

21CSC202J - OPERATING SYSTEMS

LAB MANUAL



II YEAR / III Semester

Academic Year: 2024-2025 ODD SEMESTER

**FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

RAMAPURAM CHENNAI- 600 089

LIST OF EXPERIMENTS

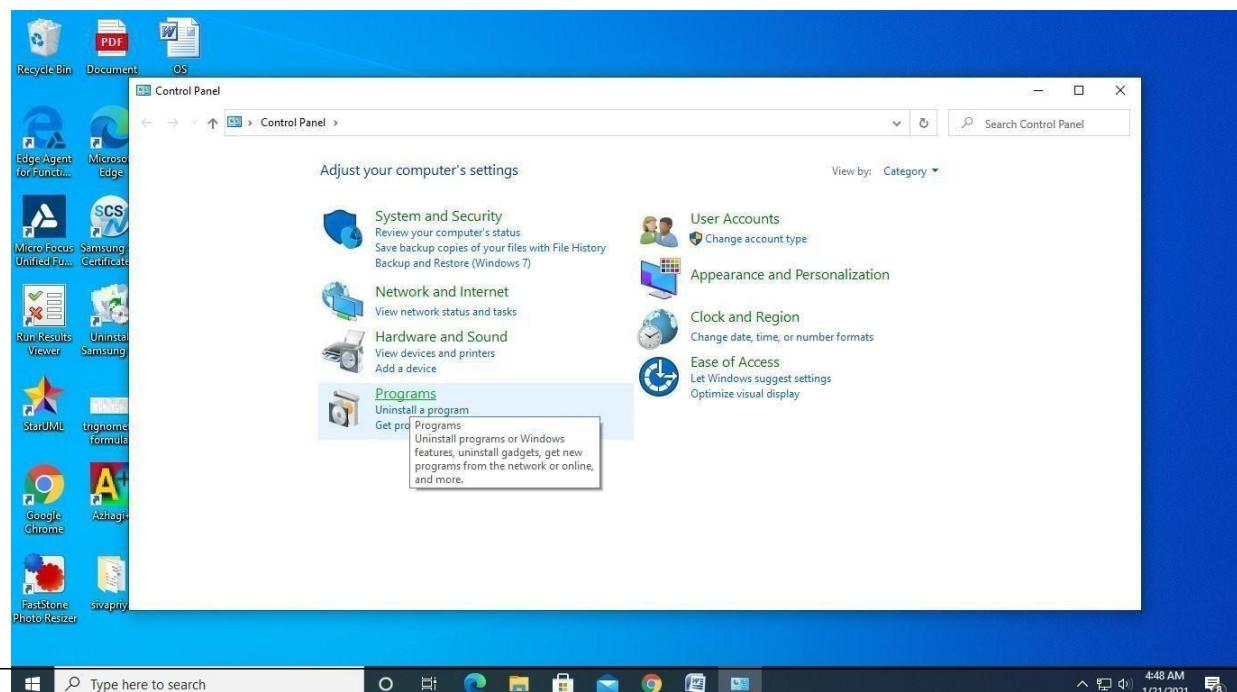
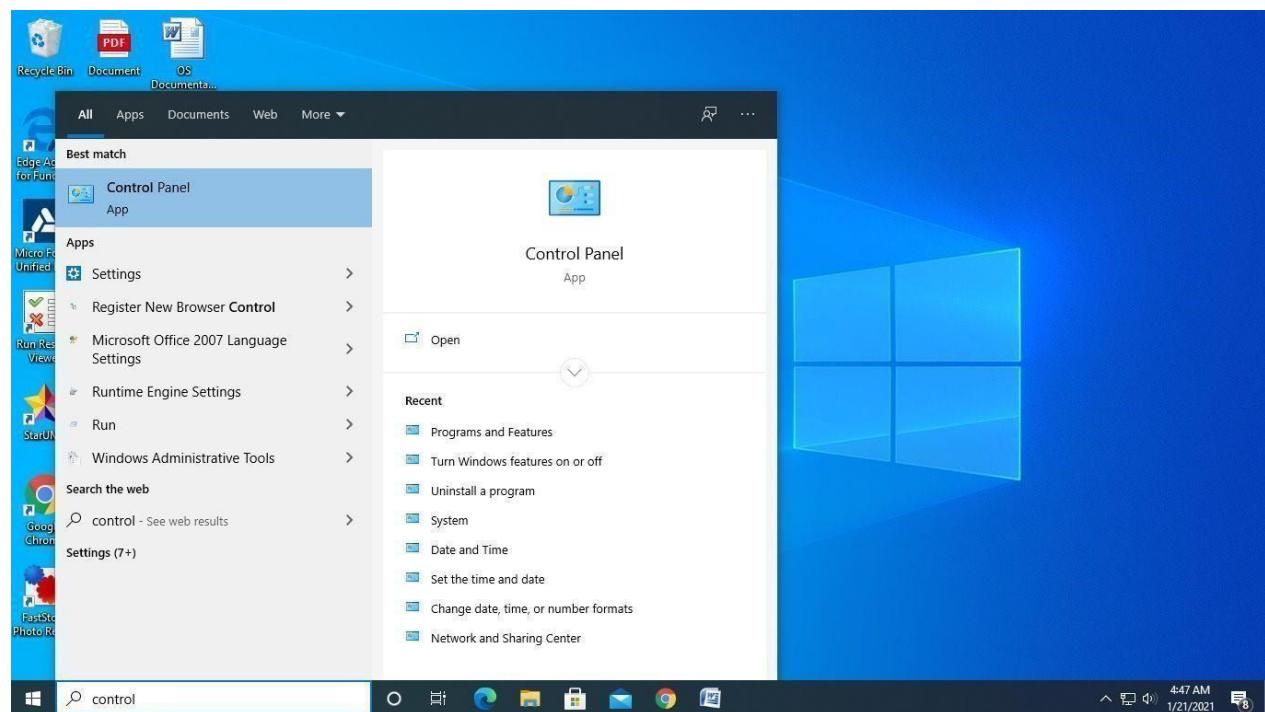
Ex.No.	Experiment Name
1	A. Operating system Installation, B. Basic Linux commands, C.Filter and admin commands
2	Process Creation using fork() and Usage of getpid(), getppid(), wait() functions
3	Multithreading
4	Mutual Exclusion using semaphore and monitor
5	Reader-Writer problem
6	Dining Philosopher 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

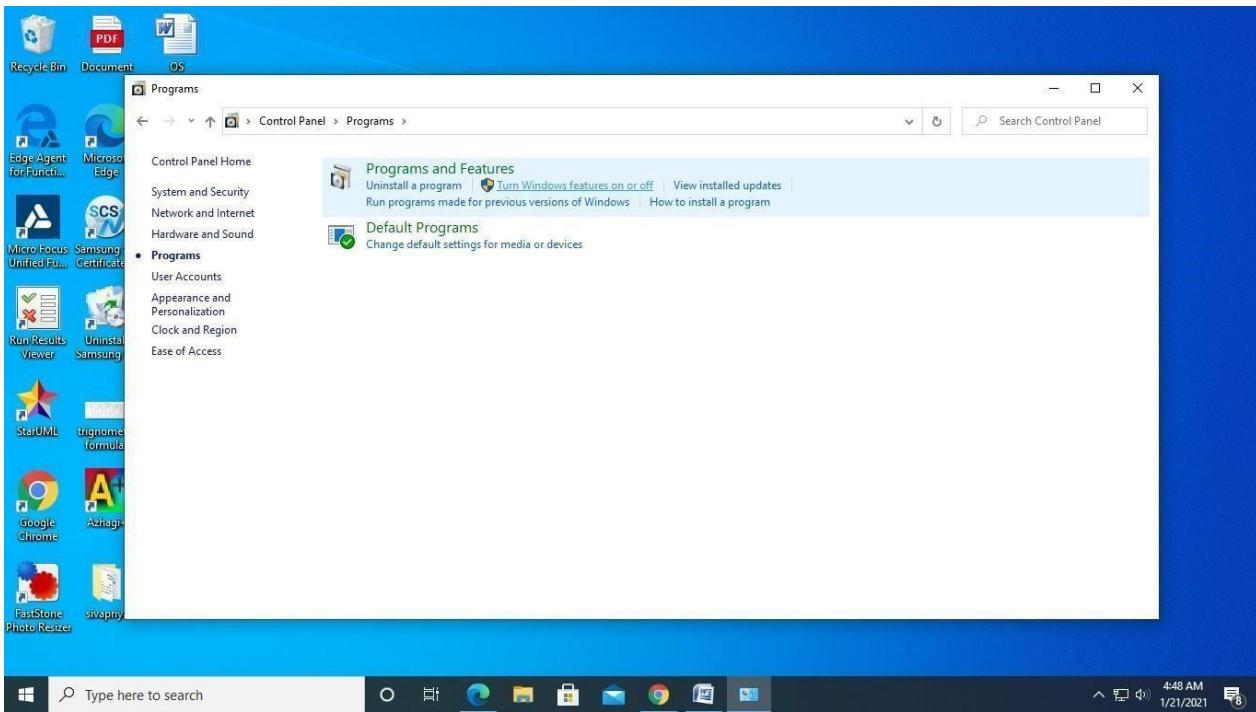
EX.No.1A

Operating System Installation

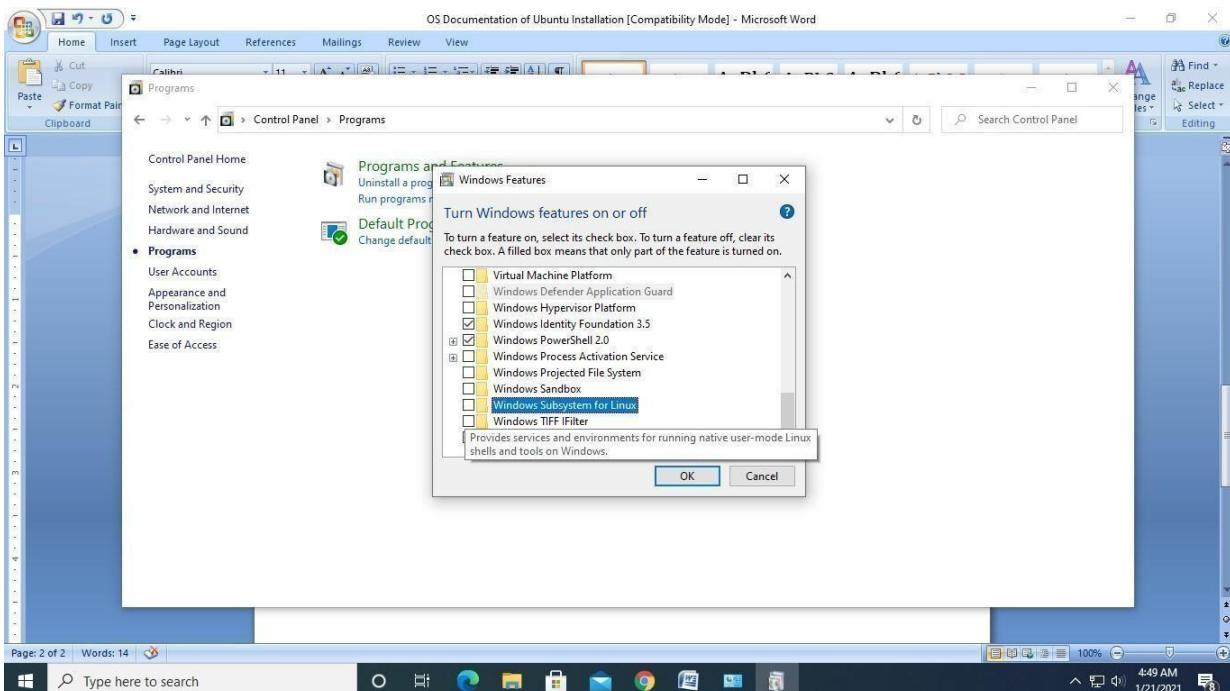
Ubuntu installation guidelines in windows

Installing Ubuntu in windows 10- 64 bit

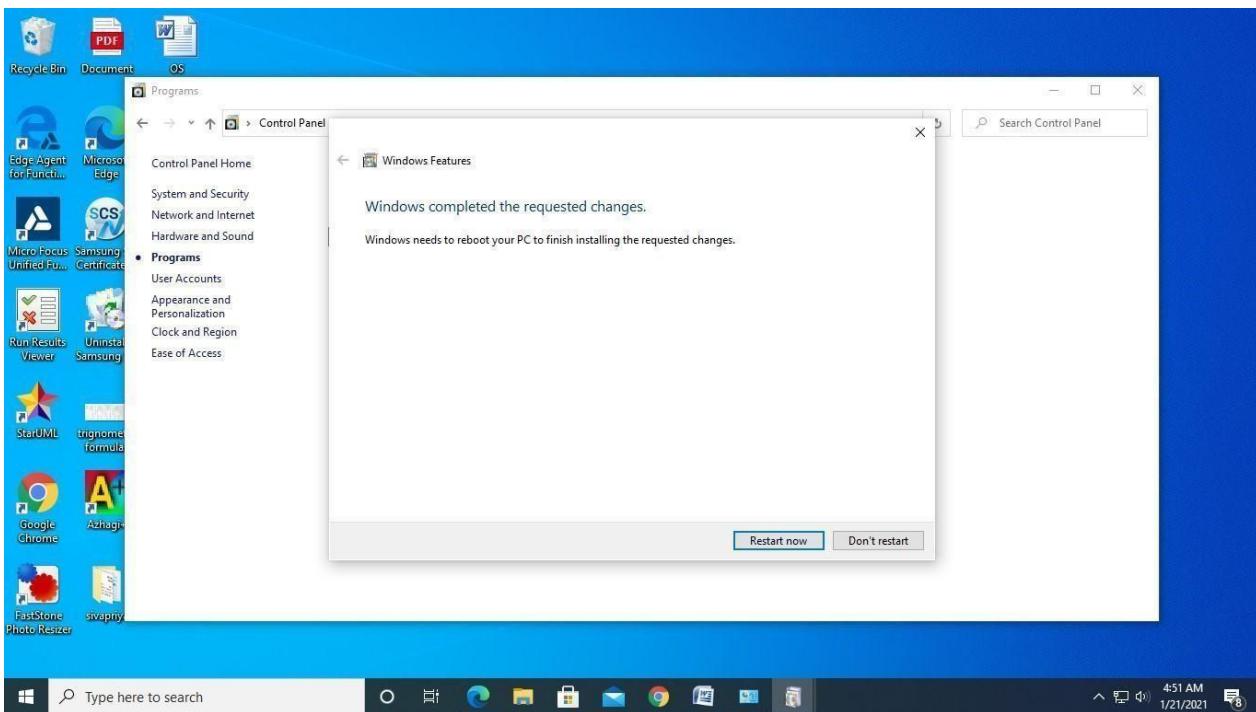




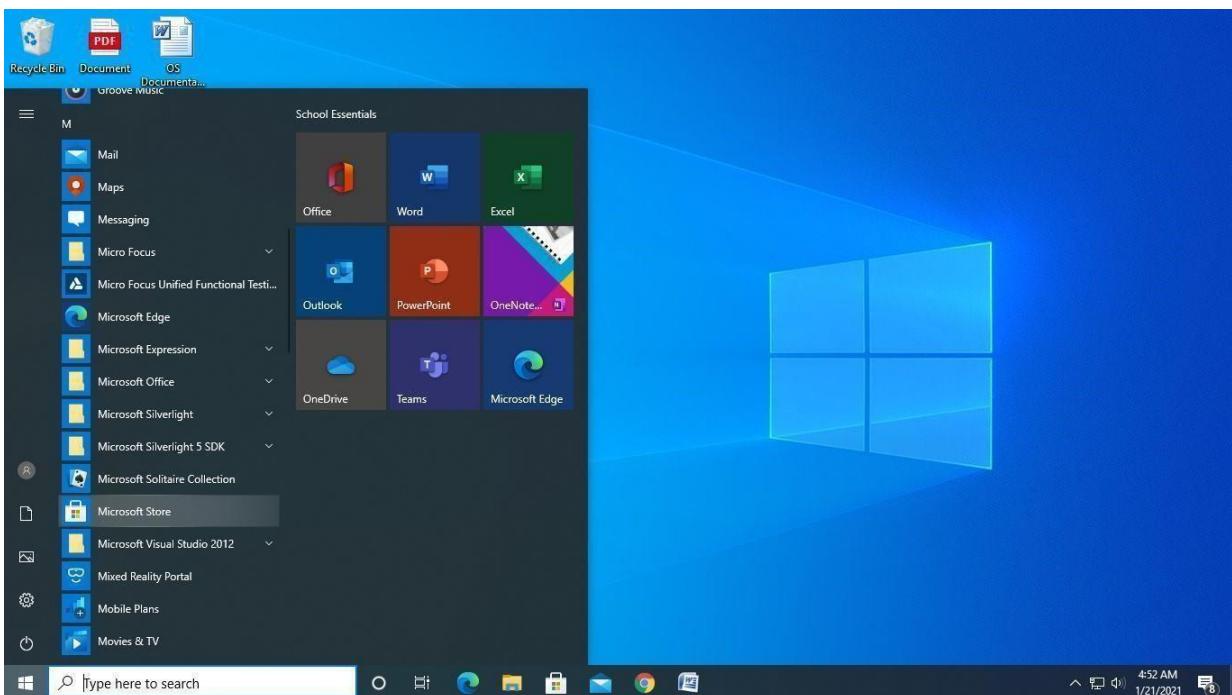
Select Turn Windows features On or Off



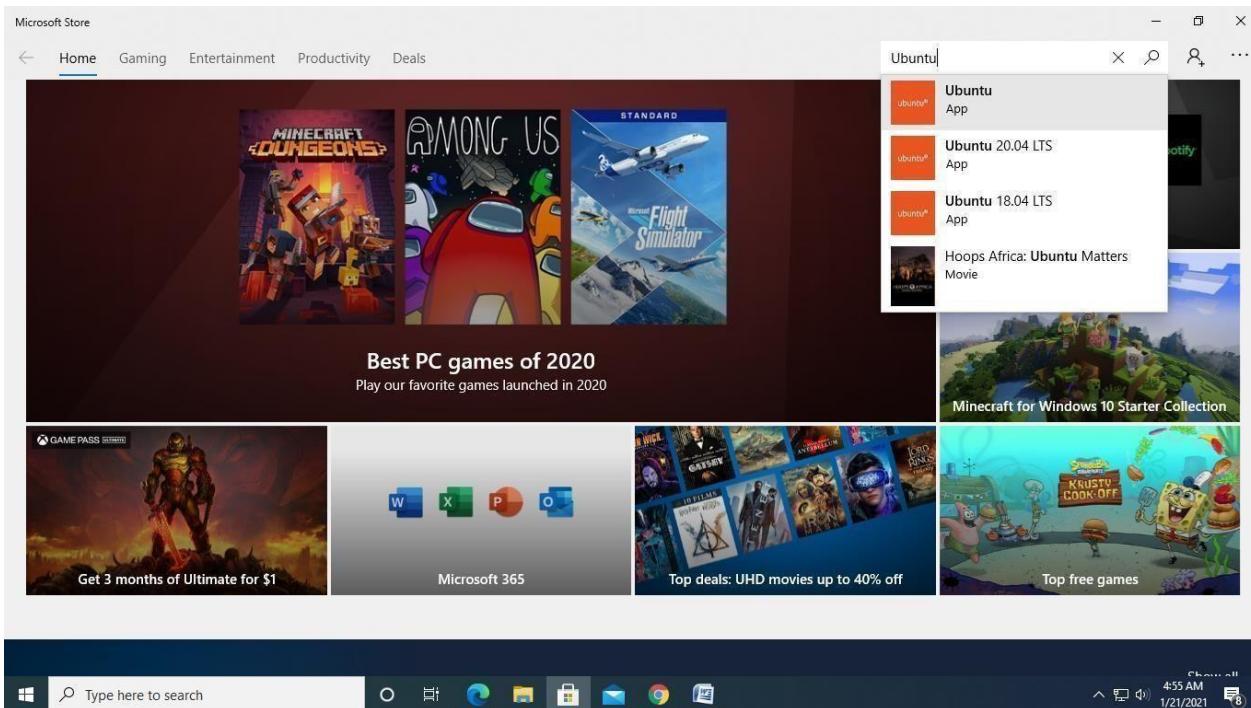
Select Windows subsystem for linux then press Ok



Now restart the PC to apply the changes



Choose Microsoft Store and search for Ubuntu and Install it



Microsoft Store

You own this app.

Ubuntu

Canonical Group Limited • Developer tools > Utilities

[Install](#) [Wish list](#)

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

More

ESRB Rating: E

OVERVIEW SYSTEM REQUIREMENTS REVIEWS RELATED

Available on

PC

Type here to search

Click on Install

Microsoft Store

Home Gaming Entertainment Productivity Deals

Downloading Ubuntu.

Get more info about faster downloads

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

ESRB

Wish list

Overview System Requirements Reviews Related

Available on

PC

Type here to search

4:55 AM 1/21/2021

Microsoft Store

Home Gaming Entertainment Productivity Deals

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

ESRB

Wish list

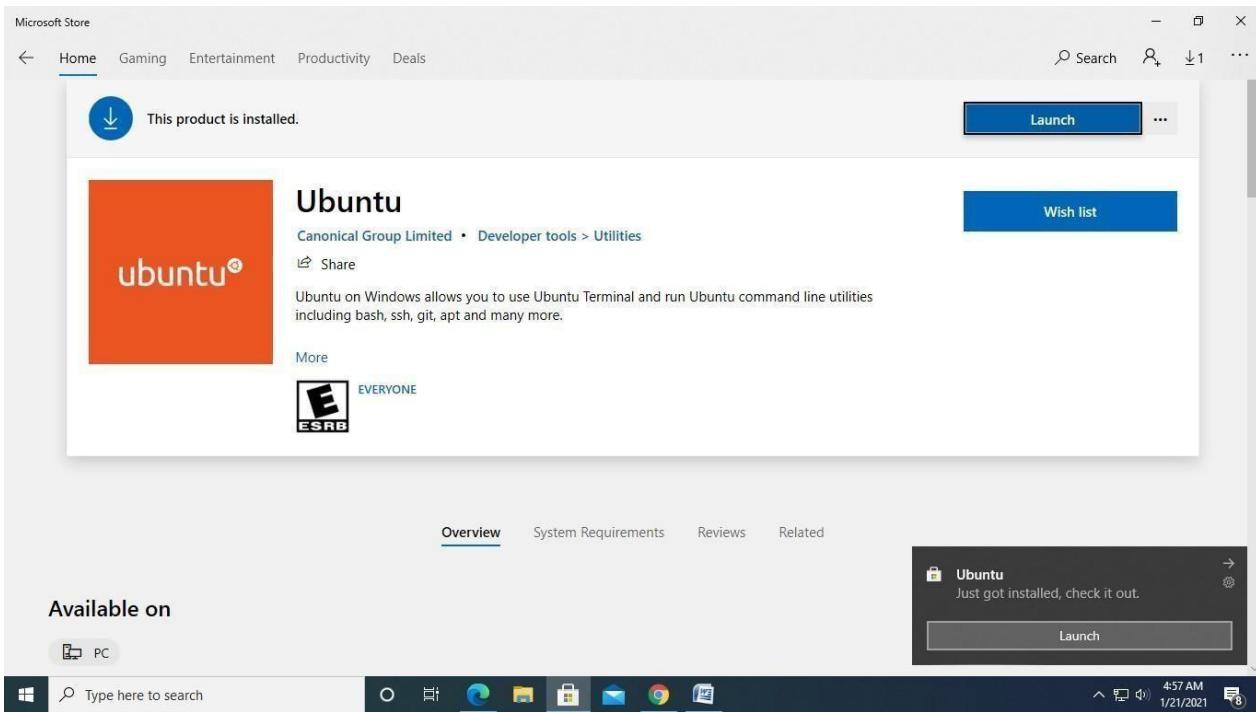
Overview System Requirements Reviews Related

Available on

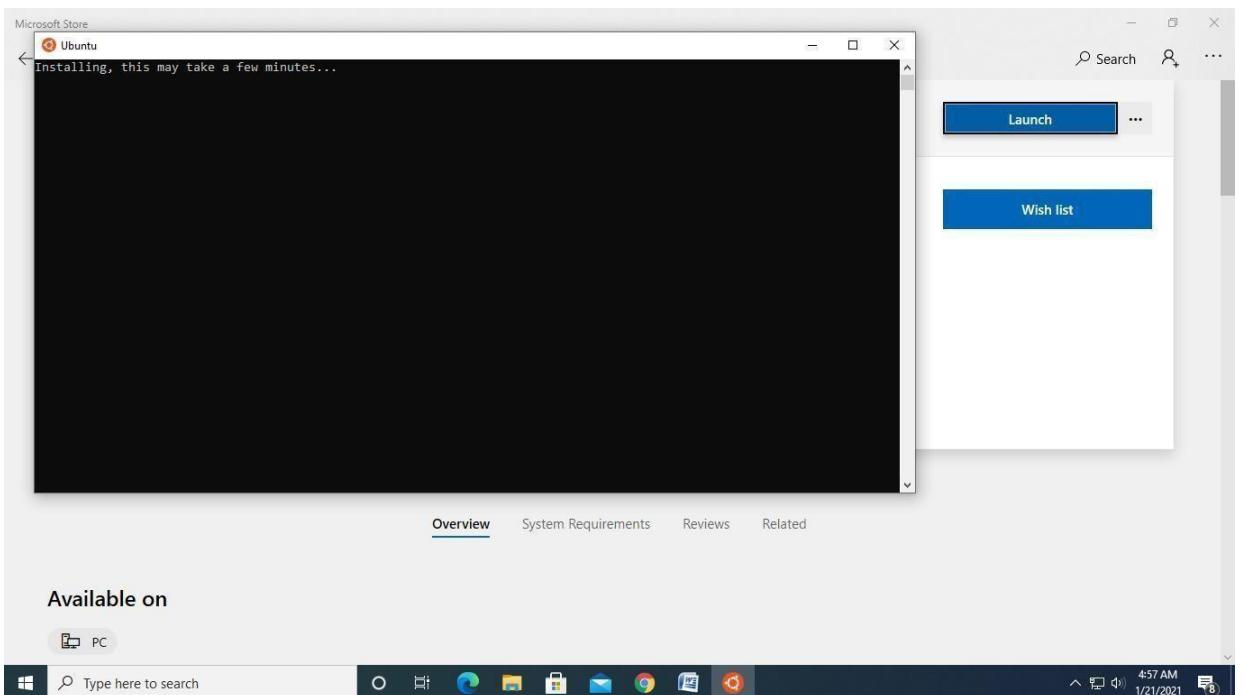
PC

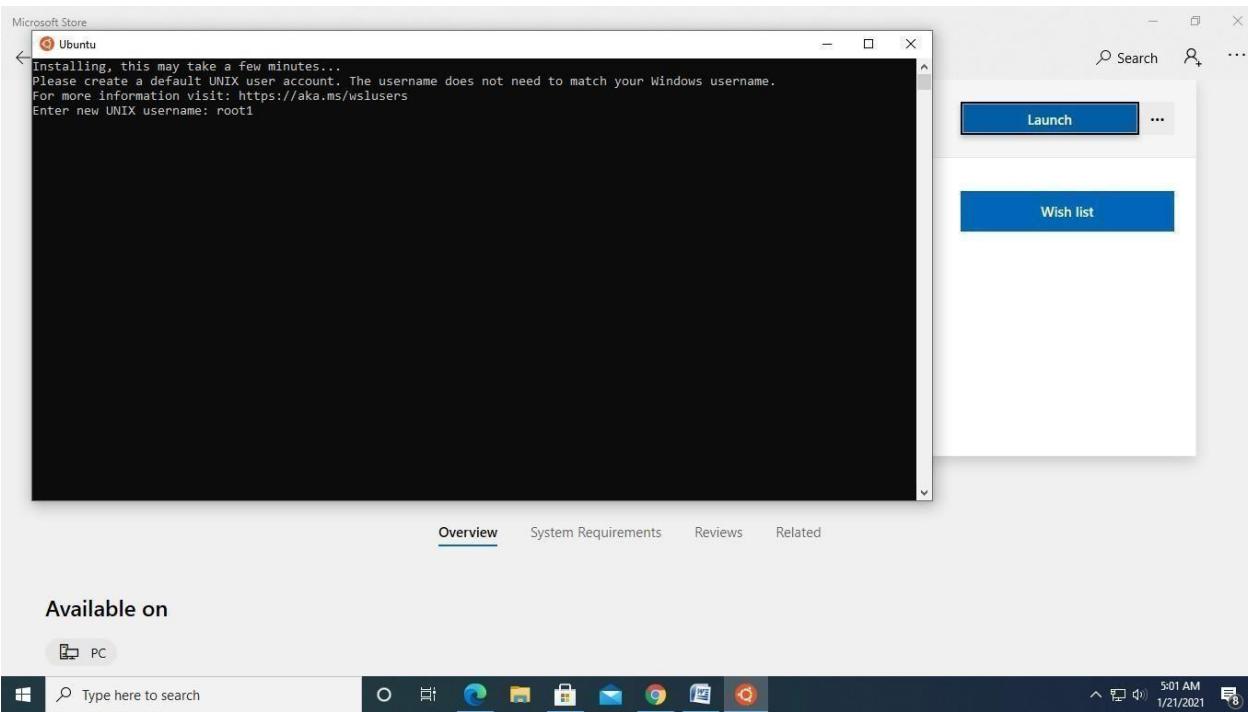
Type here to search

4:56 AM 1/21/2021



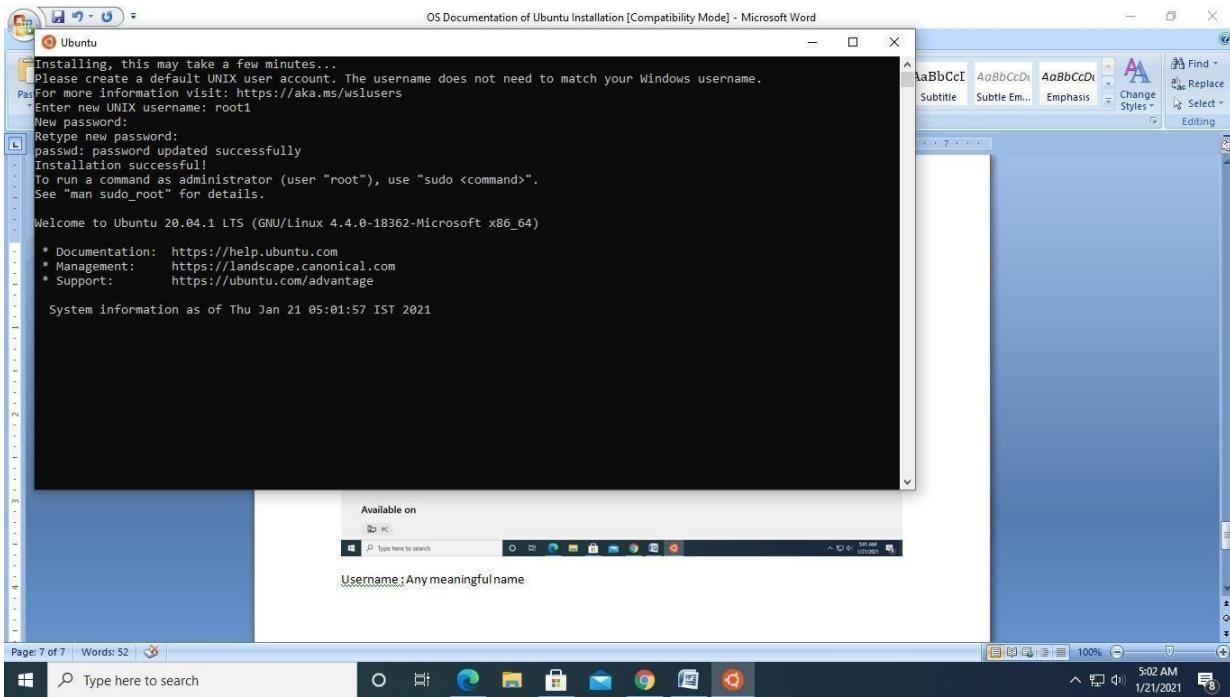
Now Launch the Ubuntu

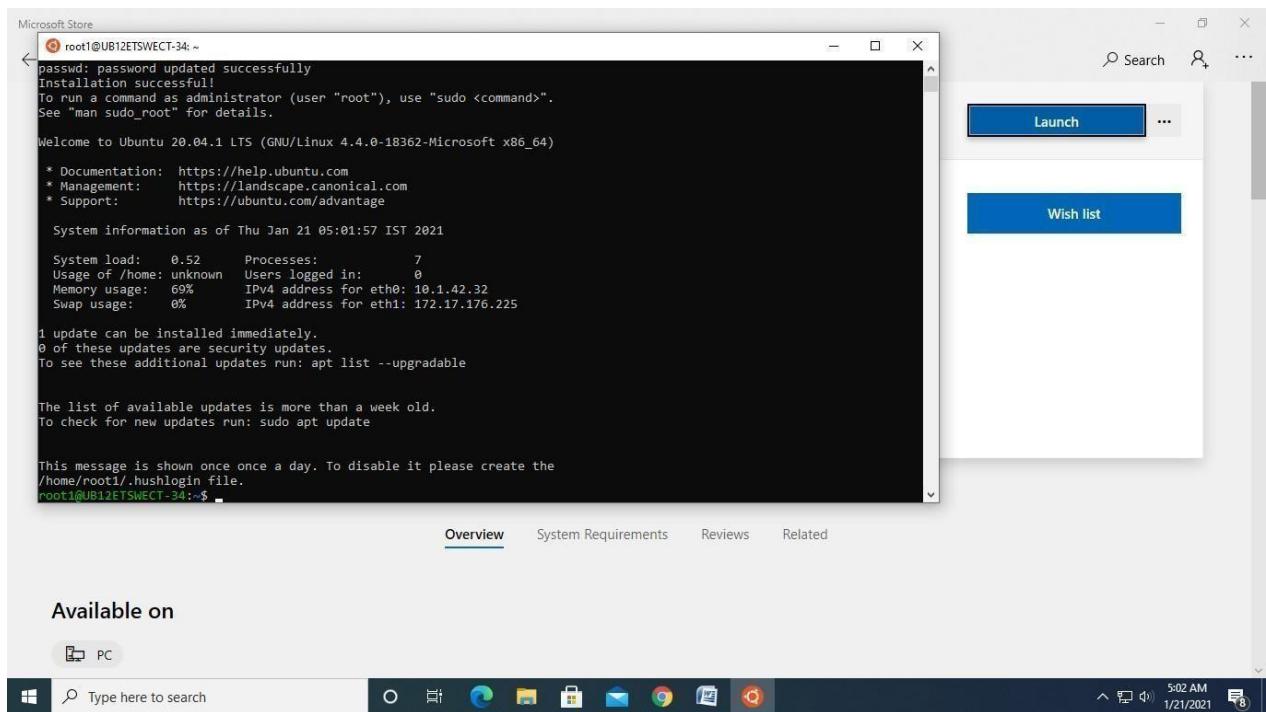




Username : Any meaningful

name Password: any name





Finally you will get the prompt on successful installation of Ubuntu in windows 10 64 bit

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

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] [+/-] [rwx] 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; $4+0+0 = 4$

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

FILTERS

1. cut

- Used to cut characters or fields from a file/input

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

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

cut **-d**delimitchar **-f**fields filename

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 field 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 fields are separated by *dchar*
-n sorts the data as number
+1n skip first field and sort the file by second field 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.

Q1. Write a command to cut 5 to 8 characters of the file *f1*.

\$

Q2. Write a command to display user-id of all the users in your system.

\$

Q3. Write a command to check whether the user *judith* is available in your system or not.
(use grep)

\$

Q4. Write a command to display the lines of the file *f1* starts with SRM.

\$

Q5. Write a command to sort the file /etc/passwd in descending order

\$

Q6. Write a command to display the unique lines of the sorted file *f21*. Also display the number of occurrences of each line.

\$

Q7. Write a command to display the lines that are common to the files *f1* and *f2*.

\$

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

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

Aim :

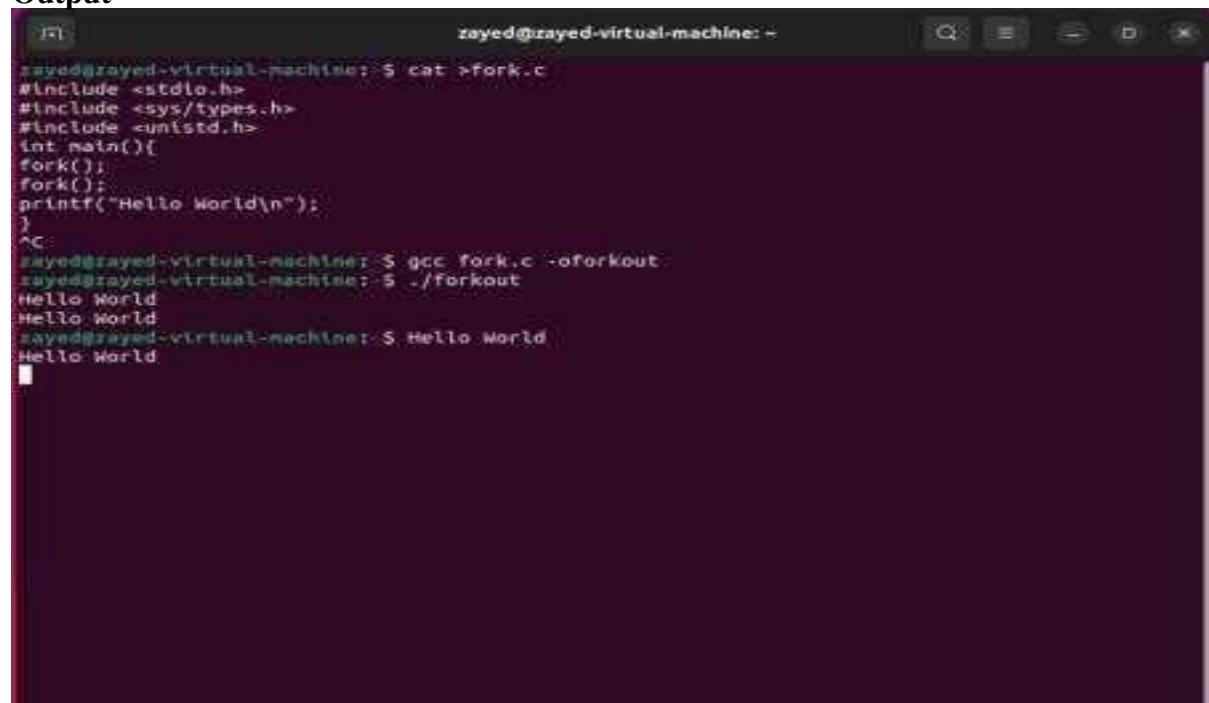
To write a program for process Creation using fork() and usage of getpid(), getppid(), wait() function.

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 -o forkout
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 :



The terminal window shows the following session:

```

zayed@zayed-virtual-machine: ~
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 &gt;
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:



```
zayed@zayed-virtual-machine: ~ cat >work.c
#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;
}
^C
zayed@zayed-virtual-machine: $ gcc work.c -o workout
zayed@zayed-virtual-machine: $ ./workout
Parent pid = 3310
Child pid = 3311
zayed@zayed-virtual-machine: $
```

Result :

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

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

EX.NO. 3

Multithreading and pthread in C

Aim :

To implement and study Multithreading and pthread in C

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:~$
```

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

Result :

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

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 conditional 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 implemented as it was not required because the
    // philosophers are implemented using threads and not processes.
```

```
if(pshared == 1){  
    printf("Cannot handle semaphores shared between processes!!! Exiting\n");  
    return -1;  
}  
  
// Initialising the semaphore conditonal variable  
pthread_cond_init(&sema->conditional_variable, NULL);  
  
// Initialising the semaphore count variable  
sema->cnt = val;  
  
// Initialising 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 condition
    // 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 philosopher 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 philosopher so
        // that the philosopher can access these forks whenever they are free

        wait(&bowls);
}

```

```

    // This puts a wait condition on the forks to be used by the current philosopher so
    // that the philosopher can access these forks whenever they are free

    wait(&forks[*i]);
    wait(&forks[(*i+4)%5]);
    sleep(1);

    // Print the philosopher 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 philosopher threads that the bowls are available for
    // eating signal(&bowls);

    // This signals the other philosopher 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 philosopher threads

    pthread_t phil[5];
    i = 0;

    // Creating the philosopher 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(phi[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

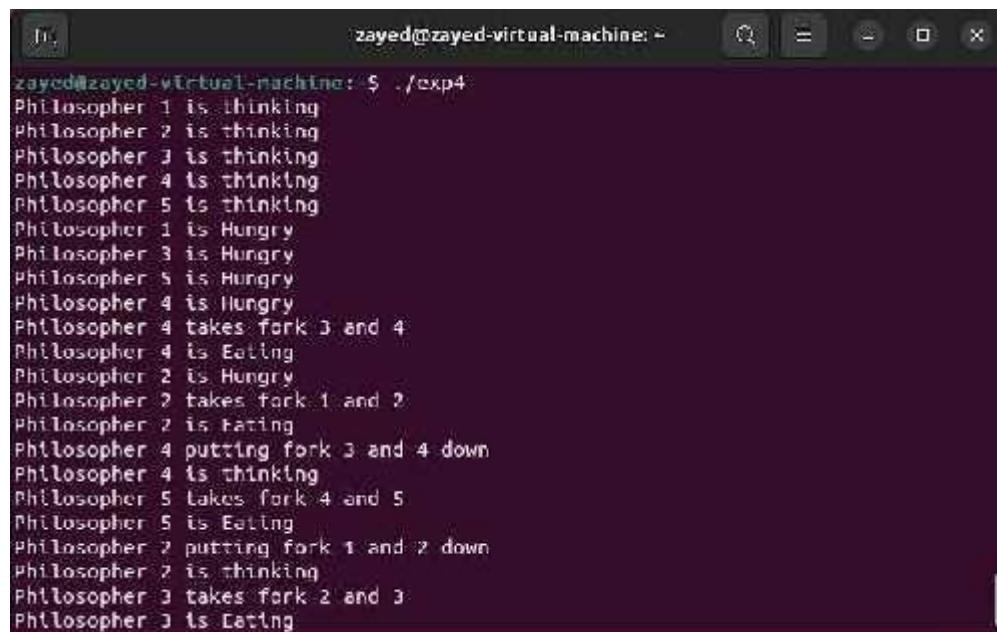
```

```

// 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

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

EX.NO. 5

Reader-Writer problem

Aim :

To study the Reader – Writer problem

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);

    }

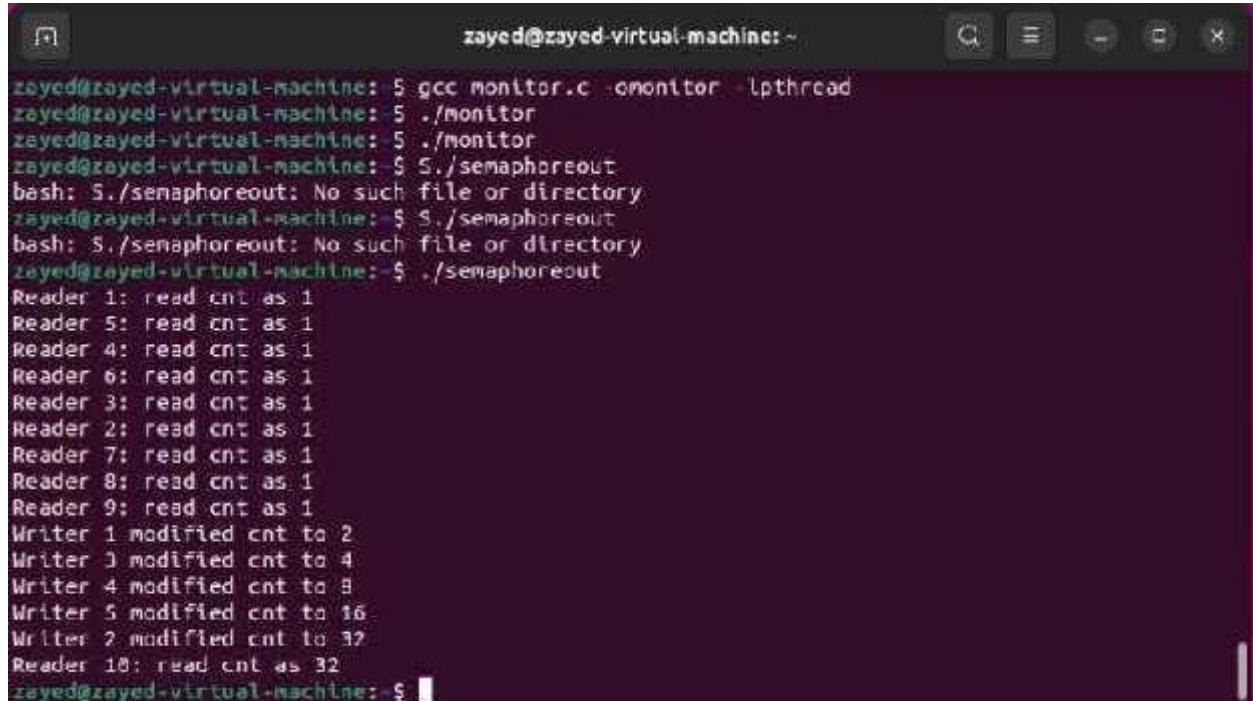
    pthread_mutex_destroy(&mutex);

    sem_destroy(&wrt);

    return 0;
}

```

Output:



```
zayed@zayed-virtual-machine:~$ gcc monitor.c -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.

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

EX:NO:6 Dining Philosopher's Problem

Aim:

To implement and study Dining Philosopher's Problem

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(phiros_thread_ids[i], NULL);
    }

    //Create threads for the philosophers.
    for(i = 0; i < PHILOSOPHER_NUM; i++)
    {
        pthread_create(&phiros_thread_ids[i], NULL, pickup_forks, (void *)&i);
    }

    sleep(run_time);

    for(i = 0; i < PHILOSOPHER_NUM; i++)
    {
        pthread_cancel(phiros_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("Philosopher %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("Philosopher %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
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 5 is thinking.
Philosopher 5 is thinking.
Philosopher 5 is thinking.
Philosopher 4 is eating meal 1.
Philosopher 4 is thinking.
Philosopher 2 is eating meal 1.
Philosopher 4 is eating meal 2.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is eating meal 3.
Philosopher 2 is eating meal 2.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 2 is eating meal 3.
Philosopher 4 is eating meal 10.
Philosopher 0: 6 meals
Philosopher 1: 6 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.

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

EX:NO:7 Bankers Algorithm for Deadlock avoidance

Aim :

To implement and study Bankers Algorithm for Deadlock Avoidance Problem

Program :

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("Allocation 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[1][1] = 2
Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4
Enter instances for max[2][1] = 5
Enter instances for max[2][2] = 6
Enter instances for max[3][0] = 76
Enter instances for max[3][1] = 8

```

```
zayedhaque@fedora:~
```

```
Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4
Enter instances for max[2][1] = 5
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
Allocation 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
```

```
^C
[zayedhaque@fedora ~]$
```

Result :

Successfully implemented the concepts of Banker's Algorithm.

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

EX: NO:8 FCFS and SJF Scheduling

Aim :

To study the concepts of FCFS and SJF Scheduling

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%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 fcfs0
[zayedhaque@fedora ~]$ ./fcfs0
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 allotting 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 :

```

zayedhaque@fedora:-
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.

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

EX: NO:9 Priority and Round Robin Scheduling

Aim :

To study the concepts of Priority and Round Robin Scheduling

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,pos,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\tBurst Time\tWaiting Time\tTurn Around Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
total+=tat[i];
printf("\n P%d\t%d\t%d\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 :

```

zayedhaque@fedora:~$ gcc sem.c -o semo
zayedhaque@fedora:~$ ./semo
Enter Total Number of Process:3

Enter Burst Time and Priority For
Enter Process 1: 1
2

Enter Process 2: 2
2

Enter Process 3: 3
3

Process      Burst Time      Waiting Time      Turn Around Time
P1            1                0                  1
P2            2                1                  3
P3            3                3                  6

Average Waiting Time = 1.00
Avg Turn Around Time = 3.00
[zayedhaque@fedora ~]$ 

```

Round Robin Scheduling

```
#include<stdio.h>

int main()
{
    //Input no of processes
    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);

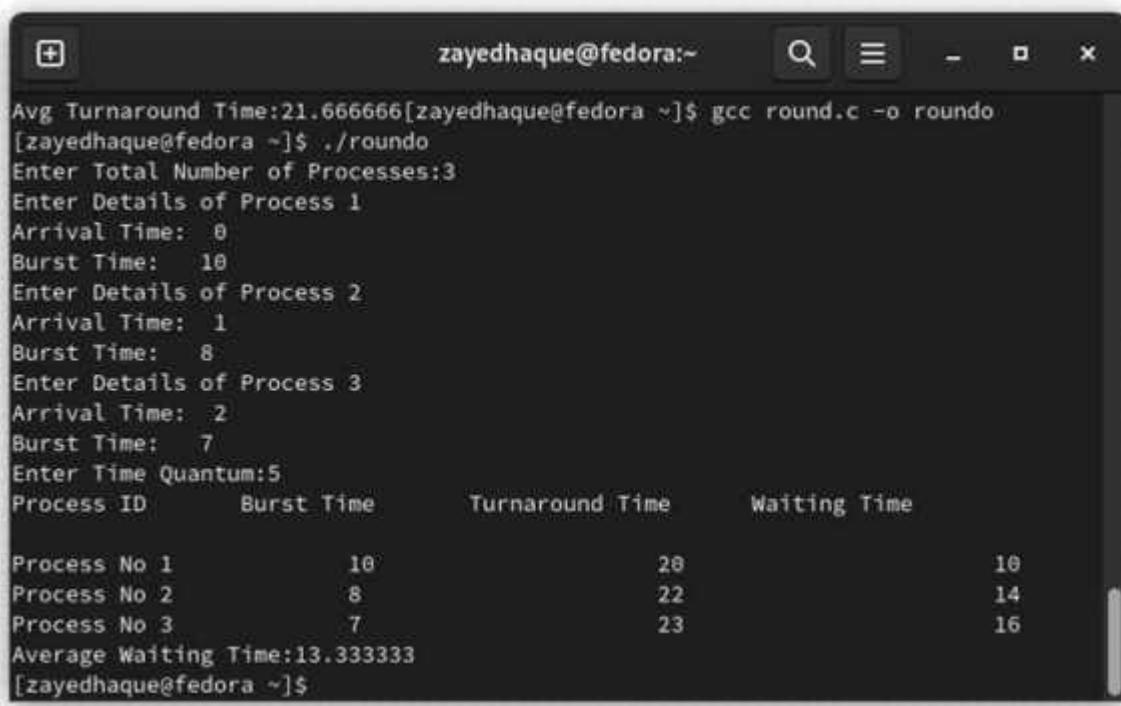
    //Total indicates total time
    //counter indicates which process is executed
    int total = 0, counter = 0,i;
    printf("Process ID      Burst Time      Turnaround Time      Waiting Time\n");
    for(total=0, i = 0; x!=0; )
    {
        // define the conditions
        if(temp_burst_time[i] <= time_slot && 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 %d \t %d \t %d \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 :



```
Avg Turnaround Time:21.666666[zayedhaque@fedora ~]$ gcc round.c -o roundo
[zayedhaque@fedora ~]$ ./roundo
Enter Total Number of Processes:3
Enter Details of Process 1
Arrival Time: 0
Burst Time: 10
Enter Details of Process 2
Arrival Time: 1
Burst Time: 8
Enter Details of Process 3
Arrival Time: 2
Burst Time: 7
Enter Time Quantum:5
Process ID      Burst Time      Turnaround Time      Waiting Time
Process No 1          10            28                  10
Process No 2          8             22                  14
Process No 3          7             23                  16
Average Waiting Time:13.333333
[zayedhaque@fedora ~]$
```

Result :

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

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

EX: NO:10 FIFO Page Replacement Algorithm

Aim :

To implement FIFO page replacement algorithm

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
            enqueue 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]=temp
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 :

```
sayed@sayed-virtual-machine: ~/Desktop/OS/5XP 10
```

```
Page Fault = 5  
Hit = 8  
sayed@sayed-virtual-machine:~/Desktop/OS/5XP 10$ ./L2FIFO
```

```
Enter the no. of pages : 15  
Enter the size of frame : 3  
Enter the page's value :  
3  
8  
2  
3  
9  
1  
6  
3  
8  
9  
3  
2  
1  
3  
Current Frame: -1 0 1  
Current Frame: -1 0 3  
Current Frame: 0 0 3  
Current Frame: 0 0 3  
CURRENT FRAME: 0 0 1  
CURRENT FRAME: 0 0 3  
CURRENT FRAME: 0 0 2  
Current Frame: 0 0 2  
Current Frame: 1 0 3  
Current Frame: 0 1 3  
Current Frame: 0 1 3  
Current Frame: 0 1 3  
Frame at the end :  
b[0] = 1  
b[1] = 6  
b[2] = 9  
b[3] = 3  
b[4] = 3  
Page Fault = 10  
Hit = 8
```

Result :

Successfully implemented page replacement using FIFO algorithm

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

EX: NO:11 LRU AND LFU Page Replacement Algorithm

Aim :

To implement page replacement using LRU and LFU

Program :

LRU

```
print("Enter the number of frames: ",end="")
capacity = int(input())
f,st,fault,pf = [],[],0,'No'
print("Enter the reference string: ",end="") s =
list(map(int,input().strip().split()))
print("\nString/Frame →\t",end="")
for i in range(capacity):
    print(i,end=' ')
print("Fault\n ↓\n")
for i in s:
    if i not in f:
        if len(f) < capacity:
            f.append(i)
            st.append(len(f)-1)
        else:
            ind = st.pop(0)
            f[ind] = i
            st.append(ind)
        pf = 'Yes' fault
        += 1
    else:
        st.append(st.pop(st.index(f.index(i))))
        pf = 'No'
    print(" %d\t%"i,end="")
    for x in f:
        print(x,end=' ')
    for x in range(capacity-len(f)):
        print(' ',end=' ')
    print("%s"%pf)
print("\nTotal Requests: %d\nPage Faults: %d"%(len(s),fault))
print("Page Hit: ",len(st))
```

Output :

```
PS D:\Classes\OS> python -u "d:\Classes\OS\lru.py"
Enter the number of frames: 5
Enter the reference string: 3 8 2 3 9 1 6 3 8 9 7 6 2 1 3

String|Frame + 8 1 2 3 4 Fault
+
3      7      Yes
8      7 8     Yes
2      7 8 2    Yes
3      7 8 2    No
9      7 8 2 9   Yes
1      7 8 2 9 1  Yes
6      7 6 2 9 1  Yes
3      7 6 2 9 1  No
8      7 6 8 9 1  Yes
9      7 6 8 9 1  No

String|Frame + 8 1 2 3 4 Fault
+
3      7      Yes
8      7 8     Yes
2      7 8 2    Yes
3      7 8 2    No
9      7 8 2 9   Yes
1      7 8 2 9 1  Yes
6      7 6 2 9 1  Yes
3      7 6 2 9 1  No
8      7 6 8 9 1  Yes
9      7 6 8 9 1  No
3      7 6 8 9 1  No
6      7 6 8 9 1  No
2      7 6 8 9 2  Yes
1      7 6 1 9 2  Yes
3      7 6 1 9 2  No

Total Requests: 15
Page Faults: 9
Page Hits: 5
```

LFU

```
print("Enter the number of frames: ",end="")
capacity = int(input())
f,st,fault,pf = [],[],0,'No'
print("Enter the reference string: ",end="")
s = list(map(int,input().strip().split()))
print("\nString/Frame →\t",end="")
for i in range(capacity):
    print(i,end=' ')
print("Fault\n↓\n")
for i in s:
    if i not in f:
        if len(f)<capacity:
            f.append(i)
            st.append(len(f)-1)
        else:
            ind = st.pop(0)
            f[ind] = i
    else:
        pf += 1
print("Page Faults: ",pf)
print("Page Hits: ",len(s)-pf)
```

```

    st.append(ind)
    pf = 'Yes'
    fault += 1
else:
    st.append(st.pop(st.index(f.index(i))))
    pf = 'No'
    print(" %d\t%6i,end=")
    for x in f:
        print(x,end=' ')
    for x in range(capacity-len(f)):
        print(' ',end=' ')
    print(" %s"%pf)
print("\nTotal Requests: %d\nPage Faults: %d"%(len(s),fault))
print("Page Hit: ",len(st))

```

Output

```

String|Frame →  0 1 2 3 4 Fault
↓

```

3	3	Yes
8	3 8	Yes
2	3 8 2	Yes
3	3 8 2	No
9	3 8 2 9	Yes
1	3 8 2 9 1	Yes
6	3 6 2 9 1	Yes
3	3 6 2 9 1	No
8	3 6 8 9 1	Yes
9	3 6 8 9 1	No
3	3 6 8 9 1	No
6	3 6 8 9 1	No
2	3 6 8 9 2	Yes
1	3 6 1 9 2	Yes
3	3 6 1 9 2	No

Total Requests: 15

Page Faults: 9

Page Hit: 5

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.

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 :

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

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\05> python -u "d:\Classes\05\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\05>
```

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.

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\disk.py
DISK SCHEDULING:
Provide number of T/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
    48  seeked
    45  seeked
    27  seeked
    24  seeked
    4  seeked
126
Avg seek time for look was 11.45454545454545
***** C-LOOK *****

    58  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

Sequential File Allocation

Algorithm :

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.

Step 8: 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 bl. block: 8
file allocation end bl. 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: 1

Enter the name of file #3: d
Enter the size (kB) of file #3: 1

FILE_Filename   FILE_SIZE      BLOCKS_OCCUPIED
a              2          20-21
b              1          29-30-31
d              1          25-26-27-28
File allocation completed. [Exiting.
(rstudio) 15 U:\Classes\OS> []

```

Indexed File Allocation

Algorithm :

Step 1: Start.

Step 2: Let n be the size of the buffer

Step 3: check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the buffer

Step 6: If the buffer is full the producer has to wait

Step 7: Check there is any consumer.

Step 8: If no the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait

Step 9: If the buffer is empty, the consumer has to wait.
Step 10: Repeat checking for the producer and consumer.

Step 11: Terminate the process.

Step 11. Terminate the process

Program :

Program :

```

}
}
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 ($?) | g++ indexed.cpp -o indexed | ; if ($?) | ./indexed |
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 Today - 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

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");
    printf("+-----+");
    printf("\n Directory files \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");
}

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:Exit\n>>");
        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 :

```
(restudio) PS D:\Classes\OS> cd "D:\Classes\OS\" ; IF ($?) { gcm -file .\osdir.ps1 ; IF ($?) { .\osdir } }  
+-----+  
Directory:files  
+-----+  
OS  
ttzayed  
+-----+  
  
  
choose the option  
1:Insert a file  
2:Display Files  
3:Delete File  
4:Search File  
5:Exit(0)>4  
  
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(0)>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(0)>5  
(restudio) PS D:\Classes\OS> █
```

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("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: 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",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[dir[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(dcmt==0)
printf("\nNo Directory's ");
else
{ printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{ printf("\n%5s\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%5s",dir[i].fname[k]);
}
} break;
default:exit(0);
}
}
return;}
```

Output

```
(root@root:~) PS D:\Classes\ADS> cd "D:\Classes\ADS\" ; if ($?) { gci -ecdir, -ocdir } ; if ($?) { .\verdir }
```

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

Enter name of directory -- 05
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 -- 05
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
05

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)