SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203

COLLEGE OF ENGINEERING & TECHNOLOGY

SCHOOL OF COMPUTING

DEPARTMENT OF DATASCIENCE AND BUSINESS SYSTEMS

LAB REPORT

Course Code: **21CSC202J**

Course Name: **OPERATING SYSTEMS**

Name :	GODFREY ASHWANTH
Reg.no :	RA2112704010020
Class :	MTech [Integrated] – Computer Science and Engineering
Year /Semester:	II / III
Specialization:	Data Science

November 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this is a bonafide record of practical work done by _____, Reg.No. _____ pursuing Second year M.Tech (Integrated), Computer Science and Engineering with _____ Specialization in the Subject - **Operating Systems (21CSC202J)** during the End Semester University Examinations , November 2022.

Faculty in-charge

Dr.K.Shantha Kumari

Head of the Department

Dr.M.Lakshmi

Dept. of DSBS

Submitted for the University Examination held on _____

Internal Examiner-1

Internal Examiner-2

Table of Contents

Exp.No:01 Name:Godfrey Ashwanth

Date:16/08/22 RegNo:RA2112704010020

INSTALLATION OF VARIOUS OS & BASIC UNIX SHELL COMMANDS

Aim:

1. A) To Study about the various features of Operating System.

    1. Installation of windows
    2. Installation of Linux
    3. Installation of dual OS
    4. Installation of Linux in windows as virtual machine

1. B) To study about Linux shell commands

Description of the Exercise:

Operating system is a software that acts as an intermediate between computer hardware and computer user.

Unix shell commands are one of the 4 layers of unix architecture enabling human interaction with the os to intimate it to begin certain processes by giving commands through interpreter.

Procedure:

1.To install Linux:

Info collected from https://techcommunity.microsoft.com

Install using Command Prompt

Step 1:
Start **CMD** with administrative privileges.
Step 2:
Execute "wsl --install" command.
Step 3:
Run "wsl -l -o" to list other Linux releases.
Step 4:
Then install your favorite Linux distribution, use "wsl --install -d NameofLinuxDistro."

Install Using Windows Features

Step 1:
Open the Start menu and type "Windows features" into the search bar and click on "Turn Windows Features On or Off".
Step 2:
Tick the "Windows Subsystem for Linux" checkbox and press the “OK” button.
Step 3:
When the operation is complete, it will be asked to restart your computer.
Step 4:
The Linux distribution can be launched from the Start menu.

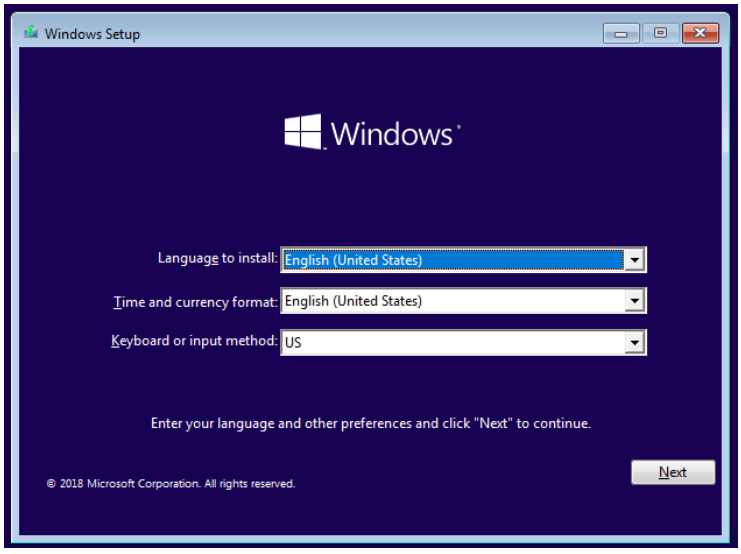
2.Installation of dual OS:

Info collected from https://opensource.com/article/18/5/dual-boot-linux

Step 1. Create a partition to dual boot Ubuntu

To install Ubuntu along with Windows 11 in dual boot, first you need to create a partition for Ubuntu. This is how you can do it:

- Open Disk Management by using Windows + X and selecting Disk Management.
- Next, select a drive, right-click on it and choose Shrink Volume.
- Then, Set the *Enter the amount of space to shrink* 50000MB or more and click Shrink.
- Now, click the space that you just created and choose New Simple Volume.
- Set the *Simple volume size in MB* to the maximum value and click Next.
- Finally, use a drive letter and label and click Finish.



Step 2. Create a bootable USB drive for Ubuntu on Windows 11

To dual boot Windows 11 and Ubuntu, you need a bootable USB drive, and to create a bootable USB flash drive, we use an application by the name of Rufus.

Step 4. Install Ubuntu along with Windows 11 in dual boot

After completing the above three steps, it is time to install Ubuntu along with your Windows 11 system. Ubuntu will be installed on the new partition that you created.

- First, unplug the bootable USB drive, plug it back in and restart your PC.
- After booting, the system will show you with options to use the device. Select the inserted USB drive by pressing Enter.
- Now, in the new window that opens, select **Install Ubuntu.**
- Select a language and a keyboard layout for your Ubuntu.

4.To install VMware:

Info collected from https://www.educba.com/install-vmware/

Step 1. To download and install the VMware product visit the official website of VMware. https://www.vmware.com/in.html.Hover on the Downloads tab, here you will find various products.

Step 2. Click on Free Product Trials & Demo >> Workstation Pro. It will be redirected to the download page. (Similarly, then select any product which you want to install.)

Step 3. Once the download is complete, run the .exe to install VMware Workstation. Popup will appear.Once Initialization gets completes, Click on Next. Accept the terms and click Next

Step 4. In the next screen, It will ask for some additional features, it is not mandatory to check this box. Click on Next.

Step 5. On the next screen, some checkboxes are populated, Check them as per your requirement. Click on Next.At this step, VMware Workstation is ready to install. Click on Install.

Step 6. Once the installation gets completed you will see the following dialogue box. Click on Finish. If you have purchased the product and have a license key, then you can click on License to enter the key.

Step 7. Upon Finish, For the first time opening, if it is not entered the License key in step 7, then it will ask for a license key. Then go for the trial version which is available free for 15 to 30 days. Click on Continue. Make sure the Admin rights for this in Windows.

1.B) Linux shell commands

Programs:

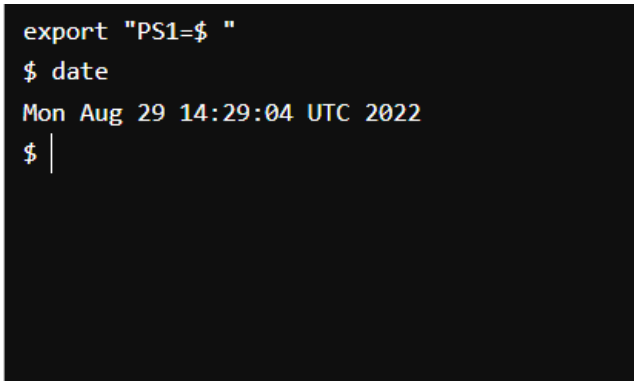
1.Date-

NAME: DATE- print or set the system date and time

SYNTAX: date

DESCRIPTION: Display the current time in the given format or set the system date.

OUTPUT:



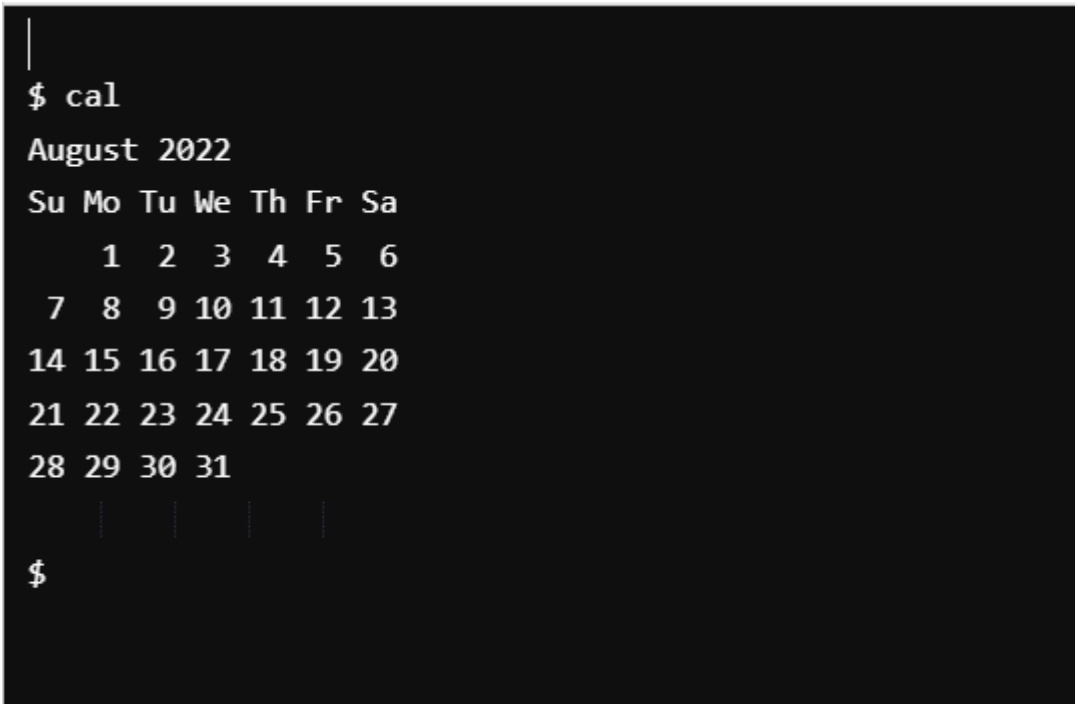
2. Calendar-

NAME: calendar

SYNTAX: cal

DESCRIPTION: Display a simple calendar

OUTPUT:



3. Cd-

NAME: cd – Change Directory

SYNTAX: cd dirname

DESCRIPTION: Change the directory which we use to work with.

OUTPUT:

```
export "PS1=$ "
$ cd new1
bash: cd: new1: No such file or directory
$
```

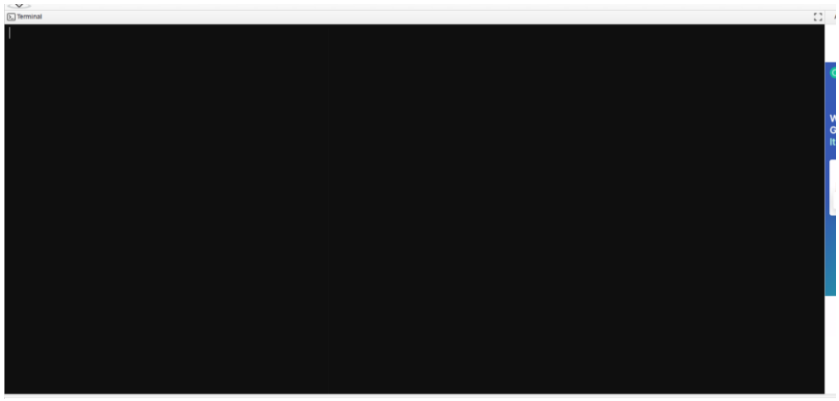
4. Clear-

NAME: clear

SYNTAX: clear

DESCRIPTION: Clears the content of the command prompt.

OUTPUT:



5. Last-

NAME: last

SYNTAX: last

DESCRIPTION: Displays the list of last logged-in users for a month.

OUTPUT:

```
~/OrchidDifferentTheories$ last

wtmp begins Wed Mar 23 05:17:59 2022
```

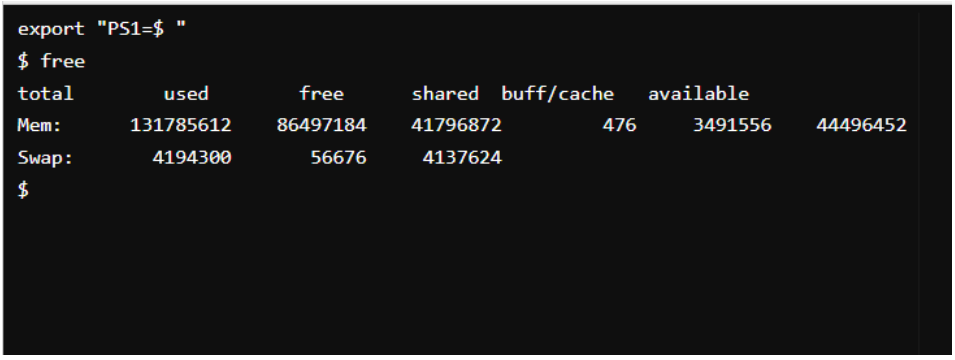
6.Free-

NAME: free

SYNTAX: free

DESCRIPTION: Displays the total amount of free and used physical and swap memory in the system

OUTPUT:



7. df-

NAME: df

SYNTAX: df

DESCRIPTION: Displays the amount of disk space available on the filesystem with each file name's argument

OUTPUT:

```
~/OrchidDifferentTheories$ df
Filesystem
 1K-blocks       Used  Available Use% Mounted on
overlay
1719415016   43590332 1588409492   3% /
tmpfs
     65536          0      65536   0% /dev
tmpfs
  32926368          0   32926368   0% /sys/fs/cgroup
tmpfs
   6585276      19592    6565684   1% /io
/dev/conman/358
   2469888        920    2464684   1% /tmp
overlay
   2469888        920    2464684   1% /nix
/dev/root
  25215872   13323344   11876144  53% /mnt/cacache
/dev/mapper/docker
1719415016   43590332 1588409492   3% /etc/hosts
shm
     65536          0      65536   0% /dev/shm
/dev/disk/by-id/google-cacache-1656701361997-us-west1-b
8454026148 7053067336 1400942428  84% /mnt/cacache/nix
devtmpfs
  32922040          0   32922040   0% /dev/tty
tmpfs
  32926368          0   32926368   0% /proc/acpi
tmpfs
  32926368          0   32926368   0% /proc/scsi
```

8. useradd-

NAME: useradd

SYNTAX: useradd

DESCRIPTION: It edits /etc/passwd, /etc/shadow, /etc/group and /etc/gshadow files for the newly created user accounts

OUTPUT:

```
~/OrchidDifferentTheories$ useradd
Usage: useradd [options] LOGIN
       useradd -D
       useradd -D [options]

Options:
  -b, --base-dir BASE_DIR       base directory for the h
ome directory of the
                                new account
  -c, --comment COMMENT         GECOS field of the new a
ccount
  -d, --home-dir HOME_DIR       home directory of the ne
w account
  -D, --defaults                print or change default
useradd configuration
  -e, --expiredate EXPIRE_DATE  expiration date of the n
ew account
  -f, --inactive INACTIVE       password inactivity peri
od of the new account
  -g, --gid GROUP               name or ID of the primar
y group of the new
                                account
  -G, --groups GROUPS           list of supplementary gr
```

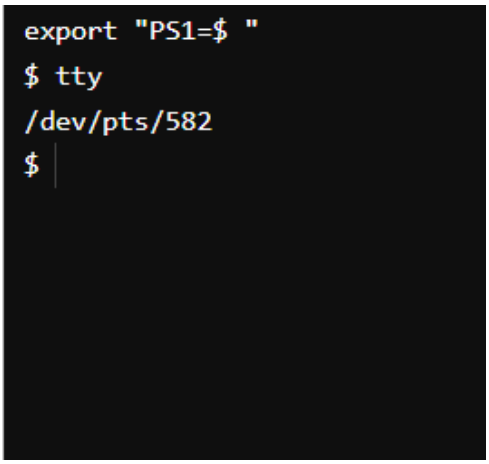
9.Terminal Name-

NAME: tty

SYNTAX: tty

DESCRIPTION: Used to display the terminal path name.

OUTPUT:



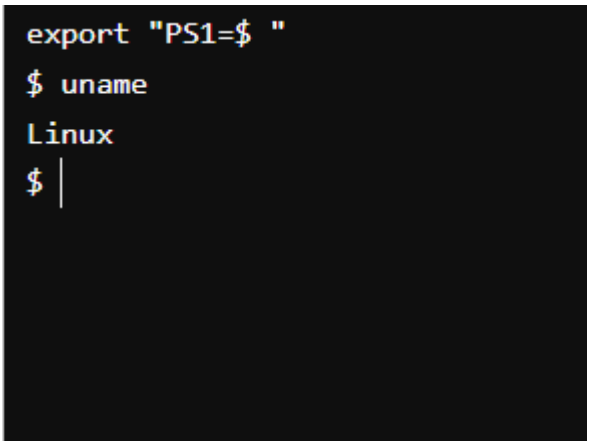
10.Uname-

NAME: Uname

SYNTAX: uname

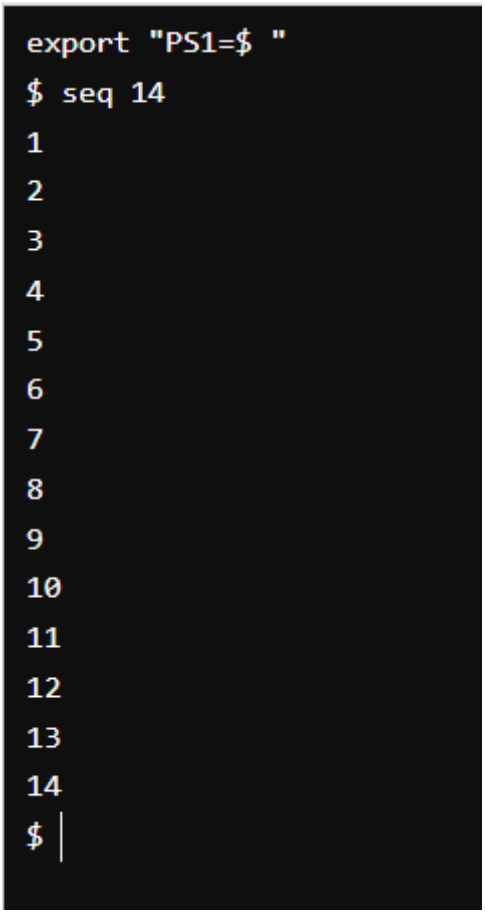
DESCRIPTION: Used to display the name of the system being used.

OUTPUT:



CRIPTION: print a sequence of numbers

OUTPUT:



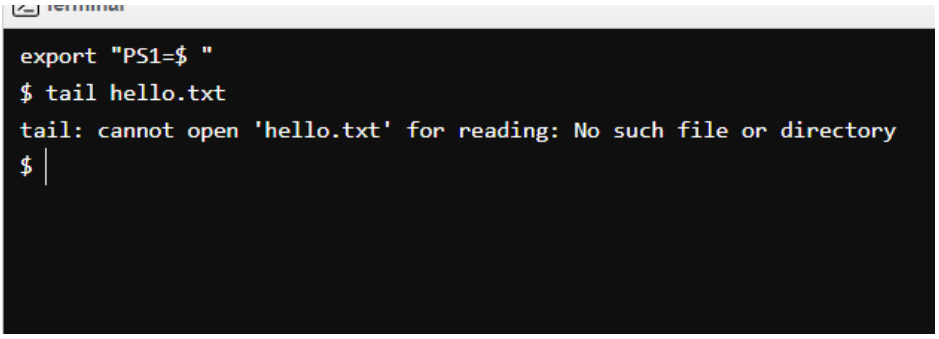
12.Tail-

NAME: tail S

SYNTAX: tail filename

DESCRIPTION: Displays the last ten lines in the file.

OUTPUT:

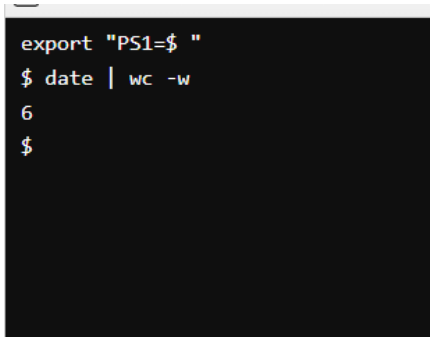


13 NAME: PIPE-|

SYNTAX: cmd1 | cmd2 | cmd3

DESCRIPTION: Makes the output of one command as input for another command.

OUTPUT:



14 Sort-

NAME: sort

(i) SYNTAX: sort filename

DESCRIPTION: Sorts the content of the file in ascending order.

OUTPUT:

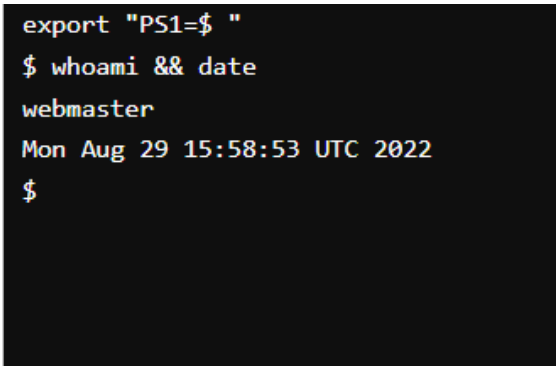
15 And-

NAME: and - & &amp.

SYNTAX: cmd1 & & cmd2

DESCRIPTION: Used to combine more than one commands.

OUTPUT:



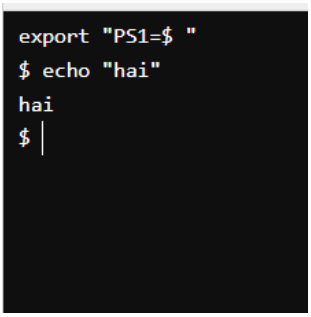
16 Echo-

NAME: echo – displays a line of text.

SYNTAX: echo "……."

DESCRIPTION: Displays the statement within double quotes.

OUTPUT:



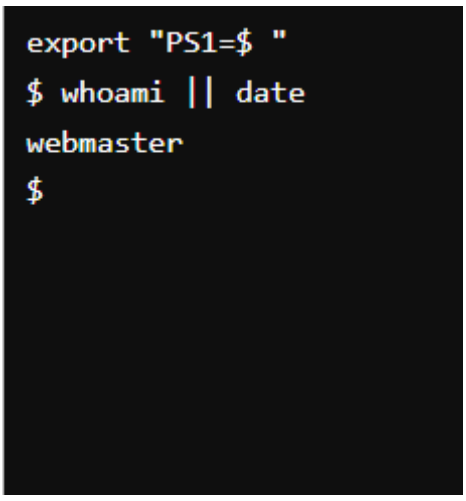
17. Or-

NAME: or - ||

SYNTAX: cmd1 || cmd2

DESCRIPTION: Displays the output for one command which is true

OUTPUT:



18. cp-

NAME: cp – copy files and directories

SYNTAX: cp fi f2

DESCRIPTION: Copies f1 to f2

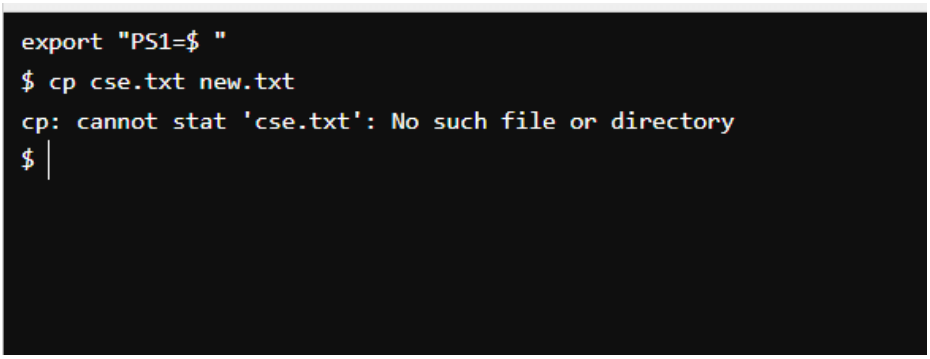
OUTPUT:

11.Sequence-

NAME: seq

SYNTAX: seq starting_value ending_value

DES



19 Touch-

NAME: touch

SYNTAX: touch filename

DESCRIPTION: Creates an empty file.

OUTPUT

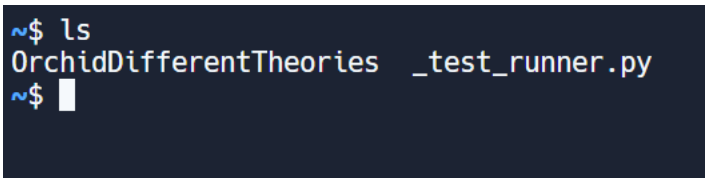
20 List-

NAME: LIST – list directory contents

(i)SYNTAX: ls

DESCRIPTION: List information about the Files (the current directory by default).

OUTPUT:



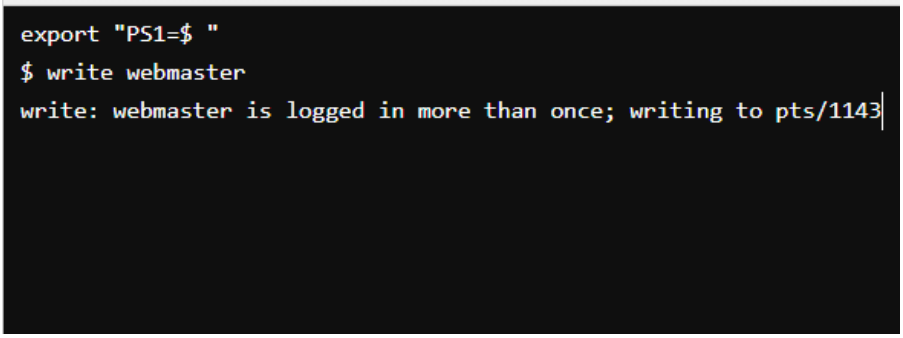
21 Write-

NAME: write

SYNTAX: write login name

DESCRIPTION: Used to communicate with other logged in users.

OUTPUT



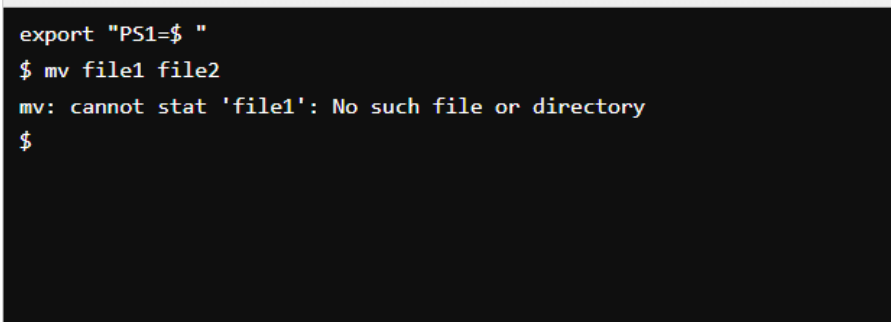
22.mv-

NAME: mv – move(rename) files

SYNTAX: mv f1 f2

DESCRIPTION: Renames Source to Destination

OUTPUT:



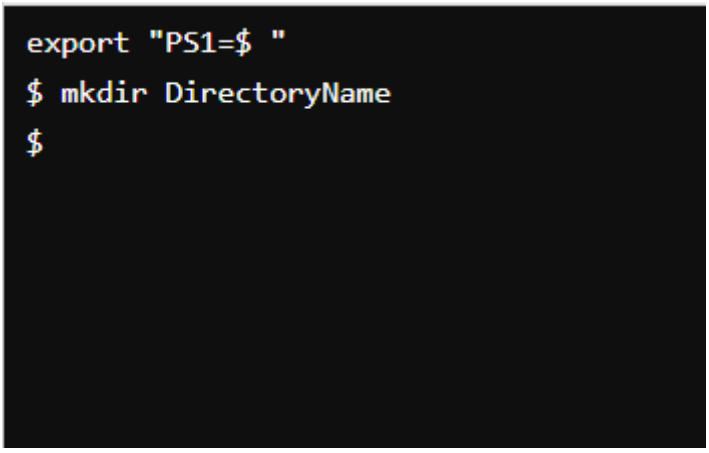
23.mkdir-

NAME: mkdir – makes directory

SYNTAX: dir DirectoryName

DESCRIPTION: Creates the directory if they do not exist already

OUTPUT:



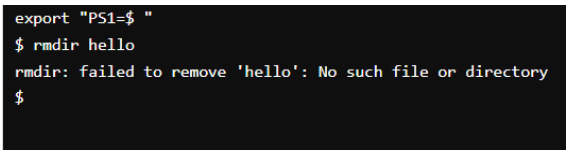
24.rmdir-

NAME: rmdir – removes directory

SYNTAX: rmdir DirectoryName

DESCRIPTION: Removes the directory, only it is empty

OUTPUT:



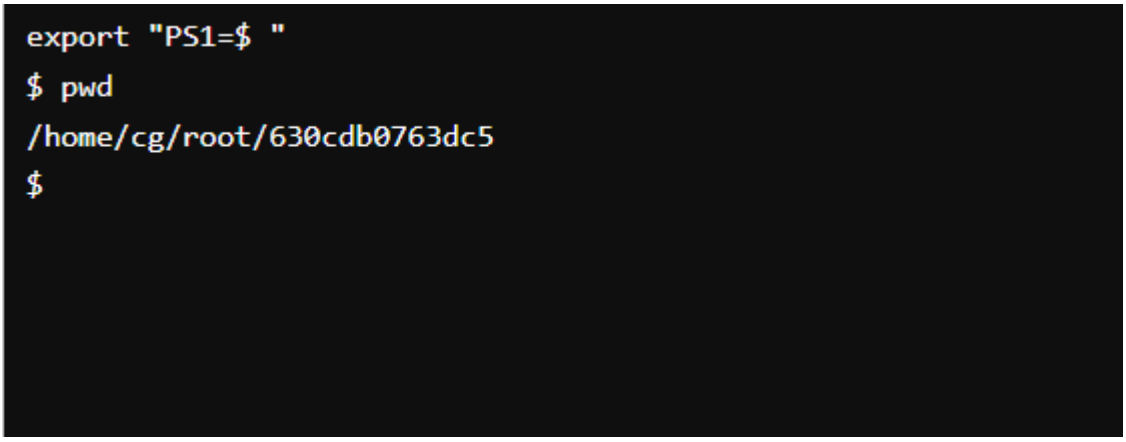
25.Pwd-

NAME: pwd – Present Working Directory displays the name of the current/working directory

SYNTAX: pwd

DESCRIPTION: Displays the name of the current/working directory

OUTPUT:



Result:

We have successfully installed various os and performed 25 shell commands and received the output for the commands.

ExpNo:02 Name:Godfrey Ashwanth
Date:16/08/22 RegNo:RA2112704010020

Process Creation using fork() and Usage of getpid(), getppid(), wait() functions

Aim: To study the process creation using fork(),getpid(),getppid(),wait() function

Description :

The fork() in C. Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.

Zombie process:

A process which has finished its execution but still has an entry in the process table to report to its parent process is known as a zombie process.

Orphan process

The process whose parent process has finished (Completed execution) or terminated and do not exists in the process table are called orphan process.

Code:

Input 1:

```
#include<stdio.h>

#include <unistd.h>

int main()

{int pid,ppid;

 fork();

 fork();

pid=getpid();

ppid=getppid();

printf("process id %d",pid);

printf("parent process id %d",ppid);

return 0;

}
```

Output:

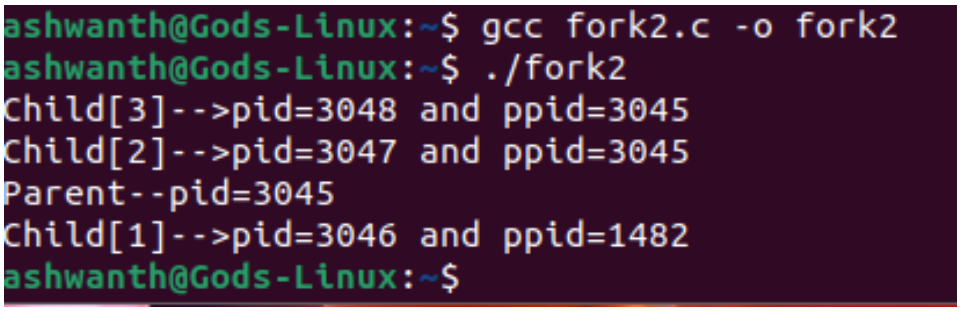
```
ashwanth@Gods-Linux:~$ gcc fork.c -o fork
ashwanth@Gods-Linux:~$ ./fork
Process ID: 4684
Parent Process ID: 4360
Process ID: 4685
Parent Process ID: 4684
Process ID: 4687
Parent Process ID: 1442
ashwanth@Gods-Linux:~$ Process ID: 4686
Parent Process ID: 4684

ashwanth@Gods-Linux:~$
```

Input2:

```
#include<stdio.h>
#include <unistd.h>
#include<stdlib.h>
int main()
{  int pid, pid1,pid2;
 pid=fork();
 if(pid==0){
  sleep(3);
 printf("Child[1] pid=%d ppid=%d\n", getpid(),getppid());
 }
 else{
 pid1=fork();
  if(pid1==0){
  sleep(2);
  printf("Child [2] pid=%d ppid=%d\n",getpid(), getppid());
}
  else{
  pid2=fork();
 if(pid2==0){
  printf("Child[3] pid=%d ppid=%d\n",getpid(), getppid());
}
else
```

```
  sleep(3);
printf("Parent pid=%d\n", getpid());}}}
return 0;
}
```

Output:

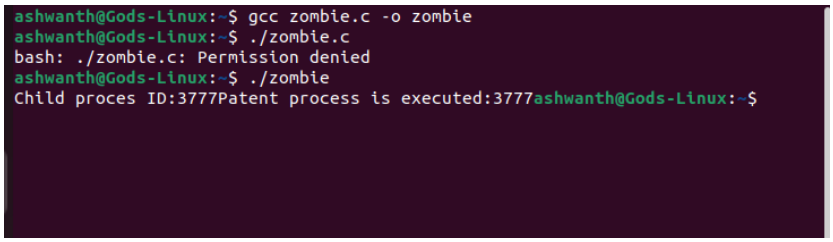


Zombie :

```
 #include <stdlib.h>
 #include <sys/types.h>
 #include <unistd.h>
 #include <stdio.h>
 int main()
 {
        int pid=fork();
        int id=getpid();
        int ppid=getppid();
    if(pid>0){
      sleep(50);
      printf("Parent id %d",ppid);}
        else{
    printf("Parent id %d",ppid);
    printf("Child id %d",pid);
        exit(0);
        }
        return 0;
```

```
}
```

Output:



Orphan:

```
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>
 int main()
{
        int pid = fork();
        if (pid > 0)
        printf("in parent process");
        else if (pid == 0)
        {
      sleep(30);
      printf("in child process");
        }
        return 0;
}
```

Output:

Result:

Hence all the process creation functions are executed successfully.Thus, ZOMBIE and ORPHAN process is created

ExpNo:03 Name:Godfrey Ashwanth

Date:23/08/22 RegNo:RA2112704010020

Multithreading and pthread in C

Aim: To perform Multithreading and pthreads in C with pthread function.

Description: A thread is a single sequence stream within in a process. As threads have some of the properties of processes, they are sometimes called lightweight processes. In short, thread is a unit of a process.

Thread is a programming technique to improve application performance through parallel processes. For example, in a browser, multiple tabs can be different threads. MS word uses multiple threads, one thread to format the text, other thread to process inputs and so on.

Threads operate faster than processes due to following reasons:

1. Thread creation is much faster
2. Context switching between threads is much faster
3. Threads can be terminated easily
4. Communication between threads is faster

Pthreads function:

- pthread_create
- pthread_join
- pthread_cancel
- pthread_exit

POSIX threads

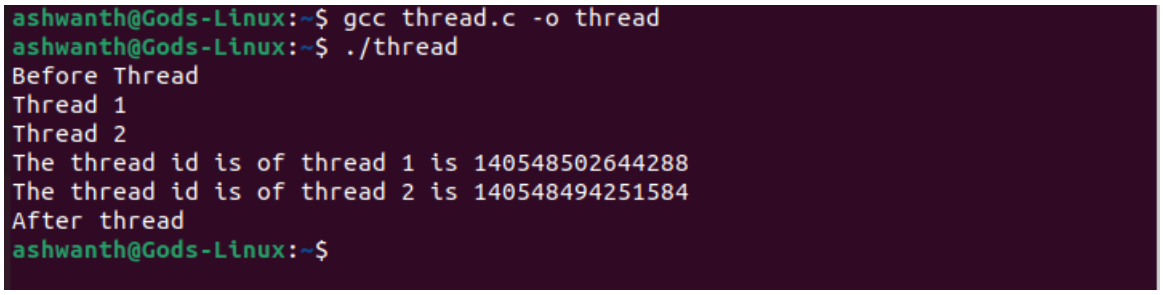
A process is an execution environment in an operating system. A process has code and data segments which are initialized from a program during an exec system call. A process has a thread of execution, wherein instructions are executed as per the value of the program counter register.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>.
#include <pthread.h>
void *Thread1(void *vargp)
{
    sleep(1);
    printf("Thread  1\n");
    return NULL;
}
void *Thread2(void *vargp)
{
    sleep(1);
    printf("Thread  2\n");
    return NULL;
}
int main()
{
    pthread_t thread_id1,thread_id2;
    printf("Before Thread\n");
    pthread_create(&thread_id1, NULL, Thread1, NULL);
   pthread_create(&thread_id1, NULL, Thread1, NULL);
    pthread_join(thread_id1, NULL);
   pthread_join(thread_id2, NULL);
    printf("After Thread\n");
```

```
    exit(0);
}
```

Output:



Result: Hence all the codes are executed successfully and the output is verified.

ExpNo:04 Name:Godfrey Ashwanth

Date:30/08/22 RegNo:RA2112704010020

Mutual Exclusion Using Mutex,Semaphore And Monitors

Aim:

1. Critical Section and Synchronization Techniques

2.Mutual Exclusion and semaphore and monitors

Description:

Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data.

Sections of a Program in OS: Following are the four essential sections of a program:

1. Entry Section: This decides the entry of any process.

2. Critical Section: This allows a process to enter and modify the shared variable.

3. Exit Section: This allows the process waiting in the Entry Section, to enter into the Critical Sections and makes sure that the process is removed through this section once it’s done executing.

4. Remainder Section: Parts of the Code, not present in the above three sections are collectively called Remainder Section.

Mutual Exclusion: When a process/thread is executing its critical section no other processes are allowed to execute its critical section.

Semaphore is a data handling technique which is very useful in process synchronization and multithreading.

A monitor is a synchronization mechanism that allows threads to have:

 mutual exclusion – only one thread can execute the method at a certain point in time, using locks

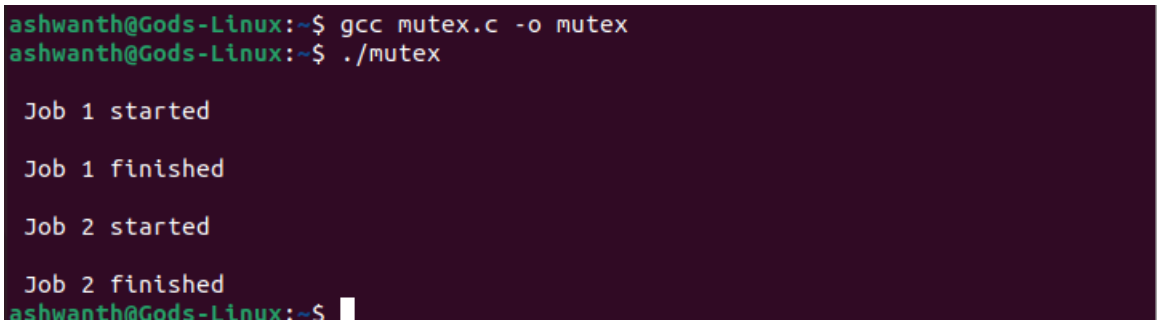
 cooperation – the ability to make threads wait for certain conditions to be met, using wait-set

Code:

Mutex:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include <unistd.h>
pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void* doSomeThing(void *arg)

{
pthread_mutex_lock(&lock);
unsigned long i = 0;
counter += 1;
printf"\n Job %d started\n", counter);
for(i=0; i<(0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
pthread_mutex_unlock(&lock);
return NULL;
}
int main(void)
{
int i = 0;
int err;
if (pthread_mutex_init(&lock, NULL) != 0)
{
printf("\n mutex init failed\n");
return 1;
}
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
printf("can't create thread :[%s]", strerror(err));
i++;
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_mutex_destroy(&lock);
return 0;
}
```

Output:

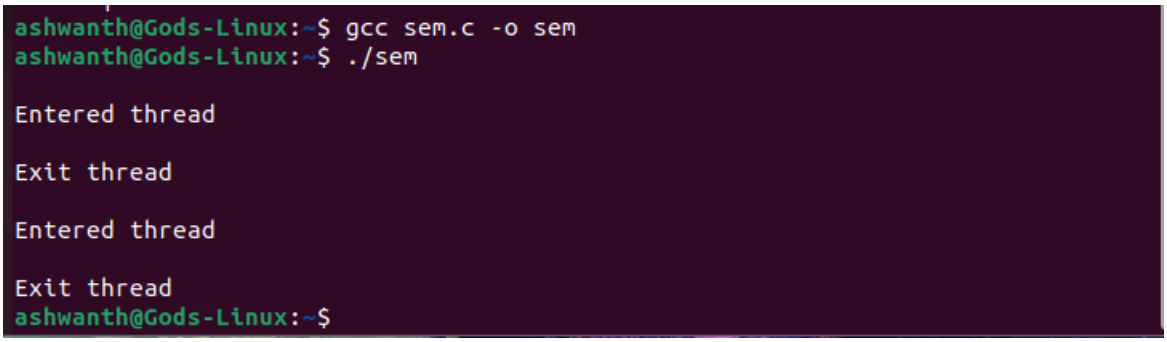


Semaphore:

```
#include < stdio.h>
#include <pthread.h>
#include < semaphore.h>
#include <unistd.h>
sem_t mutex;
void* thread(void* arg)
{
sem_wait(&mutex);
printf"\nEntered thread\n");
sleep(4);
//signal
printf("\n Exit thread\n");
sem_post(&mutex);
}

int main()
{
sem_init(&mutex, 0, 1);
```

```
pthread_t t1,t2;
pthread_create(&t1,NULL,thread,NULL);
sleep(2);
pthread_create(&t2,NULL,thread,NULL);
pthread_join(t1,NULL);
pthread_join(t2,NULL);
sem_destroy(&mutex);
return 0;
}
```

Output:



Monitors:

```
#Code in JAVA
public class ThreadA {
   public static void main(String[] args){
      ThreadB b = new ThreadB();
      b.start();

      synchronized(b){
         try{
            System.out.println("Waiting for b to complete...");
            b.wait();
         }catch(InterruptedException e){
```

```
            e.printStackTrace();
        }

        System.out.println("Total is: " + b.total);
    }
  }
}

class ThreadB extends Thread{
  int total;
  @Override
  public void run(){
    synchronized(this){
        for(int i=0; i<100 ; i++){
            total += i;
        }
        notify();
    }
  }
}
```

Output:

```
Waiting for b to complete...
Total is: 4950
```

Result: Thus,we have performed the following program for mutual exclusion using semaphore and monitor.

ExpNo:05 Name:Godfrey Ashwanth
Date:06/09/22 RegNo:RA2112704010020

Reader-Writer

Aim: To study about reader and writer problem based on mutex concept.

Description:

The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets.

The readers-writers problem is used for managing synchronization among various reader and writer process so that there are no problems with the data sets, i.e. no inconsistency is generated.

Code:

```
#Code in python
import threading as thread
import random

global x
x=0

lock = thread.Lock()

def Reader():
    global x
    print('Reader is Reading!')
    lock.acquire()
    print('Shared Data:', x)
    lock.release()
    print()


def Writer():
    global x
    print('Writer is Writing!')
    lock.acquire()
    x += 1
    print('Writer is Releasing the lock!')
    lock.release()
    print()



if __name__ == '__main__':
```

```
    for i in range(0, 10):
        randomNumber = random.randint(0, 100)
        if(randomNumber > 50):
            Thread1 = thread.Thread(target = Reader)
            Thread1.start()
        else:
            Thread2 = thread.Thread(target = Writer)
            Thread2.start()

Thread1.join()
Thread2.join()
```

Output:

```
Reader is Reading!
Shared Data: 0

Writer is Writing!
Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Reader is Reading!
Shared Data: 5

Writer is Writing!
Reader is Reading!
Writer is Releasing the lock!

Reader is Reading!
Shared Data: 6

Shared Data: 6
```

Result: Hence the code is executed successfully and the output is verified.

ExpNo:06 Name: Godfrey Ashwanth

Date:13/9/202 RegNo:RA2112704010020

DINING PHILOSOPHER

Aim:

To perform dining philosopher's problem

Description:

The dinning philosopher problem of synchronization which says that 5 philosophers are sitting around a table and their job is to think alternatively.

Code:

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

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
	int i,a[5];
	pthread_t tid[5];

	sem_init(&room,0,4);

	for(i=0;i<5;i++)
		sem_init(&chopstick[i],0,1);

	for(i=0;i<5;i++){
		a[i]=i;
		pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
	}
	for(i=0;i<5;i++)
		pthread_join(tid[i],NULL);
}
```

```
void * philosopher(void * num)
{
        int phil=*(int *)num;

        sem_wait(&room);
        printf("\nPhilosopher %d has entered room",phil);
        sem_wait(&chopstick[phil]);
        sem_wait(&chopstick[(phil+1)%5]);

        eat(phil);
        sleep(2);
        printf("\nPhilosopher %d has finished eating",phil);

        sem_post(&chopstick[(phil+1)%5]);
        sem_post(&chopstick[phil]);
        sem_post(&room);
}

void eat(int phil)
{
        printf("\nPhilosopher %d is eating",phil);

}
```

Output:

Result:

Dining philosopher's problem has been executed.

ExpNo:07 Name:Godfrey Ashwanth
Date: 19/9/22 RegNo:RA2112704010020

Banker's Algorithm

Aim:

To perform dining
banker's algorithm

Description:

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. The Banker's algorithm is run by the operating system whenever a process request's resources. The algorithm prevents deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur).

Algorithm:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true goto step (2)

4) if Finish [i] = true for all i then the system is in a safe state

Code:

```
#code in python

import numpy as np
def check(i):

    for j in range(no_r):

       if(needed[i][j]>available[j]):

          return 0

    return 1

no_p = 5

no_r = 4

Sequence = np.zeros((no_p,),dtype=int)
visited = np.zeros((no_p,),dtype=int)

allocated = np.array([[4,0,0,1],[1,1,0,0],[1,2,5,4],[0,6,3,3],[0,2,1,2]])

maximum = np.array([[6,0,1,2],[1,7,5,0],[2,3,5,6],[1,6,5,3],[1,6,5,6]])

 needed = maximum - allocated
 available = np.array([3,2,1,1])
 count = 0
while( count < no_p ):

    temp=0

    for i in range( no_p ):
```

```
        if( visited[i] == 0 ):

            if(check(i)):

                Sequence[count]=i;
                count+=1
                visited[i]=1
                temp=1
                for j in range(no_r):

                    available[j] +=
    allocated[i][j] if(temp == 0):
        break

if(count < no_p):

    print('The system is Unsafe')
  else:
    print("The system is Safe")

    print("Safe Sequence: ",Sequence)


    print("Available Resource:",available)
```

Output:

```
The system is Safe
Safe Sequence:  [0 2 3 4 1]
Available Resource: [ 9 13 10 11]
> 
```

Result:

Banker's algorithm problem has been executed.

ExpNo: 08 Name:Godfrey Ashwanth

Date: 26/09/22 RegNo:RA2112704010020

FCFS And SJF Scheduling

Aim:

To perform FCFS and SJF Scheduling

Description:

First come first server is the simplest type of algorithm. It is a non-preemptive algorithm i.e the process can not be interrupted once it starts executing. The FCFS is implemented with the help of a FIFO Queue. Shortest Job First is leased upon the burst time of the process. The processes are put into the ready queue based on their burst times. Its preemtipve version is called Shortest Remaining Time First (SRTF)

Algorithm:

Fcfs:-

1- Input the processes along with their burst time(bt)

and arrival time(at)

2- Find waiting time for all other processes i.e. for

a given process i:

wt[i] = (bt[0] + bt[1] +..... bt[i-1]) - at[i]

3- Now find turn around time

= waiting_time + burst_time for all processes

4- Average waiting time =

total_waiting_time / no_of_processes

5- Average turn around time =

total_turn_around_time / no_of_processes

SJF:-

Sort all the processes according to the arrival time.

Then select that process that has minimum arrival time and minimum Burst time.

After completion of the process make a pool of processes that arrives afterward till the completion of the previous process and select that process among the pool which is having minimum Burst time.

Code:

```
#fcfs
#include<stdio.h>
Int waitingtime(int proc[],int n, int brust_time[],int wait_time[]){
wait_time [0]=0;
for(int i=1;i<n;i++)
        wait_time[i]=burst_time[i-1]+wait_time[i-1];
        return 0;
}
Int turnaroundtimie(int proc[],int n,int brust_time[],int wait_time[],int tot[])
{
Int I;
For(i=0;i<n;i++){
        Tat[i]=birst_time[i]+wait_time[i];
Return 0;
}
Int algorithm(int proc[],int n, int brust_time[])
{
Int wait_time[n]; int tat[n];total_wat=0;
```

```
Total_tat=0;
Int I;
Waitingtime(proc,n,burst_time,wait_time);
Turnaoundaroundtime(proc,n,burst_time,wait_time,tat);
For(i=0;i<n;i++)
        Total_wat=total_wat+wait_time[i];
        Total_tat=total_tat+tat[i];
        Printf("%d\t %d \t\t %d\t %d"); I,brust_time[i],wait_time[i],total[i];
Int main(){
Int proc[]={1,2,3}
Int n=size of proc/size of proc[0];
Int brust_time[]={5,8,12}
Waiting_time(proc,n,burst_time);
Return 0;
}

#SJF
#incliude<stdio.h>
Int main(){
Int A[100][4];
Int I,j,n,total=0,idex,temp;
Float avg_wat,avg_tat;
Printf("Enter Brust Time \n")
For(int i=0;i<n;i++){
        Printf("p%d",i++);
        Scanf("%d",&A[i]A[j]);
        P[i]=i+1;
}
For(i=0;i<n;i++){
Pos=I;for(j=i+1;j<n;j++){
        If(A[j][i]<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;
For(int i=1;i<n;i++){
A[1][2]=0;
For(j=0;j<n;j++)
        A[1][2]+=A[j][i];
Total+=A[i][2];
}
Avg_wat=(float) total/n;
Total=0;
Printf("P BT WT TAT \n");
For(i=0;i<n;i++)
        A[i][3]=A[i][1]+A[i][2];
        Total+=A[i][3];
Printf("P %d P2%d P3 %d P4 %d",
A[i][0],A[i][1],A[i][2]);
}
Avg_tat=(float) total/n;
}
}
```

Output:

FCFS

```
Processes  Burst   Waiting Turn around

1          5              0       5
2          8              5       13
3          12             13      25Average waiting time = 6.000000
Average turn around time = 14.333333
```

SJF

```
Enter number of process: 4
Enter Burst Time:
P1: 2
P2: 3
P3: 4
P4: 1
P      BT     WT     TAT
P4      1     0      1
P1      2     1      3
P2      3     3      6
P3      4     6      10
Average Waiting Time= 2.500000
```

Result: Hence the code is executed successfully and the output is verified.

ExpNo:09 Name:Godfrey Ashwanth

Date:03/10/22 RegNo:RA2112704010020

Priority And Round Robin Scheduling

Aim: To implement priority and round robin scheduling algorithm in linux

Description: Priority scheduling, a number is assigned to each process. The lower the number, higher the priority. The priority is not always set as the inverse of the CPU burst time. Rather, it can be internally or externally set. But yes, the scheduling is done on the basis of priority of the process where the process with more urgent is processed first, followed by the ones with lesser priority in order. Round Robin is the preemptive scheduling in which every process get executed in a cycle way i.e in this a particular, time slice is allotted to each process which is known as time quantum.

Algorithm:

Priority:-

First input the processes with their arrival time, burst time and priority.

First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.

Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number.

Once all the processes have been arrived, we can schedule them based on their priority.

Round Robin:-

Step 1: Organize all processes according to their arrival time in the ready queue. The queue structure of the ready queue is based on the FIFO structure to execute all CPU processes.

Step 2: Now, we push the first process from the ready queue to execute its task for a fixed time, allocated by each process that arrives in the queue.

Step 3: If the process cannot complete their task within defined time interval or slots because it is stopped by another process that pushes from the ready queue to execute their task due to arrival time of the next process is reached. Therefore, CPU saved the previous state of the process, which helps to resume from the point where it is interrupted. (If the burst time of the process is left, push the process end of the ready queue).

Step 4: Similarly, the scheduler selects another process from the ready queue to execute its tasks. When a process finishes its task within time slots, the process will not go for further execution because the process's burst time is finished.

Step 5: Similarly, we repeat all the steps to execute the process until the work has finished.

Code:

Priority:

```
//Implement Of Priority Scheduling In c++
#include<bits/stdc++.h>

Using namespace std;

Struct Process{
Int pid;
Int bt;
int priority;
};

Bool sortprocess(process a,process b){
Return (a.priority >b.priority );}

Void findwaitingtime(process proc[],int n,int wt[]){
Wt[0]=0;      //waittime for 1st process
For(int i=1;i<n;i++)
        Wt=[i]=proc[i-1].bt+wt[i-1];
}

Void findturnaroundtime(process proc[],int n. int wt[],int tat)
{
For(int in=0;i<n;i++)
```

```
        Tat[i]=proc[i].bt+wt[i];
}

Void findaveragetime(process proc[],int n){
Int wt[n],tat[n],total_wt=0,total_tat=0;
Findingwaitingtime(proc,n,wt);
Findturnaoundtime(proc,n,wt,tat);
Cout<<"\n process"<<"Burst time"<<"Waiting time"<<"Turn Aound time\n";

For(int i=0;i<n,i++){
Total_wt=total_wt+wt[i];
Total_tat=totaltat+tat[i];
Cout<<" "<<proc[i].pid<<"\t\t"<<proc[i].bt<<"\t\t"<<tat[i]<<end}

Cout<<"\nAverage Waiting time="<<(float)total_wt/(float)n;
Count<<"\nAverage turn around time="<<(float)total_tat/(float)n;
}

Void priorityscheduling(process.proc[],int n){
Sort(proc,proc+n,sort process);
Count<<"Order in which process gets executed\n";
For(int i=0;i<n,i++)
        Count<<proc[i].pid<<" ";
Findaroundtime(proc,n);
}

Int main(){
Process.proc[]={{1,10,2},{2,5,0},{3,8,1}};
Int n=sizeof proc/size of proc[0];
Priorityscheduling(proc,n);

Return 0;
}
```

Round Robin:

```
//Implementation of round robin
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)
{

        int rem_bt[n];
        for (int i = 0 ; i < n ; i++)
                rem_bt[i] = bt[i];
        int t = 0; // Current time

        while (1)
        {
                bool done = true;

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

```
                        if (rem_bt[i] > 0)
                        {
                                done = false; // There is a pending process
                                if (rem_bt[i] > quantum)
                                {
                                        t += quantum;
                                        rem_bt[i] -= quantum;
                                }
                                else
                                {
                                        t = t + rem_bt[i];
                                        wt[i] = t - bt[i];
                                        rem_bt[i] = 0;
                                }
                        }
                }
                if (done == true)
                break;
        }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{

        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[], int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt, quantum);
        findTurnAroundTime(processes, n, bt, wt, tat);
        cout << "Processes "<< " Burst time "
                << " Waiting time " << " Turn around time\n";
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                        << wt[i] <<"\t\t " << tat[i] <<endl;
        }
        cout << "Average waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}
int main()
{

        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];
        int burst_time[] = {10, 5, 8};
        int quantum = 2;
        findavgTime(processes, n, burst_time, quantum);
        return 0;
```

}

Output:

Priority:

```
Order in which processes gets executed 1 3 2 Processes  Burst time  Waiting time
    Turnaround time 1       10   0       10
 3      8    10      18
 2      5    18      23
Average waiting time = 9.33333
Average turn around time = 17
```

Round Robin:

```
Processes  Burst time  Waiting time  Turn around time
1       10   13      23
 2      5    10      15
 3      8    13      21
Average waiting time = 12
Average turn around time = 19.6667
```

Result:

Hence the code executed successfully and the output is verified.

ExpNo:10 Name:Godfrey Ashwanth

Date:12/10/22 RegNo:RA2112704010020

FIFO Page Replacement Algorithm

Aim: To perform FIFO page replacement algorithm

Description:

FIFO which is also called First In First Out is one of the types of Replacement Algorithms. This algorithm is used in a situation where an Operating system replaces an existing page with the help of memory by bringing a new page from the secondary memory. FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue. The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added.

Algorithm:

Step 1. Start to traverse the pages.

Step 2. If the memory holds fewer pages, then the capacity else goes to step 5.

Step 3. Push pages in the queue one at a time until the queue reaches its maximum capacity or all page requests are fulfilled.

Step 4. If the current page is present in the memory, do nothing.

Step 5. Else, pop the topmost page from the queue as it was inserted first.

Step 6. Replace the topmost page with the current page from the string.

Step 7. Increment the page faults.

Step 8. Stop

Code:

```
#CODE IN PYTHON

from queue import Queue

def Faults(input, n, q):

    print("Incoming \t pages")


    # incoming stream item in set or not

    s = set()

    # Queue created to store pages in FIFO manne

    # we will use queue to note order of entry of incoming page

    queue = Queue()


    page_faults = 0

    for i in range(n):


        # if set has lesser item than frames i.e. set can hold more items

        if len(s) < q:


            # If incoming item is not present, add to set

            if input[i] not in s:

                s.add(input[i])
```

```
        # increment page fault
        page_faults += 1
        # Push the incoming page into the queue
        queue.put(input[i])

# If the set is full then we need to do page replacement remove first item from both
else:
    # If incoming item is not present
    if input[i] not in s:
        # remove the first page from the queue
        val = queue.queue[0]
        queue.get()
        # Remove from set
        s.remove(val)
        # insert incoming page to set
        s.add(input[i])
        # push incoming page to queue
        queue.put(input[i])
        # Increment page faults
        page_faults += 1

print(input[i], end="\t\t")
for q_item in queue.queue:
    print(q_item, end="\t")

print()
```

```
    return page_faults

input = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1]

n = len(input)

q = 3 #the number of frames

page_faults = Faults(input, n, q)

hits = n - page_faults

print("\nPage Faults: " ,page_faults)

print("Hit: " ,hits)
```

OutPut:

```
ashwanth@Gods-Linux:~$ python3 fifo.py
Incoming         pages
7                7
0                7               0
1                7               0               1
2                0               1               2
0                0               1               2
3                1               2               3
0                2               3               0
4                3               0               4
2                0               4               2
3                4               2               3
0                2               3               0
3                2               3               0
2                2               3               0
1                3               0               1
Page fault 11
Hit 3
```

Result: Hence the code is executed successfully and the output is verified.

ExpNo: 11 Name:Godfrey Ashwanth

Date:18/10/22 RegNo:RA2112704010020

LRU AND LFU PAGE REPLACEMENT ALGORITHM

Aim: To perfom page replacement algorithm using LRU &LFU Method.

Description:

LRU:- Least Recently Used. It keeps track of page usage in the memory over a short period of time. It works on the concept that pages that have been highly used in the past are likely to be significantly used again in the future. It removes the page that has not been utilized in the memory for the longest time.

LFU :- The LFU page replacement algorithm stands for the Least Frequently Used. In the LFU page replacement algorithm, the page with the least visits in a given period of time is removed. It

replaces the least frequently used pages. If the frequency of pages remains constant, the page that comes first is replaced first

Algorithm:

LFU:-

1. Start

2. Read Number Of Pages And Frames
3.Read Each Page Value
4. Search For Page In The Frames
5. If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is Leastly Used
7.Print Page Number Of Page Faults
8.Stop

LRU:-Let capacity be the number of pages thatmemory can hold. Let set be the current set of pages in memory.

1- Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the recent occurred index of each page in a map called indexes.

c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Find the page in the set that was least recently used. We find it using index array.

We basically need to replace the page with minimum index.

b) Replace the found page with current page.

c) Increment page faults.

d) Update index of current page.

Return page faults.

Code:

LRU:

```
#include <stdio.h>

//user-defined function
int findLRU(int time[], int n)
{
  int i, minimum = time[0], pos = 0;

  for (i = 1; i < n; ++i)
  {
    if (time[i] < minimum)
    {
      minimum = time[i];
      pos = i;
    }
  }

  return pos;
}

//main function
int main()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
  printf("Enter number of frames: ");
```

```
scanf("%d", &no_of_frames);

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

printf("Enter reference string: ");

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

for (i = 0; i < no_of_frames; ++i)
{
  frames[i] = -1;
}

for (i = 0; i < no_of_pages; ++i)
{
  flag1 = flag2 = 0;

  for (j = 0; j < no_of_frames; ++j)
  {
    if (frames[j] == pages[i])
    {
      counter++;
      time[j] = counter;
      flag1 = flag2 = 1;
      break;
    }
  }
```

```
if (flag1 == 0)
{
  for (j = 0; j < no_of_frames; ++j)
  {
    if (frames[j] == -1)
    {
      counter++;
      faults++;
      frames[j] = pages[i];
      time[j] = counter;
      flag2 = 1;
      break;
    }
  }
}

if (flag2 == 0)
{
  pos = findLRU(time, no_of_frames);
  counter++;
  faults++;
  frames[pos] = pages[i];
  time[pos] = counter;
}

printf("\n");

for (j = 0; j < no_of_frames; ++j)
{
  printf("%d\t", frames[j]);
```

```
    }
  }

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

  return 0;
}
```

LFU:

```
#include<stdio.h>
void print(int frameno,int frame[])
{
        int j;
        for(j=0;j<frameno;j++)
        printf("%d\t",frame[j]);
        printf("\n");
}
int main()
{
        int i,j,k,n,page[50],frameno,frame[10],move=0,flag,count=0,count1[10]={0},
                repindex,leastcount;
        float rate;
        printf("Enter the number of pages\n");
        scanf("%d",&n);
        printf("Enter the page reference numbers\n");
        for(i=0;i<n;i++)
        scanf("%d",&page[i]);
        printf("Enter the number of frames\n");
        scanf("%d",&frameno);
        for(i=0;i<frameno;i++)
        frame[i]=-1;
```

```
printf("Page reference string\tFrames\n");
for(i=0;i<n;i++)
{
	printf("%d\t\t\t",page[i]);
	flag=0;
	for(j=0;j<frameno;j++)
	{
		if(page[i]==frame[j])
		{
			flag=1;
			count1[j]++;
			printf("No replacement\n");
			break;
		}
	}
	if(flag==0&&count<frameno)
	{
		frame[move]=page[i];
		count1[move]=1;
		move=(move+1)%frameno;
		count++;
		print(frameno,frame);
	}
	else if(flag==0)
	{
		repindex=0;
		leastcount=count1[0];
		for(j=1;j<frameno;j++)
		{
			if(count1[j]<leastcount)
			{
```

```
                repindex=j;
                leastcount=count1[j];
            }
        }

        frame[repindex]=page[i];
        count1[repindex]=1;
        count++;
        print(frameno,frame);
    }
}
rate=(float)count/(float)n;
printf("Number of page faults is %d\n",count);
printf("Fault rate is %f\n",rate);
return 0;
}
```

Output:

LRU:

```
ashwanth@Gods-Linux:~$ gcc lru.c -oput
ashwanth@Gods-Linux:~$ ./put
Enter number of frames:3
Enter number of pages:6
Enter reference string:2
4
6
8
1
4

        2
        2       4
        2       4       6
        8       4       6
        8       1       6
        8       1       4


Total page falut=6ashwanth@Gods-Linux:~$ s
```

LFU:

```
ashwanth@Gods-Linux:~$ gcc lfu.c -oput
ashwanth@Gods-Linux:~$ ./put
Enter the number of pages:7
Enter the page refernce number
3
4
5
3
2
1
5
Enter the number of frames:
3
page reference string   frames
3                       3       -1      -1
4                       3       4       -1
5                       3       4       5
3                       No replacement
2                       3       2       5
1                       3       1       5
5                       No replacement
Number of page fault is:5ashwanth@Gods-Linux:~$
```

Result:

Thus, LRU and LFU page replacement algorithm is executed successfully.

ExpNo: 12 Name:Godfrey Ashwanth

Date:18/10/22 RegNo:RA211704010020

Best fit and Worst fit memory management policies

Aim: To perform worst fit & best fit memory management policy.

Description:-

Best Fit:-

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Worst Fit:-

In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Algorithm:

Best fit:-

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(bockSize[1],
blockSize[2],.....blockSize[n]) >
processSize[current], if found then assign
it to the current process.

5- If not then leave that process and keep checking
the further processes.

Worst fit:-

1- Input memory blocks and processes with sizes.

2- Initialize all memory blocks as free.

3- Start by picking each process and find the
maximum block size that can be assigned to

current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.

5- If not then leave that process and keep checking the further processes.

Code:-

Best:

```
#include<stdio.h>

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
   {
printf("Block no.%d:",i);
      scanf("%d",&b[i]);
   }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
   {
```

```
        printf("Process no.%d:",i);
        scanf("%d",&p[i]);
    }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

Worst:-

```
#include <stdio.h>
#define MAX 25
```

```
int main()
{
    int frag[MAX],b[MAX],f[MAX],i,j,nb,nf,temp,highest=0;
    static int bf[MAX],ff[MAX];

    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);

    printf("Enter the number of files:");
    scanf("%d",&nf);

    printf("\nEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++)
    {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
    }

    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }

    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(bf[j]!=1)
```

```
        {
            temp=b[j]-f[i];

           if(temp>=0)
            if(highest<temp)
            {
             ff[i]=j; highest=temp; }
              }
             }
              frag[i]=highest;
               bf[ff[i]]=1;
                highest=0;
                 }
                  printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
                  for(i=1;i<=nf;i++)
                   printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getchar();
}
```

Output:

Best Fit:

```
                        Memory Managment Schme -Best Fir
Enter the number of blocks:4
Enter the number of processes:3

Enter the size of blocks:-
Block no 1:214
Block no 2:234
Block no 3:200
Block no 4:432

Enter the size of the process:-
Process no.1:250
Process no.2:460
Process no.3:100

process_no      Process_size    Block_no        Block_size      Fragment
1               250             4               432             182ashwanth@God
s-Linux:~$ s
```

Worst Fit:-

```
        Memory managment scheme -Worst fit
Enter the number of blocks:4
Enter the number of files:3

Enter the size the blocks:-
Block 1:100
Block 2:200
Block 3:217
Block 4:250
Enter the size of the files:
Flie 1:200
Flie 2:50
Flie 3:120

File_no:        File_size:      Block_no:       Block_size:     Fragment
1               200             2               200             0
2               50              1               100             50
3               120             3               217             97ashwanth@Gods
-Linux:~$
```

Result:-

Thus, the worst fit and best fit algorithm has been executed successfully.

ExpNo:13 Name:Godfrey Ashwanth
Date:26/10/22 RegNo:RA2112704010020

Disk Scheduling Algorithms

Aim: To implement Disk Scheduling algorithm in C language using Linux

Description:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller.There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms.

Seek Time:Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write.

Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.

Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time:

Disk Access Time = Seek Time +Rotational Latency + Transfer Time

FCFS

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.Let us understand this with the help of an example**.**

SSTF

In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system. Let us understand this with the help of an example.

SCAN

In this algorithm, the disk arm moves in a particular direction till the end and serves all the requests in its path, then it returns to the opposite direction and moves till the last request is found in that direction and serves all of them

C-SCAN

This algorithm is the same as the SCAN algorithm. The only difference between SCAN and C-SCAN is, it moves in a particular direction till the last and serves the requests in its path. Then, it returns in the opposite direction till the end and doesn't serve the request while returning. Then, again reverses the direction and serves the requests found in the path. It moves circularly.

LOOK

In this algorithm, the disk arm moves in a particular direction till the last request is found in that direction and serves all of them found in the path, and then reverses its direction and serves the requests found in the path again up to the last request found. The only difference between SCAN and LOOK is, it doesn't go to the end it only moves up to which the request is found.

CODE:

FCFS:

```
#include<bits/stdc++.h>
using namespace std; int
main(){
    int i,j,k,n,m,sum=0,x,y,h; cout<<"Enter
    the size of disk\n"; cin>>m;
    cout<<"Enter number of requests\n";
    cin>>n;
    cout<<"Enter the requests\n";
    // creating an array of size n
    vector <int> a(n);
    for(i=0;i<n;i++){
        cin>>a[i];
    }
    for(i=0;i<n;i++){
        if(a[i]>m){
            cout<<"Error, Unknown position "<<a[i]<<"\n";
            return 0;
        }
    }
    cout<<"Enter the head position\n";
    cin>>h;

    // head will be at h at the starting int
    temp=
```

```
cout<<temp;
for(i=0;i<n;i++){
        cout<<" -> "<<a[i]<<' ';
        // calculating the difference for the head movement
        sum+=abs(a[i]-temp);
        // head is now at the next I/O request
        temp=a[i];
    }
    cout<<'\n';
    cout<<"Total head movements = "<< sum<<'\n'; return 0;
 }
```

SSTF:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
        printf("Enter the number of Requests\n"); scanf("%d",&n);
        printf("Enter the Requests sequence\n");
        for(i=0;i<n;i++)
        scanf("%d",&RQ[i]);
        printf("Enter initial head position\n");
        scanf("%d",&initial);

        // logic for sstf disk scheduling

        /* loop will execute until all process is completed*/
        while(count!=n)
        {
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
        d=abs(RQ[i]-initial);
        if(min>d)
        {
                min=d;
                index=i;
        }
```

```
        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];

        // 1000 is for max
        // you can use any number
        RQ[index]=1000; count++;
        }

        printf("Total head movement is %d",TotalHeadMoment); return 0;
}
```

SCAN scheduling:

```
#include<bits/stdc++.h>
using namespace std; int
main(){
        int i,j,k,n,m,sum=0,x,y,h;
        cout<<"Enter the size of disk\n";
        cin>>m;
        cout<<"Enter number of requests\n";
        cin>>n;
        cout<<"Enter the requests\n";
        vector <int> a(n),b;
        for(i=0;i<n;i++){
        cin>>a[i];
        }
        for(i=0;i<n;i++){
        if(a[i]>m){
        cout<<"Error, Unknown position "<<a[i]<<"\n"; return
        0;
        }
        }
        cout<<"Enter the head position\n";
        cin>>h;
        int temp=h;
        a.push_back(h);
        a.push_back(m);
        a.push_back(0);
```

```
        sort(a.begin(),a.end());
        for(i=0;i<a.size();i++){
        if(h==a[i])
        break;
        }
        k=i;
if(k<n/2){ for(i=k;i<a.size();i++){
        b.push_back(a[i]);
        }
        for(i=k-1;i>=0;i--){
        b.push_back(a[i]);
        }
        }
       else{ for(i=k;i>=0;i--){
        b.push_back(a[i]);
        }
        for(i=k+1;i<a.size();i++){
        b.push_back(a[i]);
        }
        }
        temp=b[0]; cout<<temp;
        for(i=1;i<b.size();i++){
        cout<<" -> "<<b[i];
        sum+=abs(b[i]-temp);
        temp=b[i];
        }
        cout<<'\n';
        cout<<"Total head movements = "<< sum<<'\n'; cout<<"Average
        head movement = "<<(float)sum/n<<'\n'; return 0;
```

CScan:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
        printf("Enter the number of Requests\n"); scanf("%d",&n);
        printf("Enter the Requests sequence\n");
        for(i=0;i<n;i++)
        scanf("%d",&RQ[i]);
        printf("Enter initial head position\n");
        scanf("%d",&initial);
        printf("Enter total disk size\n");
        scanf("%d",&size);
        printf("Enter the head movement direction for high 1 and for low 0\n");
        scanf("%d",&move);

        // logic for C-Scan disk scheduling

        /*logic for sort the request array */
        for(i=0;i<n;i++)
        {
        for( j=0;j<n-i-1;j++)
        { if(RQ[j]>RQ[j+1])
        {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
        }
```

```
}
}

int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
break;
}
}

// if movement is towards high value
if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
/*movement max to min disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial=0;
for( i=0;i<index;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];

}
}
// if movement is towards low value else
```

```
        {
        for(i=index-1;i>=0;i--)
        {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
        }
        // last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0); initial
        =size-1;
        for(i=n-1;i>=index;i--)
        {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];

        }
        }

        printf("Total head movement is %d",TotalHeadMoment); return 0;
}
```

Look:

```
#include<bits/stdc++.h>
using namespace std; int
main(){
    int i,j,k,n,m,sum=0,x,y,h; cout<<"Enter
    the size of disk\n"; cin>>m;
    cout<<"Enter number of requests\n";
    cin>>n;
    cout<<"Enter the requests\n";
    vector <int> a(n),l;
    for(i=0;i<n;i++){
        cin>>a[i];
    }
    for(i=0;i<n;i++){
        if(a[i]>m){
            cout<<"Error, Unknown position\n"; return 0;
        }
    }
    cout<<"Enter the head position\n";
    cin>>h;
    a.push_back(h);
```

```
    sort(a.begin(),a.end());
    for(i=0;i<a.size();i++){
        if(h==a[i])
            break;
    }
    k=i;
    if(k<n/2){
          for(i=k;i<a.size();i++){
              l.push_back(a[i]);
        }
        for(i=k-1;i>=0;i--){

l.push_back(a[i]);
        }
    }
    else{
        for(i=k;i>=0;i--){
            l.push_back(a[i]);
        }
        for(i=k+1;i<a.size();i++){
            l.push_back(a[i]);
        }
    }
    int temp=l[0]; cout<<temp;
    for(i=1;i<l.size();i++){
        cout<<" -> "<<l[i]<<' ';
        sum+=abs(l[i]-temp);
        temp=a[i];
    }
    cout<<'\n';
    cout<<"Total head movements = "<< sum<<'\n'; return 0;
```

Output:

FCFS:

```
Enter the size of disk
50
Enter number of requests
4
Enter the requests
8
23
13
2
Enter the head position
0
0 -> 8  -> 23  -> 13  -> 2
Total head movements = 44
```

SSTF:

```
Enter the number of Request
6
Enter the Requests sequence
22
33
1
99
4
2
Enter initial head position
55
Total head movement is 152
```

Scan:

```
Enter the size of disk
50
Enter number of requests
6
Enter the requests
25
30
35
11
22
13
Enter the head position
25
25 -> 22 -> 13 -> 11 -> 0 -> 25 -> 30 -> 35 -> 50
Total head movements = 75
Average head movement = 12.5
```

Cscan:

```
Enter the number of Requests
6
Enter the Requests sequence
22
45
33
1
44
32
Enter initial head position
45
Enter total disk size
50
Enter the head movement direction for high 1 and for low 0
1
```

Look:

```
Enter the size of disk
199
Enter number of requests
8
Enter the requests
98
183
37
122
14
124
54
57
Enter the head position
53
53 -> 54  -> 57  -> 98  -> 122  -> 124  -> 183  -> 37  -> 14
Total head movements = 481
```

Result:

Hence implementation of Disk scheduling algorithms is executed successfully.

ExpNo: 14 Name:Godfrey Ashwanth

Date:08/11/22 RegNo:RA2112704010020

Sequential And Index File Allocation

Aim:To perfom sequential and index file allocation.

Description:

Index file allocation:

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.

Sequential file allocation:

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

Algorithm:

Sequential file allocation:-

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: Allocate the required locations to each in sequential order.

b). Check whether the required locations are free from the selected location

Step 5: Print the results fileno, length , Blocks allocated.

Step 6: Stop the program

Indexed file allocation:-

STEP 1: Start the program.

STEP 2: Get information about the number of files.

STEP 3: Get the memory requirement of each file.

STEP 4: Allocate the memory to the file by selecting random locations.

STEP 5: Check if the location that is selected is free or not.

STEP 6: If the location is allocated set the flag = 1, and if free set flag = 0.

STEP 7: Print the file number, length, and the block allocated.

STEP 8: Gather information if more files have to be stored.

STEP 9: If yes, then go to STEP 2.

STEP 10: If no, Stop the program.

Code:

```
#include<stdio.h
> int main()
{
    char name[10][30];
    int start[10],length[10],num

printf("Enter the number of files to be allocated\n");
scanf("%d",&num);
    int count=0,k,j;
    for(int i=0;i<num;i++)
    {
        printf("Enter the name of the file
        %d\n",i+1); scanf("%s",&name[i][0]);
        printf("Enter the start block of the file
        %d\n",i+1); scanf("%d",&start[i]);
        printf("Enter the length of the file
        %d\n",i+1); scanf("%d",&length[i]);

        for(j=0,k=1;j<num && k<num;j++,k++)
        {
            if(start[j+1]<=start[j] || start[j+1]>=length[j])
            {

            }
            else
            {
                count++;
            }
        }
        if(count==1)
```

```
        {
            printf("%s cannot be allocated disk space\n",name[i]);
        }
    }
    printf("File Allocation Table\n");
    printf("%s%40s%40s\n","File Name","Start Block","Length")

printf("%s%50d%50d\n",name[0],start[0],length[0]);

    for(int i=0,j=1;i<num && j<num;i++,j++)
    {
        if(start[i+1]<=start[i] || start[i+1]>=length[i])
            {
                printf("%s%50d%50d\n",name[j],start[j],length[j]);
            }
    }
    return 0;
}
```

Indexed file allocation:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int f[50], index[50],i, n, st, len, j, c, k,
ind,count=0;
for(i=0;i<50;i+
+) f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
```

```
scanf("%d",&n);
}
else
printf("%d index is already allocated \n",ind); goto x;
}
y: count=0;
for(i=0;i<n;i+
+)
{
scanf("%d", &index[i]);
if(f[index[i]]==0
) count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d          >%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated
\n"); printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1
) goto x;
else
```

exit(0);

}

Output:

Sequence:

```
ashwanth@Gods-Linux:~$ gcc seq.c -oput
ashwanth@Gods-Linux:~$ ./put
Enter the number of files to be alloted
2
Enter the name of the file 1
a
Enter the start block of the file 1
2
Enter the length of thew file 1
6
Enter the name of the file 2
b
Enter the start block of the file 2
10
Enter the length of thew file 2
8
File Allocation Table
File name                                       Start Block
                        Length
a                                                 2
                    6
b,                                                10
                     8
```

Index:

```
ashwanth@Gods-Linux:~$ gcc index.c -oput
ashwanth@Gods-Linux:~$ ./put
Enter the index block:5
Enter no of blocks needed and no of files for the index 5 on the disk:
4
1
2
3
4
Allocated
File Indexed
5------>1:1
5------>2:1
5------>3:1
5------>4:1
Do you want to enter more file(Yes-1/No-0)
```

Result:

Thus, sequential and index file allocation has been executed.

ExpNo:15 Name:Godfrey Ashwanth

Date:14/11/22 RegNo:RA2112704010020

File organization schemes for single level and two level directory

Aim:To perform file organization schemes for single level and two level directory

Description:

The directory contains information about the files, including attributes, location and ownership. Sometimes the directories consisting of subdirectories also. The directory is itself a file, owned by the o.s and accessible by various file management routines.

Single level:

It is the simplest of all directory structures, in this the directory system having only one directory, it consisting of the all files. Sometimes it is said to be root directory.

Two level :

The problem in single level directory is different users may be accidentally using the same names for their files. To avoid this problem, each user need a private directory.

Algorithm:

1. Single Level Directory:
    - Start the program.
    - Declare the count, filename.
    - Read no. of files.
    - Read the file name.
    - Declare the root.
    - Display the files.
    - Stop the program.

2. Two Level Directory:
    - Start the program.
    - Declare the count, filename.
    - Read no. of files.
    - Read the file name.
    - Declare the root directory.
    - Display the files.
    - Stop the program.

Code:

Single:

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

struct
{
        char dname[10],fname[10][10];
        int fcnt;
} dir;

void main()
{
        int i,ch;
        char f[30];
        // clrscr();
        dir.fcnt = 0;
        printf("\nEnter name of directory -- ");
        scanf("%s", dir.dname);

        while(1)
        {
                printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\nEnter your choice -- ");

                scanf("%d",&ch);

                switch(ch)
                {
                        case 1: printf("\nEnter the name of the file -- ");
```

```
        scanf("%s",dir.fname[dir.fcnt]);
        dir.fcnt++;
        break;

case 2: printf("\nEnter the name of the file -- ");
        scanf("%s",f);
        for(i=0;i<dir.fcnt;i++)
        {
                if(strcmp(f, dir.fname[i])==0)
                {
                        printf("File %s is deleted ",f);
                        strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
                        break;
                }
        }

        if(i==dir.fcnt)
                printf("File %s not found",f);
        else
                dir.fcnt--;
        break;

case 3: printf("\nEnter the name of the file -- ");
        scanf("%s",f);
        for(i=0;i<dir.fcnt;i++)
        {
                if(strcmp(f, dir.fname[i])==0)
                {
                        printf("File %s is found ", f);
                        break;
```

```
                    }
                }
                if(i==dir.fcnt)
                    printf("File %s not found",f);
                break;

            case 4: if(dir.fcnt==0)
                printf("\nDirectory Empty");
                else
                {
                    printf("\nThe Files are -- ");
                    for(i=0;i<dir.fcnt;i++)
                        printf("\t%s",dir.fname[i]);
                }
                break;

            default: exit(0);
        }
    }
//getch();
}
```

Two:-

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

struct
{
   char dname[10],fname[10][10];
```

```
    int fcnt;
}dir[10];

void main()
{
    int i,ch,dcnt,k;
    char f[30], d[30];
    // clrscr();
    dcnt=0;

    while(1)
    {
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
        printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter 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)
                    {
```

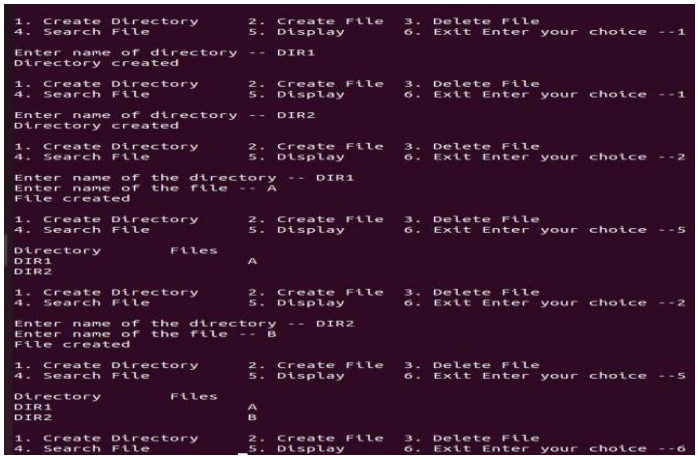
```
            printf("Enter name of the file -- ");
            scanf("%s",dir[i].fname[dir[i].fcnt]);
            dir[i].fcnt++;
            printf("File created");
            break;
          }
        if(i==dcnt)
          printf("Directory %s not found",d);
        break;

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(dcnt==0)
```

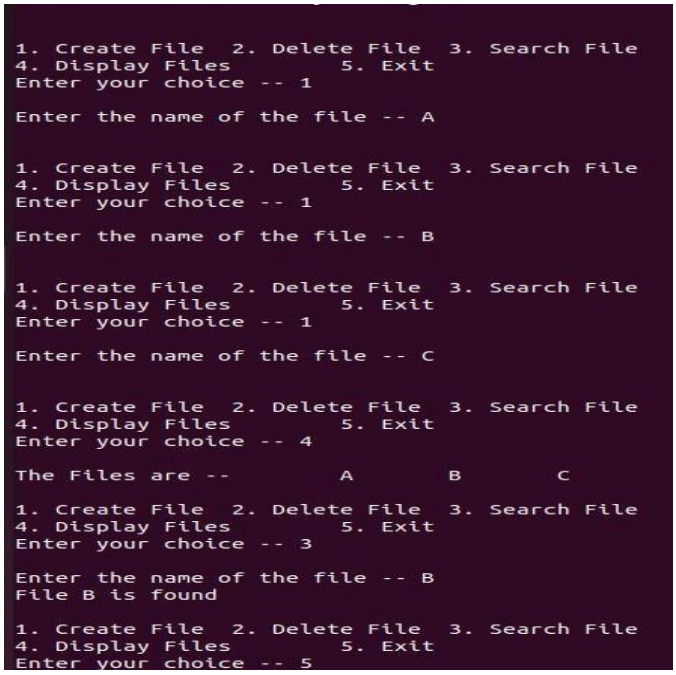
```
            printf("\nNo Directory's ");
        else
        {
            printf("\nDirectory\tFiles");
            for(i=0;i<dcnt;i++)
            {
                printf("\n%s\t\t",dir[i].dname);
                for(k=0;k<dir[i].fcnt;k++)
                printf("\t%s",dir[i].fname[k]);
            }
        }
        break;
    default:exit(0);
  }
}
// getch();
}
```

Output:

Single:



Two level:



Result:

Thus**,** File organization schemes for single level and two level directory is executed successfully **.**

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SCHOOL OF COMPUTING

DEPARTMENT OF DATASCIENCE AND BUSINESS SYSTEMS

21CSC202J OPERATING SYSTEMS

MINI PROJECT REPORT

Title

Name: Godfrey Ashwanth C

Register Number: RA2112704010020

Mail ID: gc606@srmist.edu.in

Department: Data Science And Business Systems

Specialization: Data Science

Semester: 3rd

Team Members

Mugesh Raj K **RA2112704010028**

Abstract

Scheduling of processes/work is done to finish the work on time. CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

Chapter 1

Introduction

This project is divided in two parts

- CPU Scheduling
- Memory Allocation

In CPU scheduling, it is designed in such a way that the program will find the best technique according to its average time of processing. This further helps in saving the CPU time and the suitable way is applied. Some of the techniques are as follows

First Come First Serve (FCFS)

The simplest scheduling algorithm. FCFS 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 the next process starts only after the previous gets fully executed. Here we are considering that arrival time for all processes is 0.

Shortest Job First(SJF)

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next. However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Priority

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on a first come first served basis.

Round Robin

Round Robin is the preemptive process scheduling algorithm. Each process is provided a fixed time to execute, it is called a quantum. Once a process is executed for a given time period, it is preempted and another process executes for a given time period.Context switching is used to save states of preempted processes.

The second part of the project is about memory allocation, which is considered as an important part in allocating memory. Any type of allocation can be done here because multiple ways, as memory allocation is possible in different ways. This is project we are using two type of allocation they are as follow

Best Fit

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Worst Fit

In this allocation technique, the process traverses the whole memory and always searches for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Chapter 2

Review and limitations

Each technique has its own flaw and management of working. According to the burst time and allocated memory they differ with each other in processing the required output. They all can be used in different situations as needed. The project also focuses on using all the available methods to solve the problem with suitable conditions. Each method is implemented and tested with the input given by the user.

Limitations

FCFS
This scheduling method is nonpreemptive, that is, the process will run until it finishes.

SJF
Can't implement this algorithm for CPU scheduling for the short term as we can't predict the length of the upcoming CPU burst

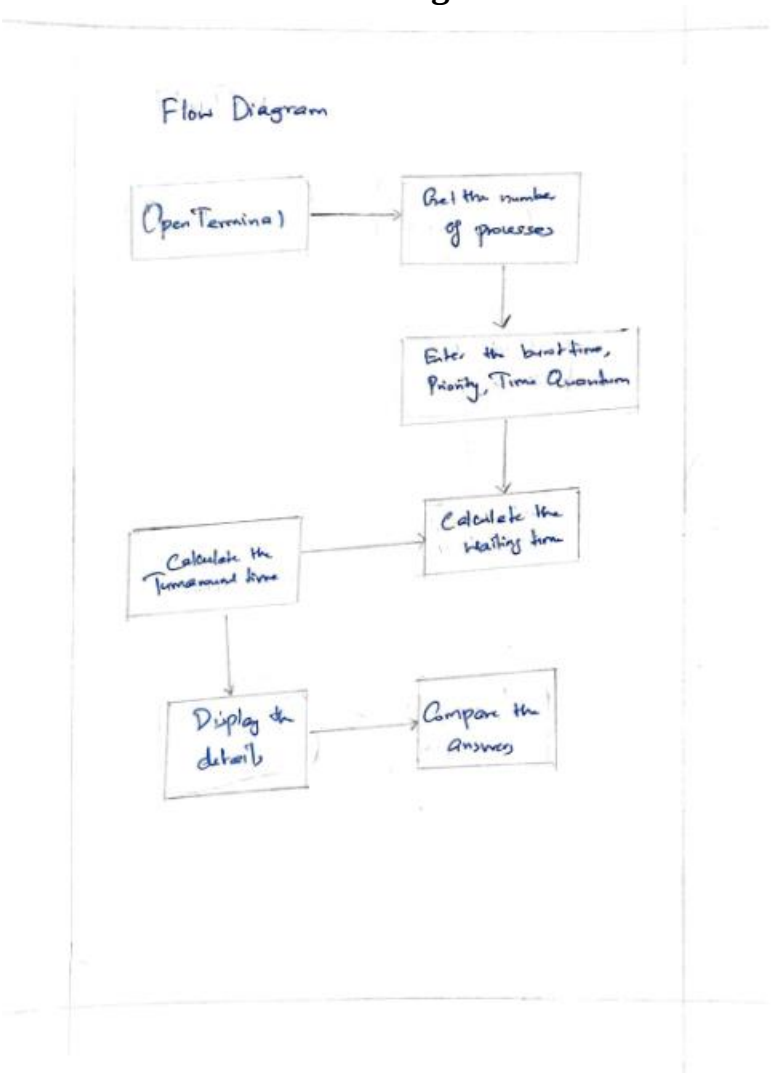
Priority
We lose all the low-priority processes if the system crashes. This process can cause starvation if high-priority processes take too much CPU time.

Round Robin
Setting the quantum too short increases the overhead and lowers the CPU efficiency, but setting it too long may cause a poor response to short processes.

Best fit
It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

Worst fit
It is also a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.

Chapter 3
Flow Diagram



Chapter-4
Modules Description

Modules Used:

- ➔ unistd.h
- ➔ stdio.h
- ➔ Stdlib.h

Unistd.h:

In the C and C++ programming languages, unistd.h is the name of the header file that provides access to the POSIX operating system API. It is defined by the POSIX.1 standard, the base of the Single Unix Specification, and should therefore be available in any POSIX-compliant operating system and compiler. For instance, this includes Unix and Unix-like operating systems, such as GNU variants, distributions of Linux and BSD, and macOS, and compilers such as GCC and LLVM.

On Unix-like systems, the interface defined by unistd.h is typically made up largely of system call wrapper functions such as fork, pipe and I/O primitives (read, write, close, etc.).

Unix compatibility layers such as Cygwin and MinGW also provide their own versions of unistd.h. In fact, those systems provide it along with the translation libraries that implement its functions in terms of win32 functions. E.g. In Cygwin, a header file can be found in /usr/include that sub-includes a file of the same name in /usr/include/sys. Not everything is defined in there but some definitions are done by references to the GNU C standard library headers (like stddef.h) which provide the type size_t and many more. Thus, unistd.h is only a generically defined adaptive layer that might be based upon already existing system and compiler specific definitions. This has the general advantage of not having a possibly concurrent set of header file defined, but one that is built upon the same root which, for this reason, will raise much fewer concerns in combined usage cases.

Stdio.h:

The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header <stdio.h>.[1] The

functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s,[2] and officially became part of the Unix operating system in Version 7.[3]

The I/O functionality of C is fairly low-level by modern standards; C abstracts all file operations into operations on streams of bytes, which may be "input streams" or "output streams". Unlike some earlier programming languages, C has no direct support for random-access data files; to read from a record in the middle of a file, the programmer must create a stream, seek to the middle of the file, and then read bytes in sequence from the stream.

The stream model of file I/O was popularized by Unix, which was developed concurrently with the C programming language itself. The vast majority of modern operating systems have inherited streams from Unix, and many languages in the C programming language family have inherited C's file I/O interface with few if any changes (for example, PHP).

Stdlib.h

The itoa (integer to ASCII) function is a widespread non-standard extension to the standard C programming language. It cannot be portably used, as it is not defined in any of the C language standards; however, compilers often provide it through the header <stdlib.h> while in non-conforming mode, because it is a logical counterpart to the standard library function atoi.

void itoa(int input, char *buffer, int radix)
itoa takes the integer input value input and converts it to a number in base radix. The resulting number (a sequence of base-radix digits) is written to the output buffer buffer.

Depending on the implementation, itoa may return a pointer to the first character in buffer, or may be designed so that passing a null buffer causes the function to return the length of the string that would have been written into a valid buffer.

For converting a number to a string in base 8 (octal), 10 (decimal), or 16 (hexadecimal), a Standard-compliant alternative is to use the standard library function sprintf.

Chapter 5
Implementation Requirements

The Requirements:

- ➔ Linux Environment
- ➔ Virtual Machine
- ➔ GCC compiler
- ➔ Base Knowledge in C

Linux:

Linux is an open-source Unix-like operating system based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds. Linux is typically packaged as a Linux distribution.

Distributions include the Linux kernel and supporting system software and libraries, many of which are provided by the GNU Project. Many Linux distributions use the word "Linux" in their name, but the Free Software Foundation uses the name "GNU/Linux" to emphasize the importance of GNU software, causing some controversy.

Popular Linux distributions include Debian, Fedora Linux, and Ubuntu, which in itself has many different distributions and modifications, including Lubuntu and Xubuntu. Commercial distributions include Red Hat Enterprise Linux and SUSE Linux Enterprise. Desktop Linux distributions include a windowing system such as X11 or Wayland, and a desktop environment such as GNOME or KDE Plasma. Distributions intended for servers may omit graphics altogether, or include a solution stack such as LAMP. Because Linux is freely redistributable, anyone may create a distribution for any purpose.

Linux was originally developed for personal computers based on the Intel x86 architecture, but has since been ported to more platforms than any other operating system. Because of the dominance of the Linux-based Android on smartphones, Linux, including Android, has the largest installed base of all general-purpose operating systems, as of May 2022. Although Linux is, as of November 2022, used by only around 2.6 percent of desktop computers, the Chromebook, which runs the Linux kernel-based Chrome OS, dominates the US K–12 education market and represents nearly 20 percent of sub-$300 notebook sales in the US.Linux is the leading operating system on servers (over 96.4% of the top 1 million web servers' operating systems are Linux), leads other big iron systems such as mainframe computers, and is the only OS used on TOP500 supercomputers (since November 2017, having gradually eliminated all competitors).

Linux also runs on embedded systems, i.e. devices whose operating system is typically built into the firmware and is highly tailored to the system. This includes routers, automation controls, smart home devices, video game consoles,televisions (Samsung and LG Smart TVs use Tizen and WebOS, respectively), automobiles (Tesla, Audi, Mercedes-Benz, Hyundai, and Toyota all rely on Linux), spacecraft (Falcon 9's and Dragon 2's avionics use a customized version of Linux), and rovers (the Mars 2020 mission).

Gcc:

The GNU Compiler Collection (GCC) is an optimizing compiler produced by the GNU Project supporting various programming languages, hardware architectures and operating systems. The Free Software Foundation (FSF) distributes GCC as free software under the GNU General Public License (GNU GPL). GCC is a key component of the GNU toolchain and the standard compiler for most projects related to GNU and the Linux kernel. With roughly 15 million lines of code in 2019, GCC is one of the biggest free programs in existence. It has played an important role in the growth of free software, as both a tool and an example.

When it was first released in 1987 by Richard Stallman, GCC 1.0 was named the GNU C Compiler since it only handled the C programming language.[1] It was extended to compile C++ in December of that year. Front ends were later developed for Objective-C, Objective-C++, Fortran, Ada, D and Go, among others. The OpenMP and OpenACC specifications are also supported in the C and C++ compilers.

GCC has been ported to more platforms and instruction set architectures than any other compiler, and is widely deployed as a tool in the development of both free and proprietary software. GCC is also available for many embedded systems, including ARM-based and Power ISA-based chips.

As well as being the official compiler of the GNU operating system, GCC has been adopted as the standard compiler by many other modern Unix-like computer operating systems, including most Linux distributions. Most BSD family operating systems also switched to GCC shortly after its release, although since then, FreeBSD, OpenBSD and Apple macOS have moved to the Clang compiler,[9] largely due to licensing reasons. GCC can also compile code for Windows, Android, iOS, Solaris, HP-UX, AIX and DOS.

State which Algorithm is the most effective and time saving for the following processes
Use atleast four Algorithms:

Processes	Burst Time	Arrival Time	Priority
P1	2	2	
P2	6	5	
P3	4	0	
P4	7	0	
P5	4	7	

i) Round Robin with Time Quantum = 3
ii) Priority Scheduling
iii) Shortest Job First - Non Preemptive
iv) First Come First Serve - Non Preemptive

i) Round Robin

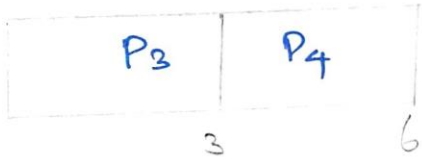
TQ = 3 units

By looking at the table Two processes - P_3, P_4 arrive at the same time - '0'.

The Ready Queue:

P_3, P_4.

Gantt chart:



Now $P_3 = 4 - 3 = 1$

$P_4 = 7 - 3 = 4$

By the time P_4 finishes P_1 arrives at time 2 so the ready queue becomes:

P_3, P_4, P_1

since P_3 has still 1 unit of time left, it gets added to the ready queue:

P_3, P_4, P_1, P_3, P_2

The Gantt chart:

P_3 finishes executing.

P_1 finishes executing

Ready Queue becomes:

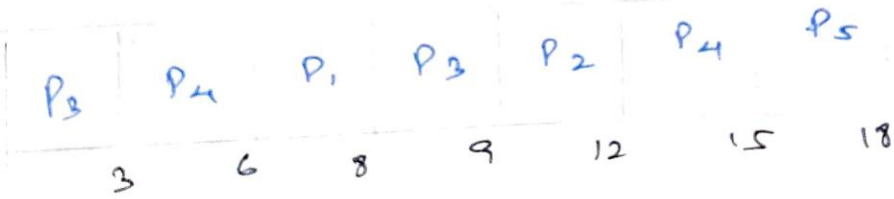
$P_3, P_4, P_1, P_3, P_2, P_4$

Since P_4 has 4 more units left to execute
By the time P_2's TQ was finished,
P_5 arrives in the ready Queue.

So the Ready Queue Now becomes:

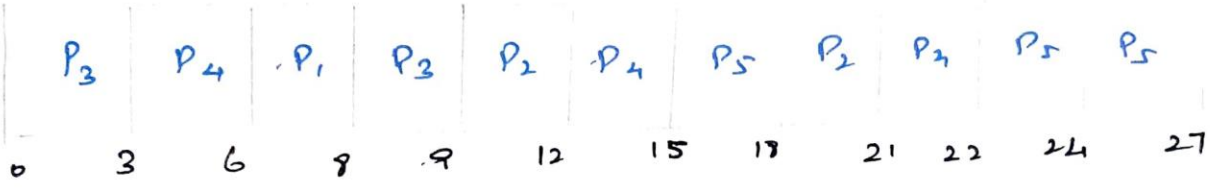
$P_3, P_4, P_1, P_3, P_2, P_4, P_5$

The Gantt chart now:



P_2 still has 3 units left,
P_4 still has 1 unit left,
P_5 still has 3 units left.

FINAL GANTT CHART:



P	Burst Time	Arrival Time	Turnaround Time	Waiting Time
P_1	2	2	6	4
P_2	6	5	16	10
P_3	4	0	9	5
P_4	7	0	22	15
P_5	8	7	20	12

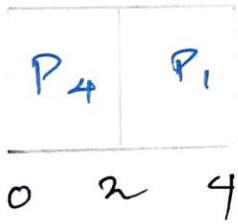
Avg Waiting Time = 9.2
Avg. Turnaround Time = 14.6

ii) Priority Scheduling:

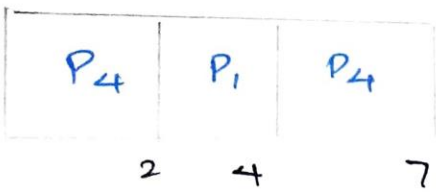
Two process arrive at the same time, - P_3, P_4
& P_3 comes first, but P_4 is of higher priority

Gantt chart

P_4 executes for 2 unit, then by the time 2, P_1 arrives and is of the highest priority so P_4 is preempted by P_1



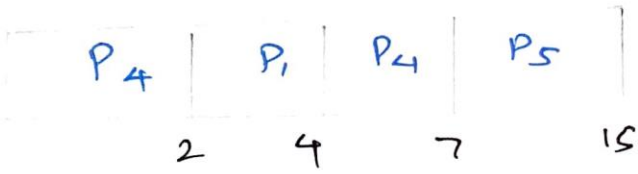
P_1 executes for 2 units, by the time 4, P_1 has finished its process, so P_4 continues its process since it has a higher priority, & P_5 hasn't arrived yet.



P_2 arrive on time 5, since it is of low priority, it does not preempt P_4

P_4 executes till '7', and is preempted by P_5, because it is of higher priority.

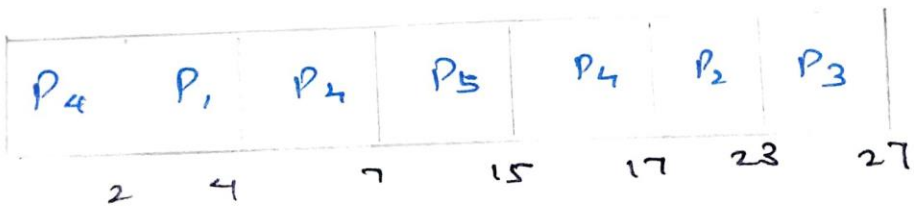
The gantt chart becomes:



P_5 executes until it completes.

After P_5, P_4 continues its execution till it finishes and then, it is followed by P_2 (which is of next priority) & then the lowest priority gets executed (P_3)

FINAL GANTT CHART:



P	Burst Time	Arrival Time	Turnaround Time	Waiting Time
1	2	2	2	0
2	6	5	18	12
3	4	0	27	23
4	7	0	17	10
5	8	7	8	0

Avg. waiting Time : $\frac{12+23+10}{5}$

$= 9$

Avg. Turnaround time $= 14.4$

(iii) Shortest Job First:

Sorting based on the burst time:-

Process		B.T
P_1	-	2
P_3	-	4
P_2	-	6
P_4	-	7
P_5	-	8

Sorting based on Arrival Time

				B.T
P_3	-	0	-	4
P_4	-	0	-	7
P_1	-	2	-	2
P_2	-	5	-	6
P_5	-	7	-	8

P_3, P_4 arrive at the same time
comparing their burst time, P_3 is executed first

P_3	P_3	P_{21}	P_2	P_4	P_5
2	4	6	12	19	27

P	Burst Time	Arrival Time	Turnaround Time	Waiting Time
1	2	2	4	2
2	6	5	7	1
3	4	0	4	0
4	7	0	12	5
5	8	7	20	12

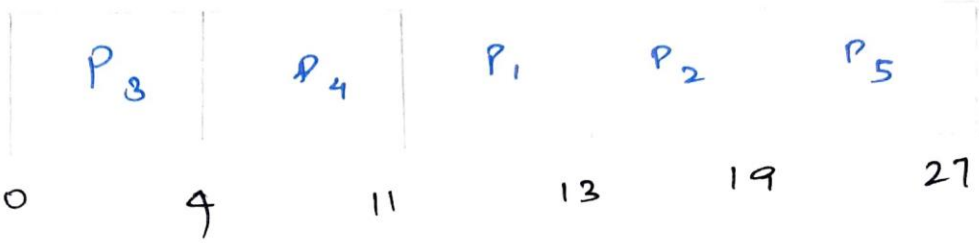
Avg. Wt time = $20/5 = 4$

Avg TAT time = $47/5 = 9.4$

iv) FCFS.

$P_3 \rightarrow P_4 \rightarrow P_1 \rightarrow P_2 \rightarrow P_5$

Gantt chart:



P	Burst Time	Arrival Time	Turnaround Time	Waiting Time
1	2	2	11	9
2	6	5	14	8
3	4	0	4	0
4	7	0	11	4
5	8	7	20	12

Avg wt time = 6.6

Avg TAT time = 12

Comparing all the four Algorithms.

S.No	Algorithm	Avg. Wt time	Avg TAT time
1	FCFS	6.6	12
2	SRTF Preemptive	4	9.4
3	Priority - Preemptive	9	14.4
4	Round Robin (TQ-3) - Preemptive	9.2	14.6

Conclusion.

Applying all the four scheduling algorithms

① → FCFS CPU scheduling. (NP)

② → SRTF CPU scheduling (P)

③ → Priority CPU scheduling (P)

④ → Round Robin with TQ - 3 (P)

①: Using FCFS

It seems to be ~~the~~ fair and the avg. Turnaround Time doesn't seem to be so bad Compared to Round Robin scheduling

Wt time : 6.6 ms

Turnaround time : 12

②: Using SRTF.

This seems to be the best approach among all the algorithms I have used. This algorithm provides the shortest Waiting time and also the shortest Turnaround Time

Waiting Time (Avg) :- 4 ms (shortest)

Turnaround time (Avg) = 9.4 ms (shortest)

(3) Using Priority Scheduling

Using this approach yields better results than Round Robin Scheduling both in terms of average waiting time and average turn around time.

Waiting Time (Avg) = 8 ms

Turn Around Time (Avg) = 14.4 ms

But this approach does worse than FCFS and SRTF Scheduling.

(4) Using Round Robin with Time Quantum = 3

This approach yields the worst result out of these 4 algorithms, yielding a:

Avg waiting Time = 9.2

Avg Turnaround Tim = 14.6

Hence, Shortest Remaining Time proves to be the best approach out of these 4 algorithms

State which fit is efficient and memory saving for the following blocks and process size.

Memory size in order :- (All in KB)

100, 50, 30, 120, 35

Process size in order :- (All in KB)

40, 10, 30, 60

i) Best fit

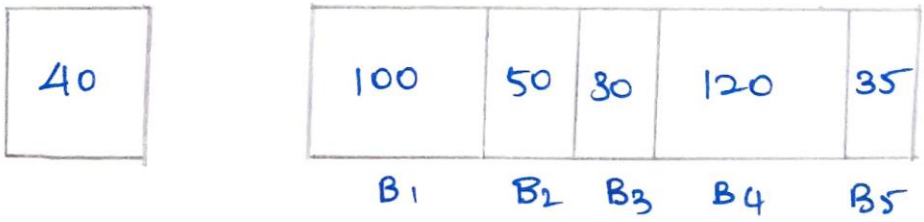
ii) Worst fit.

100	50	30	120	35
Block1	Block2	Block3	Block4	Block5

40	10	30	60
Process1	Process2	Process3	Process4

Best fet Allocation

P_1

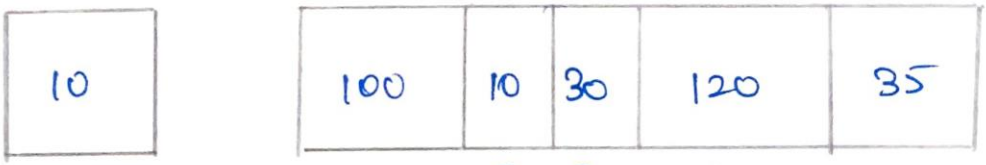


Block size	Available	Memory wastage
B1	Yes	100 − 40 = 60
B_2	Yes	50 − 40 = 10
B_3	NO	—
B_4	Yes	120 − 40 = 80
B_5	NO	—

$B_2 = 50 - 40 = 10$ KB

↳ It is considered the best

P_2



Block size	Available	Memory wastage
B_1	yes	100 − 10 = 90
B_2	NO	—
B_3	yes	30 − 10 = 20
B_4	yes	120 − 10 = 110
B_5	yes	35 − 10 = 25

$B_2 = 30 - 10 = 20$KB → It best

P_3

30

100	10	20	120	35
B_1	B_2	B_3	B_4	B_5

Block size	Available	Memory wastage
B_1	Yes	100 - 30 = 70
B_2	-	
B_3	-	
B_4	Yes	120 - 30 = 90
B_5	Yes	35 - 30 = 5

B_5 = 35 - 30 = 5KB → Is best

P_4

60

100	10	20	120	35
B_1	B_2	B_3	B_4	B_5

Block size	Available	Memory wastage
B_1	Yes	100 - 60 = 40
B_2	NO	-
B_3	NO	-
B_4	Yes	120 - 60 = 60
B_5	NO	-

B_1 = 100 - 60 = 40 → Is best

Worst fit: Allocation

P_1

40

100	50	30	120	35
B_1	B_2	B_3	B_4	B_5

Block Size	Available	Memory Wastage
B_1	Yes	100 - 40 = 60
B_2	Yes	50 - 40 = 10
B_3	NO	-
B_4	Yes	120 - 40 = 80
B_5	NO	-

$B_4 = 120 - 40 = 80$ → Worst fit

P_2

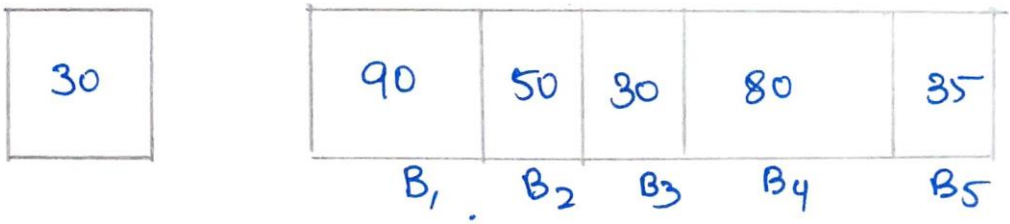
10

100	50	30	80	35
B_1	B_2	B_3	B_4	B_5

Block Size	Available	Memory wastage
B_1	Yes	100 - 10 = 90
B_2	Yes	50 - 10 = 40
B_3	Yes	30 - 10 = 20
B_4	Yes	80 - 10 = 70
B_5	Yes	35 - 10 = 25

$B_1 = 100 - 10 = 90$ → Worst fit.

P_3

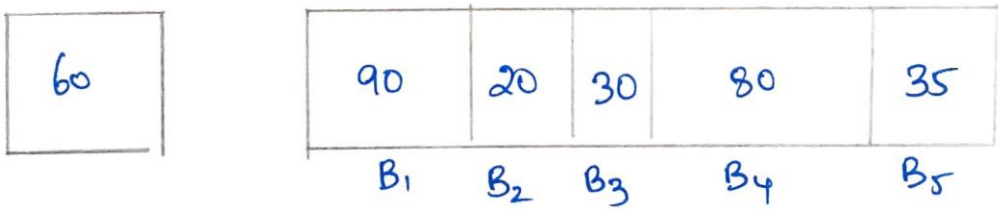


Block Size	Available	Memory Wastage
B_1	Yes	90-30 = 60
B_2	Yes	50-30 = 20
B_3	Yes	30-30 = 0
B_4	Yes	80-30 = 50
B_5	Yes	35-30 = 5

Here B_1 is avoided with B_4 because they are already filled with process. Hence

$B_2 = 50 - 30 = 20$ → Best fit

P_4



Only B_3 & B_5 are available, But the process size is bigger than the block size. Hence, P_4 (60) remains unallocated.

As it can not be fit by any free blocks.

The total memory present in the block after allocation is

i) Best fit

$100 * 40 + 10 + 20 + 120 + 5$

$\Rightarrow$ 195 KB

ii) Worst fit

$90 + 20 + 30 + 80 + 35$

[Excluding P_4 (60) as it can not be allocated]

$\Rightarrow$ 255 KB

Hence by comparing these two values we can conclude that Best fit is more efficient than worst fit (only in this case of process size & block size).

The efficiency may depend on the size and vary for different cases.

Output

FCFS:

```
mugesh@mugesh-linux:~$ gcc FCFS.c
mugesh@mugesh-linux:~$ ./a.out
Processes  Burst   Waiting Turn around

1          5               0       5
2          8               5       13
3          12              13      25Average waiting time = 6.000000
Average turn around time = 14.333333
mugesh@mugesh-linux:~$
```

SJF:

```
mugesh@mugesh-linux:~$ gcc sjf.c
mugesh@mugesh-linux:~$ ./a.out
Enter number of process: 4
Enter Burst Time:
P1: 2
P2: 3
P3: 4
P4: 1
P      BT      WT      TAT
P4      1      0       1
P1      2      1       3
P2      3      3       6
P3      4      6       10
Average Waiting Time= 2.500000
mugesh@mugesh-linux:~$
```

Priority:

```
mugesh@mugesh-linux:~$ gcc priority.c
mugesh@mugesh-linux:~$ ./a.out
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:99
Priority:1

P[2]
Burst Time:65
Priority:2

P[3]
Burst Time:4
Priority:1

P[4]
Burst Time:5
Priority:3

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              99                  0                  99
P[3]              4                   99                 103
P[2]              65                  103                168
P[4]              5                   168                173

Average Waiting Time=92
Average Turnaround Time=135
```

Round Robin

```
mugesh@mugesh-linux:~$ gcc round.c
mugesh@mugesh-linux:~$ ./a.out
Enter Total Process:     6
Enter Arrival Time and Burst Time for Process Process Number 1 :0
3
Enter Arrival Time and Burst Time for Process Process Number 2 :2
4
Enter Arrival Time and Burst Time for Process Process Number 3 :4
2
Enter Arrival Time and Burst Time for Process Process Number 4 :6
2
Enter Arrival Time and Burst Time for Process Process Number 5 :8
2
Enter Arrival Time and Burst Time for Process Process Number 6 :10
2
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[3]    |       2       |       0
P[4]    |       2       |       0
P[5]    |       2       |       0
P[6]    |       2       |       0
P[1]    |       13      |       10
P[2]    |       13      |       9

Average Waiting Time= 3.166667
mugesh@mugesh-linux:~$
```

Best Fit:

```
mugesh@mugesh-linux:~$ gcc bf.c
mugesh@mugesh-linux:~$ ./a.out

Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:9
Block no.5:3

Enter the size of the processes :-
Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               5               3               2
2               4               3               5               1
3               7               4               9               2
4               12              2               15              3mugesh@muge
```

Worst Fit:

```
mugesh@mugesh-linux:~$ ./a.out
enter block size and process size5
4
Enter the blocks100
500
200
300
600
Enter the processes200
400
100
420

Process No.     Process Size    Block no.
   1            200             5
   2            400             2
   3            100             5
   4            420             Not Allocated
mugesh@mugesh-linux:~$
```

CONCLUSION

We have successfully implemented the concepts of *CPU SCHEDULING AND MEMORY ALLOCATION* in OS , and have concluded that every methods is useful in its own way, and have also provided the outputs run on a example set. This program is also capable of finding the mentioned statistics for any other dataset.

Appendix A

Source Code:

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;           //contains process number
   }

   //sorting burst time, priority and process number in ascending order using selection sort
   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; //waiting time for first process is zero

    //calculate waiting time
    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;      //average waiting time
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];         //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;      //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

return 0;
}
```

Round Robin:

```
#include<stdio.h>

int main()
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
```

```
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
 }
 printf("Enter Time Quantum:\t");
 scanf("%d",&time_quantum);
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 for(time=0,count=0;remain!=0;)
 {
        if(rt[count]<=time_quantum && rt[count]>0)
        {
        time+=rt[count];
        rt[count]=0;
        flag=1;
        }
        else if(rt[count]>0)
        {
        rt[count]-=time_quantum;
        time+=time_quantum;
        }
        if(rt[count]==0 && flag==1)
        {
        remain--;
        printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
        wait_time+=time-at[count]-bt[count];
        turnaround_time+=time-at[count];
        flag=0;
        }
        if(count==n-1)
        count=0;
        else if(at[count+1]<=time)
        count++;
        else
        count=0;
 }
 printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
 printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

 return 0;
}
```

Best fit:

```
#include&lt;stdio.h&gt;

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf(&quot;\n\t\t\tMemory Management Scheme - Best Fit&quot;);
printf(&quot;\nEnter the number of blocks:&quot;);
scanf(&quot;%d&quot;,&amp;nb);
printf(&quot;Enter the number of processes:&quot;);

scanf(&quot;%d&quot;,&amp;np);
printf(&quot;\nEnter the size of the blocks:-\n&quot;);
for(i=1;i&lt;=nb;i++)
{
printf(&quot;Block no.%d:&quot;,i);
scanf(&quot;%d&quot;,&amp;b[i]);
}
printf(&quot;\nEnter the size of the processes :-\n&quot;);
for(i=1;i&lt;=np;i++)
{
printf(&quot;Process no.%d:&quot;,i);
scanf(&quot;%d&quot;,&amp;p[i]);
}
for(i=1;i&lt;=np;i++)
{
for(j=1;j&lt;=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp&gt;=0)
if(lowest&gt;temp)
{
parray[i]=j;
lowest=temp;
}
}

}
fragment[i]=lowest;
```

```
barray[parray[i]]=1;
lowest=10000;
}
printf(&quot;\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment&quot;);
for(i=1;i&lt;=np &amp;&amp; parray[i]!=0;i++)
printf(&quot;\n%d\t\t%d\t\t%d\t\t%d\t\t%d&quot;,i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

Worst:-

```
#include &lt;stdio.h&gt;
#define MAX 25

int main()
{
int frag[MAX],b[MAX],f[MAX],i,j,nb,nf,temp,highest=0;
static int bf[MAX],ff[MAX];

printf(&quot;\nEnter the number of blocks:&quot;);
scanf(&quot;%d&quot;,&amp;nb);

printf(&quot;Enter the number of files:&quot;);
scanf(&quot;%d&quot;,&amp;nf);

printf(&quot;\nEnter the size of the blocks:-\n&quot;);
for(i=1;i&lt;=nb;i++)

{
printf(&quot;Block %d:&quot;,i);
scanf(&quot;%d&quot;,&amp;b[i]);
}

printf(&quot;Enter the size of the files :-\n&quot;);
for(i=1;i&lt;=nf;i++)
{
printf(&quot;File %d:&quot;,i);
scanf(&quot;%d&quot;,&amp;f[i]);
}

for(i=1;i&lt;=nf;i++)
{
for(j=1;j&lt;=nb;j++)
{
```

```
if(bf[j]!=1)
{
temp=b[j]-f[i];

if(temp&gt;=0)
if(highest&lt;temp)
{
ff[i]=j; highest=temp; }
}
}
frag[i]=highest;

bf[ff[i]]=1;
highest=0;
}
printf(&quot;\nFile_no:\tFile_size
:\tBlock_no:\tBlock_size:\tFragement&quot;);
for(i=1;i&lt;=nf;i++)
printf(&quot;\n%d\t\t%d\t\t%d\t\t%d\t\t%d&quot;,i,f[i],ff[i],b[ff[i]],frag[i]);
getchar();
}
```

Appendix B
Github Profile

Your Repositories (github.com)

CPU Scheduling in Operating System | Studytonight

https://github.com/godfreyash