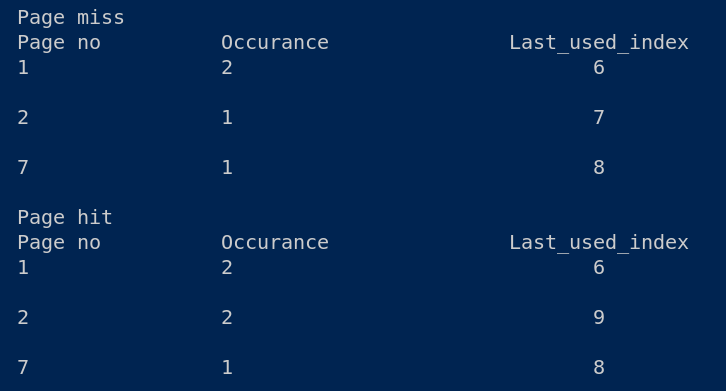
**Output:**

****

****

****

****

****

**Optimal Page replacement:**

**Algorithm:**

1: Get the size of frames and no of requests from the user. Initialise fault and hit variable to 0.

2: For every request call add\_reference function.

3:In add\_reference function checks the reference array to find it the page has already been allocated.

If already allocated display “page hit for that reference”

Else:

If there is empty space present in reference array store the request in that place and display “Fault for the reference”

Else: call the predict function by passing the current request.

The predict function returns the index of the request which is used least in the future. Replace that index with the current reference.

4:Do step 3 till all the requests have been consumed.

5:Diplay the number of hits and number of misses.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

int pointer;

int faults,hits;

int frame\_size,i,number\_of\_references;

int frame[1000],references[1000];

void print(){

    int i;

    for(int i=0;i<frame\_size;i++){

        if(frame[i]==-1){

            printf("-");

        }else{

            printf("%d ",frame[i]);

        }int reference[number\_of\_references];

    }

    printf("\n");

}

int predict(int reference\_length,int page\_no,int start){

    int pos=-1,farthest=start,i;

    for(i=0;i<frame\_size;i++){

        int j;

        for(j=start;j<reference\_length;j++){

            if(frame[i]==references[j]){

                if(j>farthest){

                    farthest=j;

                    pos=i;

                }

                break;

            }

        }

        if(j==page\_no){

            return i;

        }

    }

    if(pos==-1){

        return 0;

    }else{

        return pos;

    }

}

void add\_reference(int reference,int current\_position,int reference\_length){

    int i;

    bool allocated=false;

    for(i=0;i<frame\_size;i++){

        if(frame[i]==reference){

            printf("    hit for %d |",reference);

            hits++;

            allocated=true;

            break;

        } else if(frame[i]==-1){

            frame[i]=reference;

            printf("Fault for %d |",reference);

            faults++;

            allocated=true;

            break;

        }

    }

    if(allocated==false){

        int j=predict(reference\_length,current\_position,current\_position+1);

        frame[j]=reference;

        printf("Fault for %d|",reference);

        faults++;

    }

    print();

}

int main(){

    printf("Enter frame size: ");

    scanf("%d",&frame\_size);

    for(i=0;i<frame\_size;i++)

    {

        frame[i] = -1;

    }

    print(frame\_size,frame);

    printf("Enter the number of references: ");

    scanf("%d",&number\_of\_references);

    for(i=0;i<number\_of\_references;i++)

    {

        scanf("%d",&references[i]);

        add\_reference(references[i],i,number\_of\_references);

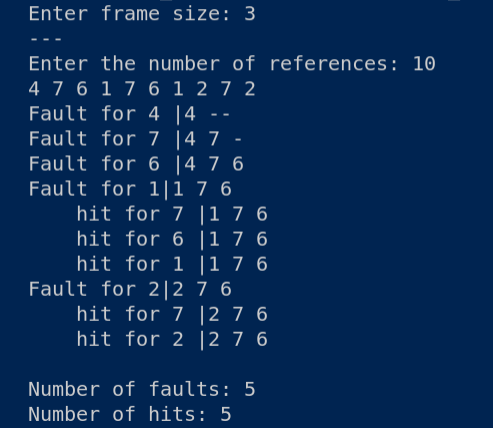
    }

    printf("\nNumber of faults: %d \nNumber of hits: %d\n",faults,hits );

   return 0;

}

**Output:**

****

**Result:**

Thus the page replacement algorithm has been executed successfully.

OS LAB EX 10

Designing a Virtual Memory Manager

**Aim:** to demonstrate paging

**Problem Description:**

Paging is a memory-management technique that provides the non-contiguous address space

in main memory.In this technique physical memory is broken into fixed-sized blocks called

frames and logical memory is divided into blocks of the same size called pages. When a

process is to be executed, its pages are loaded into any available memory frames from the

backing store.

The address generated by CPU is called logical address and it is divided into two parts a page

number (p) and a page offset (d). The page number is used as an index into a page table. The

page table contains the base address of each page in physical memory. The combination of

base address and page offset is used to map the page in physical memory address. Hardware

decides the page size.

**Algorithm:**

1: Start the process

2: Declare page number, page table, frame number and process size.

3: Read the process size, total number of pages

4: Read the relative address

5: Calculate the physical address

6: Display the address

7: Stop the process

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int main()

{

  srand(time(NULL));

  int frames,frame\_size,pages,page\_size;//16,4,5

  printf("Give the no. of frames: ");

  scanf("%d",&frames);

  printf("Give the frame size: ");

  scanf("%d",&frame\_size);

  page\_size=frame\_size;

  printf("Give the no. of pages: ");

  scanf("%d",&pages);

  int starting\_addr;

  printf("Starting address: ");

  scanf("%d",&starting\_addr);

  int page\_table[pages][2];

  printf("Initial state of page\_table: \n");

  for(int i=0,j=starting\_addr;i<pages;i++,j+=page\_size)

  {

    page\_table[i][0]=-1;

    page\_table[i][1]=j;

    printf("%d %d\n",page\_table[i][0],page\_table[i][1]);

  }

  int flag=0;

  for(int i=0;i<pages;i++)

  {

    int temp;

    do

    {

      temp=rand()%16;

    }while(flag & (1<<temp));

    flag=(1<<temp)|flag;

    page\_table[i][0]=temp;

  }

  printf("After allocation: \n");

  for(int i=0;i<pages;i++)

  {

    printf("%d %d\n",page\_table[i][0],page\_table[i][1]);

  }

  int page\_no,offset;

  do

  {

    printf("Page\_number(bet 0 and 4): ");

    scanf("%d",&page\_no);

    if(page\_no>4 && page\_no<0)

      printf("Illegal value!!\n");

  }while(page\_no>4 && page\_no<0);

  do

  {

    printf("Offset(bet 0 and 3): ");

    scanf("%d",&offset);

    if(offset>3 && offset<0)

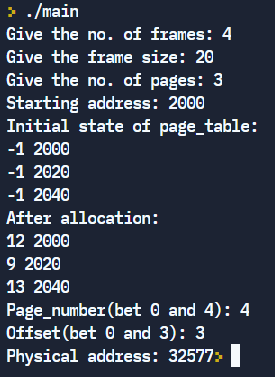
      printf("Illegal value!!\n");

  }while(page\_no>3 && page\_no<0);

  printf("Physical address: %d",page\_table[page\_no][1]+offset);

}

**Output:**



**Result:**

Thus the paging hardware has been executed successfully.

OS LAB EX 11

Implementation Of Disk Scheduling Algorithms

**Aim:**

To simulate FIFO, SSTF, scan disk scheduling algorithm.

**Problem Description:**

**FIFO disk scheduling:** In First In First Out scheduling, order of arrival disk request is maintained while processing the disk.

**SSTF disk scheduling**: In Shortest Seek Time First, that request next which requires least number of head movements from its current position regardless of the direction.

**Scan disk scheduling:** Thehead starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk.

**FIFO:**

**Algorithm:**

1:Store the initial position of head in seek variable.

2:Store the number of disk requests in no\_disk\_entry variable.

3:Store the disk requests in the disk\_head array

4:Initialise variable x value as seek

5:Initlialise f\_seek value to 0.

6.For every request find the absolute difference between x and the current disk request,and add the same to f\_seek.initialise the value of x to current disk request.

7.Display f\_seek as final result.

**SSTF:**

**Algorithm:**

1:Initalise f\_seek to 0.

2:Initialise y to seek and create index variable,temp array which has same record as disk\_head array.

3:Run a while loop till value of x matches no\_disk\_entry

Initialise mini variable to 111111

Run a for loop to find the minimum absolute difference between current temp index value and y,the tie is broken by choosing the right most request.

Add the difference to f\_seek

Make that request as y

Change the value of index to INT\_MAX

4:Display f\_seek as final seek value.

**SCAN**

**Algorithm:**

1:Add three new entries in the disk\_head array which are 0 , the seek value,the upper bound of the disk.

2:Initialise f\_seek=0.

3.Sort the array

4.Get the choice from the user whether to check from 1.Left 2.Right

5.If choice ==1

Display the f\_seek as sum of last entry and seek

Print the disk request from seek pos to 0 index and from seek index to last index as the order of execution

Else

Display f\_seek as the difference between last entry in disk\_head array and seek value and sum it with (the difference between the last disk\_head entry and 2nd disk\_head entry).

Print the disk request from idx+1 to end and idx-1 to the beginning as the order of execution.

**Code:**

#include<stdio.h>

#include<math.h>

#include<limits.h>

int no\_disk\_entry,seek,f\_seek;

int disk\_head[100];

int temp\_disk[100];

void fcfs(){

f\_seek=0;

int x=seek;

    for(int i=0;i<no\_disk\_entry;i++){

        f\_seek+=abs(disk\_head[i]-x);

        x=disk\_head[i];

    }

    printf("The seek time in fcfs  is %d\n",f\_seek);

}

void sstf(){

    f\_seek=0;

    int x=0,index,y=seek;

    while(x!=no\_disk\_entry){

        int mini=99999;

        for(int i=0;i<no\_disk\_entry;i++){

            if(abs(y-temp\_disk[i])<=mini && temp\_disk[i]!=INT\_MAX)

            {

                if((y-temp\_disk[i]) == mini) // breaking tie - right direction

                {

                    continue;

                }

                mini=abs(y-temp\_disk[i]);

                index=i;

            }

            }

            printf("%d %d\n",mini,index);

            f\_seek+=mini;

            y=temp\_disk[index];

            temp\_disk[index]=INT\_MAX;

            x++;

    }

     printf("The seek time in sstf  is %d\n",f\_seek);

}

void sort(int arr[],int n){

    for(int i=0;i<n;i++){

        for(int j=0;j<n-i-1;j++){

            if(arr[j]>arr[j+1]){

                int temp=arr[j];

                arr[j]=arr[j+1];

                arr[j+1]=temp;

            }

        }

    }

}

void scan(){

    int upper\_bound;

    printf("Enter the upperbound value\n");

    scanf("%d",&upper\_bound);

    f\_seek=0;

    int c;

    disk\_head[no\_disk\_entry]=0;

    disk\_head[no\_disk\_entry+1]=seek;

    disk\_head[no\_disk\_entry+2]=upper\_bound;

    sort(disk\_head,no\_disk\_entry+3);

    int idx;

    for(int i=0;i<=no\_disk\_entry+2;i++){

        if(disk\_head[i]==seek){

            idx=i;

            break;

        }

    }

    printf("Enter choice \n 1.left 2.right\n");

    scanf("%d",&c);

    if(c==1){

        printf("Total seek is %d\n",disk\_head[no\_disk\_entry+2]+seek );

        printf("Order of execution\n");

        for(int i=idx-1;i>=0;i--){

            printf("%d \n",disk\_head[i]);

        }

        for(int i=idx+1;i<=no\_disk\_entry+1;i++){

            printf("%d \n",disk\_head[i]);

        }

    }else{

         printf("Total seek is %d\n",disk\_head[no\_disk\_entry+2]-seek + disk\_head[no\_disk\_entry+2]-disk\_head[1]);

         printf("Order of execution\n");

         for(int i=idx+1;i<=no\_disk\_entry+2;i++){

            printf("%d \n",disk\_head[i]);

        }

         for(int i=idx-1;i>0;i--){

            printf("%d \n",disk\_head[i]);

        }

    }

}

int main(){

    printf("Enter the no.of disk access\n");

    scanf("%d",&no\_disk\_entry);

    printf("Enter the inital seek\n");

    scanf("%d",&seek);

    for(int i=0;i<no\_disk\_entry;i++){

        printf("Enter :");

        scanf("%d",&disk\_head[i]);

        temp\_disk[i]=disk\_head[i];

    }

    printf("\n Seek time in FCFS disk scheduling\n");

    fcfs();

    printf("\nSSTF disk scheduling\n");

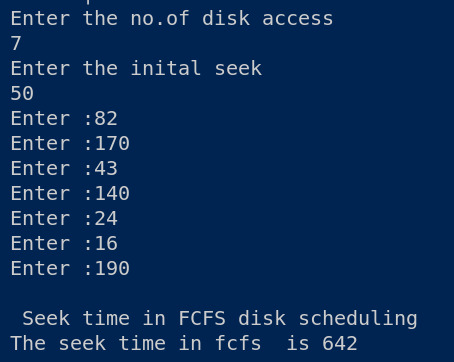
    sstf();

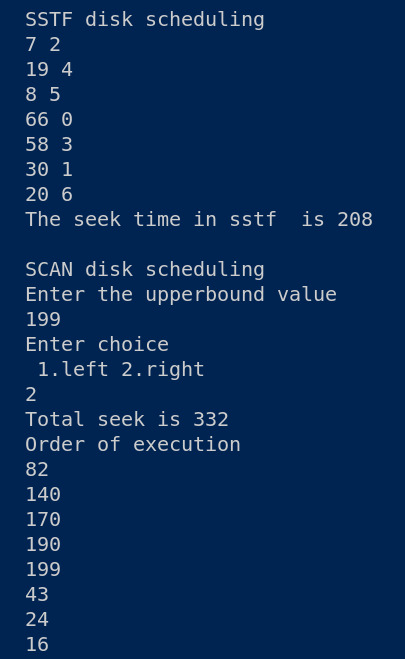
     printf("\nSCAN disk scheduling\n");

    scan();

}

**Output:**

****

****

**Result:**

Thus the disk scheduling algorithm has been executed successfully.

OS LAB EX 12

Implementation Of File Allocation and File Organisation Methods

**Aim:**

To simulate sequential. Linked and Indexed file allocation.

To simulate single, two level and hierarchal file organisation.

**Problem Description:**

**Sequential File Allocation**: Each file occupies a contiguous set of blocks on the disk.

**Linked File Allocation**: Each file is a linked list of disk blocks which**need not be**contiguous

**Indexed File Allocation:** a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.

**Single level file organisation:** A single directory has all the files under it.

**Two level file organisation**: There is a directory per the number of users to store their files.

**Hierarchal file organisation:** There are directories as well as sub directories for each user to store their files.

**Sequential file allocation:**

**Algorithm:**

1:Create an empty array to store the file as file\_allocated.

2:Create st,len ,i,j and choice variables, initialise c to 0.

3:create a goto start statement x where c is initialised as 0.

4:Get the start address and block size of each file

5:Loop through file\_allocated array to check for empty space of size of the request

If continuous space of required size is free store the file in that location.And display “File allocated ” message

Else display file not allocated

6:Get choice from user whether to enter more files 1.yes 2.no

If 1 goto x

Else exit

**Code:**

#include<stdio.h>

#include<string.h>

int main(){

char directory\_name[100],files[10][1000],name[100];

int n;

printf("Enter the directory name\n");

scanf("%s",directory\_name);

printf("Enter the number of files\n");

scanf("%d",&n);

int x=0,j=0,f=0,c;

while(x!=n){

    start:

    f=0;

    printf("Enter the file name\n");

    scanf("%s",name);

    for(int i=0;i<n;i++){

        if(strcmp(name,files[i])==0){

            f=1;

            break;

        }

    }

    if(!f){

        strcpy(files[j++],name);

        x++;

    }else{

        printf("File already exists\n Do you want to enter again 1.yes 2.no\n");

        scanf("%d",&c);

        if(c){

            goto start;

        }else{

            break;

        }

    }

}

printf("The directory name is %s\n",directory\_name);

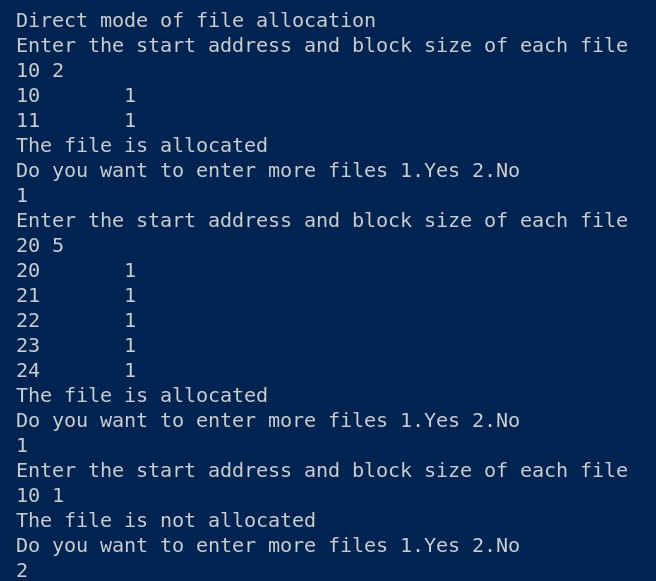
printf("The files are \n");

for(int i=0;i<x;i++){

    printf("%s \n",files[i]);

}}

**Output:**

****

**Linked File Allocation:**

**Algorithm:**

1:Create a struct list which has block of size int and struct list\* next.Typedef that struct to list.

2:Create a struct file which has char array n of size 10,s of type int ,b of ty int and List\*head.

3:Create a array of structure of type File,integer array freeb of size 100,mem,bsize,n,nf,c,r,f of type integer.

4:Get the size of memory block and store in mem variable.

5:Get the size of each block and store in bsize variable

6:Print the number of blocks as mem/bsize.

7:Initalise n as m/bsize.

8:Run a for loop till n/3 and use random function to fill up freeb.

9:Run a for loop to display the free blocks in freeb array where freeb value is 0.

10:Get the number of files and store in f.

11:Run a for loop till f

Get the name of file and store in files[c].n

Get the size and store in files[c].s

Check if files[c].s/bsize is greater than files[c].b

If true increment (files[c].b)

If files[c].b>nf

Display cant allocate

Else

Create a List pointer t and p

Loop through files[c].b and randomly choose freeb where it is free and decrement nf

In case of first allocation make files[c].head as t and p as files[c].head

Else make p->next as t and p as t

If there is no free space decrement the iterator

If nf==0

Display “Memory over”

12:Display the allocated files using for loop by looping over files.head

**Code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct list {

    int block;

    struct list \*next;

}List;

typedef struct file {

    char n[10];

    int s, b;

    List\* head;

}File;

File files[100];

int freeb[100], mem, bsize, n, nf, c, r, f;

List\* newnode(int n)

{

    List\* p = (List\*)malloc(sizeof(List));

    p->block = n;

    p->next = NULL;

    return p;

}

void print(List\* head)

{

    for(List\* t = head; t != NULL; t = t->next)

    {

        printf("%d",t->block);

        if(t->next == NULL) printf("\n");

        else printf("->");

    }

}

int main()

{

    printf("Enter size of memory in KB: ");

    scanf("%d", &mem);

    printf("Enter size of block in KB: ");

    scanf("%d", &bsize);

    printf("No. of blocks = %d\n", mem/bsize);

    n = mem/bsize;

    nf = n;

    for(int i = 0; i <= n/3; i++) {

        r = random()%n;

        if(freeb[r] == 1) {

            i--;

        }

        else {

            freeb[r] = 1;

            nf--;

        }

    }

    printf("Free blocks:\n");

    for(int i = 0; i < n; i++) {

        if(freeb[i] == 0) printf("%d ", i);

    }

    printf("\nEnter no. of files: ");

    scanf("%d", &f);

    for(int i = 0; i < f; i++) {

        printf("Enter name of file %d: ", i+1);

        scanf("%s", (files[c].n));

        printf("Enter size in KB: ");

        scanf("%d", &files[c].s);

        files[c].b = files[c].s/bsize;

        if(files[c].s\*1.0/bsize > files[c].b) (files[c].b)++;

        if(files[c].b > nf) {

            printf("Can't allocate!\n");

            i--;

        }

        else {

            List \*t, \*p;

            for(int j = 0; j < files[c].b; j++) {

                r = random()%n;

                if(freeb[r] == 0) {

                    freeb[r] = 1;

                    nf--;

                    t = newnode(r);

                    if(j == 0) {

                        files[c].head = t;

                        p = files[c].head;

                    }

                    else {

                        p->next = t;

                        p = t;

                    }

                }

                else j--;

            }

            c++;

        }

        if(nf == 0) {

            printf("Memory over!\n");

            f = c;

            break;

        }

    }

    printf("\nFile allocation:\n");

    for(int i = 0; i < f; i++) {

        printf("File %s:\n",files[i].n);

        print(files[i].head);

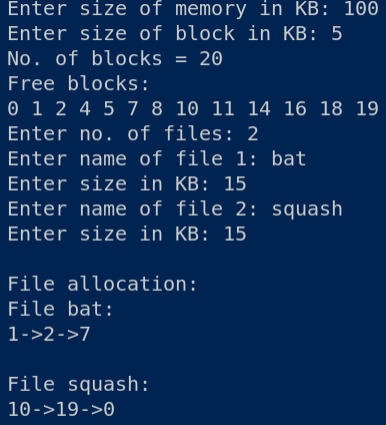
        printf("\n");

    }

    return 0;

}

**Output:**

****

**Indexed File Allocation:**

**Algorithm:**

1:Create a struct file with char array n of size 10,s ,b i of size int.

2:Create a struct indexblock with int blockid and int blocks array of size 100.

3:Create array of struct of type indexblock of size 100 and array of struct of type files of size 100,freeb array of size 100,mem,bsize,n,nf,c,r,f of integer type.

4:Get the size of memory block and store in mem variable.

5:Get the size of each block and store in bsize variable

6:Print the number of blocks as mem/bsize.

7:Initalise n as m/bsize.

8:Run a for loop till n/3 and use random function to fill up freeb.

9:Run a for loop to display the free blocks in freeb array where freeb value is 0.

10:Get the number of files from the user

11:Run a for loop to get the name of the file and size of the file

initialise files[c].b=files[c].s/bsize

Check if files[c].s/bsize is greater than files[c].b

If true increment (files[c].b)

If files[c].b+1>nf

Display cant allocate

Else: choose a random block id using random function and initialize files[c].i=r

Loop through files[c].b and randomly choose freeb where it is free and decrement nf.Initialise that index[c].blocks[j] as r.

Else

Decrement the iterator

If nf==0 Display memory over.

12:Display the indexed file allocation table using for loop.

**Code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct file {

    char n[10];

    int s, b;

    int i;

}File;

typedef struct indexblock {

    int blockid;

    int blocks[100];

} IndexBlock;

IndexBlock index[100];

File files[100];

int freeb[100], mem, bsize, n, nf, c, r, f;

void print(int i)

{

    printf("Index block no.: %d\nBlock table:\n", index[i].blockid);

    for(int j = 0; j < files[i].b; j++) {

        printf("%d ", index[i].blocks[j]);

    }

}

int main()

{

    printf("Enter size of memory in KB: ");

    scanf("%d", &mem);

    printf("Enter size of block in KB: ");

    scanf("%d", &bsize);

    printf("No. of blocks = %d\n", mem/bsize);

    n = mem/bsize;

    nf = n;

    for(int i = 0; i <= n/3; i++) {

        r = random()%n;

        if(freeb[r] == 1) {

            i--;

        }

        else {

            freeb[r] = 1;

            nf--;

        }

    }

    printf("Free blocks:\n");

    for(int i = 0; i < n; i++) {

        if(freeb[i] == 0) printf("%d ", i);

    }

    printf("\nEnter no. of files: ");

    scanf("%d", &f);

    for(int i = 0; i < f; i++) {

        printf("Enter name of file %d: ", i+1);

        scanf("%s", files[c].n);

        printf("Enter size in KB: ");

        scanf("%d", &files[c].s);

        files[c].b = files[c].s/bsize;

        if(files[c].s\*1.0/bsize > files[c].b) (files[c].b)++;

        if(files[c].b + 1 > nf) {

            printf("Can't allocate!\n");

            i--;

        }

        else {

            do {

                r = random()%n;

            }while(freeb[r] == 1);

            index[c].blockid = r;

            files[c].i = r;

            freeb[r] = 1;

            nf--;

            for(int j = 0; j < files[c].b; j++) {

                r = random()%n;

                if(freeb[r] == 0) {

                    freeb[r] = 1;

                    nf--;

                    index[c].blocks[j] = r;

                }

                else j--;

            }

            c++;

        }

        if(nf == 0) {

            printf("Memory over!\n");

            f = c;

            break;

        }

    }

    printf("\nFile Allocation:\n");

    for(int i = 0; i < f; i++) {

        printf("File %s:\n",files[i].n);

        print(i);

        printf("\n");

    }

    return 0;

}

**Output:**

****

**Single level file organisation:**

**Algorithm:**

1:Create a char array of name directory\_name of size 100,2-D char array named file and name array of size 100.

2:Declare int variable n

3:Get the directory name and store into directory\_name.

4:Get the number of files and store into n.

5:Create and initialise integer variables x as 0,j as 0,f as 0 and c.

6:Run a while loop till x equals n

Create a goto variable start

Initialise f as 0

Get the file name and store into name

Run a for loop to check if the name of file already exists,If so display File already exists.Do you want to enter again 1.yes 2.No

If yes goto start else break

If file is not found store the name of file into files array.

7:Display the single level file organisation using for loop

**Code:**

#include<stdio.h>

#include<string.h>

int main(){

char directory\_name[100],files[10][1000],name[100];

int n;

printf("Enter the directory name\n");

scanf("%s",directory\_name);

printf("Enter the number of files\n");

scanf("%d",&n);

int x=0,j=0,f=0,c;

while(x!=n){

    start:

    f=0;

    printf("Enter the file name\n");

    scanf("%s",name);

    for(int i=0;i<n;i++){

        if(strcmp(name,files[i])==0){

            f=1;

            break;

        }

    }

    if(!f){

        strcpy(files[j++],name);

        x++;

    }else{

        printf("File already exists\n Do you want to enter again 1.yes 2.no\n");

        scanf("%d",&c);

        if(c){

            goto start;

        }else{

            break;

        }

    }

}

printf("The directory name is %s\n",directory\_name);

printf("The files are \n");

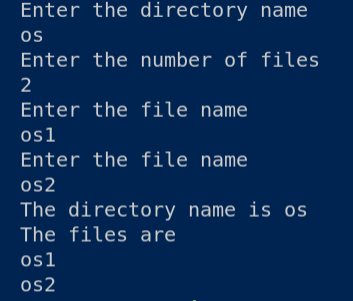
for(int i=0;i<x;i++){

    printf("%s \n",files[i]);

}

}

**Output:**

****

**Two level file organisation**:

**Algorithm:**

1:Create a struct with char array dname of size 10,2-D char array.

2:Create array of struct dir

3:Create integer variables I,ch,dcnt,k,char array f of size 30 and d of size 30.

4:Initialise dcnt to 0.

5:Run a while loop and get the user choice 1.Create a directory 2.Create file 3.Display 4.Exit

Case 1: Get the name of the directory .Initialise dire[dcnt].fcnt to 0.

Print “directory created”.

Case 2:Get the name of the directory and check if such directory exists using a for loop. If exist Get the name of the file from the user and store the same to dir[i].fname[dir[i].fcnt].Increment dir[i].fcnt .

Else display “Directory not found”.

6:Display the two level file organisation

**Code:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct {

    char dname[10],fname[10][10];

    int fcnt;

}dir[10];

int main(){

    int i,ch,dcnt,k;

    char f[30],d[30];

    dcnt=0;

    while(1){

        printf("\n1.Create Directory\t2.Create file\t3.Display\t4.Exit\nEnter the choice :");

        scanf("%d",&ch);

        switch(ch){

            case 1:printf("\nEnter the name of the directory--");

            scanf("%s", dir[dcnt].dname);

            dir[dcnt].fcnt=0;

            dcnt++;

            printf("Directory created");

            break;

            case 2:printf("Enter the name of the directory--");

            scanf("%s",d);

            for(i=0;i<dcnt;i++){

                if(strcmp(d,dir[i].dname)==0){

                    printf("Enter the name of the file\n");

                    scanf("%s",dir[i].fname[dir[i].fcnt]);

                    dir[i].fcnt++;

                    printf("File created\n");

                    break;

                }

            }

            if(i==dcnt){

                printf("Directory %s not found",d);

            }

            break;

            case 3:

                if(dcnt==0){

                    printf("\n No directory\n");}else{

                    for(int i=0;i<dcnt;i++){

                        printf("\n%s\t\t",dir[i].dname);

                        for(int k=0;k<dir[i].fcnt;k++){

                            printf("\t%s",dir[i].fname[k]);

                        }

                }}

                break;

            case 4:exit(0);

            default:exit(0);

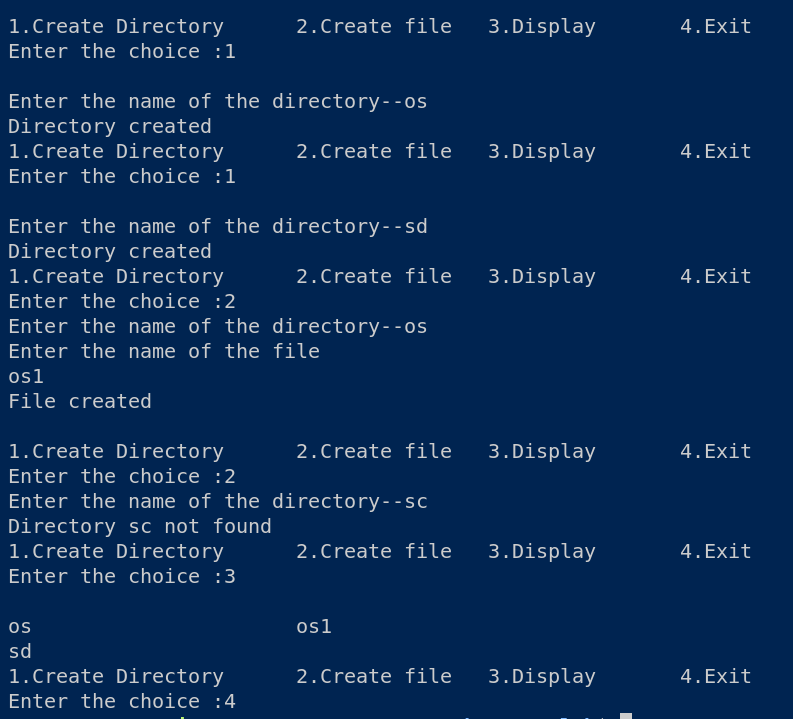
        }

    }

    return 0;

}

**Output:**

****

**Hierarchal file organisation:**

**Algorithm:**

1:Create struct node with char N of size 25,df of type int,struct node\*pc and \*ps. Create global array of struct pointers A of size 20,intger variable in as 0 and c as 0.

2:Create nu,nc , i , j ,k integer variables.

3:Create struct variable hdr and allocate memory using malloc.Initialise hdr->df as 1 and pc and ps as NULL.

4:Get the number of users and store into nu.Call create function by passing hdr,nu

Create(struct node \*P,int N)

Create I,struct node\*temp,\*t.

Initialise tmp as P

Loop till N and allocate memory for t.

Get the name and ask if it is 1.dir 2.file

If 1 make A[c] as t and increment c.

Make t->pc and t->ps point to NULL.

If 0 make tmp->p as t and tmp as t

Else make tmp->ps as t and tmp as t.

5:Display the Hierarchal file organisation using a for loop.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    char N[25];

    int df;

    struct node \*pc;

    struct node \*ps;

};

struct node \*A[20];

int in = 0, c = 0;

void create(struct node \*P,int N)

{

    int i;

    struct node \*tmp,\*t;

    tmp = P;

    for(int i=0;i<N;i++)

    {

        t = malloc(sizeof(struct node));

        printf("Enter name : ");

        scanf("%s",t->N);

        printf("Is it a dir(1) or file(0) : ");

        scanf("%d",&t->df);

        if(t->df == 1)

        {

            A[c] = t;

            c++;

        }

        t->pc = NULL;

        t->ps = NULL;

        if(i==0)

        {

            tmp->pc = t;

            tmp = t;

        }

        else

        {

            tmp->ps = t;

            tmp = t;

        }

    }

}

void display(struct node \*P)

{

    int i;

    P = P->pc;

    do

    {

        printf("\n%s(%d)",P->N,P->df);

        if(P->df == 1 && P->pc != NULL)

            display(P);

        P = P->ps;

    } while(P != NULL);

}

int main()

{

    int nu,nc;

    int i,j,k;

    struct node \*hdr;

    hdr = malloc(sizeof(struct node));

    hdr->df = 1;

    hdr->pc = NULL;

    hdr->ps = NULL;

    printf("Enter no.of users : ");

    scanf("%d",&nu);

    create(hdr,nu);

    for(in = 0;in < c;in++)

    {

        printf("Enter no. of child nodes for %s : ",A[in]->N);

        scanf("%d",&nc);

        create(A[in],nc);

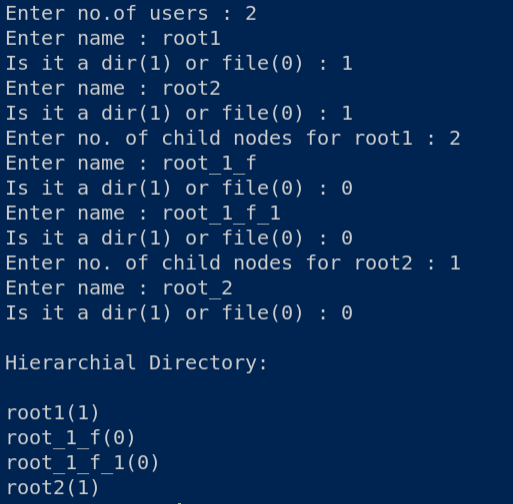
    }

    printf("\nHierarchial Directory:\n ");

    display(hdr);

}

**Output:**

****

**Result:**

Thus the file allocation and organisation techniques has been compiled and executed successfully.

OS LAB

Mini Project-Simulation Of Distributed File system

**Aim:**

Multiple clients can access the server remotely and can issue remote procedural commands to read the files in the server.

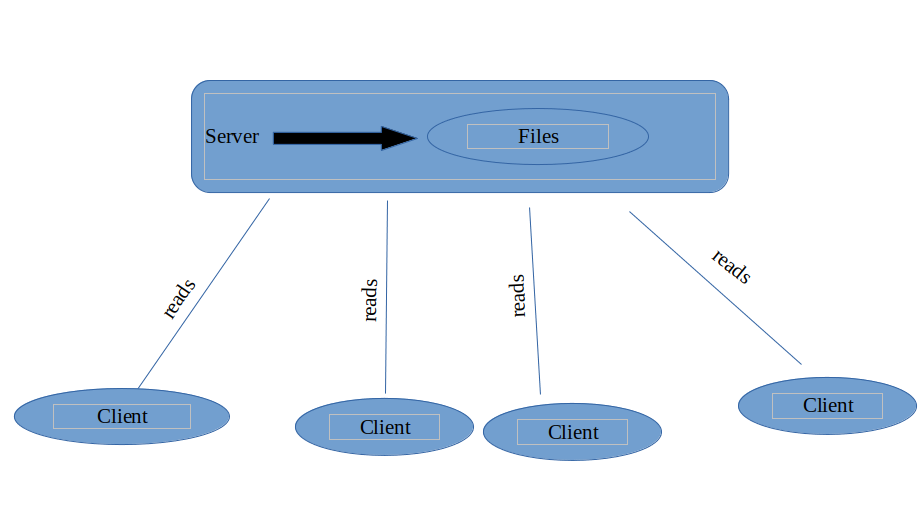
**Problem Description:**

This project is a rudimentary implementation of Network file system, a type of distributed file system where multiple clients access the server via TCP connection and access(read) the files in it.

Main Concepts involved:

* Threads: implemented using pthread library for handling multiple clients
* Socket Programming: For communication between server and client in 2 different locations! Main header files used for socket programming:
* Sys/socket.h
* Sys/types.h
* Arpa/inet.h
* Netinet/in.h
* Errno.h(for handling errors like timeout)
* Other concepts and header files:
* Stdio.h : for standard input and output
* Stdlib.h : Standard library for functions like sleep, exit, getc()
* String.h : for manipulation of strings sent between user and server
* Unistd.h : it is the standard unix function for miscellaneous functions and constants.

Design:



**Code:**

**Server.cpp**

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<string.h>

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<bits/stdc++.h>

#include <dirent.h>

#include<fstream>

#define PORT 5000

#define PORT\_DISK 6000//port for socket that communicate with the warehouse

#define MAX\_DISK\_LENGTH 100

#define DISK 3//no. of connected disks can be deleted as and when required

using namespace std;

int socketforclient,socketfordisks;

struct sockaddr\_in servaddr,serfordisk,cliaddr;

void find\_disk\_send\_request(int contoclient,char search\_file[]){

char disk\_msg[50];

char line[256];

FILE\* demo;

int count = 0;

string ln;

ifstream file(search\_file);

while(getline(file,ln))

count++;

int arr[1];

arr[0] = count;

send(contoclient,arr,sizeof(arr),0);

demo = fopen(search\_file, "r");

if(demo == NULL)

{

char fail[] = "File not found";

send(contoclient,fail,sizeof(fail),0);

}

else{

while (fgets(line, sizeof(line), demo)) {

strcpy(disk\_msg,line);

send(contoclient,disk\_msg,sizeof(disk\_msg),0);

}

}

}

void\* name\_as\_you\_want(void \*arg)

{

int contoclient = \*(int \*)arg;

char user\_name[50],password[50],client\_file[50];

char get\_username[]="Enter the username :";

char get\_password[]="Enter the password :";

char authentication\_message[50]="Success";

send(contoclient,get\_username,sizeof(get\_username),0);

read(contoclient,user\_name,sizeof(user\_name));

send(contoclient,get\_password,sizeof(get\_password),0);

read(contoclient,password,sizeof(password));

if(strcmp(password,"1234")==0){

send(contoclient,authentication\_message,sizeof(authentication\_message),0);

char request\_file[]="Send the path of desired file without '/'";

send(contoclient,request\_file,sizeof(request\_file),0);

read(contoclient,client\_file,sizeof(client\_file));

} else{

char failed[]="Access Denied";

send(contoclient,failed,sizeof(failed),0);

}

close(contoclient);

pthread\_detach(pthread\_self());

return NULL;

}

int main()

{

if((socketforclient=socket(AF\_INET,SOCK\_STREAM,0))<0)

{

perror("Error in socket creation!!\n");

exit(0);

}//passive socket for client communication

if((socketfordisks=socket(AF\_INET,SOCK\_STREAM,0))<0)

{

perror("Error in socket creation!!\n");

exit(0);

}//passive socket for disk communication

memset(&servaddr,0,sizeof(servaddr));

memset(&cliaddr,0,sizeof(cliaddr));

memset(&serfordisk,0,sizeof(serfordisk));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(9000);

serfordisk.sin\_family = AF\_INET;

serfordisk.sin\_addr.s\_addr = INADDR\_ANY;

serfordisk.sin\_port = htons(PORT\_DISK);//init servaddr and serfordisk

if(bind(socketforclient,(struct sockaddr \*)&servaddr,sizeof(servaddr))<0)

{

printf("Bind Failed!!");

exit(0);

}//binding port no. with socket - req for client side

if(bind(socketfordisks,(struct sockaddr \*)&serfordisk,sizeof(serfordisk))<0)

{

printf("Bind Failed!!");

exit(0);

}// same procedure but for connecting with disks

listen(socketforclient,5);//TCP queue for clients

listen(socketfordisks,10);//TCP queue for disks

pthread\_t pid;

socklen\_t len;

while(1)

{

int contoclient=accept(socketforclient,(struct sockaddr \*)&cliaddr,&len);//accept the connect of client

printf("Thread");

pthread\_create(&pid,NULL,&name\_as\_you\_want,(void \*)&contoclient);//thread created for auth and subsequently other things

}//ever running loop for accomodating with clients

void\* val;

pthread\_join(pid, &val);

return 0;

}

**Client.c**

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <errno.h>

int main(int argc, char const \*argv[])

{

int sockfd, confd,len;

struct sockaddr\_in servaddr,cli;

sockfd = socket(AF\_INET,SOCK\_STREAM,0);//Creating a socket for client

if(sockfd<0)

{

perror("Socket Not created!!");

}

else

{

printf("Socket Successfully created!\n");

}

memset(&servaddr,0,sizeof(servaddr));//Flushing previously stored values in servaddr

memset(&cli,0,sizeof(cli));//Flushing previously stored values in cli

int port;//for getting port number from cmd args

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

// sscanf(argv[2],"%d",&port);//basically fscanf but passing string instead of file concepts

servaddr.sin\_port = htons(9000);

int conval = connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr));//Connection Establishment between client and server

if(conval < 0)

{

perror("Server Connection failed...\n");

if(errno==ETIMEDOUT)

{

printf("Connection Timed out!!!\n");

exit(0);

}

exit(0);

}

else

{

printf("Connection Established!!\n");

char server\_username[100],server\_password[50],authentication[50],file\_request[50],server\_reply[50];

char username[50];

char password[50];

recv(sockfd,&server\_username,sizeof(server\_username),0);

scanf(" %[^\n]", username);

send(sockfd,username,sizeof(username),0);

recv(sockfd,&server\_password,sizeof(server\_password),0);

scanf(" %[^\n]", password);

send(sockfd,password,sizeof(password),0);

recv(sockfd,&authentication,sizeof(authentication),0);

printf("%s\n",authentication);

if(strcmp(authentication,"Success")==0)

{

recv(sockfd,&file\_request,sizeof(file\_request),0);

char filepath[50];

scanf(" %[^\n]", filepath);

send(sockfd,filepath,sizeof(filepath),0);

int arr[1];

int lines;

recv(sockfd,&arr,sizeof(arr),0);

lines = arr[0];

printf("No.of lines : %d\n",lines);

for(int i=0;i<lines;i++)

{

recv(sockfd,&server\_reply,sizeof(server\_reply),0);

printf("%s",server\_reply);

}

}

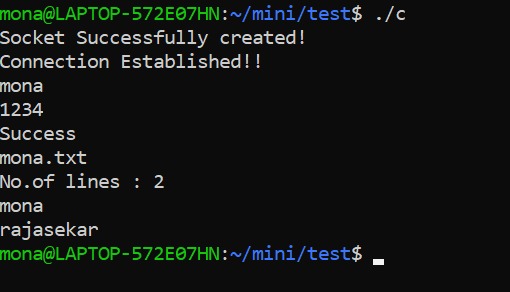
close(sockfd);

}

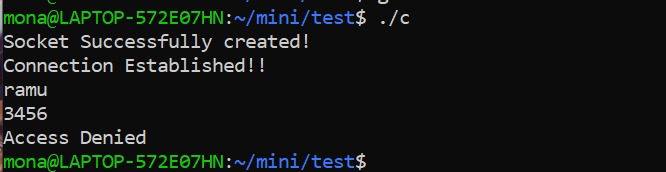
}

**Output:**

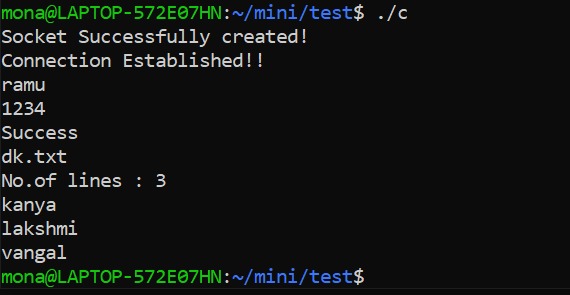
Single client accessing

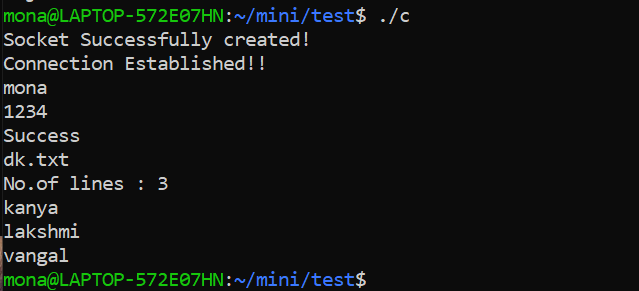


Client entering wrong password



Multiple client accessing file at same time





**Result:**

Thus the distributed file system has been simulated and executed.