21CSC202J - OPERATING SYSTEMS

LAB MANUAL

II YEAR / III Semester

FACULTY OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CHENNAI- 600 089

LIST OF EXPERIMENTS

Ex.No.	Experiment Name
1	A. Operating system Installation, B. Basic Linux commands, C.Filter and admin commands
2	Process Creation using fork() and Usage of getpid(), getppid(), wait() functions
3	Multithreading
4	Mutual Exclusion using semaphore and monitor
5	Reader-Writer problem
6	Dining Philosopher problem
7	Bankers Algorithm for Deadlock avoidance
8	FCFS and SJF Scheduling
9	Priority and Round robin scheduling
10	FIFO Page Replacement Algorithm
11	LRU and LFU Page Replacement Algorithm
12	Best fit and Worst fit memory management policies
13	Disk Scheduling algorithm
14	Sequential and Indexed file Allocation
15	File organization schemes for single level and two level directory

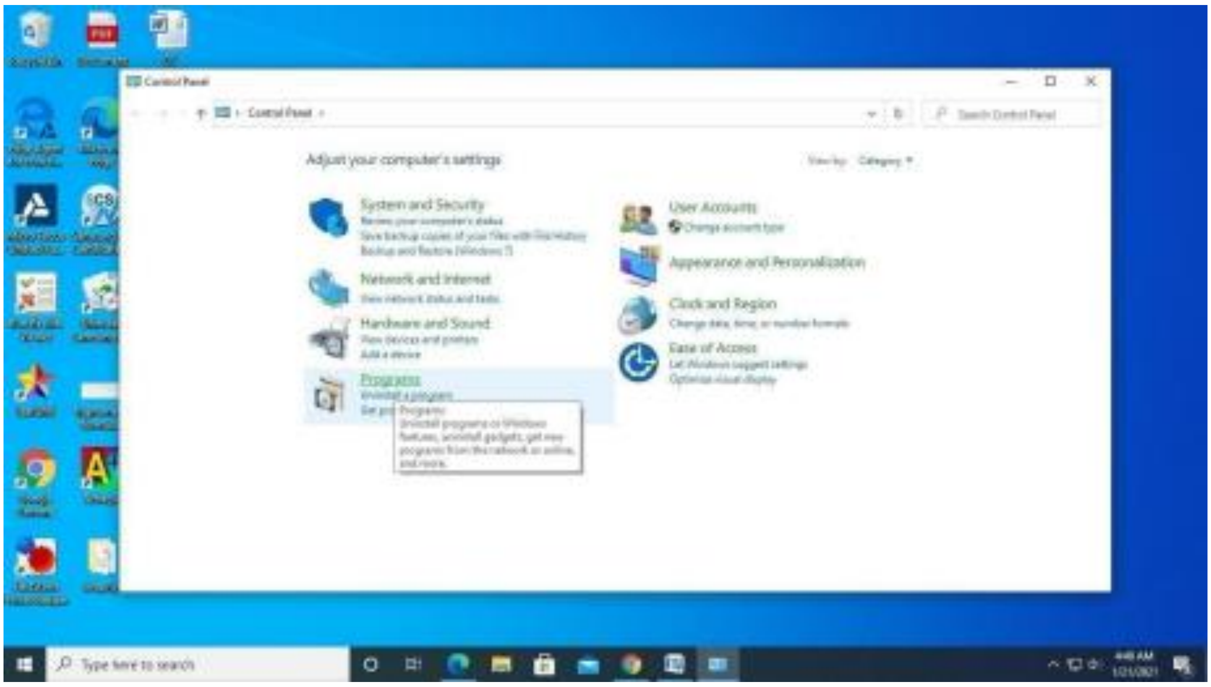
Ex No. 1a	OPERATING SYSTEM INSTALLATION

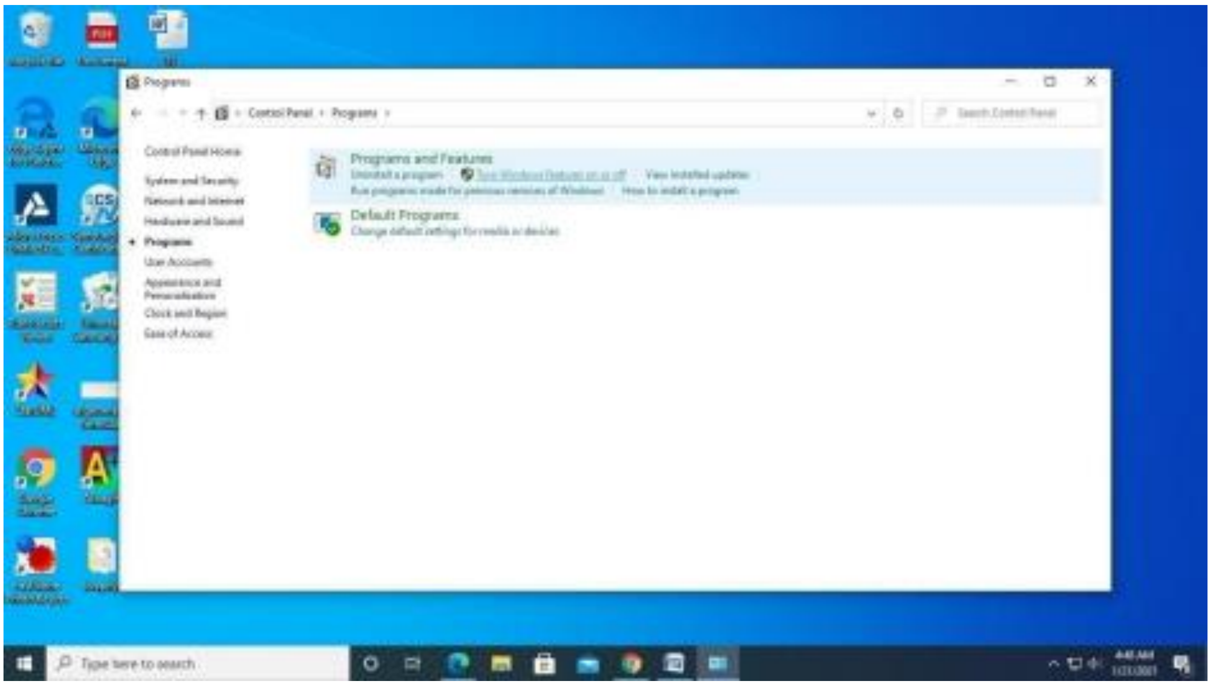
Linux operating system can be installed as either dual OS in your system or you can install through a virtual machine (VM).

Installation of Ubuntu in yourWindows OS through a Virtual machine

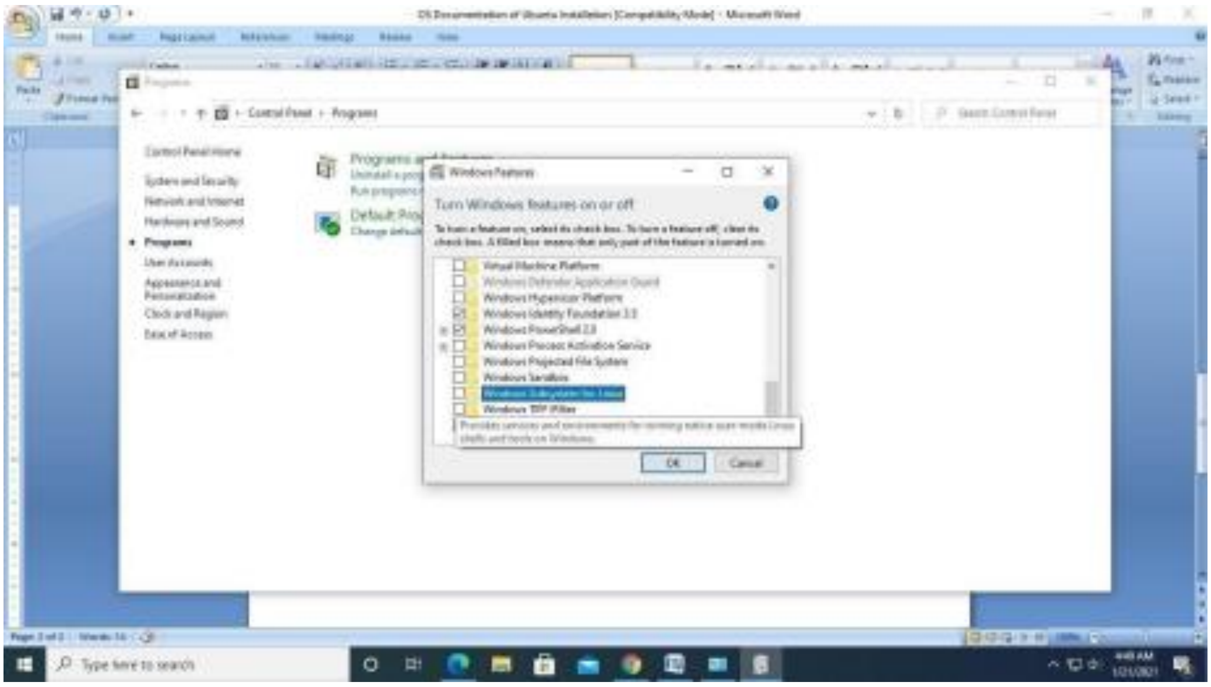
Steps

1. Download VMware Player or Workstation recent version.
2. Download Ubuntu LTS recent version.
3. Install VM ware Player in your host machine.
4. Open VMware Workstation and click on "New Virtual Machine".
5. Select "Typical (recommended)" and click "Next".
6. Select "Installer disc image (ISO)", click "Browse" to select the Ubuntu ISO file, click "Open" then "Next".
7. You have to type in "Full name", "User name" that must only consist of lowercase and numbers then you must enter a password. After you finished, click "Next".
8. You can type in a different name in "Virtual machine name" or leave as is and select an appropriate location to store the virtual machine by clicking on "Browse" that is next to "Location" -- you should place it in a drive/partition that has at least 5GB of free space. After you selected the location click "OK" then "Next".
9. In "Maximum disk size" per Ubuntu recommendations you should allocate at least 5GB -- double is recommended to avoid running out of free space.
10. Select "Store virtual disk as a single file" for optimum performance and click "Next".
11. Click on "Customize" and go to "Memory" to allocate more RAM -- 1GB should suffice, but more is always better if you can spare from the installed RAM.
12. Go to "Processors" and select the "Number of processors" that for a normal computer is 1 and "Number of cores per processor" that is 1 for single core, 2 for dual core, 4 for quad core and so on -- this is to insure optimum performance of the virtual machine.
13. Click "Close" then "Finish" to start the Ubuntu install process.
14. On the completion of installation, login to the system





Turn Windows features On or Off


Windows Features

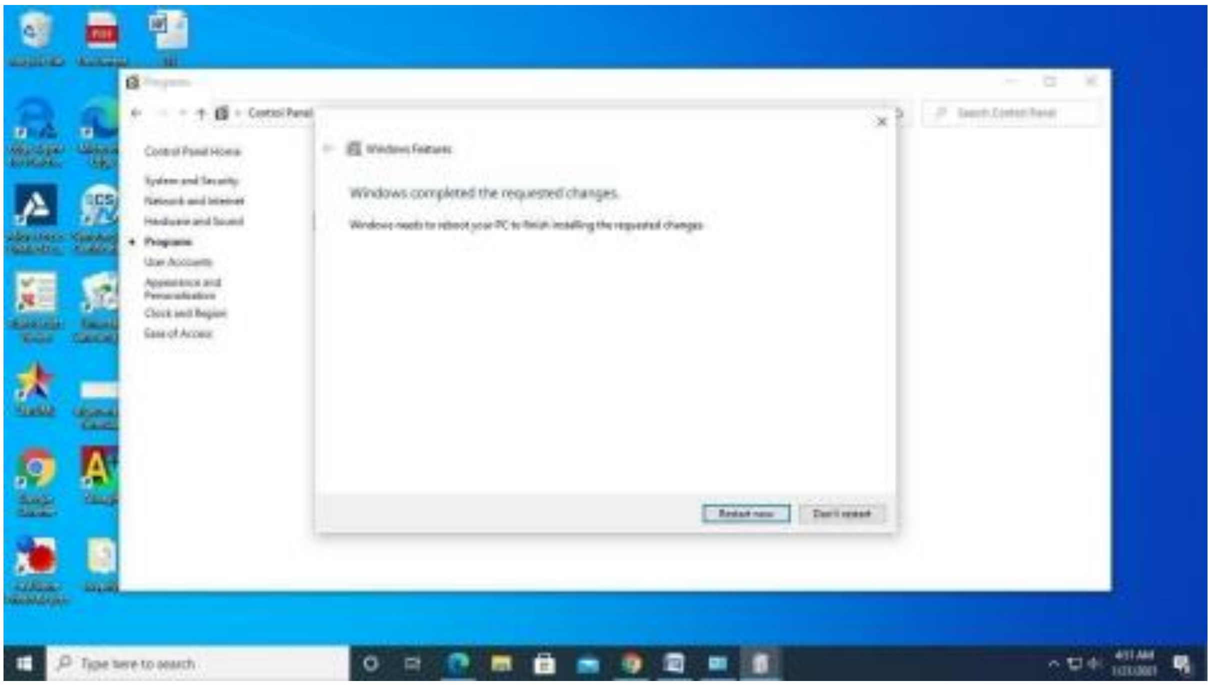
Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] Virtual Machine Platform
- [ ] Windows Defender Application Guard
- [ ] Windows Hypervisor Platform
- [✓] Windows Identity Foundation 3.5
- [✓] Windows PowerShell 2.0
- [ ] Windows Process Activation Service
- [ ] Windows Projected File System
- [ ] Windows Sandbox
- [ ] Windows Subsystem for Linux
- [ ] Windows TIFF IFilter

Provides services and environments for running native user-mode Linux shells and tools on Windows.

Windows subsystem for linux then press Ok

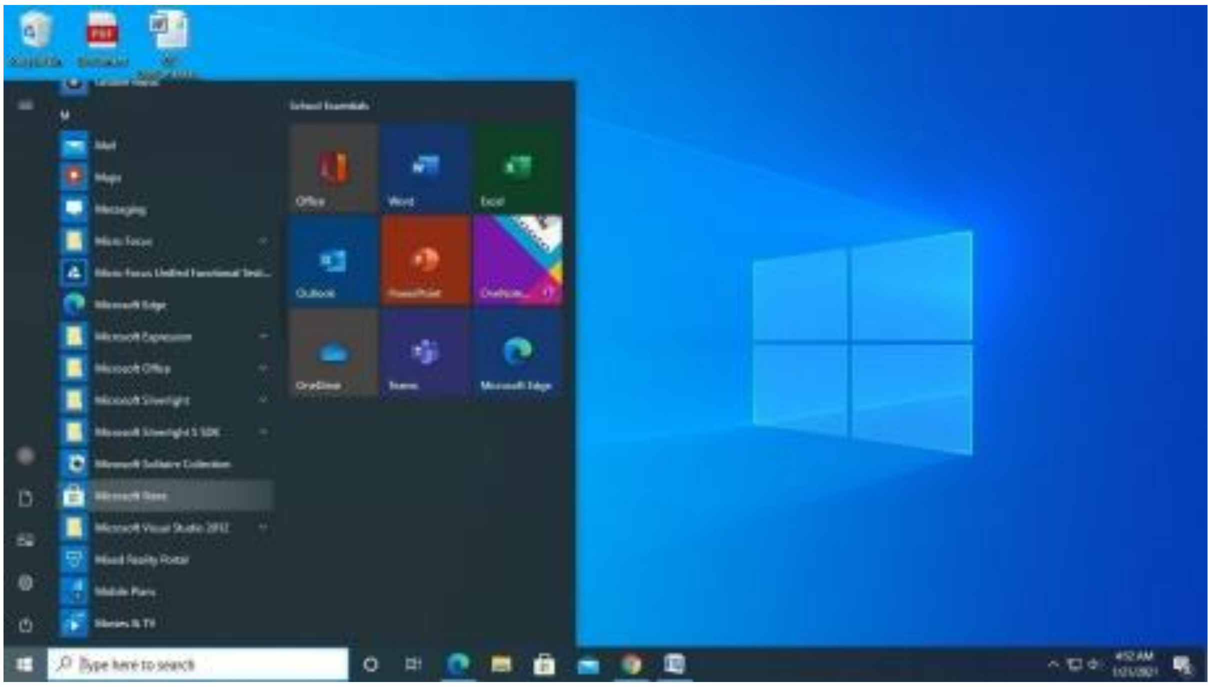

Windows Features

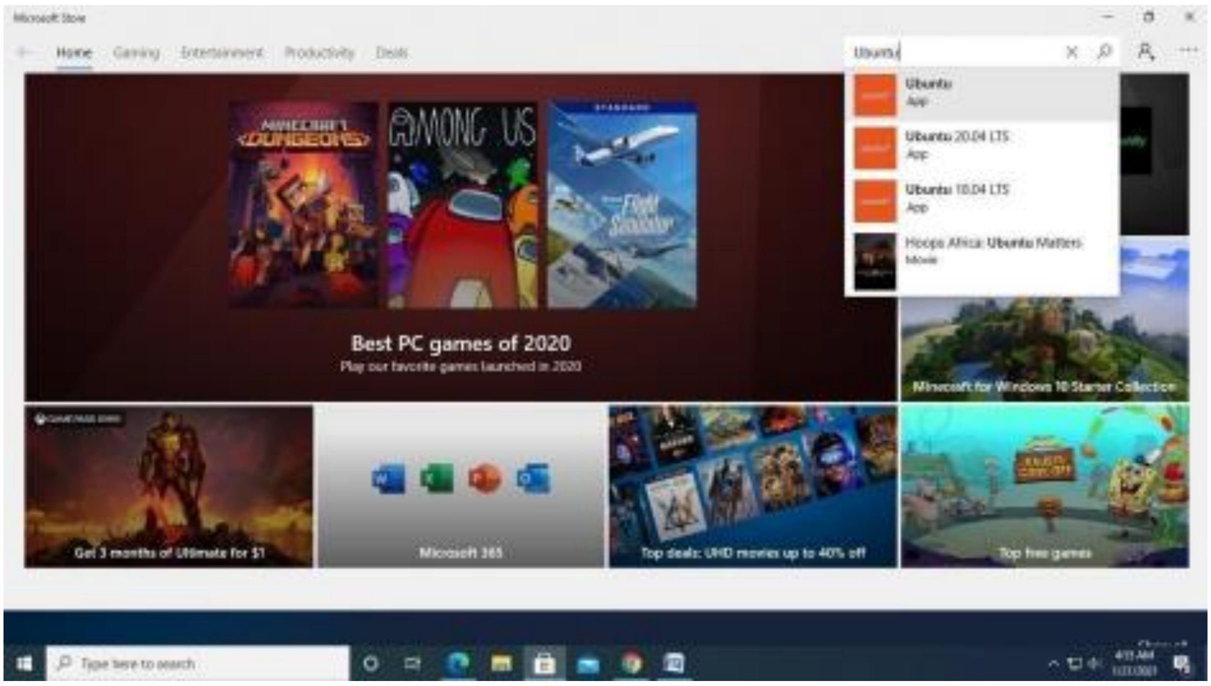
Windows completed the requested changes.

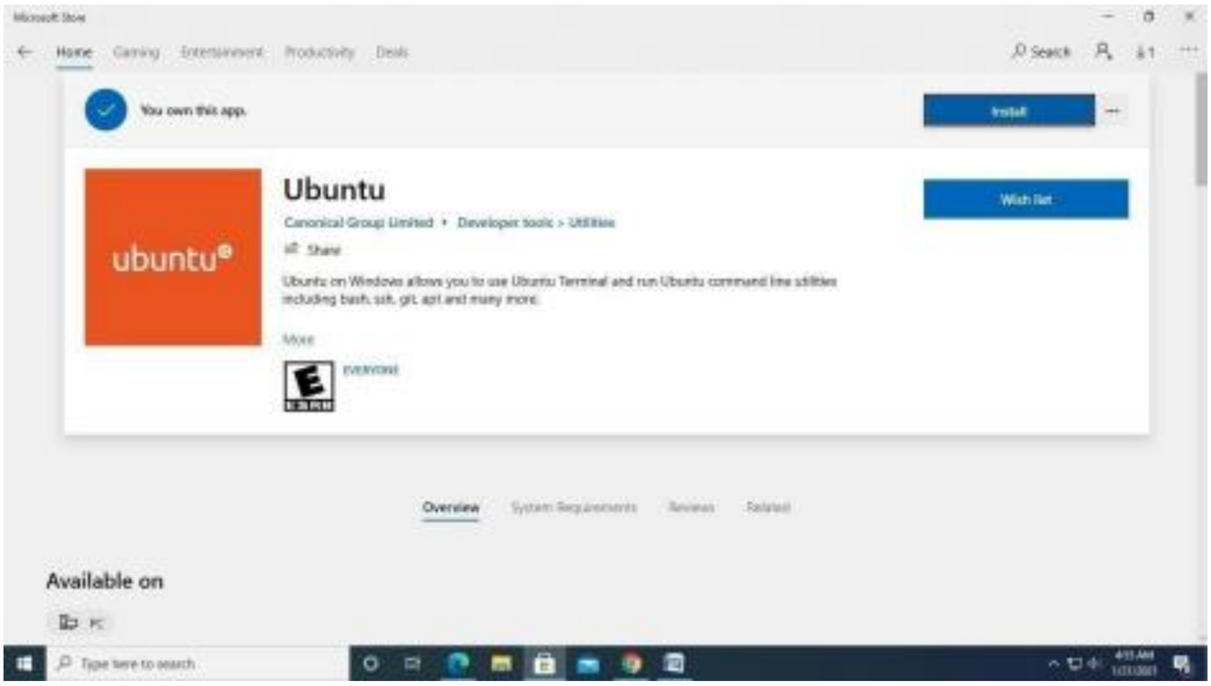
Windows needs to reboot your PC to finish installing the requested changes.

restart the PC to apply the changes

Choose Microsoft Store and search for Ubuntu and Install it

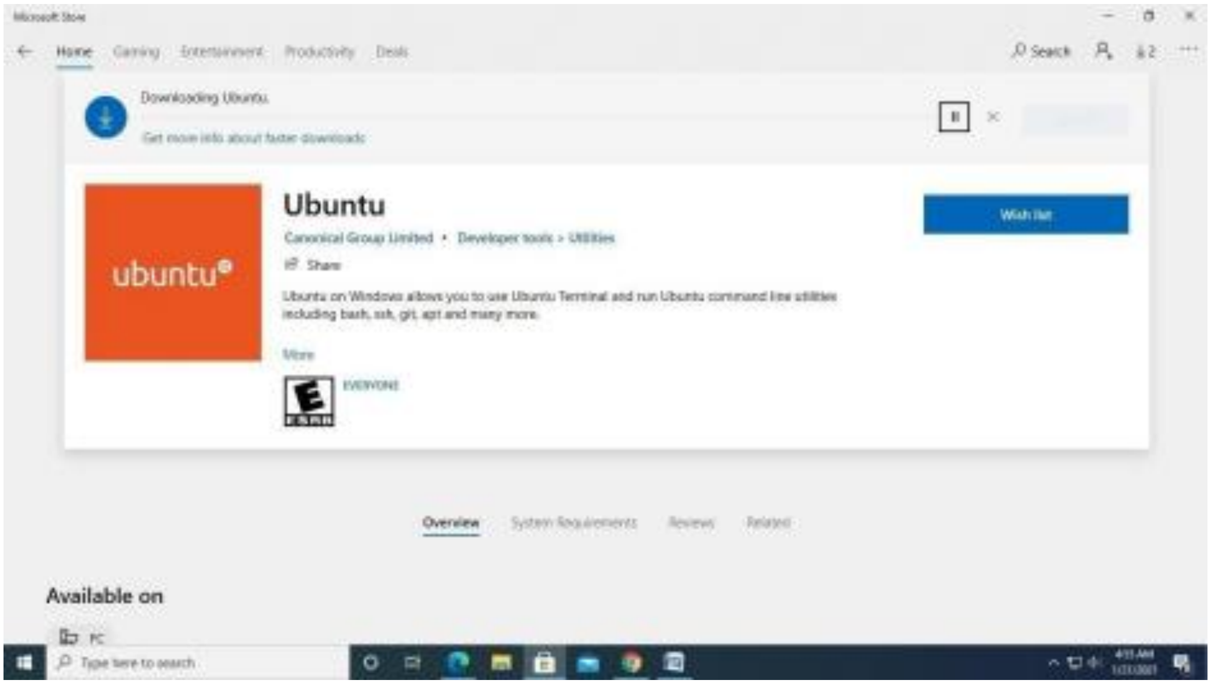

School Essentials
Mail
Maps
Messaging
Microsoft Edge
Microsoft Office
Microsoft Store
Mixed Reality Portal
Movies & TV
Office
Word
Excel
Outlook
PowerPoint
OneNote
OneDrive
Teams
Microsoft Edge
Type here to search


Microsoft Store
Home
Gaming
Entertainment
Productivity
Deals
Ubuntu
Ubuntu
App
Ubuntu 20.04 LTS
App
Ubuntu 18.04 LTS
App
Hoops Africa: Ubuntu Matters
Movie
Best PC games of 2020
Play our favorite games launched in 2020
Minecraft for Windows 10 Starter Collection
Get 3 months of Ultimate for $1
Microsoft 365
Top deals: UHD movies up to 40% off
Top free games
Type here to search

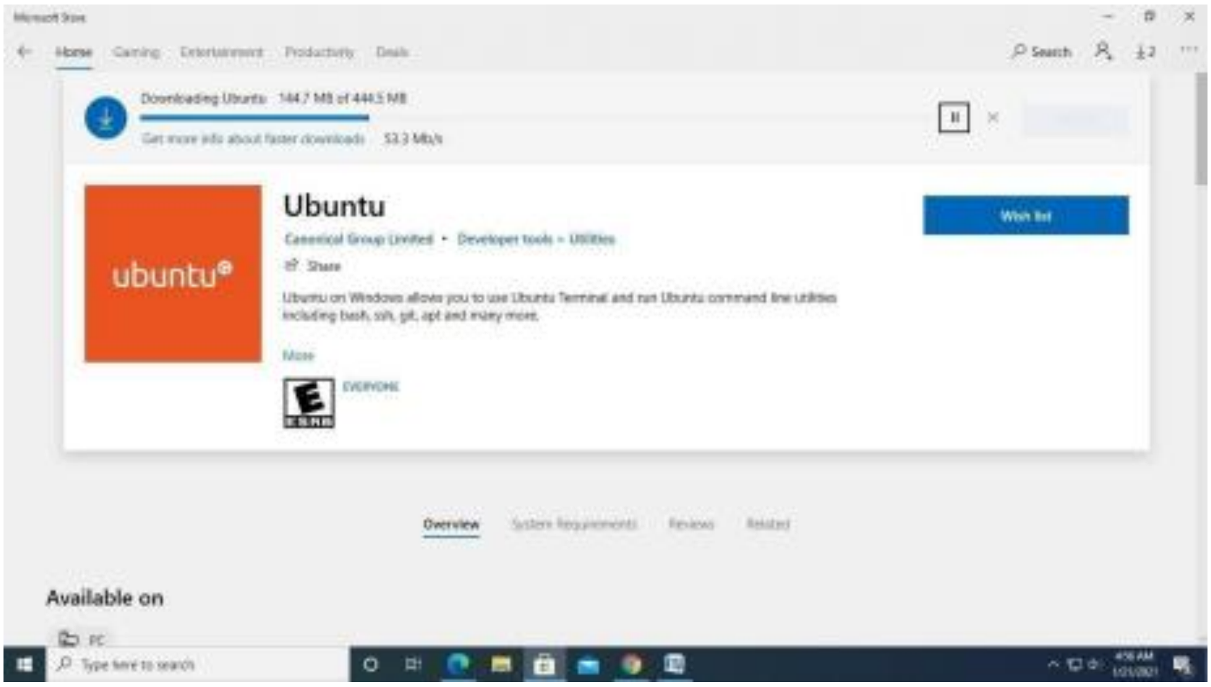


Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Click on Install



Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.



Downloading Ubuntu 144.7 MB of 444.5 MB

Get more info about faster downloads 53.3 Mb/s

Ubuntu

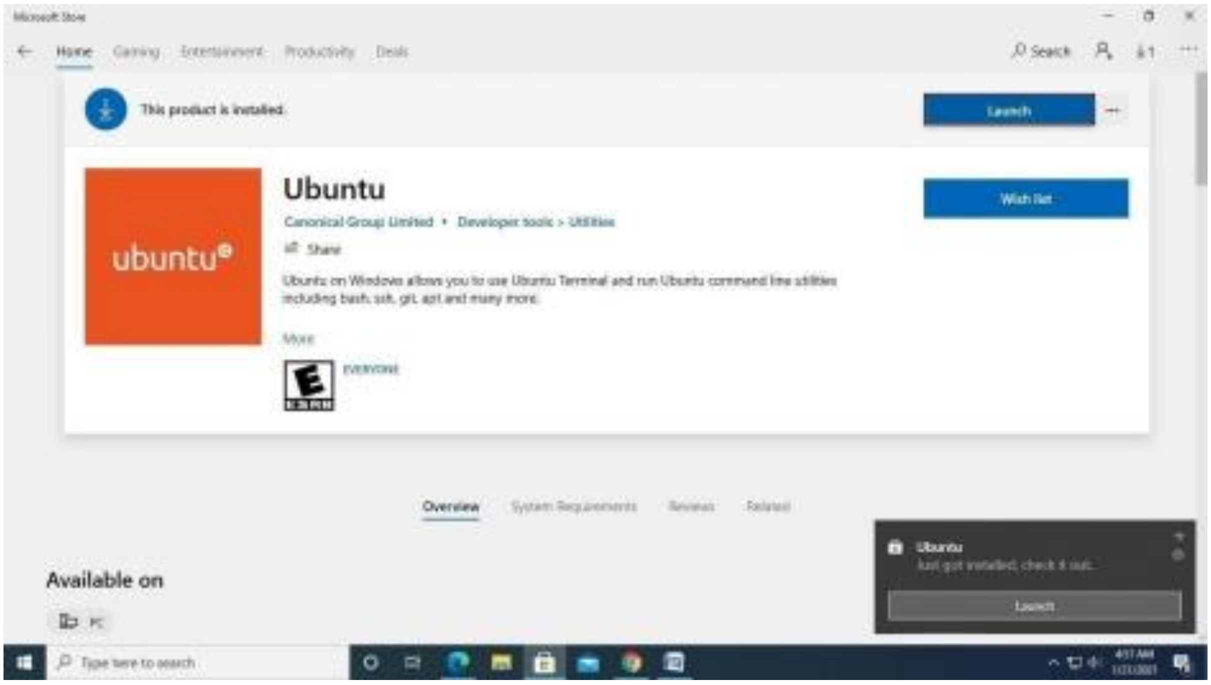
Canonical Group Limited • Developer tools > Utilities

Share

Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

More

EVERYONE



This product is installed.

Ubuntu

Canonical Group Limited • Developer tools > Utilities

Share

Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

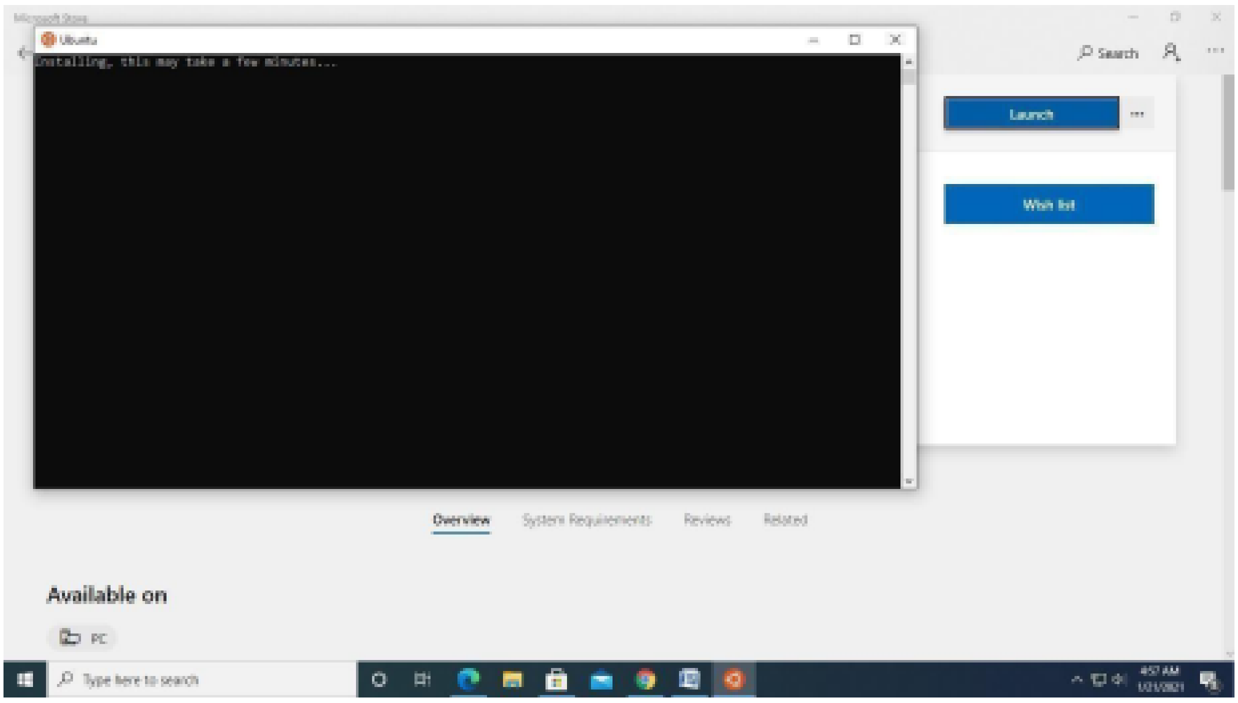
More

EVERYONE

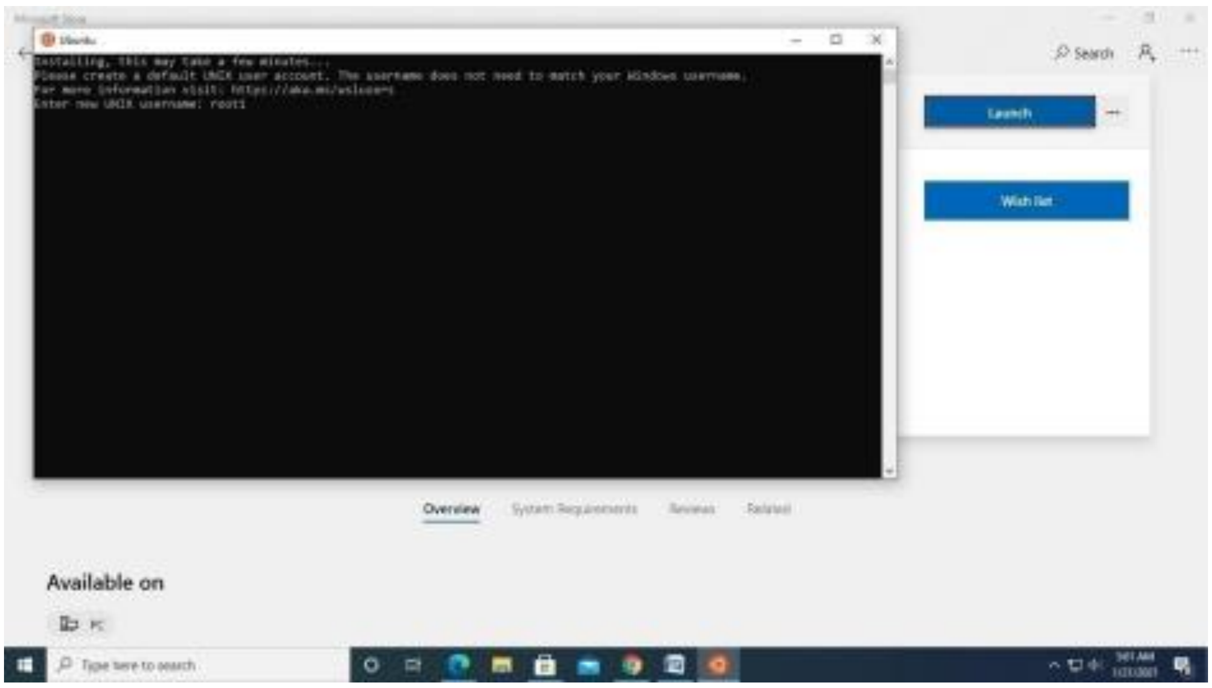
Ubuntu

Just got installed, check it out.

Now Launch the Ubuntu

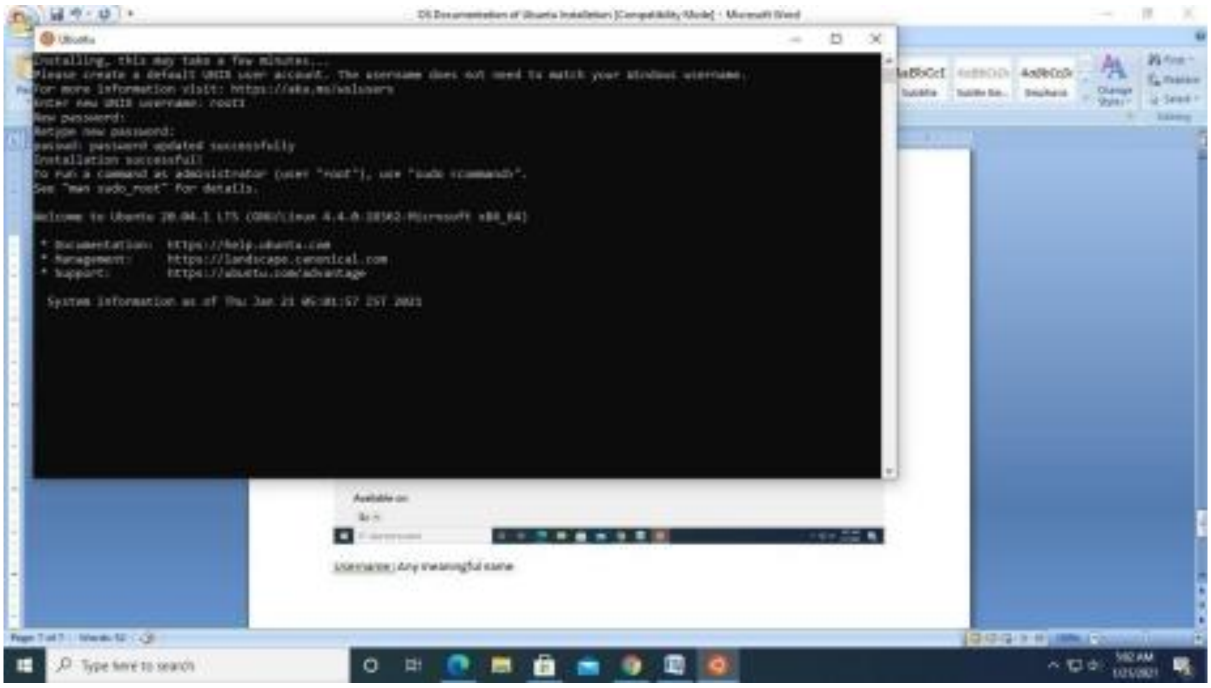


```
Installing, this may take a few minutes...
```



```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: root1
```

name : Any meaningful
name Password: any name



```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: root1
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.4.0-18362-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 21 05:01:57 IST 2021
```

Username : Any meaningful name

Result:

Finally completed installation of Ubuntu in windows 10- 64 bits successfully.

Ex. No. 1B	**BASIC LINUX COMMANDS**	**Date :**

a) Basics
1. echo SRM to display the string SRM

2. clear to clear the screen

3. date to display the current date and time

4. cal 2003 to display the calendar for the year 2003
cal 6 2003 to display the calendar for the June-2003

5. passwd to change password

b) Working with Files
1. ls list files in the present working directory
ls –l list files with detailed information (long list)
ls –a list all files including the hidden files

2. cat > f1 to create a file (Press ^d to finish typing)

3. cat f1 display the content of the file f1

4. wc f1 list no. of characters, words & lines of a file f1
wc –c f1 list only no. of characters of file f1
wc –w f1 list only no. of words of file f1
wc –l f1 list only no. of lines of file f1

5. cp f1 f2 copy file f1 into f2

6. mv f1 f2 rename file f1 as f2

7. rm f1 remove the file f1

8. head –5 f1 list first 5 lines of the file f1
tail –5 f1 list last 5 lines of the file f1

c) Working with Directories
1. mkdir elias to create the directory elias
2. cd elias to change the directory as elias
3. rmdir elias to remove the directory elias
4. pwd to display the path of the present working directory
5. cd to go to the home directory
cd .. to go to the parent directory
cd - to go to the previous working directory
cd / to go to the root directory

d) File name substitution
1. ls f? list files start with 'f' and followed by any one character *2. ls* *.c list files with extension 'c'

3. ls [gpy]et list files whose first letter is any one of the character g, p or y and followed by the word et

4. ls [a-d,l-m]ring list files whose first letter is any one of the character from a to d and l to m and followed by the word ring.

e) I/O Redirection
1. Input redirection
wc –l < ex1 □ To find the number of lines of the file ‘ex1’

2. Output redirection
who > f2 □ the output of ‘who’ will be redirected to file f2 *3. cat* >> f1 □ to append more into the file f1

f) Piping
Syntax : Command1 | command2

Output of the command1 is transferred to the command2 as input. Finally output of the command2 will be displayed on the monitor.

ex. *cat* f1 | *more* □ list the contents of file f1 screen by screen

head –6 f1 |*tail* –2 □ prints the 5th & 6th lines of the file f1.

g) Environment variables
1. echo $HOME □ display the path of the home directory

2. echo $PS1 □ display the prompt string $
3. echo $PS2 □ display the second prompt string (> symbol by default) *4. echo $LOGNAME* □ login name
5. echo $PATH □ list of pathname where the OS searches for an executable file

h) File Permission
-- chmod command is used to change the access permission of a file.

Method-1
Syntax : *chmod* [ugo] [+/-] [rwxa] filename

u : user, g : group, o : others
+ : Add permission - : Remove the permission
r : read, w : write, x : execute, a : all permissions
ex. *chmod* ug+rw f1
adding ‘read & write’ permissions of file f1 to both user and group members.

Method-2
Syntax : *chmod* octnum file1

The 3 digit octal number represents as follows
· first digit -- file permissions for the user
· second digit -- file permissions for the group
· third digit -- file permissions for others
Each digit is specified as the sum of following
4 – read permission, 2 – write permission, 1 – execute permission
ex. *chmod* 754 f1
it change the file permission for the file as follows
· read, write & execute permissions for the user ie; 4+2+1 = 7
· read, & execute permissions for the group members ie; 4+0+1 = 5 · only read permission for others ie; 4+0+0 = 4

Result:

The above commands are successfully executed and verified.

Ex. No. 1C	**FILTERS and ADMIN COMMANDS**	**Date :**

FILTERS

1. cut
▪ Used to cut characters or fileds from a file/input

Syntax : **cut -c**chars filename
-ffieldnos filename

▪ By default, tab is the filed separator(delimiter). If the fileds of the files are separated by any other character, we need to specify explicitly by **–d** option

cut -ddelimitchar **-f**fileds filname

2. grep
▪ Used to search one or more files for a particular pattern.

Syntax : **grep** pattern filename(s)

Lines that contain the *pattern* in the file(s) get displayed
pattern can be any regular expressions
More than one files can be searched for a pattern

-v option displays the lines that do not contain the *pattern*
-l list only name of the files that contain the *pattern*
-n displays also the line number along with the lines that matches the *pattern*

3. sort
▪ Used to sort the file in order

Syntax : **sort** filename

Sorts the data as text by default
Sorts by the first filed by default

-r option sorts the file in descending order
-u eliminates duplicate lines
-o filename writes sorted data into the file *fname*
-tdchar sorts the file in which fileds are separated by *dchar*
-n sorts the data as number
+1n skip first filed and sort the file by second filed numerically

4. Uniq
▪ Displays unique lines of a sorted file
Syntax : **uniq** filename
-d option displays only the duplicate lines
-c displays unique lines with no. of occurrences.
5. diff
▪ Used to differentiate two files

Syntax : **diff** f1 f2
compare two files f1 & f2 and prints all the lines that are differed between f1 & f2.

Result:

The above commands are successfully executed and verified.

EX.NO: 2	Process Creation using fork() and Usage of getpid(), getppid(), wait() functions

Aim :

To write a program for process Creation using fork() and usage of getpid(), getppid(), wait() function.

Description:

Algorithm

Step 1 : Open the terminal and edit your program and save with extension ".c"

Ex. nano test.c

Step 2 : Compile your program using gcc compiler

Ex. gcc test.c ➔ Output file will be "a.out"

(or)

gcc -o test text.c ➔ Output file will be "test"

Step 3 : Correct the errors if any and run the program

Ex. ./a.out (or) ./test

Syntax for process creation

int fork();

Returns 0 in child process and child process ID in parent process.

Other Related Functions

int getpid() ➔ returns the current process ID

int getppid() ➔ returns the parent process ID

wait() ➔ makes a process wait for other process to complete

Virtual fork

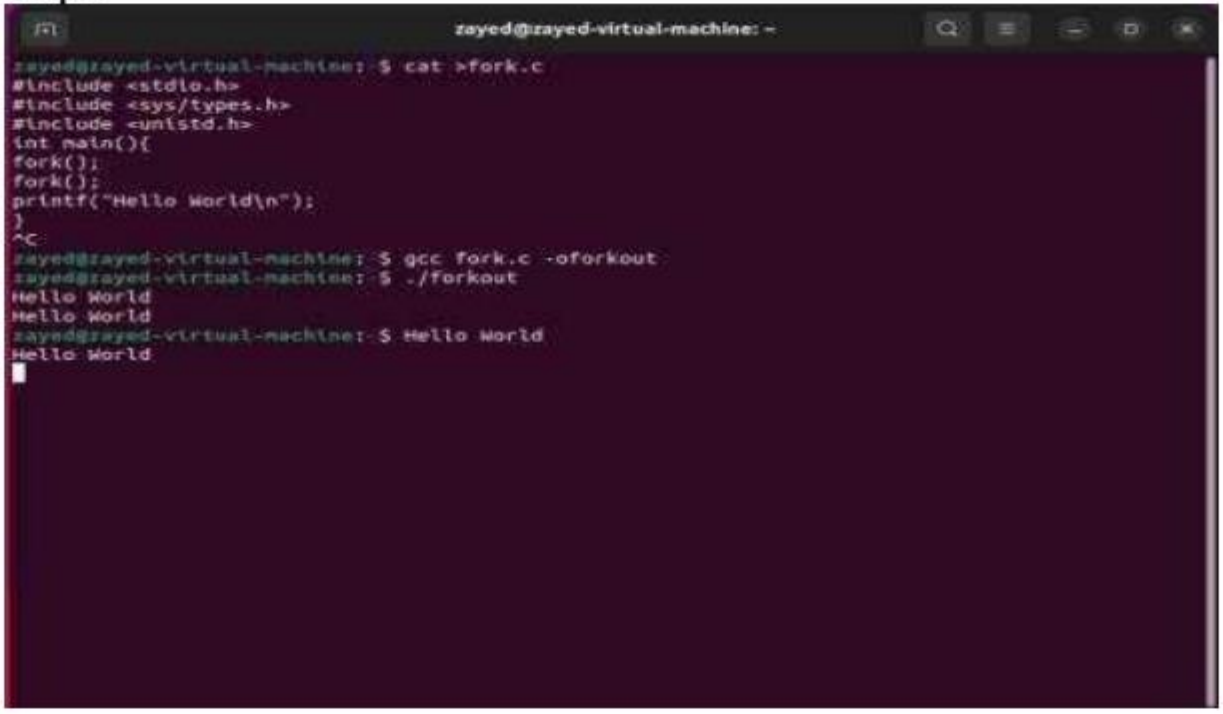
vfork() is similar to fork but both processes shares the same address space.

Program :

- **Process creating using fork()**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int
main(){
fork();
fork();
printf("Hello World\n"); }
```

Output

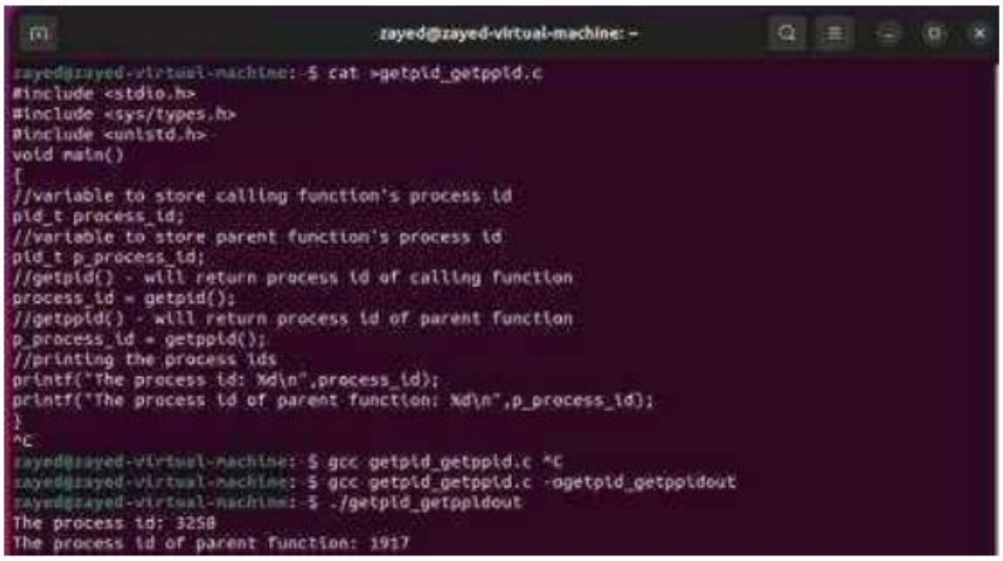


```
zayed@zayed-virtual-machine:~$ cat >fork.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
fork();
fork();
printf("Hello World\n");
}
^C
zayed@zayed-virtual-machine:~$ gcc fork.c -oforkout
zayed@zayed-virtual-machine:~$ ./forkout
Hello World
Hello World
zayed@zayed-virtual-machine:~$ Hello World
Hello World
```

- **Usage of getpid() and getppid()**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
/variable to store calling function's
int process_id, p_ process_id;
//getpid() - will return process id of calling function
process_id = getpid();
//getppid() - will return process id of parent function
p_process_id = getppid();
//printing the process ids
printf("The process id: %d\n",process_id);
printf("The process id of parent function: %d\n",p_process_id);
}
```

Output :

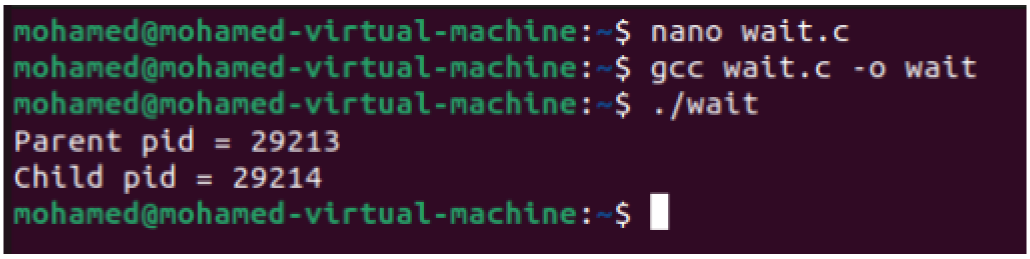


```
zayed@zayed-virtual-machine:~$ cat >getpid_getppid.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
//variable to store calling function's process id
pid_t process_id;
//variable to store parent function's process id
pid_t p_process_id;
//getpid() - will return process id of calling function
process_id = getpid();
//getppid() - will return process id of parent function
p_process_id = getppid();
//printing the process ids
printf("The process id: %d\n",process_id);
printf("The process id of parent function: %d\n",p_process_id);
}
^C
zayed@zayed-virtual-machine:~$ gcc getpid_getppid.c ^C
zayed@zayed-virtual-machine:~$ gcc getpid_getppid.c -ogetpid_getppidout
zayed@zayed-virtual-machine:~$ ./getpid_getppidout
The process id: 3258
The process id of parent function: 1917
```

• Usage of wait()

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
pid_t cpid;
if (fork()== 0)
exit(0); /* terminate child */
else
cpid = wait(NULL); /* reaping parent */
printf("Parent pid = %d\n", getpid());
printf("Child pid = %d\n", cpid);
return 0;
}
```

Output:



```
mohamed@mohamed-virtual-machine:~$ nano wait.c
mohamed@mohamed-virtual-machine:~$ gcc wait.c -o wait
mohamed@mohamed-virtual-machine:~$ ./wait
Parent pid = 29213
Child pid = 29214
mohamed@mohamed-virtual-machine:~$
```

Result :

Thus Successfully completed Process Creation using fork() and Usage of getpid(), getppid(), wait() functions.

cP SDJ 2 81 1 40 20

EX.NO. 3 Multithreading and pthread in C

Aim :

To implement and study Multithreading and pthread in C

Description:

Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.
To implement and study Multithreading and pthread in C

Multithreading models are three types

· Many to many relationship.
· Many to one relationship.
· One to one relationship.

We must include the pthread.h header file at the beginning of the script to use all the functions of the pthreads library.

The **functions** defined in the **pthreads library** include:

a. ***pthread_create:*** used to create a new thread

Syntax:

int pthread_create(pthread_t * thread,const pthread_attr_t * attr, void * (*start_routine)(void *), void *arg);

Parameters:

· **thread:** pointer to an unsigned integer value that returns the thread id of the thread created.
· **attr:** pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes.
· **start_routine:** pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used.
· **arg:** pointer to void that contains the arguments to the function defined in the earlier argument

b. ***pthread_exit:*** used to terminate a thread

Syntax:

void pthread_exit(void *retval);

Parameters: This method accepts a mandatory parameter **retval** which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

c. ***pthread_join:*** used to wait for the termination of a thread.

Syntax:

int pthread_join(pthread_t th,void **thread_return);

Parameter: This method accepts following parameters:

- **th:** thread id of the thread for which the current thread waits.
- **thread_return:** pointer to the location where the exit status of the thread mentioned in th is stored.

d. ***pthread_self:*** used to get the thread id of the current thread.

Syntax:

pthread_t pthread_self(void);

e. ***pthread_equal:*** compares whether two threads are the same or not. If the two threads are equal, the function returns a non-zero value otherwise zero.

Syntax:

int pthread_equal(pthread_t t1,

pthread_t t2);

Parameters: This method accepts following parameters:

- t1: the thread id of the first thread
- t2: the thread id of the second thread

Algorithm:

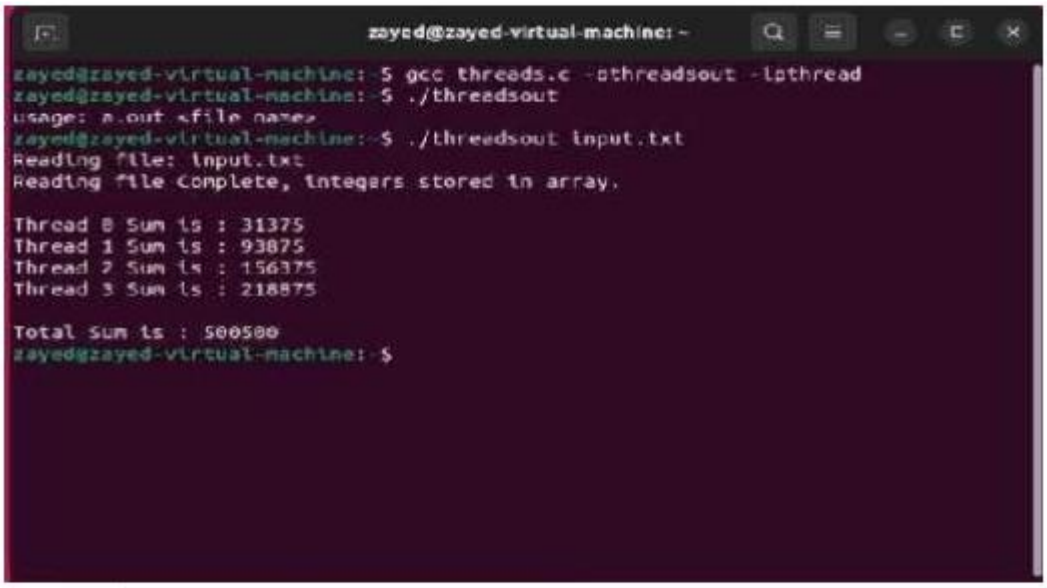
Step: 1 Start
Step: 2 Read Input File
Step: 3 Initialize Threads. Max of 4 threads
Step: 4 Execute each thread.
Step: 5 Combine results from each thread.
Step: 6 Display the total sum.
Step: 7 Stop

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAX 1000
#define MAX_THREAD 4

int array[1000];
int sum[4] = { 0 };
int arraypart = 0;

void* sum_array(void* arg)
{
    int thread_part = arraypart++;
    for (int i = thread_part * (MAX / 4); i < (thread_part + 1) * (MAX / 4); i++)
    {
        sum[thread_part] += array[i];
    }
}

void testSum()
{
  pthread_t threads[MAX_THREAD];

    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_create(&threads[i], NULL, sum_array,
    (void*)NULL); }

    // joining threads
    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_join(threads[i], NULL);
    }

      // print each thread
    for (int i = 0; i < MAX_THREAD; i++)
    {
        printf("Thread %d Sum is : %d \n",i, sum[i]);
    }
    // adding the 4 parts
     int total_sum = 0;
    for (int i = 0; i < MAX_THREAD; i++)
    {
        total_sum += sum[i];
    }
       printf("\nTotal Sum is : %d \n",total_sum);

}

void readfile(char* file_name)
{
   char ch;
   FILE *fp;
  fp = fopen(file_name,"r"); // read mode

   if( fp == NULL )
   {
     perror("Error while opening the file.\n");
      exit(EXIT_FAILURE);
   }
```

```
    char line [5]; /* line size */

    int i=0;

   printf("Reading file: ");
   fputs(file_name,stdout);
   printf("\n");
    while ( fgets ( line, sizeof line, fp) != NULL ) /* read a line
    */ {
       if (i < 1000)
       {
         array[i]=atoi(line);
       }

      i++;
    }
  fclose(fp);
   printf("Reading file Complete, integers stored in

array.\n\n"); }

int main(int argc, char* argv[])
{
   if (argc != 2) {
          fprintf(stderr,"usage: a.out <file name>\n");
          /*exit(1);*/
           return -1;
    }
   readfile(argv[1]);
  //Debug code for testing only
   testSum();
   return 0;
}
```

Output :



```
zayed@zayed-virtual-machine:~$ gcc threads.c -othreadsout -lpthread
zayed@zayed-virtual-machine:~$ ./threadsout
usage: a.out <file name>
zayed@zayed-virtual-machine:~$ ./threadsout input.txt
Reading file: input.txt
Reading file Complete, integers stored in array.

Thread 0 Sum is : 31375
Thread 1 Sum is : 93875
Thread 2 Sum is : 156375
Thread 3 Sum is : 218875

Total Sum is : 500500
zayed@zayed-virtual-machine:~$
```

Result :

Thus the Program Successfully implemented and studied Multithreading and pthread in C

EX.NO. 4 Mutual Exclusion using semaphore and monitor

Aim :

To implement Mutual Exclusion using semaphore and monitor

Description:

Semaphore

Semaphore is used to implement process synchronization. This is to protect critical region shared among multiples processes.

SYSTEM V SEMAPHORE SYSTEM CALLS

Include the following header files for System V semaphore

<sys/ipc.h>, <sys/sem.h>, <sys/types.h>

To create a semaphore array,

```
int semget(key_t key, int nsems, int semflg)
```

key ➔ semaphore id

nsems ➔ no. of semaphores in the semaphore array

semflg ➔ IPC_CREATE|0664 : to create a new semaphore

IPC_EXCL|IPC_CREAT|0664 : to create new semaphore and the

call fails if the semaphore already exists To perform operations on the semaphore sets viz., allocating resources, waiting for the resources or freeing the resources,

```
int semop(int semid, struct sembuf *semops, size_t nsemops)
```

semid ➔ semaphore id returned by semget()

nsemops ➔ the number of operations in that array

semops ➔ The pointer to an array of operations to be performed on the semaphore set. The structure is as follows

```
struct sembuf {
        unsigned short sem_num; /* Semaphore set num */ short
        sem_op;  /* Semaphore operation */ short sem_flg; /*Operation
        flags, IPC_NOWAIT, SEM_UNDO */
    };
```

Algorithm:

Step1: start the program

Step2: initialize the semaphore structures and write a *Function to initialise the semaphore variables, condition and mutual exclusion*

Step3:write an program coding for initialize the fork and   locks the mutex using pthread mutex loc() for access critical section.

Step4: write the statement for *wakes up one waiting thread and finding the value for debugging then print it.*

*Step5:*write the statement for *philospher thread creation and display the result.*

Step6: stop the program..

Program :

USING SEMAPHORE

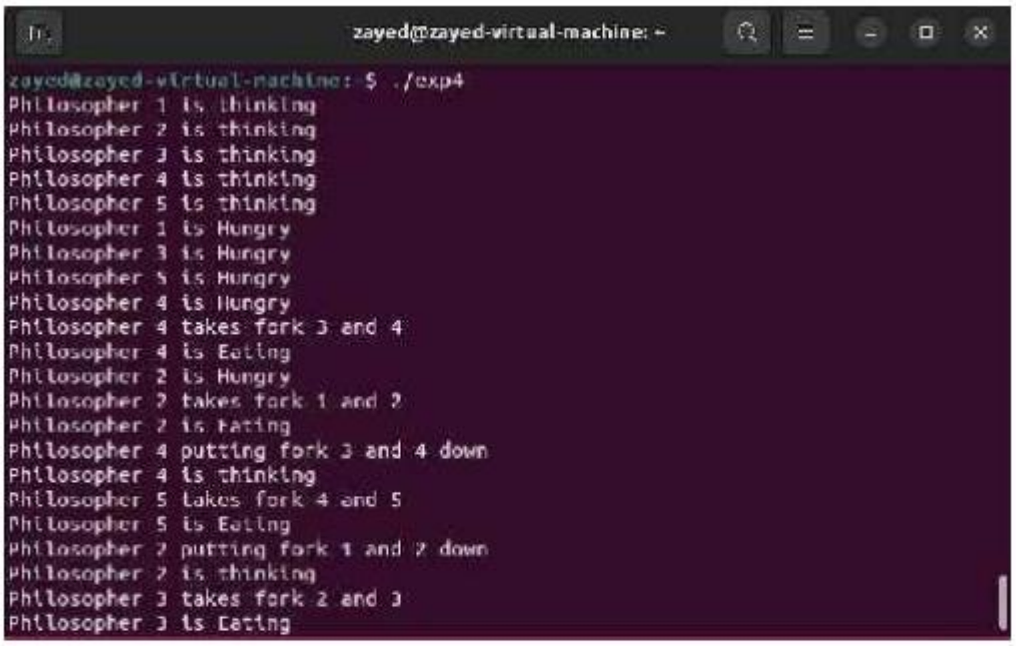
```
#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

#include<errno.h>

#include <stdlib.h>

#include<sched.h>

int philNo[5] = { 0, 1, 2, 3, 4 };

// my_semaphore structure

typedef struct {

  // Semaphore mutual exclusion variable

  pthread_mutex_t mutex;

  // Semaphore count variable

   int cnt;

  // Semaphore conditonal variable

  pthread_cond_t conditional_variable;

}
 my_semaphore;

// Function to initialise the semaphore variables
```

```
int init(my_semaphore *sema, int pshared, int val) {
  // The case when pshared == 1 is not implemeted as it was not required because the
philosphers are implemented using threads and not processes.
```

```
  if(pshared == 1){
          printf("Cannot handle semaphores shared between processes!!!
           Exiting\n"); return -1;
  }

  // Initialisng the semaphore conditonal variable
  pthread_cond_init(&sema->conditional_variable, NULL);

  // Initialisng the semaphore count variable
  sema->cnt = val;

  // Initialisng the semaphore mutual exclusion variable
  pthread_mutex_init(&sema->mutex, NULL);

  return 0;
}

int signal(my_semaphore *sema) {

  //This locks the mutex so that only thread can access the critical section at a time
  pthread_mutex_lock(&sema->mutex);
  sema->cnt = sema->cnt + 1;

  // This wakes up one waiting thread
  if (sema->cnt)
          pthread_cond_signal(&sema->conditional_variable);

  // A woken thread must acquire the lock, so it will also have to wait until we call unlock

  // This releases the mutex
  pthread_mutex_unlock(&sema->mutex);

  return 0;
}
int wait(my_semaphore *sema) {
  //This locks the mutex so that only thread can access the critical section at a time
  pthread_mutex_lock(&sema->mutex);
  // While the semaphore count variable value is 0 the mutex is blocked on the conditon
variable
  while (!(sema->cnt))
    pthread_cond_wait(&sema->conditional_variable, &sema-
    >mutex); // unlock mutex, wait, relock mutex

  sema->cnt = sema->cnt - 1;

  // This releases the mutex and threads can access mutex
  pthread_mutex_unlock(&sema->mutex);
  return 0;
}
// Print semaphore value for debugging
void signal1(my_semaphore *sema) {
  printf("Semaphore variable value = %d\n", sema->cnt);
}
// Declaring the semaphore variables which are the shared resources by the threads
my_semaphore forks[5], bowls;

//Function for the philospher threads to eat
void *eat_food(void *arg) {
    while(1) {
        int* i = arg;
        // This puts a wait condition on the bowls to be used by the current philospher so
that the philospher can access these forks whenever they are free
        wait(&bowls);

        // This puts a wait condition on the forks to be used by the current philospher so
```

```
          that the philospher can access these forks whenever they are free

          wait(&forks[*i]);

          wait(&forks[(*i+4)%5]);

          sleep(1);

     //Print the philospher number, its thread ID and the number of the forks it uses for

          eating printf("Philosopher %d with ID %ld eats using forks %d and %d\n", *i+1,
  pthread_self(), *i+1, (*i+4)%5+1);

        // This signals the other philospher threads that the bowls are available for

 eating signal(&bowls);

          // This signals the other philospher threads that these forks are available for
  eating and thus other threads are woken up
          signal(&forks[*i]);

          signal(&forks[(*i+4)%5]);

          sched_yield();

    }

  }

  void main() {
     int i = 0;
     // Initialising the forks (shared variable) semaphores

     while(i < 5){

        init(&forks[i], 0, 1);

        i++;

     }

     // Initialising the bowl (shared variable) semaphore

     init(&bowls, 0, 1);

     // Declaring the philospher threads

     pthread_t phil[5];

     i = 0;

     // Creating the philospher threads

     while(i < 5) {

        pthread_create(&phil[i], NULL, eat_food, &philNo[i]);



        i++;
    }
    i = 0;
   // Waits for all the threads to end their execution before
    ending while(i < 5) {
        pthread_join(phil[i], NULL);
        i++;
    }}
```

Output :

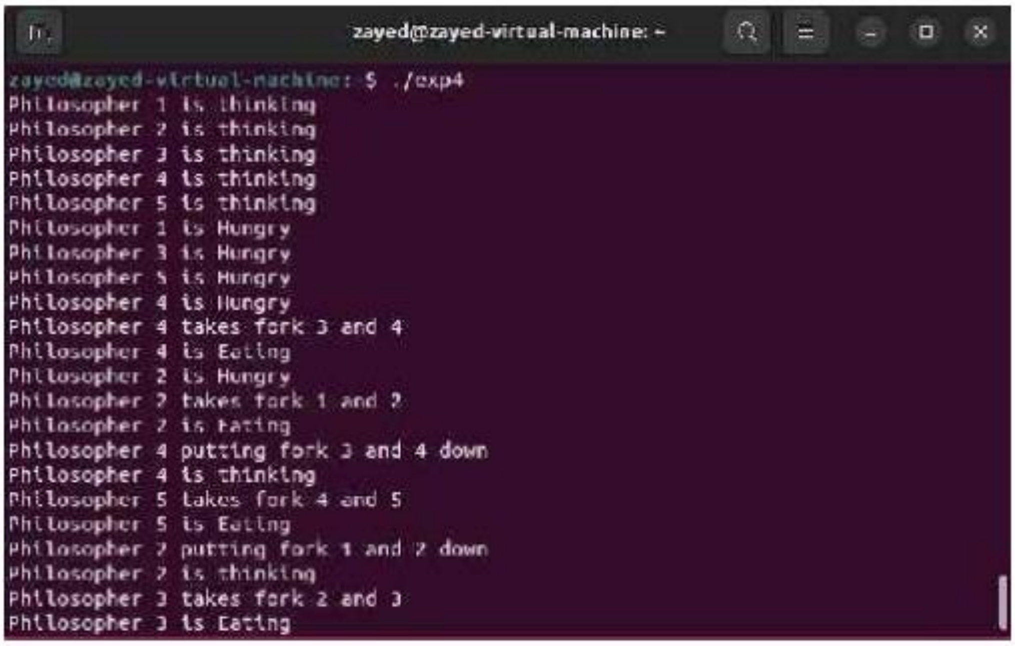


```
zayed@zayed-virtual-machine: $ ./exp4
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
```

USING MONITOR

```
monitor DP
{
    status state[5];
condition self[5];
// Pickup chopsticks
Pickup(int i)
{
    // indicate that I'm hungry
    state[i] = hungry;
    // set state to eating in test()
    // only if my left and right neighbors
    // are not eating
    test(i);


    // if unable to eat, wait to be signaled if
    (state[i] != eating)
        self[i].wait;
}
// Put down chopsticks
Putdown(int i)
{
    // indicate that I'm thinking
    state[i] = thinking;
    // if right neighbor R=(i+1)%5 is hungry and
    // both of R's neighbors are not eating, //
    set R's state to eating and wake it up by
    // signaling R's CV
        test((i + 1) % 5);
        test((i + 4) % 5);
    }
    test(int i)
    {
        if (state[(i + 1) % 5] != eating
            && state[(i + 4) % 5] != eating
            && state[i] == hungry) {
            // indicate that I'm eating
            state[i] = eating;

            // signal() has no effect during Pickup(),
            // but is important to wake up waiting
            // hungry philosophers during Putdown()
            self[i].signal();
        }
    }
    init()
    {
        // Execution of Pickup(), Putdown() and test()
        // are all mutually exclusive,
        // i.e. only one at a time can be executing
for
    i = 0 to 4
        // Verify that this monitor-based solution is
        // deadlock free and mutually exclusive in that
        // no 2 neighbors can eat simultaneously
        state[i] = thinking;
    }
}
```

Output :



```
zayed@zayed-virtual-machine: $ ./exp4
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
```

Result :

Successfully executed Mutual Exclusion using semaphore and Monitor.

EX.NO. 5 Reader-Writer problem

Aim :

To study the Reader – Writer problem

Description:

Consider a situation where we have a file shared between many people.

- ➢ If one of the person tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- ➢ However, if some person is reading the file, then others may read it at the same time.

Precisely in OS, we call this situation the readers-writer, problem.

- ➢ One set of data is shared among a number of processes
- ➢ Once a writer is ready, it performs its write. Only one writer may write at a time
- ➢ If a process is writing, no other process can read it
- ➢ If at least one reader is reading, no other process can write
- ➢ Readers may not write and only read

Algorithm

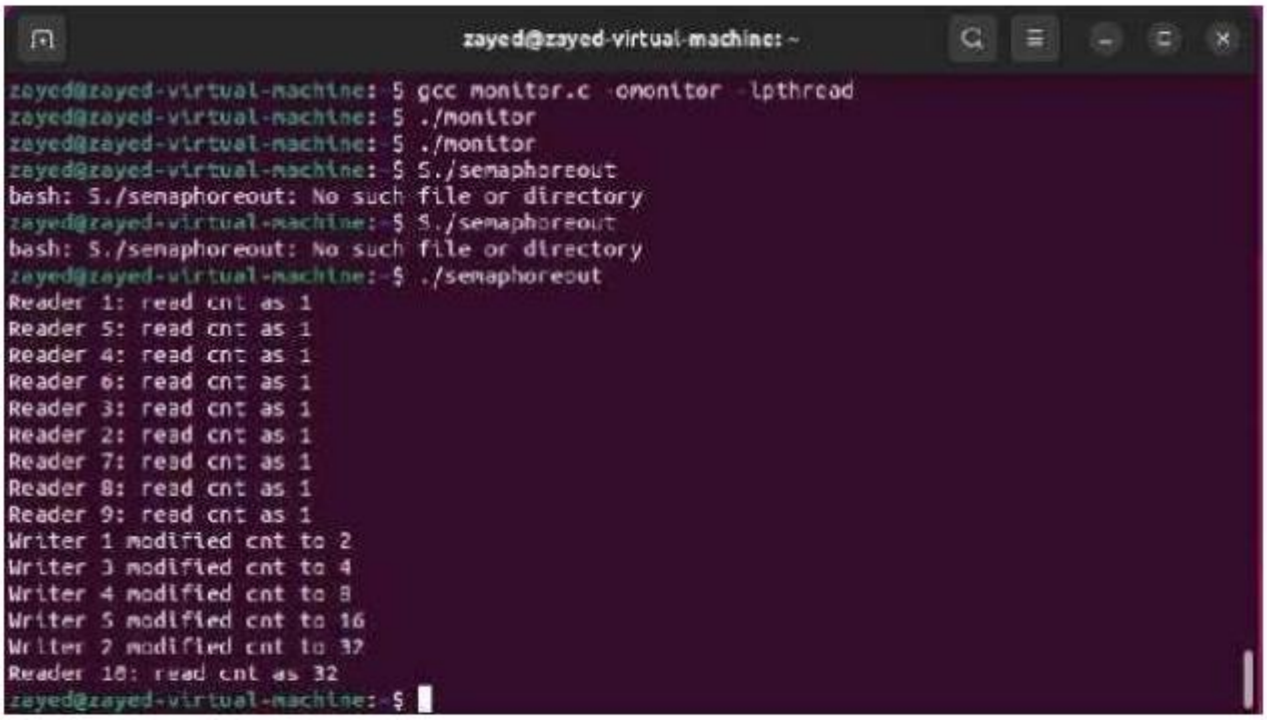
```
Step 1: Start
Step 2: Initialize the mutex, semaphore with value 1
Step 3: Create reader thread, writer thread.
Step 4: Reader Thread:
        Lock the mutex
        Update numreader;
        Check for first reader;
                Unlock the mutex;
                Perform Read operation; lock the mutex, update numreader.
        Check if last reader
                Unlock mutex.
Step 5: Writer Thread
        Wait on semaphore.
        Modify cnt;
        Signal Semaphore;
Step 6:combine the results;
Step 7:Destroy semaphore
Step 8 : Stop
```

Program :

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>
sem_t wrt;

pthread_mutex_t
```

```
mutex; int cnt = 1;

int numreader = 0;

void *writer(void *wno)

{
    sem_wait(&wrt);

    cnt = cnt*2;

   printf("Writer %d modified cnt to %d\n",(*((int )wno)),cnt);

    sem_post(&wrt);

}

void *reader(void *rno)

{

   // Reader acquire the lock before modifying numreader

   pthread_mutex_lock(&mutex);

    numreader++;

    if(numreader == 1) {

        sem_wait(&wrt); // If this id the first reader, then it will block the

    writer }

   pthread_mutex_unlock(&mutex);

   // Reading Section

   printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);
   // Reader acquire the lock before modifying numreader

   pthread_mutex_lock(&mutex);

    numreader--;

    if(numreader == 0) {

        sem_post(&wrt); // If this is the last reader, it will wake up the

    writer. }

   pthread_mutex_unlock(&mutex);

}

int main()

{
   pthread_t read[10],write[5];

   pthread_mutex_init(&mutex, NULL);

    sem_init(&wrt,0,1);

    int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer
    for(int i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }
    for(int i = 0; i < 10; i++) {
        pthread_join(read[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(write[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;

}
```

Output:



```
zayed@zayed-virtual-machine: ~
zayed@zayed-virtual-machine:~$ gcc monitor.c -omonitor -lpthread
zayed@zayed-virtual-machine:~$ ./monitor
zayed@zayed-virtual-machine:~$ ./monitor
zayed@zayed-virtual-machine:~$ $./semaphoreout
bash: $./semaphoreout: No such file or directory
zayed@zayed-virtual-machine:~$ $./semaphoreout
bash: $./semaphoreout: No such file or directory
zayed@zayed-virtual-machine:~$ ./semaphoreout
Reader 1: read cnt as 1
Reader 5: read cnt as 1
Reader 4: read cnt as 1
Reader 6: read cnt as 1
Reader 3: read cnt as 1
Reader 2: read cnt as 1
Reader 7: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Writer 1 modified cnt to 2
Writer 3 modified cnt to 4
Writer 4 modified cnt to 8
Writer 5 modified cnt to 16
Writer 2 modified cnt to 32
Reader 10: read cnt as 32
zayed@zayed-virtual-machine:~$
```

Result:

Thus Successfully provided a solution to Reader – Writer using mutex and semaphore.

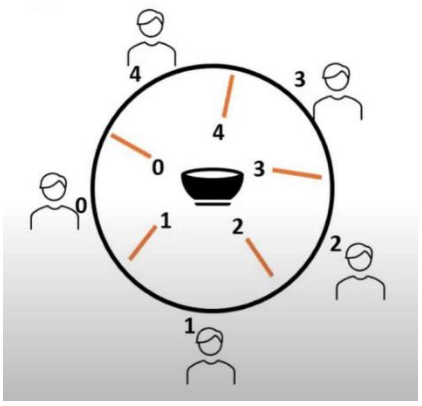
EX:NO:6 Dining Philosopher's Problem

Aim:

To implement and study Dining Philosopher's Problem

Description:

The dining – philosophers problem is considered a classic synchronization problem because it is an example of a large class of concurrency – control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation- free manner. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbours). A philosopher may pick up only one chopstick at a time. Obviously, she cam1ot pick up a chopstick that is already in the hand of a neighbour. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophers problem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence of deadlock.

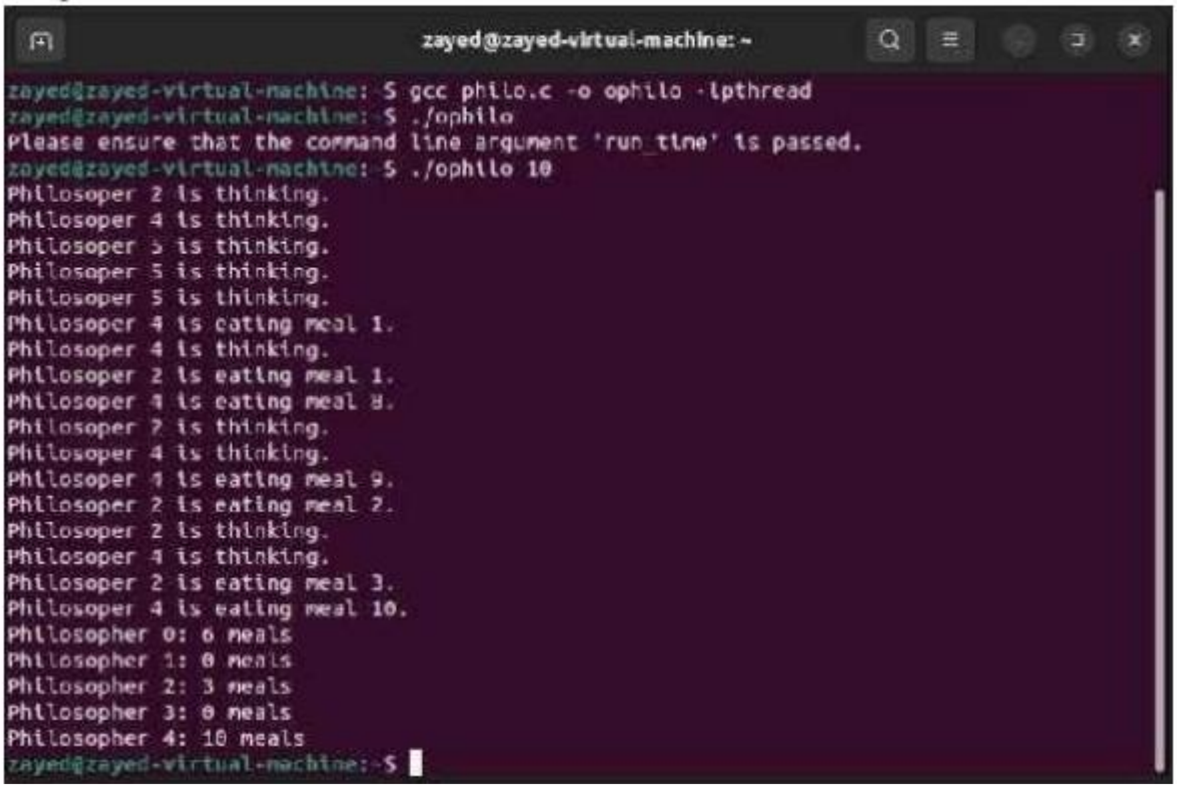


Algorithm:
Step1: Create a file named philosoher.c using "nano" command.
Step2: initiliaze the all header files
Step3:declare and defined the function of pickup_fork(),test()eat (),think() and neighbor() and left and right neighbor ().
Step 4: set the philosopher max5 ,meal max 10
Step5: write coding to state the philosphers and pthread identifiers
Step6.if the is state!=eating the it will be in wait state. Else its calls mutex_unlock() .
Step7. Write the coding for display the result of philosopher and meal state and stop the thread
Step8: Compile the file using the given command "gcc philosoher.c -o philosoher -pthread".
Step 9: use "./philosoher 10" to run the code .
Step 10: stop the program

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

//Function declarations
void *pickup_forks(void * philosopher_number); void *return_forks(void *
philosopher_number); void test(int philosopher_number);
int left_neighbor(int philosopher_number); int right_neighbor(int
philosopher_number); double think_eat_time(void);
void think(double think_time);
void eat(double eat_time);

//Constants to be used in the program.
#define PHILOSOPHER_NUM 5
#define MAX_MEALS 10
#define MAX_THINK_EAT_SEC 3
//States of philosophers.
enum {THINKING, HUNGRY, EATING} state[PHILOSOPHER_NUM];
//Array to hold the thread identifiers.
pthread_t philos_thread_ids[PHILOSOPHER_NUM];
//Mutex lock.
pthread_mutex_t mutex;

//Condition variables.
pthread_cond_t cond_vars[PHILOSOPHER_NUM];

//Array to hold the number of meals eaten for each philosopher.
int meals_eaten[PHILOSOPHER_NUM];

int main(int argc, char *argv[])
{
   //Ensure correct number of command line arguments.
   if(argc != 2)
   {
      printf("Please ensure that the command line argument 'run_time' is
   passed.\n"); }
   else
   {
      //Set command line argument value to variable run_time;
      double run_time = atof(argv[1]);

      //Initialize arrays.
      int i;
      for(i = 0; i < PHILOSOPHER_NUM; i++){
            state[i] = THINKING;
            pthread_cond_init(&cond_vars[i], NULL);
            meals_eaten[i] = 0;
        }
        //Initialize the mutex lock.
        pthread_mutex_init(&mutex, NULL);
        //Join the threads.
        for(i = 0; i < PHILOSOPHER_NUM; i++)
        {
            pthread_join(philos_thread_ids[i], NULL);
        }
        //Create threads for the philosophers.
        for(i = 0; i < PHILOSOPHER_NUM; i++)
        {
            pthread_create(&philos_thread_ids[i], NULL, pickup_forks, (void
        *)&i); }
        sleep(run_time);
        for(i = 0; i < PHILOSOPHER_NUM; i++)
        {
            pthread_cancel(philos_thread_ids[i]);
        }
        //Print the number of meals that each philosopher ate.
        for(i = 0; i < PHILOSOPHER_NUM; i++)
        {
            printf("Philosopher %d: %d meals\n", i, meals_eaten[i]);
        }
    }
    return 0;
}
void *pickup_forks(void * philosopher_number)
{
    int loop_iterations = 0;
    int pnum = *(int *)philosopher_number;
    while(meals_eaten[pnum] < MAX_MEALS)
    {
        printf("Philosoper %d is thinking.\n", pnum);
        think(think_eat_time());
        pthread_mutex_lock(&mutex);
        state[pnum] = HUNGRY;
        test(pnum);
        while(state[pnum] != EATING)
        {
            pthread_cond_wait(&cond_vars[pnum], &mutex);
        }
        pthread_mutex_unlock(&mutex);

        (meals_eaten[pnum])++;
        printf("Philosoper %d is eating meal %d.\n", pnum, meals_eaten[pnum]);
        eat(think_eat_time());
        return_forks((philosopher_number));
        loop_iterations++;
    }
}
void *return_forks(void * philosopher_number)
{
   pthread_mutex_lock(&mutex);
    int pnum = *(int *)philosopher_number;
    state[pnum] = THINKING;
    test(left_neighbor(pnum));
    test(right_neighbor(pnum));
   pthread_mutex_unlock(&mutex);
}
int left_neighbor(int philosopher_number)
{
    return ((philosopher_number + (PHILOSOPHER_NUM - 1)) %
5); }
int right_neighbor(int philosopher_number)
{
    return ((philosopher_number + 1) % 5);
}
void test(int philosopher_number)
{
    if((state[left_neighbor(philosopher_number)] != EATING) &&
        (state[philosopher_number] == HUNGRY) &&
        (state[right_neighbor(philosopher_number)] != EATING))
    {
        state[philosopher_number] = EATING;
       pthread_cond_signal(&cond_vars[philosopher_number]);
    }
}
double think_eat_time(void)
{
    return ((double)rand() * (MAX_THINK_EAT_SEC - 1)) / (double)RAND_MAX +
1; }
void think(double think_time)
{
    sleep(think_time);
}

void eat(double eat_time)
{
    sleep(eat_time);
}
```

Output:



```
zayed@zayed-virtual-machine:~$ gcc philo.c -o ophilo -lpthread
zayed@zayed-virtual-machine:~$ ./ophilo
Please ensure that the command line argument 'run_time' is passed.
zayed@zayed-virtual-machine:~$ ./ophilo 10
Philosoper 2 is thinking.
Philosoper 4 is thinking.
Philosoper 3 is thinking.
Philosoper 5 is thinking.
Philosoper 5 is thinking.
Philosoper 4 is eating meal 1.
Philosoper 4 is thinking.
Philosoper 2 is eating meal 1.
Philosoper 4 is eating meal 8.
Philosoper 2 is thinking.
Philosoper 4 is thinking.
Philosoper 4 is eating meal 9.
Philosoper 2 is eating meal 2.
Philosoper 2 is thinking.
Philosoper 4 is thinking.
Philosoper 2 is eating meal 3.
Philosoper 4 is eating meal 10.
Philosopher 0: 6 meals
Philosopher 1: 0 meals
Philosopher 2: 3 meals
Philosopher 3: 0 meals
Philosopher 4: 10 meals
zayed@zayed-virtual-machine:~$
```

Result:
Thus Successfully implemented the concepts of Dining Philosophers Problem.

EX:NO:7 Bankers Algorithm for Deadlock avoidance

Aim :
To implement and study Bankers Algorithm for Deadlock Avoidance Problem.

DESCRIPTION:
Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources. Data structures
n-Number of process,
m-number of resource types.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.
11. end

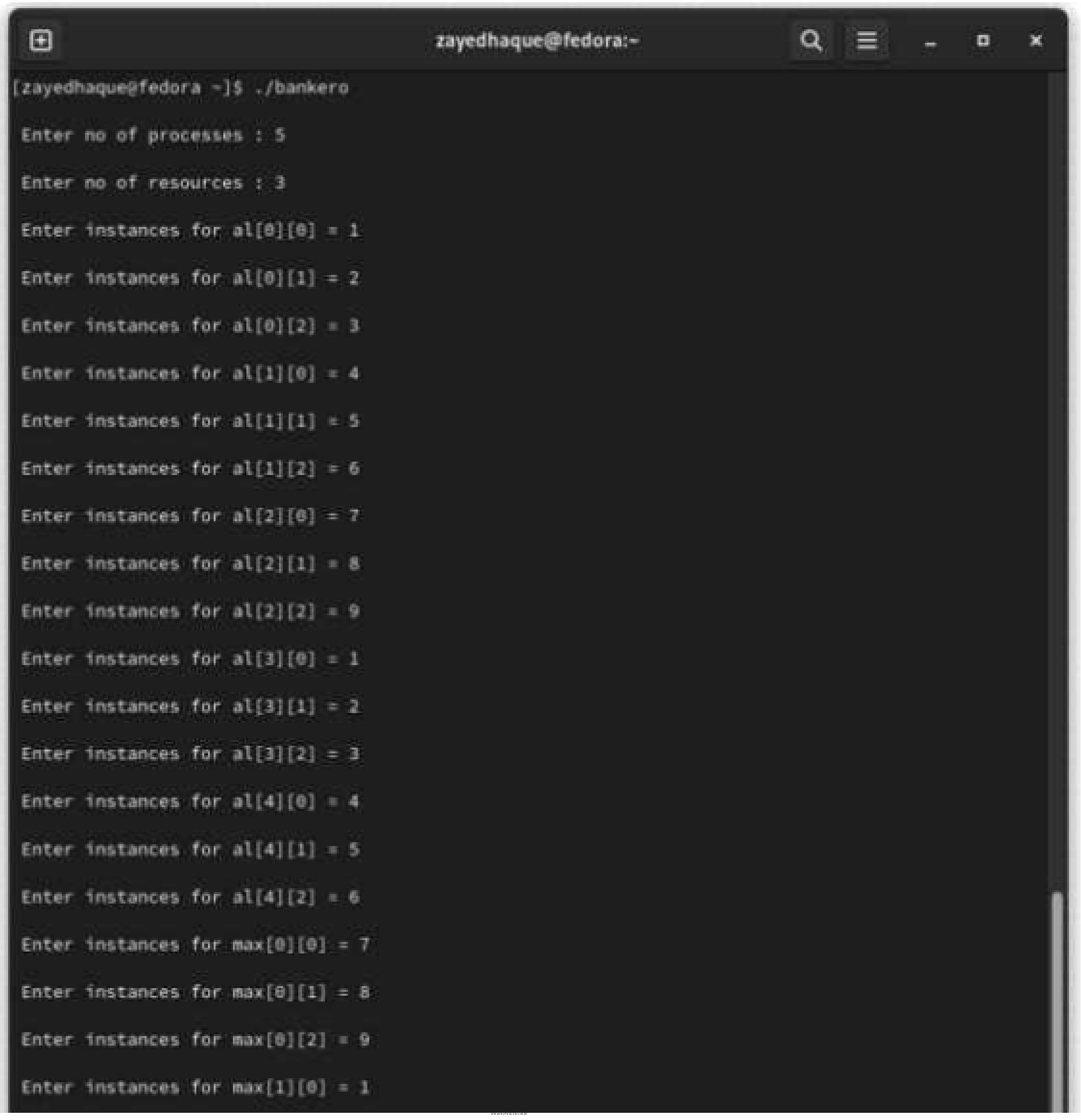
Banker's Algorithm

```
#include <stdio.h>

int m, n, i, j, al[10][10], max[10][10], av[10], need[10][10], temp, z, y, p,
k; void main()

 {

printf("\n Enter no of processes : ");
 scanf("%d", &m); // enter numbers of processes

printf("\n Enter no of resources : ");
 scanf("%d", &n); // enter numbers of resources

for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
printf("\n Enter instances for al[%d][%d] = ", i,j); // al[][] matrix is for allocated
 instances scanf("%d", &al[i][j]);
  al[i][j]=temp;
 }
```

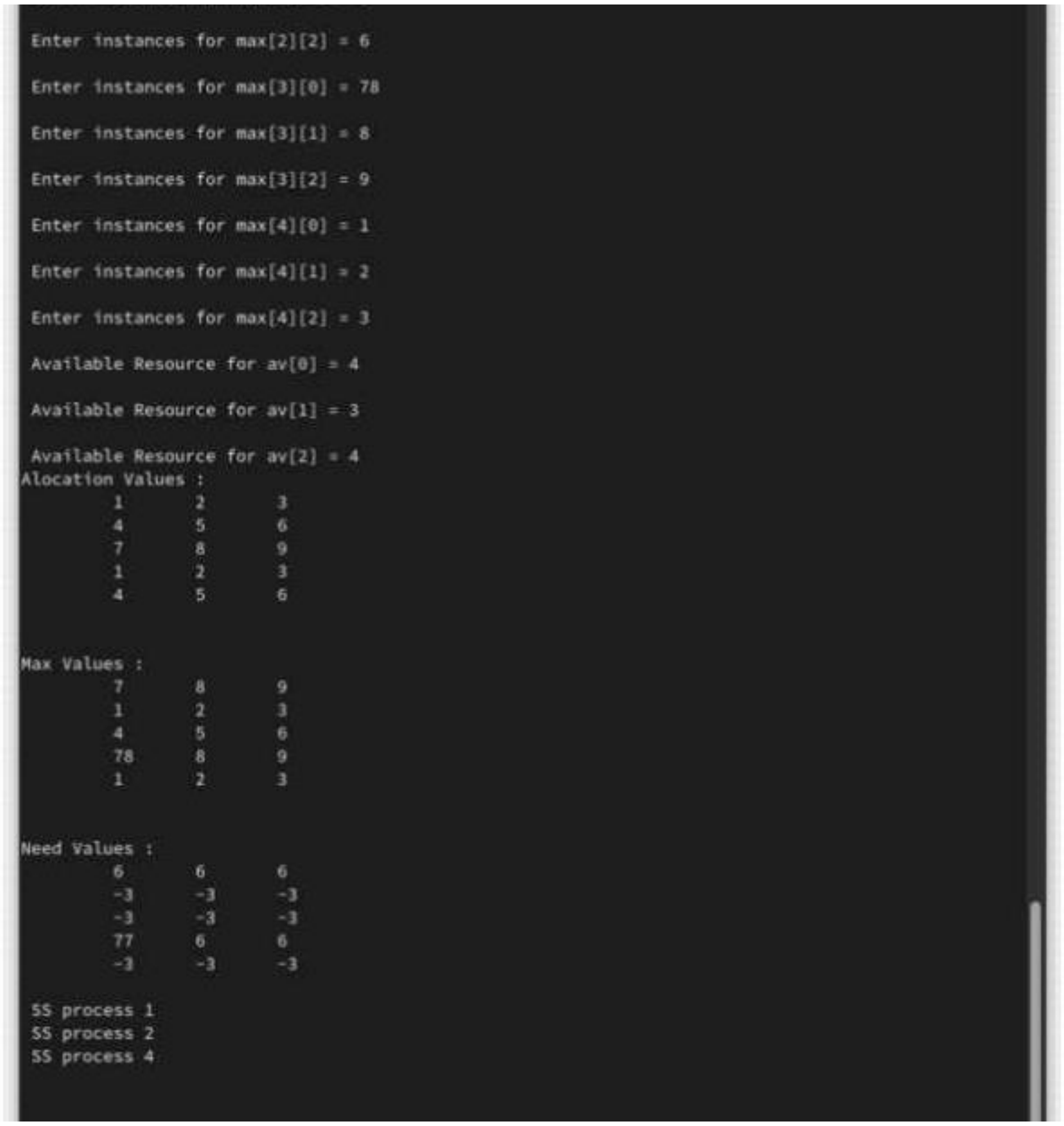
```
}
for (i = 0; i < m;
 i++) {
for (j = 0; j < n; j++) {printf("\n Enter instances for max[%d][%d] = ", i,j); // max[][]
 matrix is for max instances
 scanf("%d", &max[i][j]);
 }
 }
for (i = 0; i < n; i++) {
printf("\n Available Resource for av[%d] = ",i); // av[] matrix is for available instances
 scanf("%d", &av[i]);
            }
            // Print allocation values
printf("Alocation Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {printf(" \t %d", al[i][j]); // printing allocation matrix
            }
printf("\n");
}

printf("\n\n");

// Print max values
printf("Max Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
printf(" \t %d", max[i][j]); // printing max matrix
}
printf("\n");
}
printf("\n\n");

// Print need values
printf("Need Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
 need[i][j] = max[i][j] - al[i][j]; // calculating need matrix
printf("\t %d", need[i][j]); // printing need matrix
}
printf("\n");
}
p = 1; // used for terminating while loop
y = 0;
 while (p != 0) {
for (i = 0; i < m; i++) {
 z = 0;
for (j = 0; j < n; j++) {
 if (need[i][j] <= av[j] &&
(need[i][0] != -1)) { // comparing need with available instance
and // checking if the process is done


// or not
z++; // counter if condition TRUE
}
            }
if (z == n) { // if need<=available TRUE for all resources then
condition // is TRUE
for (k = 0; k < n; k++) {
av[k] += al[i][k]; // new work = work + allocated
}
```

```
printf("\n SS process %d", i); // Print the Process
need[i][0] = -1; // assign -1 if Process done
y++; // cont if process done
}

} // end for loop

if (y == m) { // if all done then
p = 0; // exit while loop
}} // end while printf("\n");
}
```

Output :



```
[zayedhaque@fedora ~]$ ./bankero

 Enter no of processes : 5

 Enter no of resources : 3

 Enter instances for al[0][0] = 1

 Enter instances for al[0][1] = 2

 Enter instances for al[0][2] = 3

 Enter instances for al[1][0] = 4

 Enter instances for al[1][1] = 5

 Enter instances for al[1][2] = 6

 Enter instances for al[2][0] = 7

 Enter instances for al[2][1] = 8

 Enter instances for al[2][2] = 9

 Enter instances for al[3][0] = 1

 Enter instances for al[3][1] = 2

 Enter instances for al[3][2] = 3

 Enter instances for al[4][0] = 4

 Enter instances for al[4][1] = 5

 Enter instances for al[4][2] = 6

 Enter instances for max[0][0] = 7

 Enter instances for max[0][1] = 8

 Enter instances for max[0][2] = 9

 Enter instances for max[1][0] = 1
```



```
 Enter instances for max[2][2] = 6

 Enter instances for max[3][0] = 78

 Enter instances for max[3][1] = 8

 Enter instances for max[3][2] = 9

 Enter instances for max[4][0] = 1

 Enter instances for max[4][1] = 2

 Enter instances for max[4][2] = 3

 Available Resource for av[0] = 4

 Available Resource for av[1] = 3

 Available Resource for av[2] = 4
Alocation Values :
         1       2       3
         4       5       6
         7       8       9
         1       2       3
         4       5       6


Max Values :
         7       8       9
         1       2       3
         4       5       6
         78      8       9
         1       2       3


Need Values :
         6       6       6
         -3      -3      -3
         -3      -3      -3
         77      6       6
         -3      -3      -3

 SS process 1
 SS process 2
 SS process 4
```

Result :

Successfully implemented the concepts of Banker’s Algorithm.

EX:NO:8 FCFS and SJF Scheduling

Aim :

To study the concepts of FCFS and SJF Scheduling

Description:

1. FCFS Scheduling Algorithm :

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm. First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed. Here we are considering that arrival time for all processes is 0. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time Waiting Time(W.T): Time Difference between turn around time and burst time. Waiting Time = Turn Around Time – Burst Time

Algorithm:

Step 1. Input the processes along with their burst time (bt).
Step 2. Find waiting time (wt) for all processes.
Step 3. As first process that comes need not to wait so waiting time for process 1 will be 0 i.e. wt[0] = 0.
Step 4. Find waiting time for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] Step 5. Find turnaround time = waiting_time + burst_time for all processes.
Step 6. Find average waiting time = total_waiting_time / no_of_processes
Step 7. Similarly, find average turnaround time = total_turn_around_time / no_of_processes.
Input : Processes Numbers and their burst times Output : Process-wise burst-time, waiting-time and turnaround-time Also display Average-waiting time and Average-turnaround-time

Program :

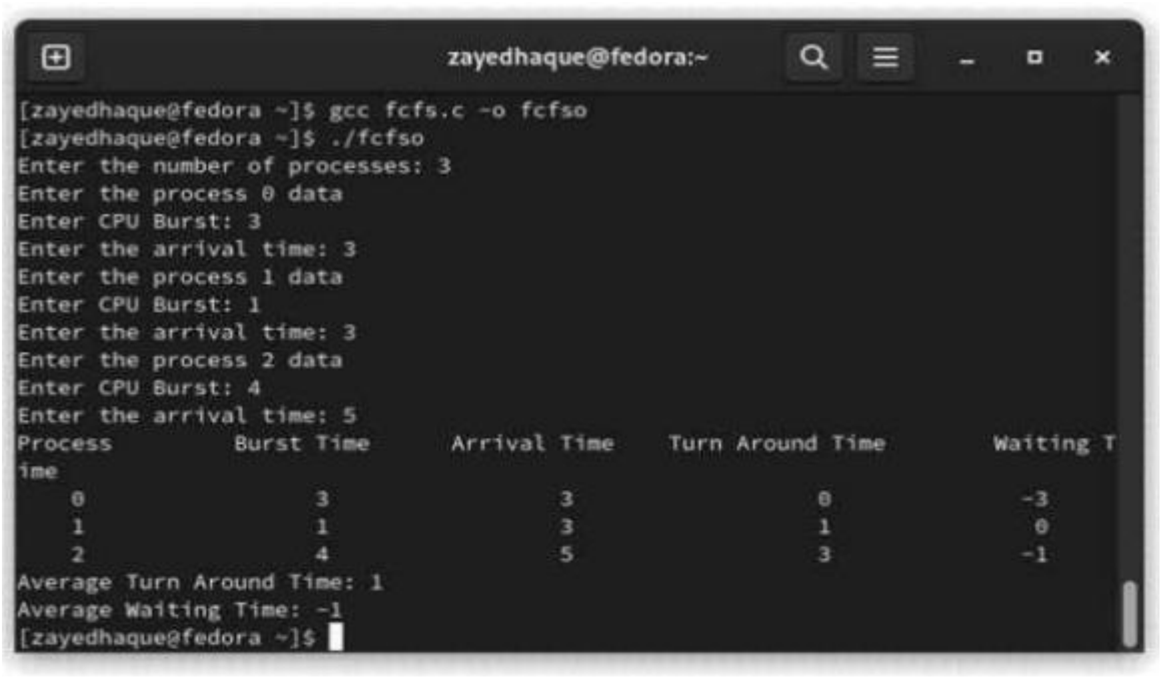
FCFS Scheduling

```
#include <stdio.h>
typedef struct fcfs
{
    int process; //Process
    Number int burst; //Burst
    Time
    int arrival; //Arrival Time
    int tat; //Turn Around
    Time int wt; //Waiting
    Time
}fcfs;
int sort(fcfs [], int);

int main()
{
    int n, i, temp = 0, AvTat = 0, AvWt = 0;
    printf ("Enter the number of processes: ");
```

```
     scanf ("%d", &n);
    fcfs arr[n]; //Array of type
    fcfs int tct[n];
    for (i = 0; i < n; i++)
     {
         arr[i].process = i;
        printf ("Enter the process %d data\n", arr[i].process);
        printf ("Enter CPU Burst: ");
         scanf ("%d", &(arr[i].burst));
        printf ("Enter the arrival time:
         "); scanf ("%d",
         &(arr[i].arrival));
     }
    //Sorting the processes according to their arrival
     time sort(arr, n);
    printf ("Process\t\tBurst Time\tArrival Time\tTurn Around
     Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
     {
         tct[i] = temp + arr[i].burst;
         temp = tct[i];


   arr[i].tat = tct[i] - arr[i].arrival;
   arr[i].wt = arr[i].tat - arr[i].burst;
   AvTat = AvTat + arr[i].tat;
   AvWt = AvWt + arr[i].wt;
   printf ("%5d\t%15d\t\t%9d\t%12d\t%12d\n", arr[i].process,
arr[i].burst,  arr[i].arrival, arr[i].tat, arr[i].wt);
  }
  printf ("Average Turn Around Time: %d\nAverage Waiting Time:
%d\n", AvTat / n,  AvWt / n);
  return 0;
}

//Bubble Sort
int sort(fcfs arr[], int n)
{
  int i, j;
  fcfs k;

  for (i = 0; i < n - 1; i++)
  {
    for (j = i + 1; j < n; j++)
```

```
            {
                //Sorting the processes according to their arrival time
                if (arr[i].arrival > arr[j].arrival)
                {
                    k = arr[i];
                    arr[i] = arr[j];
                    arr[j] = k;
                }
            }
        }
        return 0;
    }
```

Output



```
[zayedhaque@fedora ~]$ gcc fcfs.c -o fcfso
[zayedhaque@fedora ~]$ ./fcfso
Enter the number of processes: 3
Enter the process 0 data
Enter CPU Burst: 3
Enter the arrival time: 3
Enter the process 1 data
Enter CPU Burst: 1
Enter the arrival time: 3
Enter the process 2 data
Enter CPU Burst: 4
Enter the arrival time: 5
Process          Burst Time      Arrival Time    Turn Around Time        Waiting T
ime
    0                3               3                  0                   -3
    1                1               3                  1                    0
    2                4               5                  3                   -1
Average Turn Around Time: 1
Average Waiting Time: -1
[zayedhaque@fedora ~]$
```

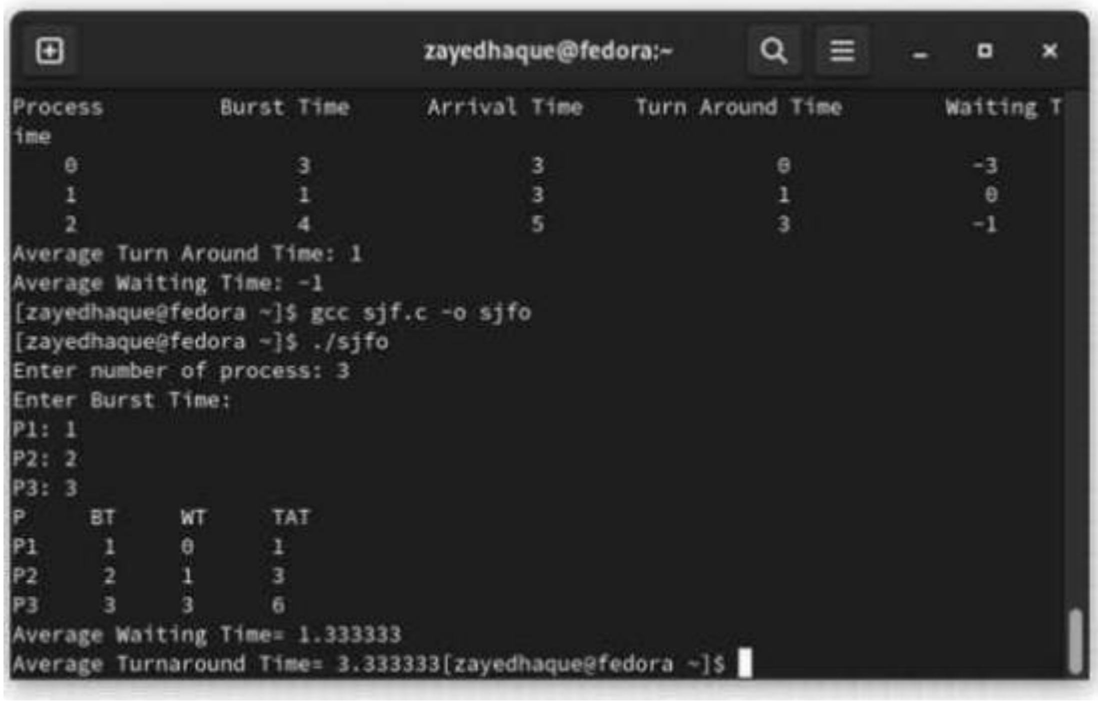
SJF Scheduling

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

```
    A[i][0] = A[index][0];
    A[index][0] = temp;
}
A[0][2] = 0;
// Calculation of Waiting Times for
(i = 1; i < n; i++) {
    A[i][2] = 0;
    for (j = 0; j < i; j++)
        A[i][2] += A[j][1];
    total += A[i][2];
}
avg_wt = (float)total / n;
total = 0;
printf("P BT WT TAT\n");
// Calculation of Turn Around Time and printing the
// data.
for (i = 0; i < n; i++) {
    A[i][3] = A[i][1] + A[i][2];
    total += A[i][3];
    printf("P%d %d %d %d\n", A[i][0],
           A[i][1], A[i][2], A[i][3]);
}
avg_tat = (float)total / n;
```

```
        printf("Average Waiting Time= %f", avg_wt);
        printf("\nAverage Turnaround Time= %f", avg_tat);
    }
```

Output :



```
Process          Burst Time       Arrival Time      Turn Around Time        Waiting T
ime
    0                 3                 3                  0                  -3
    1                 1                 3                  1                   0
    2                 4                 5                  3                  -1
Average Turn Around Time: 1
Average Waiting Time: -1
[zayedhaque@fedora ~]$ gcc sjf.c -o sjfo
[zayedhaque@fedora ~]$ ./sjfo
Enter number of process: 3
Enter Burst Time:
P1: 1
P2: 2
P3: 3
P      BT      WT      TAT
P1      1       0       1
P2      2       1       3
P3      3       3       6
Average Waiting Time= 1.333333
Average Turnaround Time= 3.333333[zayedhaque@fedora ~]$ 
```

Result : Implemented the concepts of FCFS and SJF Scheduling in C successfully.

EX:NO:9 Priority and Round Robin Scheduling

Aim :

To study the concepts of Priority and Round Robin Scheduling

Description

Priority Based Scheduling

Priority Scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority. Priority scheduling also helps OS to involve priority assignments. 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 can be decided based on memory requirements, time requirements, etc. To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the
processes. The waiting time of each process is obtained by summing up the burst times
of all the previous processes.

Round-Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than, it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

Algorithm: Priority Based Scheduling

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time

Step 6: Arrange the processes based on process priority

Step 7: For each process in the Ready Q calculate

Step 8: for each process in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step10: Stop

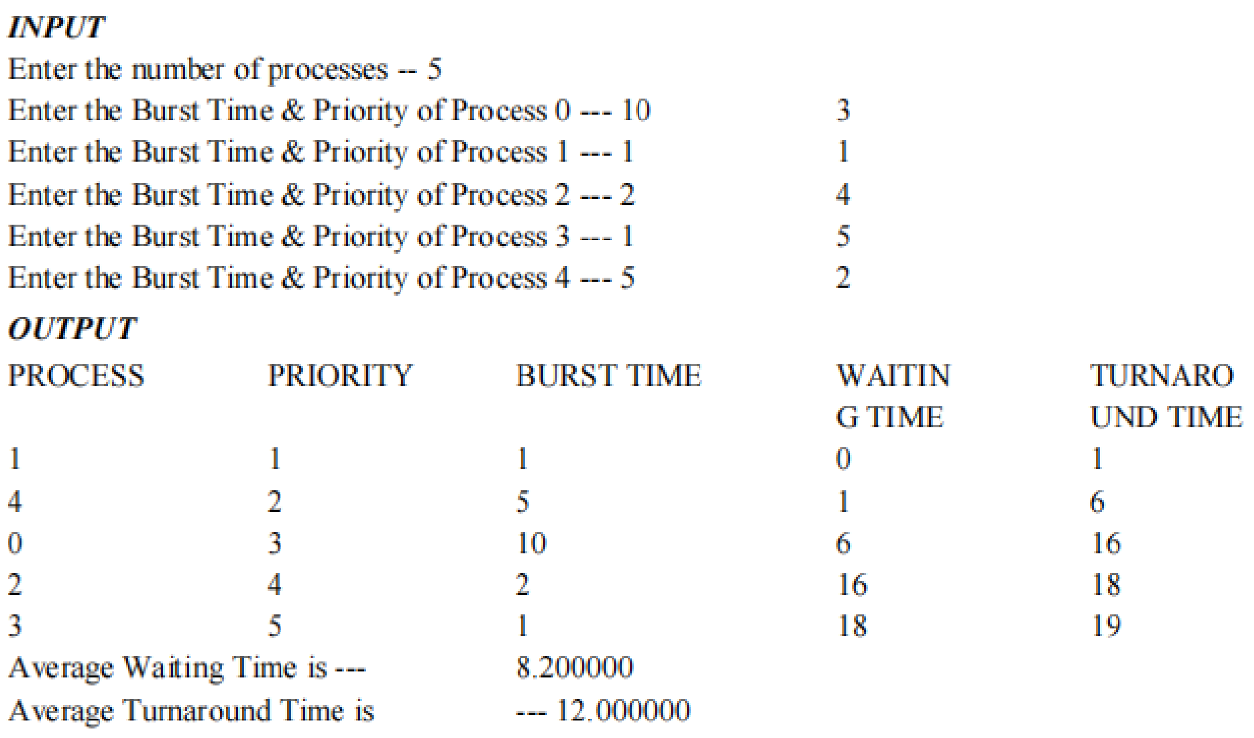
Program :

Priority Scheduling

```
#include<stdio.h>
#define max 10
int main()
{
int
i,j,n,bt[max],p[max],wt[max],tat[max],pr[max],total=0,p
os,temp; float avg_wt,avg_tat;
printf("Enter Total Number of Process:");
scanf("%d",&n);
printf("\nEnter Burst Time and Priority For ");
for(i=0;i<n;i++)
{
printf("\nEnter Process %d: ",i+1);
scanf("%d",&bt[i]);
scanf("%d",&pr[i]);
p[i]=i+1;
}
for(i=0;i<n;i++)
{ pos=i;
for(j=i+1;j<n;j++)
{
 if(pr[j]<pr[pos])
pos=j;
 } temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;
 temp=bt[i];
 bt[i]=bt[pos];
 bt[pos]=temp;
 temp=p[i];
p[i]=p[pos];
p[pos]=temp;
 } wt[0]=0;
for(i=1;i<n;i++)
 { wt[i]=0;
for(j=0;j<i;j++)
 wt[i]+=bt[j];
 total+=wt[i];
 }
 avg_wt=total/n;
 total=0;
```

```
printf("\n\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurn
Around Time"); for(i=0;i<n;i++)
 {
 tat[i]=bt[i]+wt[i];
 total+=tat[i];
printf("\n P%d\t\t%d\t\t\t%d\t\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
 avg_tat=total/n;
printf("\n\nAverage Waiting Time = %.2f",avg_wt);
printf("\nAvg Turn Around Time = %.2f\n",avg_tat);
 return 0;
 }
```

Output :



```
INPUT
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10          3
Enter the Burst Time & Priority of Process 1 --- 1           1
Enter the Burst Time & Priority of Process 2 --- 2           4
Enter the Burst Time & Priority of Process 3 --- 1           5
Enter the Burst Time & Priority of Process 4 --- 5           2
OUTPUT
PROCESS          PRIORITY         BURST TIME          WAITIN          TURNARO
                                                      G TIME          UND TIME
1                1                1                   0               1
4                2                5                   1               6
0                3                10                  6               16
2                4                2                   16              18
3                5                1                   18              19
Average Waiting Time is ---       8.200000
Average Turnaround Time is        --- 12.000000
```

Algorithm:Round Robin Scheduling

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.
Step 6: Consider the ready queue is a circular Q, calculate
a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU fromprocess(n-1)
b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).
Step 7: Calculate
c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number ofprocess Step
8: Stop the process

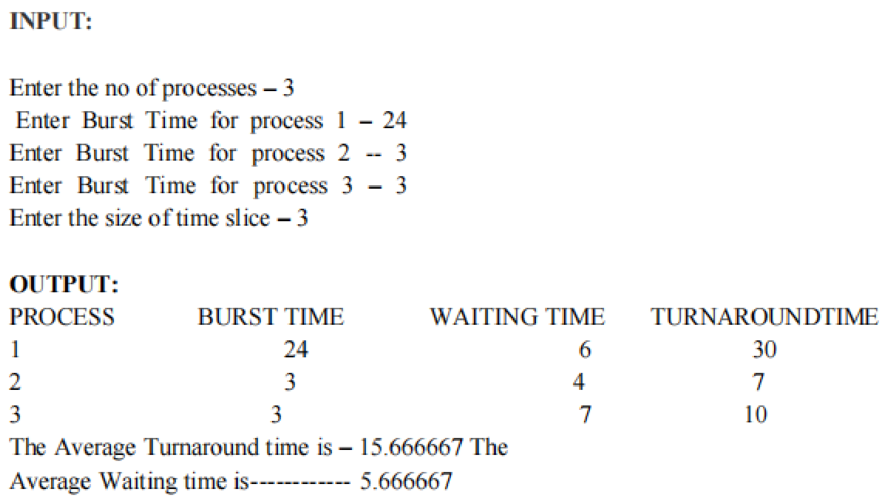
Program

```
#include<stdio.h>

int main()
```

```
{
  //Input no of processed
   int n;
  printf("Enter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n],
   temp_burst_time[n]; int x = n;

  //Input details of processes
  for(int i = 0; i < n; i++)
   {
      printf("Enter Details of Process %d \n", i + 1);
      printf("Arrival Time: ");
       scanf("%d", &arr_time[i]);
      printf("Burst Time: ");
       scanf("%d", &burst_time[i]);
       temp_burst_time[i] = burst_time[i];
   }

  //Input time slot
   int time_slot;
  printf("Enter Time Slot:");
   scanf("%d", &time_slot);

  //Total indicates total time
  //counter indicates which process is executed
   int total = 0, counter = 0,i;
  printf("Process ID Burst Time Turnaround Time Waiting
  Time\n"); for(total=0, i = 0; x!=0; )
   {
      // define the conditions
       if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
       {
          total = total + temp_burst_time[i];
          temp_burst_time[i] = 0;
          counter=1;
       }
       else if(temp_burst_time[i] > 0)
       {
          temp_burst_time[i] = temp_burst_time[i] - time_slot;
          total += time_slot;
       }
       if(temp_burst_time[i]==0 && counter==1)
       {
          x--; //decrement the process no.
```

```
                printf("\nProcess No %d \t\t %d\t\t\t%d\t\t\t %d", i+1,
                    burst_time[i],  total-arr_time[i], total-arr_time[i]-
                    burst_time[i]);
                wait_time = wait_time+total-arr_time[i]-burst_time[i];
                ta_time += total -arr_time[i];
                counter =0;
            }
            if(i==n-1)
            {
                i=0;
            }
            else if(arr_time[i+1]<=total)
            {
                i++;
            }
            else
            {
                i=0;
            }
        }
        float average_wait_time = wait_time * 1.0 / n;
        float average_turnaround_time = ta_time * 1.0 / n;
        printf("\nAverage Waiting Time:%f",
        average_wait_time);  printf("\nAvg Turnaround
        Time:%f", average_turnaround_time);  return 0;
    }
```

Output :



INPUT:

```
Enter the no of processes – 3
 Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 – 3
Enter the size of time slice – 3
```

OUTPUT:

```
PROCESS        BURST TIME        WAITING TIME     TURNAROUNDTIME
1                     24                  6              30
2                     3                   4              7
3                     3                   7              10
The Average Turnaround time is – 15.666667 The
Average Waiting time is----------- 5.666667
```

Result :

Implemented the concepts of Priority Scheduling and Round Robin Scheduling in C successfully

EX No 10	FIFO Page Replacement Algorithm

Aim :

To implement FIFO page replacement algorithm

Description:

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.

FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Algorithm:

```
Step 1. Start the process
Step 2. Read number of pages n
Step 3. Read number of page frames f
Step 4. Read page values into an array a[i]
Step 5. Initialize avail[i]=0 to check page hit
Step 6. Replace the page with circular queue, while re-placing check page availability
in the frame
Place avail[i]=1 if page is placed in the frame Count page faults
Step 7. Print the results.
Step 8. Stop the process.
```

Program :

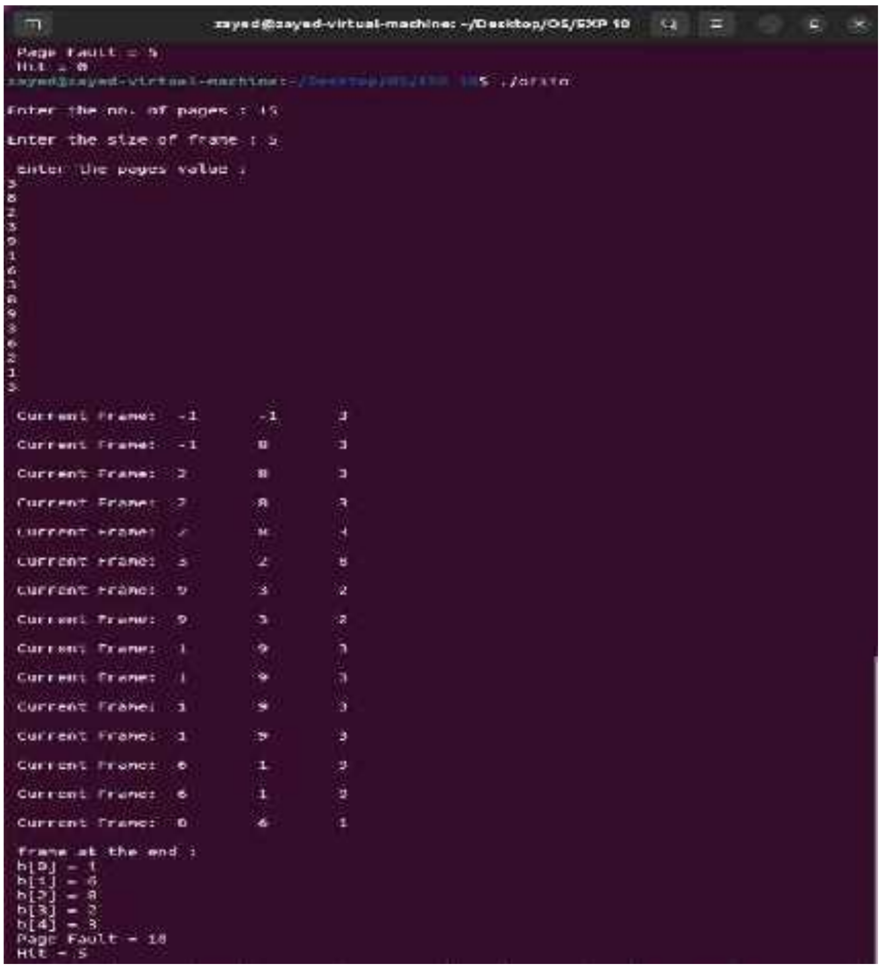
FIFO Page Replacement Algorithm

```
#include <stdio.h>
int a[100],b[100],i,n,z,f,j,pf,h,temp;
void main(){
printf("\nEnter the no. of pages : "); // no. of page  referencing
scanf("%d",&n);
printf("\nEnter the size of frame : "); // no. of page  frames
scanf("%d",&f);
printf("\n Enter the pages value :\n"); // values of page
referencing for(i=0;i<n;i++){
scanf("%d",&a[i]);
}
```

```
for(i=0;i<f;i++){ // assign values of  page frames 1 innitially
 b[i]=-1;
 }
 i=0;j=0;h=0; // i , j used for loop, h for hit count all initialized to 0
 while(i<n){
 if(b[i]=-1 && i<f ){ // when frames are empty so for starting
 enquee b[i]=a[i];
pf++; // page fault counter
 }
 else
 {
 z=0
 ;
for(j=0;j<f;j++){

           if(b[j]==a[i]){ // to check if value already
          present h++; // hit counter
           }
           else{
           z++; // if not hit count increment
           }
 }
 if(z==f){ // if no value
           matched pf++; // page
          fault counter for(j=0;j<f-1;j++){ // shifting
           values
           temp=b[j];
           b[j]=b[j+1]
           ;
           b[j+1]=tem
          p;
           }
           b[f-1]=a[i]; // insert new values
 }
 } // end else
printf("\n Current Frame: %d \t %d \t %d \n",b[2],b[1],b[0]); // frames
 value for  every iteration
 i++;
 } //end while
printf("\n frame at the end :");
for(i=0;i<f;i++){
printf("\n b[%d] = %d",i,b[i]); // frame values at the end }
printf("\n Page Fault = %d ",pf); // no. of page
faults printf("\n Hit = %d ",h); //  no. of hitS
```

```
printf("\n");
}
```

Output :



```
Page Fault = 5
Hit = 0
zayed@zayed-virtual-machine:~/Desktop/OS/EXP 10$ ./fifo

Enter the no. of pages : 15

Enter the size of frame : 5

 Enter the pages value :
3
8
2
3
9
1
6
3
8
9
3
6
2
1
3

 Current Frame:  -1      -1      3

 Current Frame:  -1      8       3

 Current Frame:  2       8       3

 Current Frame:  2       8       3

 Current Frame:  2       8       4

 Current Frame:  3       2       8

 Current Frame:  9       3       2

 Current Frame:  9       3       2

 Current Frame:  1       9       3

 Current Frame:  1       9       3

 Current Frame:  1       9       3

 Current Frame:  1       9       3

 Current Frame:  6       1       9

 Current Frame:  6       1       2

 Current Frame:  8       6       1

 Frame at the end :
 b[0] = 1
 b[1] = 6
 b[2] = 8
 b[3] = 2
 b[4] = 3
 Page Fault = 10
 Hit = 5
```

Result :

Successfully implemented page replacement using FIFO algorithm

Program (4)	Output (3)	Viva(3)	Total(10)

EX No 11	LRU AND LFU Page Replacement

Aim :

To implement page replacement using LRU and LFU

Description:

LRU replaces the page that has not been used for the longest period of time. The idea is based on the assumption that pages used recently will likely be used again soon.

Common implementations include using a linked list or a stack where each time a page is accessed, it is moved to the top. When a page needs to be replaced, the page at the bottom (least recently used) is chosen.

Assume a system has 3 page slots and the page request sequence is: 1, 2, 3, 1, 4, 5.

1. Load pages 1, 2, 3 → (1, 2, 3)
2. Page 1 accessed again → (2, 3, 1)
3. Page 4 replaces page 2 (LRU) → (3, 1, 4)
4. Page 5 replaces page 3 (LRU) → (1, 4, 5)

Algorithm:

Step 1: Start

Step 2: Initialize memory for the frames and stack arrays.

Step 3: Get Input- The number of frames and the reference string are taken as input.

Step 4: LRU Algorithm:

- Iterate over each page request
- Checks if the page is already in the frames.
- If the page is found, update the stack to mark it as recently used.
- If the page is not found, use an empty frame if available or replace the least recently used page.
- Page faults are counted, and the current state of frames and page fault status are printed for each request.

Step 5: Prints the total number of requests, page faults, and page hits.

Step 6: Stop

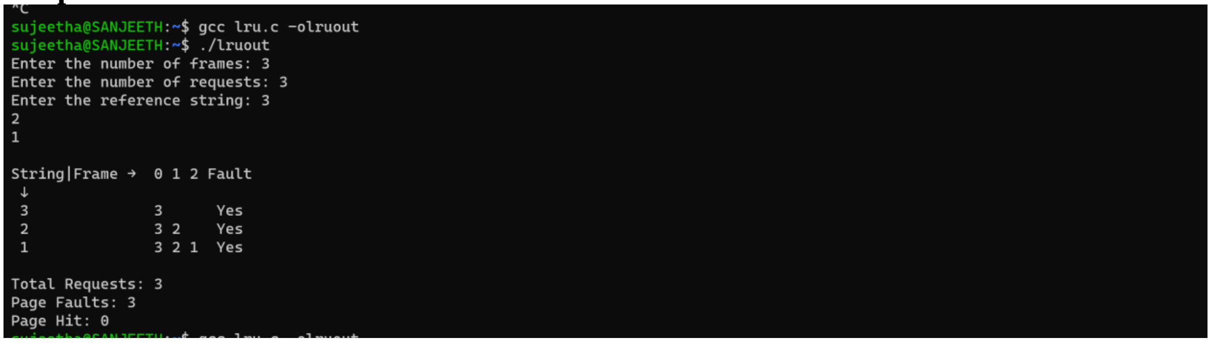
Program :
LRU

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

int main() {
    int capacity, num_requests, page_faults = 0;
    int *frames, *stack, *reference_string;
```

```
// Input the number of frames
printf("Enter the number of frames: ");
scanf("%d", &capacity);

// Allocate memory for frames and stack
frames = (int *)malloc(capacity * sizeof(int));
stack = (int *)malloc(capacity * sizeof(int));

// Initialize frames and stack
for (int i = 0; i < capacity; i++) {
    frames[i] = -1;
    stack[i] = -1;
}

// Input the reference string
printf("Enter the number of requests: ");
scanf("%d", &num_requests);
reference_string = (int *)malloc(num_requests * sizeof(int));

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

printf("\nString|Frame →\t");
for (int i = 0; i < capacity; i++) {
    printf("%d ", i);
}
printf("Fault\n ↓\n");

for (int i = 0; i < num_requests; i++) {
    int page = reference_string[i];
    int found = 0, empty_frame = -1;

    // Check if the page is already in frames
    for (int j = 0; j < capacity; j++) {
        if (frames[j] == page) {
            found = 1;
            // Move the page to the top of the stack
            for (int k = 0; k < capacity; k++) {
                if (stack[k] == j) {
                    for (int l = k; l < capacity - 1; l++) {
                        stack[l] = stack[l + 1];
                    }
                    stack[capacity - 1] = j;
                break;
            }
        }
        break;
    }
    if (frames[j] == -1 && empty_frame == -1) {
        empty_frame = j;
    }
}

if (!found) {
    if (empty_frame != -1) {
        // If there's an empty frame, use it
        frames[empty_frame] = page;
        stack[empty_frame] = empty_frame;
    } else {
        // Replace the least recently used page
        int lru_index = stack[0];
        frames[lru_index] = page;
        // Shift stack elements
        for (int j = 0; j < capacity - 1; j++) {
            stack[j] = stack[j + 1];
        }
        stack[capacity - 1] = lru_index;
    }
    page_faults++;
    printf(" %d\t\t", page);
    for (int j = 0; j < capacity; j++) {
        if (frames[j] != -1) {
            printf("%d ", frames[j]);
        } else {
            printf("  ");
        }
    }
    printf(" Yes\n");
} else {
    printf(" %d\t\t", page);
    for (int j = 0; j < capacity; j++) {
        if (frames[j] != -1) {
            printf("%d ", frames[j]);
        } else {
            printf("  ");
        }
    }
    printf(" No\n");
```

```
        }
    }

    printf("\nTotal Requests: %d\nPage Faults: %d\n", num_requests, page_faults);
    printf("Page Hit: %d\n", num_requests - page_faults);

    // Free allocated memory
    free(frames);
    free(stack);
    free(reference_string);

    return 0;
}
```

Output :



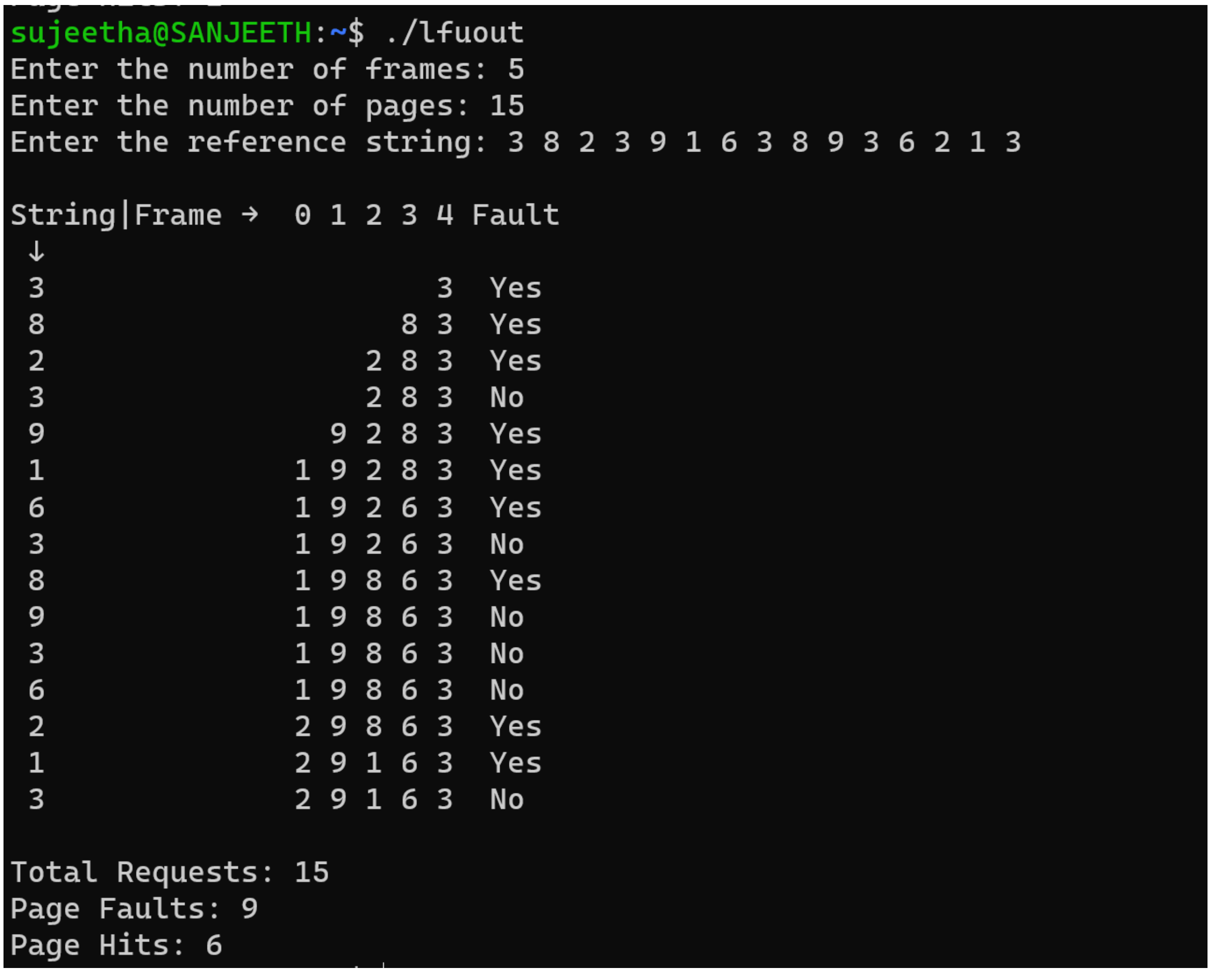
```
^C
sujeetha@SANJEETH:~$ gcc lru.c -olruout
sujeetha@SANJEETH:~$ ./lruout
Enter the number of frames: 3
Enter the number of requests: 3
Enter the reference string: 3
2
1

String|Frame →  0 1 2 Fault
 ↓
 3              3      Yes
 2              3 2    Yes
 1              3 2 1  Yes

Total Requests: 3
Page Faults: 3
Page Hit: 0
```

Algorithm:

LFU

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

typedef struct {
    int page;
    int freq;
    int time;
} Page;

void LFU(int capacity, int pages[], int n) {
    Page frames[capacity];
    int faults = 0, hits = 0;

    for (int i = 0; i < capacity; i++) {
        frames[i].page = -1;
        frames[i].freq = 0;
        frames[i].time = 0;
    }
    printf("\nString|Frame →\t");
    for (int i = 0; i < capacity; i++) {
        printf("%d ", i);
    }
    printf("Fault\n ↓\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int found = 0, index = -1;

        for (int j = 0; j < capacity; j++) {
            if (frames[j].page == page) {
                found = 1;
                frames[j].freq++;
                frames[j].time = i;
                hits++;
                break;
            }
            if (frames[j].page == -1) {
                index = j;
            }
        }

        if (!found) {
            if (index != -1) {
                frames[index].page = page;
                frames[index].freq = 1;
                frames[index].time = i;
            } else {
                int min_freq = frames[0].freq, min_time = frames[0].time;
                index = 0;

                for (int j = 1; j < capacity; j++) {
                    if (frames[j].freq < min_freq || (frames[j].freq == min_freq &&
frames[j].time < min_time)) {
                        min_freq = frames[j].freq;
                        min_time = frames[j].time;
                        index = j;
                    }
                }

                frames[index].page = page;
                frames[index].freq = 1;
                frames[index].time = i;
            }
            faults++;
            printf(" %d\t\t", page);
            for (int j = 0; j < capacity; j++) {
                if (frames[j].page != -1) {
                    printf("%d ", frames[j].page);
                } else {
                    printf("  ");
                }
            }
            printf(" Yes\n");
        } else {
            printf(" %d\t\t", page);
            for (int j = 0; j < capacity; j++) {
                if (frames[j].page != -1) {
                    printf("%d ", frames[j].page);
                } else {
                    printf("  ");
                }
            }
            printf(" No\n");
        }
    }

    printf("\nTotal Requests: %d\nPage Faults: %d\n", n, faults);
    printf("Page Hits: %d\n", hits);
}

int main() {
    int capacity, n;

    printf("Enter the number of frames: ");
    scanf("%d", &capacity);

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

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

    LFU(capacity, pages, n);

    return 0;
}
```

Output



```
sujeetha@SANJEETH:~$ ./lfuout
Enter the number of frames: 5
Enter the number of pages: 15
Enter the reference string: 3 8 2 3 9 1 6 3 8 9 3 6 2 1 3

String|Frame →  0 1 2 3 4 Fault
 ↓
 3                      3  Yes
 8                    8 3  Yes
 2                  2 8 3  Yes
 3                  2 8 3  No
 9                9 2 8 3  Yes
 1              1 9 2 8 3  Yes
 6              1 9 2 6 3  Yes
 3              1 9 2 6 3  No
 8              1 9 8 6 3  Yes
 9              1 9 8 6 3  No
 3              1 9 8 6 3  No
 6              1 9 8 6 3  No
 2              2 9 8 6 3  Yes
 1              2 9 1 6 3  Yes
 3              2 9 1 6 3  No

Total Requests: 15
Page Faults: 9
Page Hits: 6
```

Result :

Successfully implemented page replacement using LRU and LFU algorithms.

Program (4)	Output (3)	Viva(3)	Total(10)

EX:NO:12	Best Fit and Worst Fit Memory Management Policies

Aim :

To implement and study Best fit and Worst fit memory management policies in C.

Description:

The Best Fit algorithm is a type of memory management scheme used in various computing contexts, particularly in dynamic memory allocation. It is designed to manage free memory blocks efficiently by allocating the smallest available block of memory that is large enough to satisfy the request.

Maintain a list of free memory blocks with their sizes and starting addresses.

When a memory request is made, the algorithm searches the list of free blocks to find the smallest block that is large enough to fulfill the request.

This search involves checking each free block and keeping track of the smallest block that fits the requested size.

Once the smallest suitable block is found, the requested memory is allocated from this block.

If the block is larger than the requested size, it is split into two parts: one part to satisfy the request and another part that remains free.

When a memory block is freed, it is added back to the list of free blocks.

Adjacent free blocks are usually merged to form larger blocks, reducing fragmentation

Consider a system with the following free memory blocks:

- Block 1: 100 KB
- Block 2: 500 KB
- Block 3: 200 KB
- Block 4: 300 KB

If a request for 210 KB of memory is made, the Best Fit algorithm will select Block 4 (300 KB) since it is the smallest block that can satisfy the request. Block 4 will then be split into a 210 KB allocated block and a 90 KB free block.

Algorithm:

Step 1: Start

Step 2: Define function for memory blocks

Step 3: Initialize the list of memory blocks

Step 4: Allocate memory using best fit algorithm

-Find the best fit block- If no suitable block is found, allocation fails

-Allocate memory from the best fit block

- update free block

Step 5: Display the results

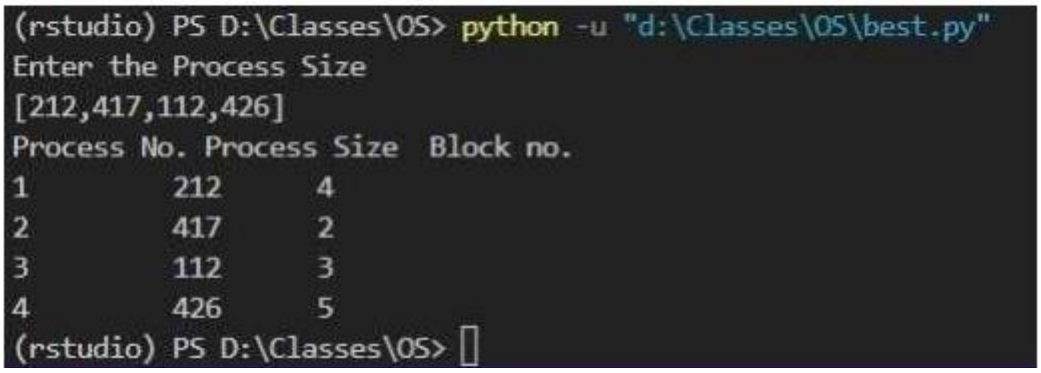
Step 6: Stop

Program :

```
# Function to allocate memory to
blocks
# as per Best fit algorithm
def bestFit(blockSize, m, processSize,
n):
    # Stores block id of the block
    # allocated to a
    process allocation
     = [-1] * n
    # pick each process and find suitable
    # blocks according to its size ad
    # assign to it
    for i in range(n):

        # Find the best fit block for
        # current process
        bestIdx = -1
        for j in range(m):
            if blockSize[j] >= processSize[i]:
                if bestIdx == -1:
                    bestIdx = j
                elif blockSize[bestIdx] > blockSize[j]:
                    bestIdx = j

        # If we could find a block for
        # current process
        if bestIdx != -1:
            # allocate block j to p[i] process
            allocation[i] = bestIdx

            # Reduce available memory in this block.
            blockSize[bestIdx] -= processSize[i]

    print("Process No. Process Size Block no.")
    for i in range(n):
        print(i + 1, " ", processSize[i],
                                end = " ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
```

```
        print("Not Allocated")
# Driver code
if name_== '_main_':
   print("Enter the Process Size")
    l=input()
    blockSize = [100, 500, 200, 300, 600]
   processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)
    bestFit(blockSize, m, processSize, n)
```

Output :



```
(rstudio) PS D:\Classes\OS> python -u "d:\Classes\OS\best.py"
Enter the Process Size
[212,417,112,426]
Process No. Process Size  Block no.
1          212      4
2          417      2
3          112      3
4          426      5
(rstudio) PS D:\Classes\OS> 
```

Worst Fit Policy

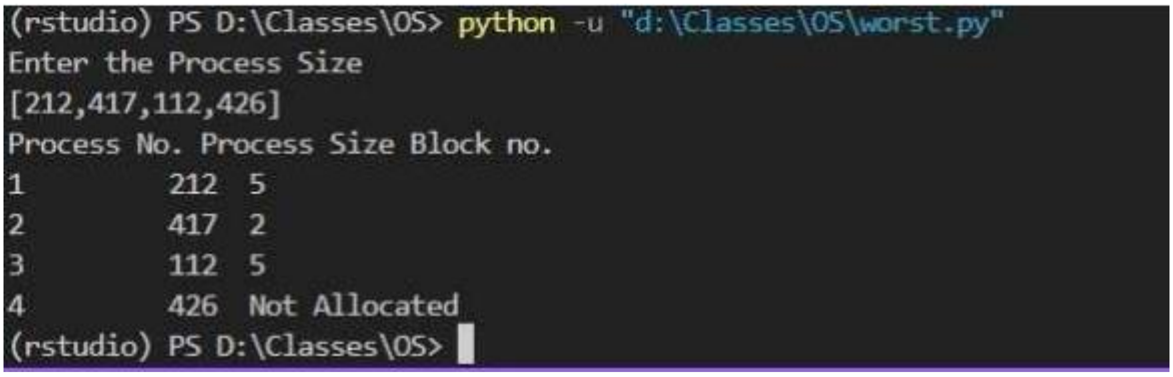
Algorithm :

Step 1: Start
Step 2: Input memory blocks and processes with sizes.
Step 3: Initialize all memory blocks as free.
Step 4: Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
Step 5: If not then leave that process and keep checking the further processes.
Step 6: Stop

Program :

```
# Function to allocate memory to blocks as
# per worst fit algorithm
def worstFit(blockSize, m, processSize, n):
    # Stores block id of the block
    # allocated to a process
    # Initially no block is assigned
    # to any process
    allocation = [-1] * n
    # pick each process and find suitable
    blocks # according to its size ad assign to
    it for i in range(n):
            # Find the best fit block for
      # current process wstIdx = -1
      for j in range(m):
                  if blockSize[j] >= processSize[i]:
                         if wstIdx == -1:
               wstIdx = j
             elif blockSize[wstIdx] < blockSize[j]:
               wstIdx = j
      # If we could find a block for
      # current process
      if wstIdx != -1:

         # allocate block j to p[i] process
         allocation[i] = wstIdx

         # Reduce available memory in this block.
         blockSize[wstIdx] -= processSize[i]

    print("Process No. Process Size
    Block no.") for i in range(n):
       print(i + 1, " ",
          processSize[i], end = " ")
       if allocation[i] != -1:
          print(allocation[i] + 1)
       else:
          print("Not Allocated")

# Driver code
if name_== '_main_':
      print("Enter the Process Size")
    l=input()
    blockSize = [100, 500, 200, 300, 600]
    processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)
    worstFit(blockSize, m, processSize, n)
```

Output :



```
(rstudio) PS D:\Classes\OS> python -u "d:\Classes\OS\worst.py"
Enter the Process Size
[212,417,112,426]
Process No. Process Size Block no.
1          212  5
2          417  2
3          112  5
4          426  Not Allocated
(rstudio) PS D:\Classes\OS>
```

Result :

Successfully implemented and studied Best fit and Worst fit memory management policies in C.

Program (4)	Output (3)	Viva(3)	Total(10)

EX No 13	Disk Scheduling Algorithm

Aim :

To implement and study disk scheduling algorithms.

Description:

One of the responsibilities of the operating system is to use the hardware efficiently. For the
disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

Algorithm:

Step 1: Start
Step 2: Prompt the user to enter the number of I/O requests (n).
Step 3: Prompt the user to enter the initial position of the disk arm (hp).
Step 4: Validate that the initial position is within the range of total cylinders (200).
Step 5: Prompt the user to enter the positions to visit (requests).
Step 6: **FCFS (First-Come, First-Served) Disk Scheduling**

- Initialize time to 0 and pos to hp.
- For each request in requests:
- Add the absolute difference between request and pos to time.
- Update pos to request.
- Calculate average seek time as time / n.
- Return the average seek time.

Step 7: **SSTF (Shortest Seek Time First) Disk Scheduling**

- Initialize time to 0, position to hp, and create a copy of requests.
- While there are requests remaining:
- For each request, push the absolute difference between position and request along with the request itself onto a heap.
- Pop the heap to get the request with the smallest seek time.
- Add the absolute difference between position and the popped request to time.
- Update position to the popped request.
- Remove the popped request from requests.

        - Calculate average seek time as time / n.
        - Return the average seek time.

Step 8: **SCAN Disk Scheduling**

- Initialize time to 0, pos to hp, end to 200, and start to 0.
- Seek from pos to end:
    - For each position from pos to end, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Add the absolute difference between pos and end to time, and update pos to end.
- Seek from end to start:
    - For each position from end to start, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Calculate average seek time as time / n.
- Return the average seek time.

Step 9: **C-SCAN (Circular SCAN) Disk Scheduling**

- Initialize time to 0, pos to hp, end to 200, and start to 0.
- Seek from pos to end:
    - For each position from pos to end, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Add the absolute difference between pos and end to time, and update pos to end.
- Seek from start to hp:
    - For each position from start to hp, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Calculate average seek time as time / n.
- Return the average seek time

Step 10: **LOOK Disk Scheduling**:

- Initialize time to 0, pos to hp, end to the maximum value in requests, and start to the minimum value in requests.
- Seek from pos to end:
    - For each position from pos to end, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Seek from end to start:
    - For each position from end to start, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Calculate average seek time as time / n.
- Return the average seek time.

Step 11: **C-LOOK (Circular LOOK) Disk Scheduling**:

- Initialize time to 0, pos to hp, end to the maximum value in requests, and start to the minimum value in requests.
- Seek from pos to end:
    - For each position from pos to end, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Add the absolute difference between pos and start to time, and update pos to start.
- Seek from start to hp:
    - For each position from start to hp, if the position is in requests, add the absolute difference between pos and the position to time, update pos, and remove the position from requests.
- Calculate average seek time as time / n.
- Return the average seek time.

Step 12: **Main Execution**:

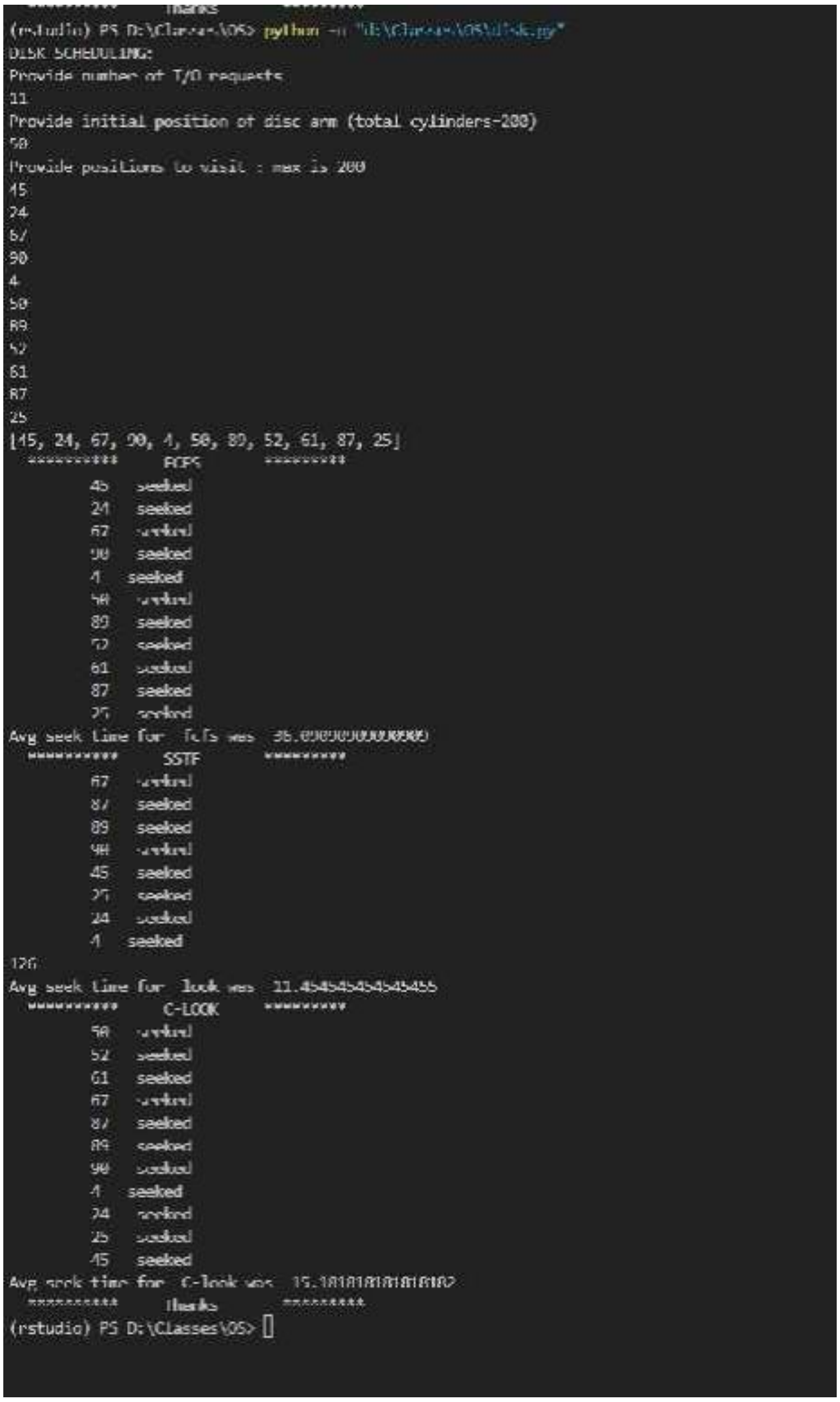
- Prompt user to provide input values for the number of I/O requests, initial disk arm position, and request positions.
- Call each disk scheduling function (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK) with the provided input and print the average seek times.
- Print a thank you message.

Step 13: Stop

Program :

```
from heapq import *
# hp is initial head position
# and requests is the list of requests
# no of cylinders is 200 def
FCFS(hp,requests):
    time = 0
    n = len(requests)
    pos = hp
    for request in requests:
        time += abs(request-pos)
        pos = request
        print(" ",pos," seeked")
    # calculate average seek time
    avg_seek_time = time / n
    return avg_seek_time
# Shortest Seek Time First
def SSTF(hp,reqs):
    requests = reqs.copy()
    time = 0
    position = hp
    n = len(requests)
    heap=[]
    while len(requests)>0:
        for r in requests:
            heappush(heap,(abs(position-r),r))
        x=heappop(heap)[1]
        time+=abs(position-x)
        position=x
        print(" ",x," seeked")
        requests.remove(x)
        heap=[]
    # calculate average seek time
    avg_seek_time = time/n
    return avg_seek_time
def SCAN(hp,reqs):
    requests = reqs.copy()
    pos = hp
    time = 0
    end=200
    start=0
    #seek from curr_pos to end
    which is 200  for i in
    range(pos,end+1):
        if i in requests:
            time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    time+=abs(pos-end)
    pos=end
    #seek back to start
    for i in range(end,start-1,-1):
        if i in requests:
            time+=abs(pos-i)
            # print(time)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    print(time)
    # calculate average seek time
    avg_seek_time = time/n
    return avg_seek_time
def C_SCAN(hp,reqs):
    requests = reqs.copy()
    pos = hp
    time = 0
    end=200
    start=0
    #seek from curr_pos to end
    which is 200  for i in
    range(pos,end+1):
        if i in requests:
            time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    time+=abs(pos-end)
    pos=end
    #seek to hp from start
    for i in range(start,hp+1):
        if i in requests:
            time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    # calculate average seek time
    avg_seek_time = time/n return
    avg_seek_time
def LOOK(hp,reqs): requests =
    reqs.copy() pos = hp
    time = 0 end=max(requests)
    start=min(requests)
    #seek from curr_pos to end which is 200 for i in
        range(pos,end+1): if i in requests: time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    #seek back to start
    for i in range(end,start-1,-1):
        if i in requests: time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
    print(time)
    # calculate average seek time avg_seek_time = time/n return
avg_seek_time  def C_LOOK(hp,reqs): requests = reqs.copy() pos
    = hp
    time = 0 end=max(requests)
    start=min(requests)
    #seek from curr_pos to max of list for i in
        range(pos,end+1): if i in requests:
        time+=abs(pos-i) pos=i
        print(" ",i," seeked")
            requests.remove(i)
    time+=abs(pos-start)
    pos=start
    #seek to hp from start
    for i in range(start,hp+1):
        if i in requests:
            time+=abs(pos-i)
            pos=i
            print(" ",i," seeked")
            requests.remove(i)
        # calculate average seek time avg_seek_time = time/n return
                        avg_seek_time
  if name_=='_main_': print("DISK SCHEDULING:")
     print("Provide number of I/O requests")
     #n is the number of I/O requests n
      = int(input())
     print("Provide initial position of disc arm (total
     cylinders=200)") hp = int(input())
     while hp>200:
     print("!!! INVALID !!! try again")
     hp = int(input())
 print("Provide positions to visit : max is 200")
  requests = []
 for i in range(n): req = int(input()) requests.append(req)
 print(requests)
 #calling the functions
 print(" ********** FCFS *********")
 print("Avg seek time for fcfs was ", FCFS(hp,requests))
 print(" ********** SSTF *********")
 print("Avg seek time for sstf was ", SSTF(hp,requests))
 print(" ********** SCAN *********")
 print("Avg seek time for scan was ", SCAN(hp,requests))
 print(" ********** C-SCAN *********")
 print("Avg seek time for C-scan was ",
     C_SCAN(hp,requests))
 print(" ********** LOOK *********")
 print("Avg seek time for look was ",
     LOOK(hp,requests))
 print(" ********** C-LOOK *********")
 print("Avg seek time for C-look was ",
     C_LOOK(hp,requests))
 print(" ********** Thanks *********")
```

Output :



```
  **********     Thanks       *********
(rstudio) PS D:\Classes\OS> python -u "d:\Classes\OS\disk.py"
DISK SCHEDULING:
Provide number of I/O requests
11
Provide initial position of disc arm (total cylinders-200)
50
Provide positions to visit : max is 200
45
24
67
90
4
50
89
52
61
87
25
[45, 24, 67, 90, 4, 50, 89, 52, 61, 87, 25]
  **********     FCFS       *********
          45   seeked
          24   seeked
          67   seeked
          90   seeked
          4   seeked
          50   seeked
          89   seeked
          52   seeked
          61   seeked
          87   seeked
          25   seeked
Avg seek time for  fcfs was  36.09090909090909
  **********     SSTF       *********
          67   seeked
          87   seeked
          89   seeked
          90   seeked
          45   seeked
          25   seeked
          24   seeked
          4   seeked
126
Avg seek time for  look was  11.454545454545455
  **********     C-LOOK       *********
          50   seeked
          52   seeked
          61   seeked
          67   seeked
          87   seeked
          89   seeked
          90   seeked
          4   seeked
          24   seeked
          25   seeked
          45   seeked
Avg seek time for  C-look was  15.181818181818182
  **********     Thanks       *********
(rstudio) PS D:\Classes\OS> ▯
```

Result :

Successfully implemented various types of Disk Scheduling Algorithms.

Program (4)	Output (3)	Viva(3)	Total(10)

EX No 14	Sequential and Indexed File Allocation

Aim :

To implement and study Sequential and Indexed File Allocation

Description:

The most common form of file structure is the sequential file in this type of file, a fixed
format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

Sequential File Allocation

Algorithm :

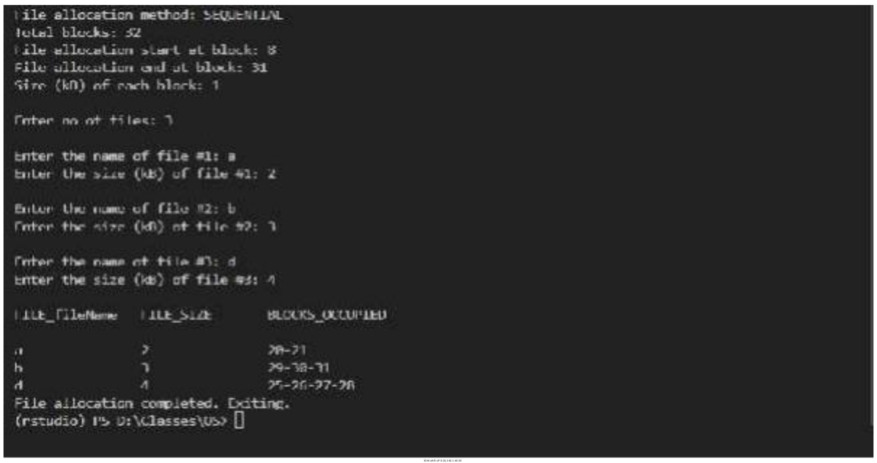
```
Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations to each in sequential order a).
Randomly select a location from available location s1= random(100);
a) Check whether the required locations are free from the selected
location.
if(b[s1].flag==0){
for
(j=s1;j<s1+p[i];j++){
if((b[j].flag)==0)count++;
}
if(count==p[i]) break;
}
b) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)
{
k[i][j]=s; j=j+1; b[s].bno=s;
b[s].flag=1;
}
Step 5: Print the results file no, length, Blocks allocated.
Step 6: Stop the program
```

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TOTAL_DISK_BLOCKS 32
#define TOTAL_DISK_INODES 8
#ifndef MAX
#define MAX 15
#endif
int blockStatus[TOTAL_DISK_BLOCKS]; // free = 0
int blockStart;
struct file_table { char fileName[20];
  int startBlock;
  int fileSize;
  int allotStatus;
};
struct file_table fileTable[TOTAL_DISK_BLOCKS - TOTAL_DISK_INODES];
int AllocateBlocks(int Size) {
  int i = 0, count = 0, inList = 0, nextBlock = 0;
  int allocStartBlock = TOTAL_DISK_INODES;
  int allocEndBlock = TOTAL_DISK_BLOCKS - 1;
  // check whether sufficient free blocks are available
  for (i = 0; i < (TOTAL_DISK_BLOCKS -
            TOTAL_DISK_INODES); i++) if (blockStatus[i] ==
            0)
            count++;
  if (count < Size)
            return 1; // not enough free blocks
  count = 0;
  while (count < Size) {
nextBlock = (rand() % (allocEndBlock - allocStartBlock + 1)) + allocStartBlock;
            for (i = nextBlock; i < (nextBlock + Size); i++) {
            if (blockStatus[i] == 0)
            count = count + 1;
            else { count =
            0; break;
            }
            }
  }
  blockStart = nextBlock;
  for (int i = 0; i < Size; i++) {
            blockStatus[blockStart + i] = 1;
  }
  if (count == Size)
            return nextBlock; // success
  else
    return 1; // not successful
}
   void main() {
  int i = 0, j = 0, numFiles = 0, nextBlock = 0, ret = 1,
  totalFileSize = 0; char s[20];
  //-- -
  char *header[] = {"FILE_fileName", "FILE_SIZE", "BLOCKS_OCCUPIED"};
  printf("File allocation method: SEQUENTIAL\n");
  printf("Total blocks: %d\n", TOTAL_DISK_BLOCKS);
  printf("File allocation start at block: %d\n",
  TOTAL_DISK_INODES);  printf("File allocation end at
  block: %d\n", TOTAL_DISK_BLOCKS - 1);  printf("Size (kB)
  of each block: 1\n\n");
  printf("Enter no of files: ");
```

```
 scanf("%d", &numFiles);
 //numFiles = 3;
 for (i = 0; i < numFiles; i++) {
        //-- -
        printf("\nEnter the name of file #%d: ", i+1);
         scanf("%s", fileTable[i].fileName); printf("Enter
         the size (kB) of file #%d: ", i+1); scanf("%d",
         &fileTable[i].fileSize);
        //strcpy(fileTable[i].fileName, "testfile");
         srand(1234);
         ret = AllocateBlocks(fileTable[i].fileSize);
        //-- -
         if (ret == 1) {
         exit(0);
         } else {
          fileTable[i].startBlock = ret;
         }
 }
 //-- -
 printf("\n%*s %*s %*s\n", -MAX, header[0], -MAX, header[1], MAX,
 header[2]); //Seed the pseudo-random number generator used by
 rand() with the value seed  srand(1234);
 //-- -
 for (j = 0; j < numFiles; j++) {
        printf("\n%*s %*d ", -MAX, fileTable[j].fileName, -MAX,
        fileTable[j].fileSize); for(int k=0;k<fileTable[j].fileSize;k++) {
        printf("%d%s", fileTable[j].startBlock+k, (k == fileTable[j].fileSize-
         1) ? "" : "-"); }
 }
 printf("\nFile allocation completed. Exiting.\n");
}
```

Output :



```
File allocation method: SEQUENTIAL
Total blocks: 32
File allocation start at block: 8
File allocation end at block: 31
Size (kB) of each block: 1

Enter no of files: 3

Enter the name of file #1: a
Enter the size (kB) of file #1: 2

Enter the name of file #2: b
Enter the size (kB) of file #2: 3

Enter the name of file #3: d
Enter the size (kB) of file #3: 4

FILE_FileName   FILE_SIZE      BLOCKS_OCCUPIED

a               2              20-21
b               3              29-30-31
d               4              25-26-27-28
File allocation completed. Exiting.
(rstudio) PS D:\Classes\OS> 
```

Indexed File Allocation :

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

```
q=random(100);
{
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
```

Step 5: Print the results file no, length ,Blocks

allocated.

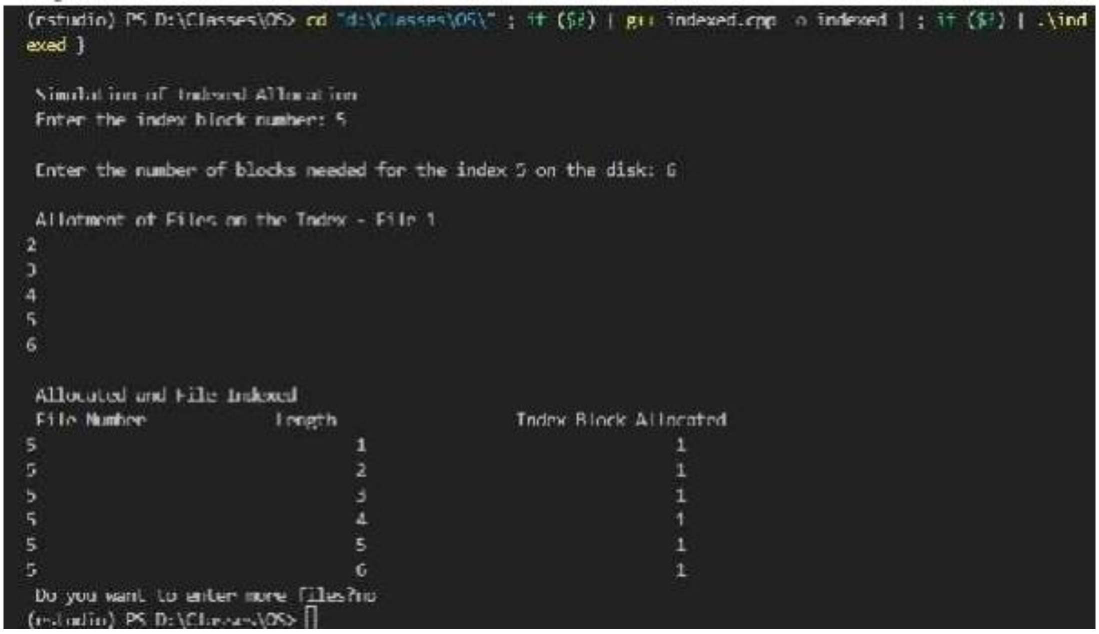
Step 6: Stop the program

Program :

```
#include <iostream>
using namespace std;
int MaximumSizeAvailable=12;
int BlockFiles[12];
int BlockIndex[12];
int IndexBlock, n;int i;int j; int k; int Choice=0;
void Allotment();
void IndexedAllocation(){
cout << "\n Enter the index block number: ";
cin >> IndexBlock;
if (BlockFiles[IndexBlock] != 1){
cout << "\n Enter the number of blocks needed for the
index " <<  IndexBlock << " on the disk: ";
cin >> n;
}
else{
cout << IndexBlock << " is already allocated
\n" << endl;  IndexedAllocation();
} Allotment();
}
void Allotment(){
int occupied = 0;
cout<<"\n Allotment of Files on the Index - File ";
for (i=0; i<n; i++){
```

```
cin >> BlockIndex[i];
if (BlockFiles[BlockIndex[i]] == 0)
occupied++;
}if (occupied == n){ //All the files are alloted and indexed for (j=0;
j<n; j++){  BlockFiles[BlockIndex[j]] = 1;
}
cout << "\n Allocated and File Indexed";
cout<<"\n File Number \t\t Length \t\t Index Block
Allocated"; for (k=0; k<n; k++){
cout <<"\n"<< IndexBlock << " \t\t\t\t " << BlockIndex[k]
<< " \t\t\t\t " << BlockFiles[BlockIndex[k]];


}
}
else{
cout << "\n File in the index is already allocated";
cout << "\n Enter another file to index"; Allotment();
}
cout << "\n Do you want to enter more files?";
cin >> Choice;
if (Choice == 1)
IndexedAllocation();
else
exit(0);
return;
}
int main()
{ cout<<"\n Simulation of Indexed Allocation";
for(int i=0;i<MaximumSizeAvailable;i++){
BlockFiles[i]=0;
BlockIndex[i]=0;
}
IndexedAllocation();
return 0;
}
```

Output :



```
(rstudio) PS D:\Classes\OS> cd "d:\Classes\OS\" ; if ($?) { g++ indexed.cpp -o indexed } ; if ($?) { .\ind
exed }

 Simulation of Indexed Allocation
 Enter the index block number: 5

 Enter the number of blocks needed for the index 5 on the disk: 6

 Allotment of Files on the Index - File 1
2
3
4
5
6


 Allocated and File Indexed
 File Number              Length                   Index Block Allocated
5                               1                                  1
5                               2                                  1
5                               3                                  1
5                               4                                  1
5                               5                                  1
5                               6                                  1
 Do you want to enter more files?no
(rstudio) PS D:\Classes\OS> 
```

Result :

Successfully implemented and studied Sequential and Indexed File Allocation.

Program (4)	Output (3)	Viva(3)	Total(10)

EX No 15	File Organization Schemes for Single and Two Level Directory

Aim :

To implement and study file organization schemes for single and two level directory.

Description:

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory.

Algorithm:

Step 1: Start

Step 2: Initialize dir.fno to 0 to keep track of the number of files

Step3: Display options to the user until they choose to exit:

1. Insert a file
2. Display files
3. Delete a file
4. Search for a file
5. Exit

Step 4: Insert file:

- Prompt the user to enter the file name.
- Store the file name in the dir.fileName array at the position indicated by dir.fno.
- Increment dir.fno by 1.

Step 5: Display files

- Print the directory name.
- Loop through the dir.fileName array up to dir.fno and print each file name.

Step 6: Delete a file

- Prompt the user to enter the name of the file to be deleted.
- Loop through the dir.fileName array to find the file.
- If the file is found:
  - Print a message indicating that the file is deleted.
  - Copy the last file in the list (dir.fileName[dir.fno - 1]) to the position of the deleted file.
  - Decrement dir.fno by 1.

Step 7: Search

- Prompt the user to enter the name of the file to be searched.
- Loop through the dir.fileName array to find the file.
- If the file is found, print its position.
- If the file is not found, print a message indicating that the file is not found.

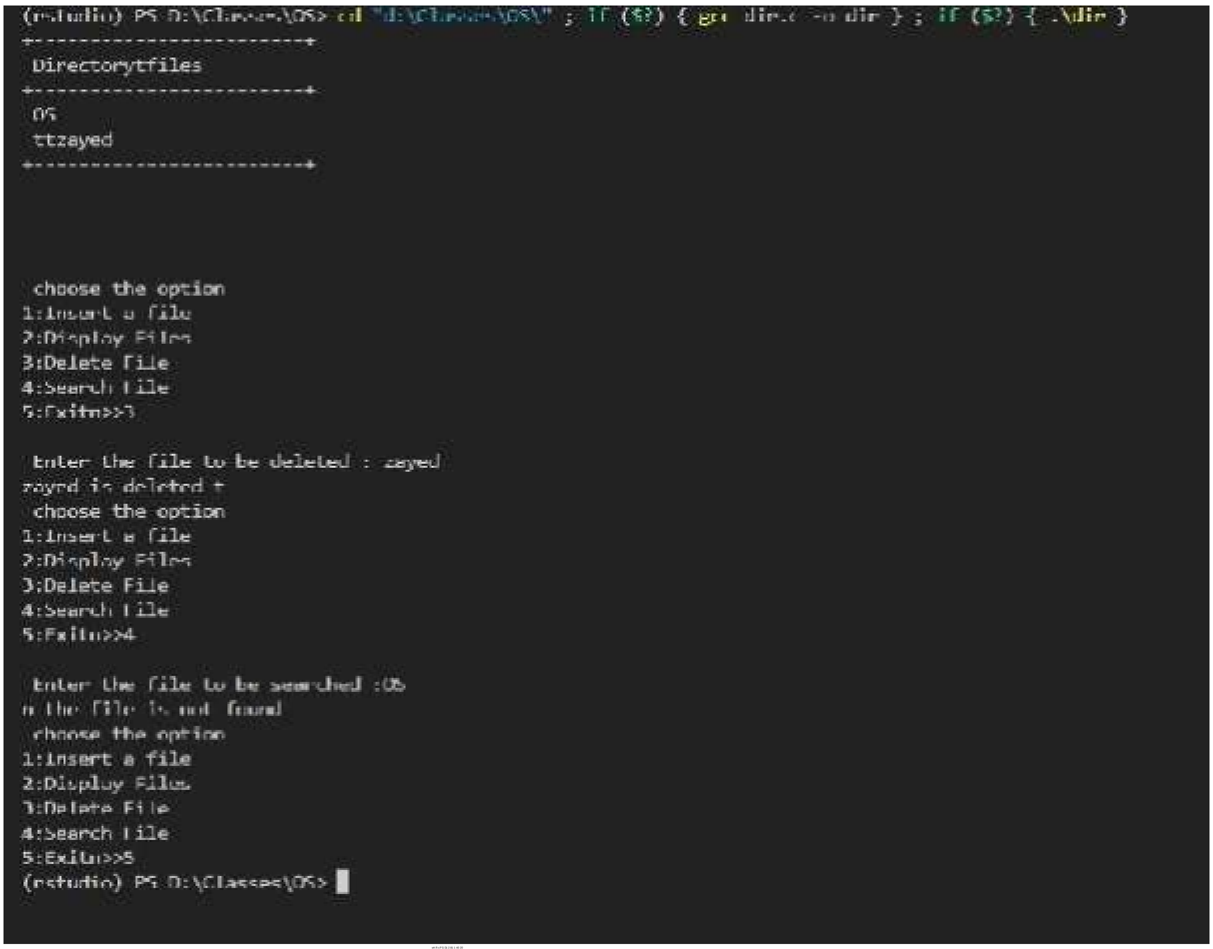
Step 8: Exit - Closes the program execution

Step 9: Stop

Single-level directory Program :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct file
{
    char fileName[15][20];
    char dirName[10];
    int fno;
};
struct file dir;
int i, n;
void InsertFile()
{
   printf("\n Enter the File name ");
    scanf("%s", dir.fileName[dir.fno]);
    dir.fno++;
}
void DisplayFiles()
{
   printf("\n\n\n\n");
   printf("+------------------------+");
   printf("\n Directorytfiles \n");
   printf("+------------------------+");
   printf("\n %s", dir.dirName);
   for (i = 0; i < dir.fno; i++)
    {
       printf("\n tt%s", dir.fileName[i]);
    }
   printf("\n+------------------------+");
   printf("\n\n\n\n");
}
void DeleteFile()
{
    char name[20];
   printf("\n Enter the file to be deleted : ");
    scanf("%s", name);
   for (i = 0; i < dir.fno; i++)
    {
        if (strcmp(dir.fileName[i], name) == 0)
        {
           printf("%s is deleted t", dir.fileName[i]);
            strcpy(dir.fileName[i], dir.fileName[dir.fno - 1]);
            dir.fno--;
        }
    }
}
void SearchFile()
{
    char name[20];
    int found = -1;
    printf("\n Enter the file to be searched :");
    scanf("%s", name);
    for (i = 0; i < dir.fno; i++)
    {
        if (strcmp(dir.fileName[i], name) == 0)
        {
            printf("\n The File is found at position %d", i + 1);
            found = 1;
            break;
        }
    }
    if (found == -1)
        printf("n the file is not found ");
}
int main()
{
    int op;
    dir.fno = 0;
    printf("\n Enter the directory name : ");
    scanf("%s", dir.dirName);
    while (1)
    {
        printf("\n choose the option \n1:Insert a file\n2:Display Files\n3:Delete
File\n4:Search File\n5:Exitn>>");
        scanf("%d", &op);
        switch (op)
        {
        case 1:
            InsertFile();
            break;
        case 2:
            DisplayFiles();
            break;
        case 3:
            DeleteFile();
            break;
        case 4:
            SearchFile();
            break;
        case 5:
            exit(0);
        }
    }
    return 0;
}
```

Output :



```
(rstudio) PS D:\Classes\OS> cd "d:\Classes\OS\" ; if ($?) { gcc dir.c -o dir } ; if ($?) { .\dir }
+-------------------------+
 Directoryfiles
+-------------------------+
 OS
 ttzayed
+-------------------------+




 choose the option
1:insert a file
2:Display Files
3:Delete File
4:Search File
5:Exit>>3

 Enter the file to be deleted : zayed
zayed is deleted !
 choose the option
1:insert a file
2:Display Files
3:Delete File
4:Search File
5:Exit>>4

 Enter the file to be searched :OS
n the file is not found
 choose the option
1:insert a file
2:Display Files
3:Delete File
4:Search File
5:Exit>>5
(rstudio) PS D:\Classes\OS>
```

Two-level directory

Algorithm:

Step 1: Start

Step 2: Set dcnt to 0 to keep track of the number of directories.

Step3: Display options to the user until they choose to exit:

1. Create a directory
2. Create a file
3. Delete a file
4. Search for a file
5. Display directories and files
6. Exit

Step 4: Create a directory

- Prompt the user to enter the directory name.
- Store the directory name in dir[dcnt].dname.
- Initialize dir[dcnt].fcnt to 0 to keep track of the number of files in the directory.
- Increment dcnt by 1.
- Print a message indicating that the directory has been created.

Step 5: Create a file

- Prompt the user to enter the name of the directory.
- Loop through the list of directories to find the specified directory.
- If the directory is found:
  - Prompt the user to enter the file name.

        - Store the file name in the fname array of the directory.
        - Increment the file count (fcnt) of the directory by 1.
        - Print a message indicating that the file has been created.
    - If the directory is not found, print a message indicating that the directory was not found.

Step 6: Delete a file

- Prompt the user to enter the name of the directory.
- Loop through the list of directories to find the specified directory.
- If the directory is found:
- Prompt the user to enter the file name.
- Loop through the list of files in the directory to find the specified file.
- If the file is found:
    - Print a message indicating that the file has been deleted.
    - Decrement the file count (`fcnt`) of the directory by 1.
    - Replace the deleted file with the last file in the list.
    - Break the loop.
- If the file is not found, print a message indicating that the file was not found.

Step 7: Search

- Prompt the user to enter the name of the directory.
- Loop through the list of directories to find the specified directory.
- If the directory is found:
    - Prompt the user to enter the file name.
    - Loop through the list of files in the directory to find the specified file.
    - If the file is found, print a message indicating that the file has been found.
    - If the file is not found, print a message indicating that the file was not found.
- If the directory is not found, print a message indicating that the directory was not found.

Step 8: Display Directories and Files

- If no directories exist, print a message indicating that there are no directories.
- Otherwise, print the list of directories and their respective files.

Step 9: Exit - Closes the program execution

Step 10: Stop

Two-level directory

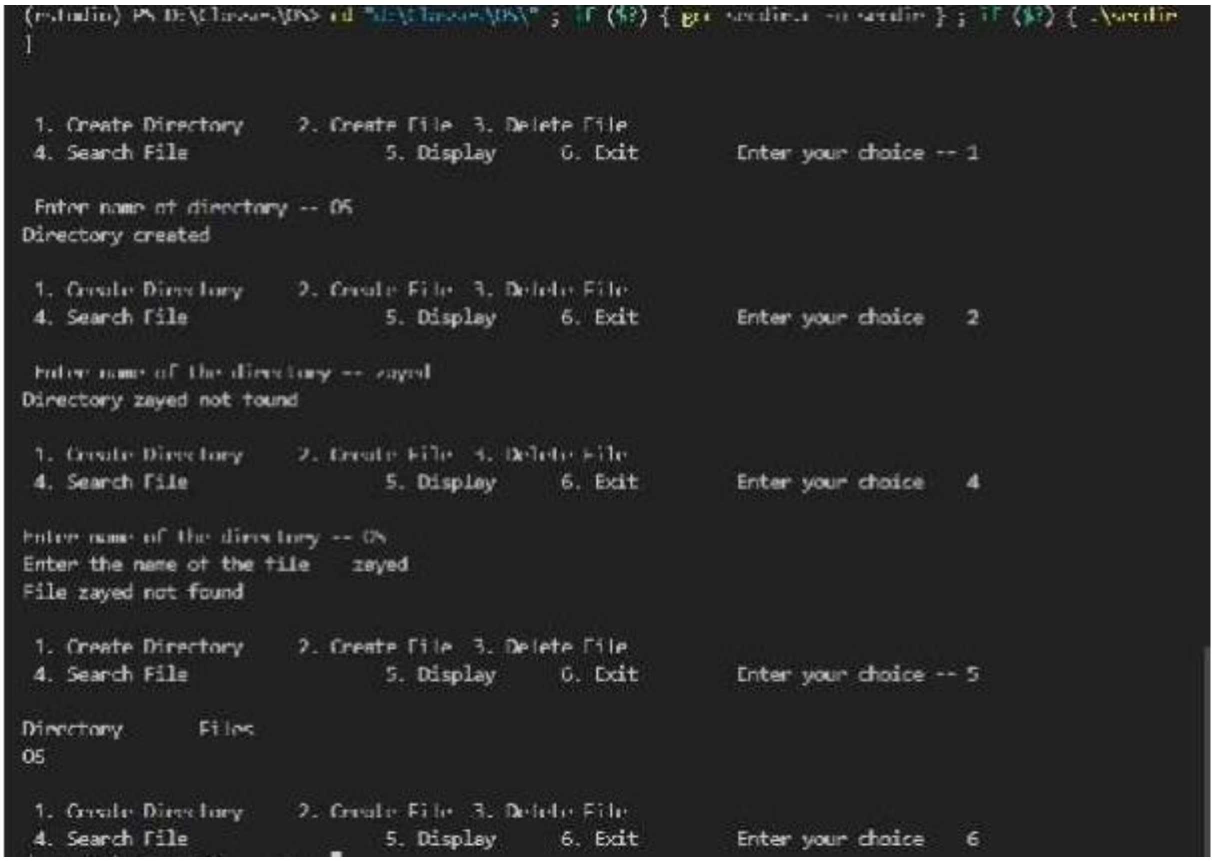
Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct
{
char dname[10],fname[10][10];
```

```
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k; char
f[30], d[30]; dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4.  Search File \t \t 5. Display \t 6. Exit \t Enter your
choice -- ");  scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
```

```
printf("File %s is deleted ",f);
 dir[i].fcnt--;
 strcpy(dir[i].fname[k],dir[i].fname[d
 ir[i].fcnt]);  goto jmp;
 }
 }
printf("File %s not found",f);
 goto jmp;
 }
 }
printf("Directory %s not found",d);
jmp : break;
 case 4: printf("\nEnter 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 -- "); scanf("%s",f);
 for(k=0;k<dir[i].fcnt;k++) {
 if(strcmp(f, dir[i].fname[k])==0)
 {
printf("File %s is found ",f);
 goto jmp1;}
 }
printf("File %s not found",f);
 goto jmp1;}}
printf("Directory %s not found",d);
jmp1: break;
 case 5: if(dcnt==0)
printf("\nNo Directory's ");
 else
 { printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
 { printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
 }
 } break;
 default:exit(0);
 }
 }
 return;}
```

Output :



```
(rstudio) PS D:\Classes\OS> cd "d:\Classes\OS\" ; if ($?) { gcc [illegible] -o [illegible] } ; if ($?) { .\[illegible]
}


 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 1

 Enter name of directory -- OS
Directory created

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice    2

 Enter name of the directory -- zayed
Directory zayed not found

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice    4

Enter name of the directory -- OS
Enter the name of the file    zayed
File zayed not found

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice -- 5

Directory       Files
OS

 1. Create Directory     2. Create File  3. Delete File
 4. Search File                  5. Display      6. Exit         Enter your choice    6
```

Result :

Successfully implemented file organization schemes for single level and two level directory.

Program (4)	Output (3)	Viva(3)	Total(10)