DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	01
DOP	04/02/2022
DOS	14/02/2022

Experiment No. 01

Aim : To explore basic Linux Commands and System Calls.

Theory :

The Linux command line is a text interface to your computer. Often referred to as the shell,terminal, console, prompt or various other names,it can give the appearance of being complex and confusing to use but once you understand it, it's really easy and helpful.

Linux provides a powerful command-line interface compared to other operating systems such as Windows and MacOS. We can do basic work and advanced work through its terminal. We can do some basic tasks such as creating a file, deleting a file, moving a file, and more.

- **ls -** list directory contentsList information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort isspecified.
- **mkdir -** creates a directory with a given nameCreate the DIRECTORY(ies), if they do not already exist.
- **cd -** change the working directoryThe cd utility shall change the working directory of the current shell executionenvironment
- **echo command -** This command is used to display a text or a string to the standard output or a file. Especially for printing purposes.

- **ps –** report a snapshot of the current processes.ps displays information about a selection of the active processes. If you want a repetitive update of the selection
and the displayed information, use top instead.
- **grep, egrep, fgrep -** print lines that match patterns.
grep searches for PATTERNS in each FILE. PATTERNSis one or more patterns separated by newline characters, and grep prints each line that matches a pattern.Typically PATTERNS should be quoted when grep is use din a shell command.
- **who command -** you display the users currently logged in to your Operating System.
- **getuid, geteuid -** get user identity
getuid() returns the real user ID of the calling process.geteuid() returns the effective user ID of the calling process.
- **setuid -** set user identity setuid()
sets the effective user ID of the calling process. If the calling process is privileged (more precisely: if the process has theCAP_SETUID capability in its user namespace), the real UID and saved set-user-ID arealso set.
- **sort -** sort lines of text files
Write sorted concatenation of all FILE(s) to standardoutput. With no FILE, or whenFILE is -, read standard input.

- **| - pipelining** A pipe is a form of redirection (transfer of standardoutput to some other destination)that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing.
- **rm -** remove files or directories

If the -I or --interactive=once option is given, and there are more than three files or the -r, -R, or --recursive are given, then rm prompts the user for whether to proceed with the entire operation. If the response is not affirmative, the entire command is aborted.

- **cal -** display a calendar

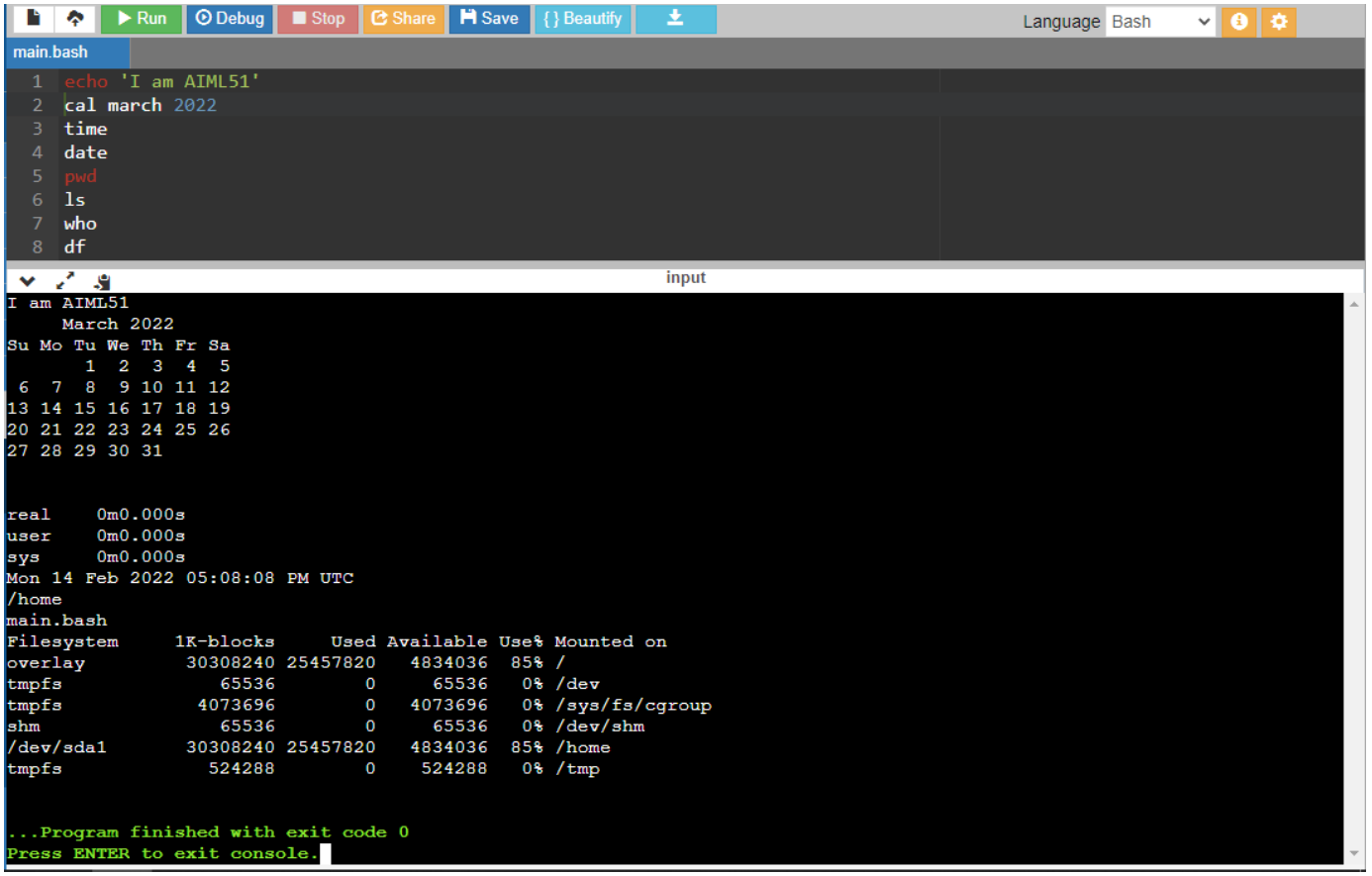
cal displays a simple calendar. If no arguments are specified, the current month isdisplayed. The month may be specified as a number(1-12), as a month name or as an abbreviated month name according to the current locales.

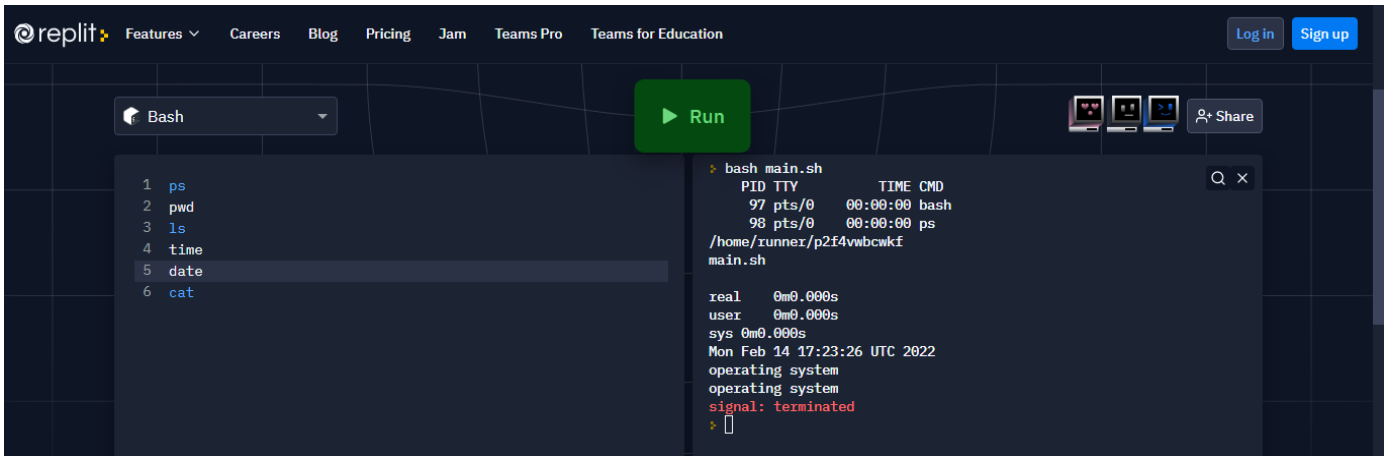
- **time -** get time in seconds

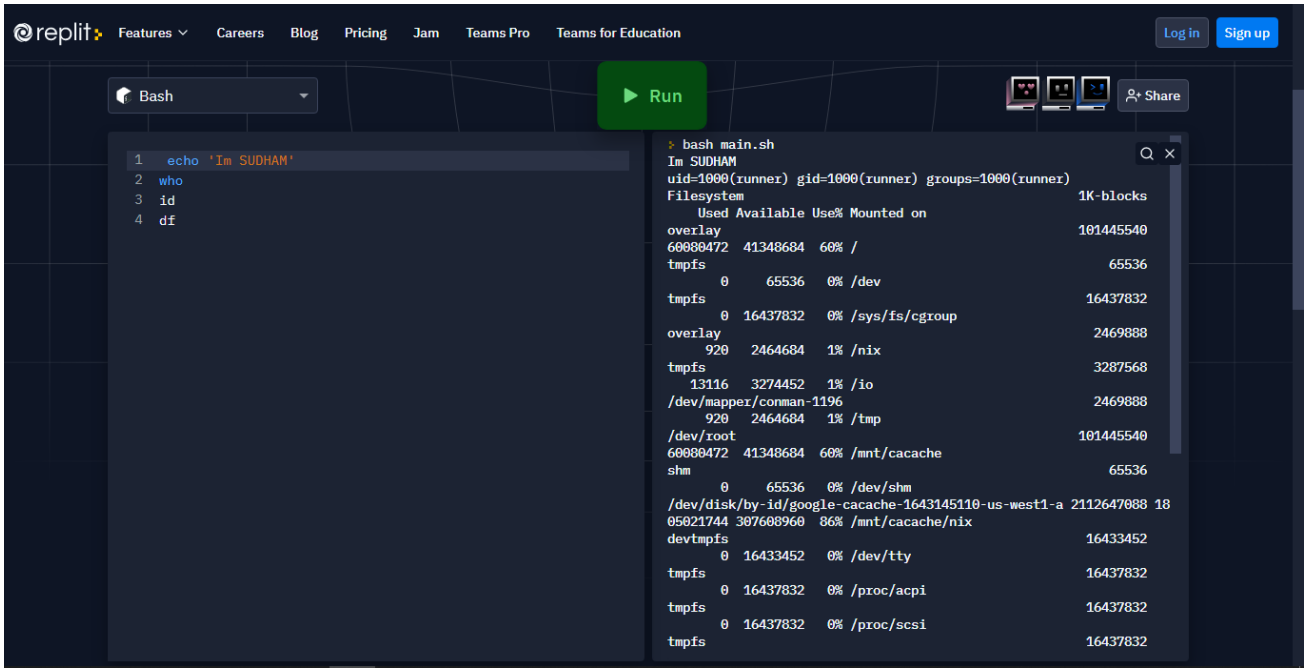
time() returns the time as the number of seconds since the Epoch, 1970-01-0100:00:00 +0000 (UTC).13.pwd - print name of current/working directoryPrint the full filename of the current working directory

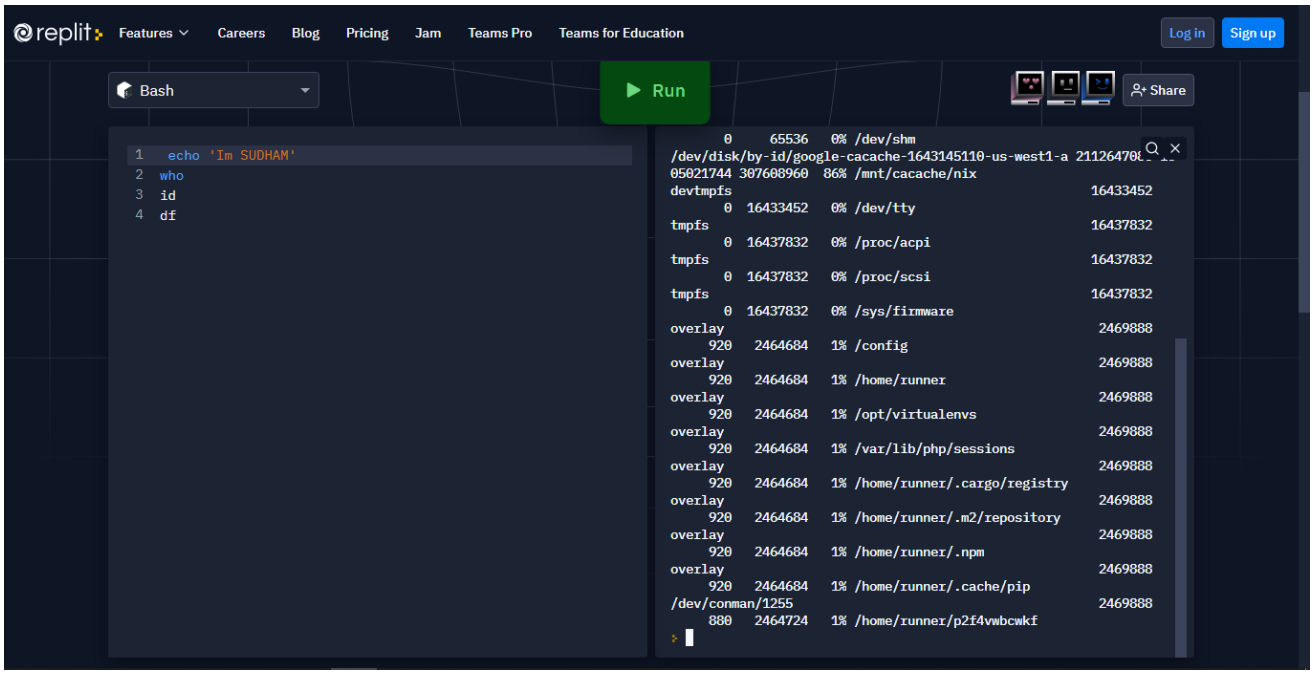
- **cat command -** This command can read, modify or concatenate text files. It also displays file contents.

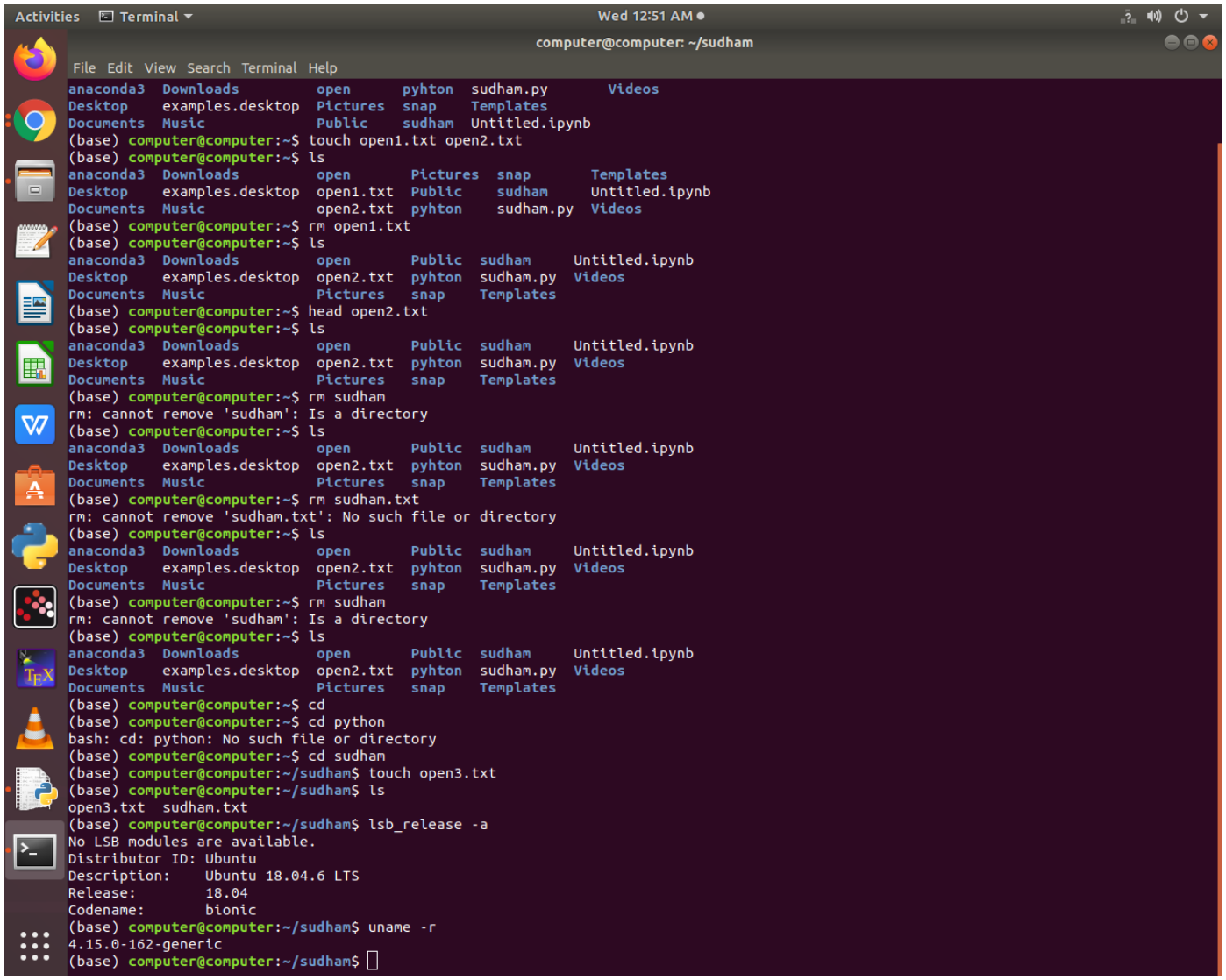
Program(input)/Output -






replit
Features
Careers
Blog
Pricing
Jam
Teams Pro
Teams for Education
Log in
Sign up
Bash
Run
Share
1 echo 'Im SUDHAM'
2 who
3 id
4 df
bash main.sh
Im SUDHAM
uid=1000(runner) gid=1000(runner) groups=1000(runner)
Filesystem 1K-blocks
Used Available Use% Mounted on
overlay 101445540
60080472 41348684 60% /
tmpfs 65536
0 65536 0% /dev
tmpfs 16437832
0 16437832 0% /sys/fs/cgroup
overlay 2469888
920 2464684 1% /nix
tmpfs 3287568
13116 3274452 1% /io
/dev/mapper/conman-1196 2469888
920 2464684 1% /tmp
/dev/root 101445540
60080472 41348684 60% /mnt/cacache
shm 65536
0 65536 0% /dev/shm
/dev/disk/by-id/google-cacache-1643145110-us-west1-a 2112647088 18
05021744 307608960 86% /mnt/cacache/nix
devtmpfs 16433452
0 16433452 0% /dev/tty
tmpfs 16437832
0 16437832 0% /proc/acpi
tmpfs 16437832
0 16437832 0% /proc/scsi
tmpfs 16437832


replit
Features
Careers
Blog
Pricing
Jam
Teams Pro
Teams for Education
Log in
Sign up
Bash
Run
Share
1 echo 'Im SUDHAM'
2 who
3 id
4 df
0 65536 0% /dev/shm
/dev/disk/by-id/google-cacache-1643145110-us-west1-a 21126470
05021744 307608960 86% /mnt/cacache/nix
devtmpfs 16433452
0 16433452 0% /dev/tty
tmpfs 16437832
0 16437832 0% /proc/acpi
tmpfs 16437832
0 16437832 0% /proc/scsi
tmpfs 16437832
0 16437832 0% /sys/firmware
overlay 2469888
920 2464684 1% /config
overlay 2469888
920 2464684 1% /home/runner
overlay 2469888
920 2464684 1% /opt/virtualenvs
overlay 2469888
920 2464684 1% /var/lib/php/sessions
overlay 2469888
920 2464684 1% /home/runner/.cargo/registry
overlay 2469888
920 2464684 1% /home/runner/.m2/repository
overlay 2469888
920 2464684 1% /home/runner/.npm
overlay 2469888
920 2464684 1% /home/runner/.cache/pip
/dev/conman/1255 2469888
880 2464724 1% /home/runner/p2f4vwbcwkf



Conclusion : We have seen different Linux commands and executed them to see how it interacts with the system; commands like creating directory, creating and deleting files, etc.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	02
DOP	08/02/2022
DOS	15/02/2022

Experiment No. 02

Aim - Linux Shell Script

- Display OS version, release number, kernel version
- Display top 10 processes in descending order
- Display processes with the highest memory usage
- Display current logged in use and log name

Theory -

A shell is a command-line interpreter and typical operations performed by shell script include file manipulation, program execution, and printing text.

- Display OS version, release number, komel version

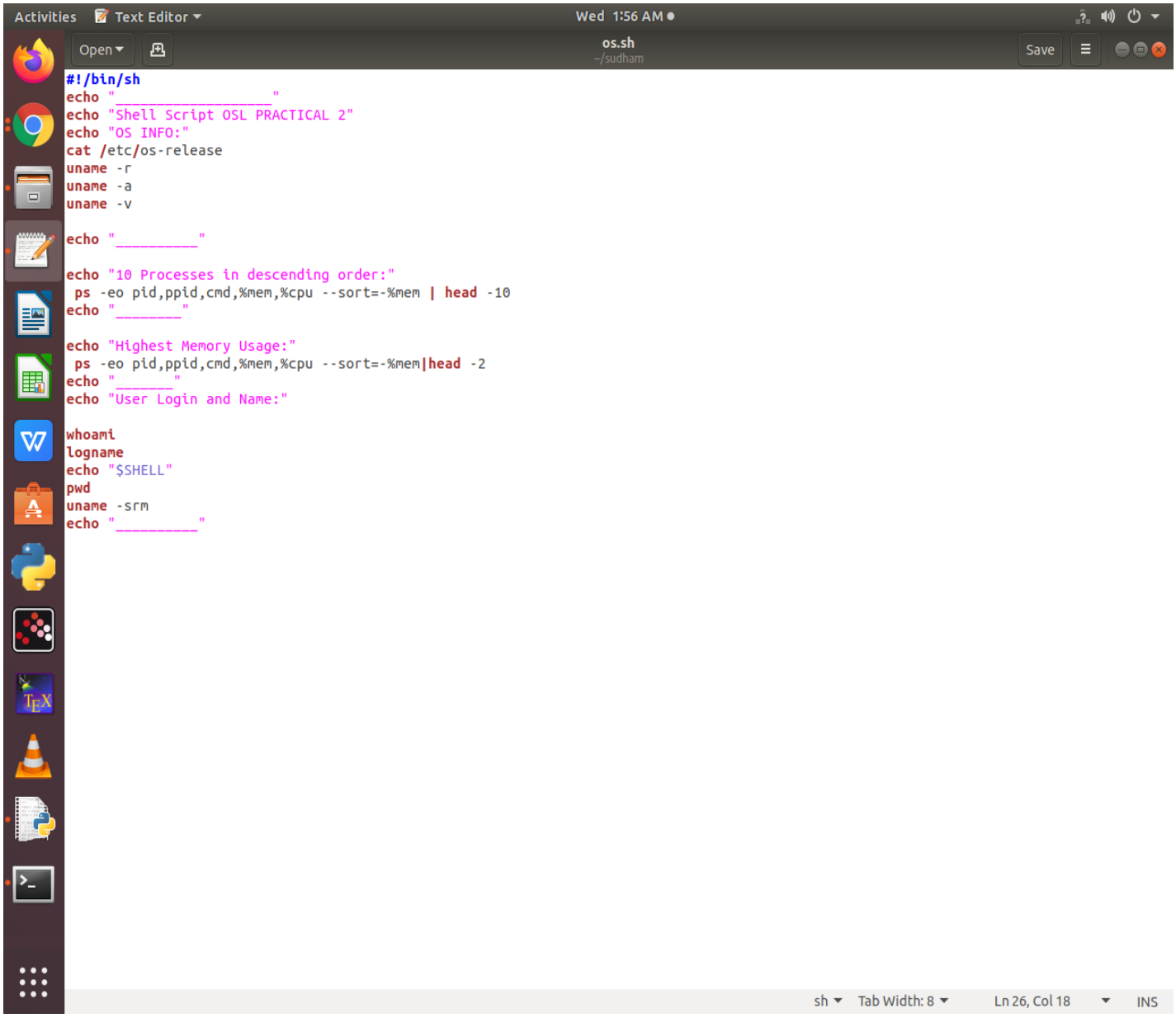
Linux commands display system information like NAME, VERSION, ID and even VERSION_ID among other things. Since Nix systems like to be command bases, there are many commands which help in our need.

- Display top 10 processes in descending order

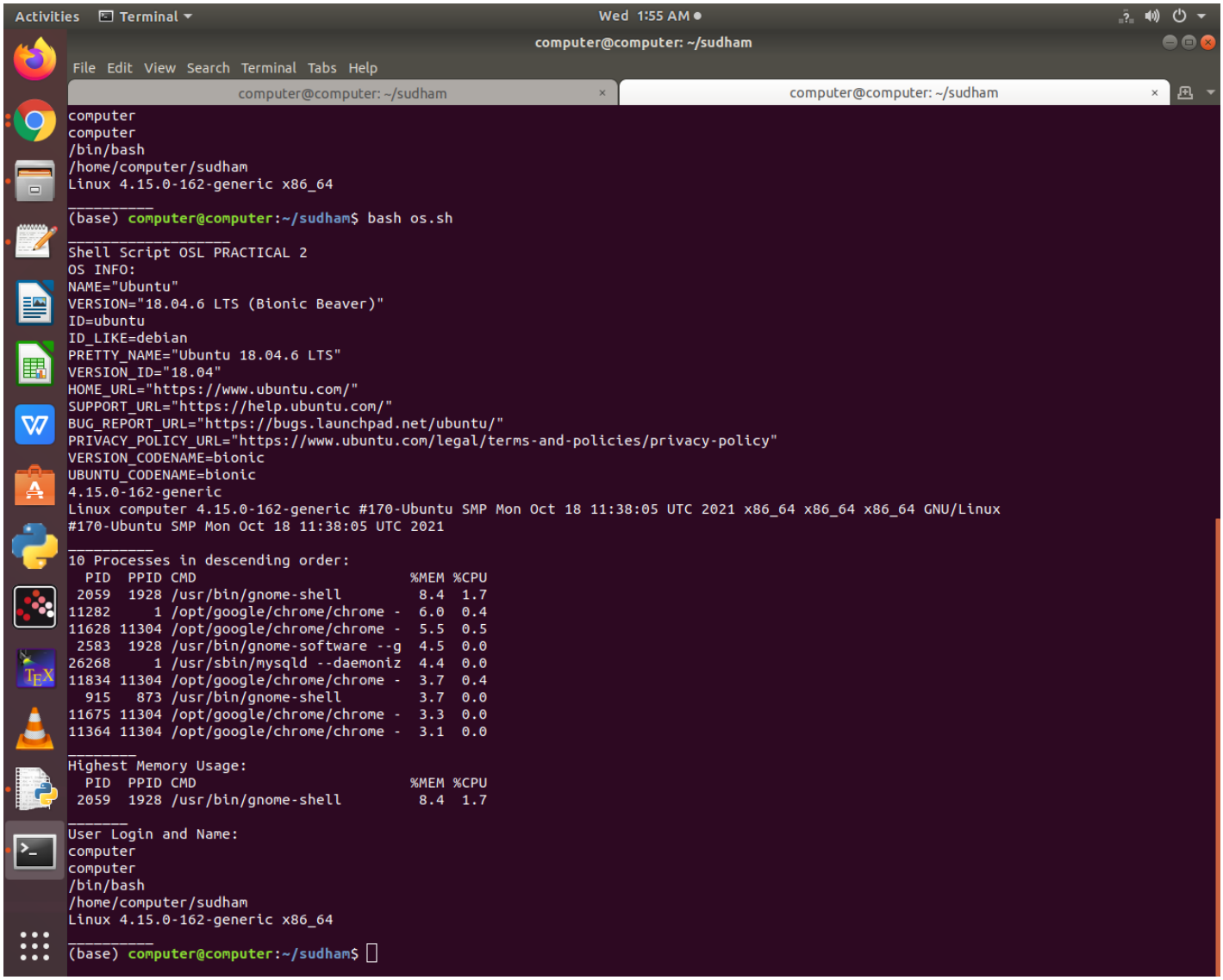
This command displays top 10 processes, there are flags that enable display more precise information like pid, ppid, mem and we've used--sortflag to sort the output according tomem i e memory usage and we've pipelined it to head to display 10 processes.

- Display processes with the highest memory usage
It displays the highest memory usage process in the system, as you can see from the below posted image, it's currently a Firefox browser.
- Display current logged in user and log name
whoami displays current user and other commands have been executed to display current BASH SHELL and pwd displays current working directory.

Program -


Activities
Text Editor
Wed 1:56 AM
Open
os.sh
~/sudham
Save
#!/bin/sh
echo "____________________"
echo "Shell Script OSL PRACTICAL 2"
echo "OS INFO:"
cat /etc/os-release
uname -r
uname -a
uname -v
echo "__________"
echo "10 Processes in descending order:"
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head -10
echo "_________"
echo "Highest Memory Usage:"
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem|head -2
echo "_______"
echo "User Login and Name:"
whoami
logname
echo "$SHELL"
pwd
uname -srm
echo "__________"
sh
Tab Width: 8
Ln 26, Col 18
INS

Output -



Conclusion - We successfully (Displayed OS version, release number, kernel version Displayed) & (top 10 processes in descending order) & processes with the highest memory usage & also Displayed current logged in use and log name through Linux Shell Script.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML
Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	03
DOP	18/02/2022
DOS	25/03/2022

Experiment No:- 3

Aim:- Linux - API implementing "CP" command

Theory:- Implementing linux "CP" command to copy one file to another ~~Do~~

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY
Mandatory arguments to long options are mandatory for short options too.
-a,-- archive same as --dR --preserve=all
--attribute-only don't copy the file data, just the attributes.
--backup make a backup of each existing destination file.
-b like --backup but does not accept an argument.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        FILE *fptr1,*fptr2;
        char filename[100], c;
        printf("Enter the filename to open for reading \n");
        scanf("%s",filename); //Open one file for reading
        fptr1=fopen(filename,"r");
        printf("Cannot open file %s\n", filename):
        if(fptr1= NULL
        exit(0);
        printf("Enter the filename to open for writing \n")
        scanf("%s", filename);
        fptr2=fopen(filename,"w");
        if(fptr2==NULL)
        {
                printf("Cannot open file %s\n",filename);
                exit(0);
        }
        c=fgetc(fptr1);
        while(c!=EOF)
        {
                fputc(c,fptr2);
                c=fgetc(fptr1);
        }
        printf("\nContents copied to %s", filename);
        fclose(fptr1);
        fclose(fptr2);
        return 0;
}
```

OUTPUT :-

CONCLUSION :

Able to run c program in linux operation system through terminals which can copy the contents of one file to another file.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	04
DOP	25/03/2022
DOS	26/03/2022

Experiment No. 4

AIM :- Process management scheduling

Write a program to demonstrate the concept of non preemptive scheduling algorithms

Write a program to demonstrate the concept of preemptive scheduling algorithms

THEORY:

FCFS (NON PREEMPTIVE):

First Come, First Served (FCFS) also known as First In, First Out (FIFO) is the CPU scheduling algorithm in which the CPU is allocated to the processes in the order they are queued in the ready queue.

SJF (PREEMPTIVE AND NON PREEMPTIVE):

Shortest job first (SJF) is a scheduling algorithm, that is used to schedule processes in an operating system. It is a very important topic in Scheduling when compared to round-robin and FCFS Scheduling.

There are two types of SJF

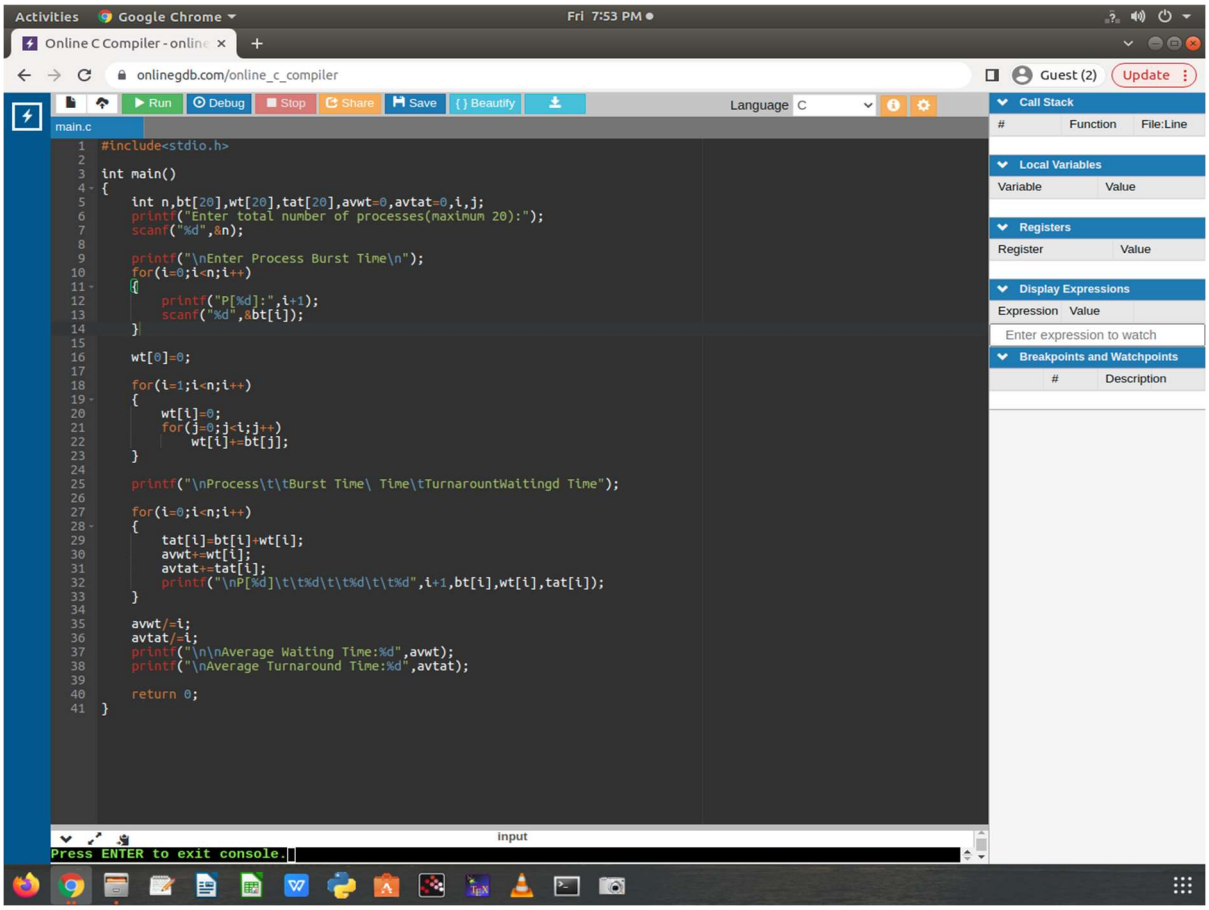
- Pre-emptive SJF
- Non-Preemptive SJF

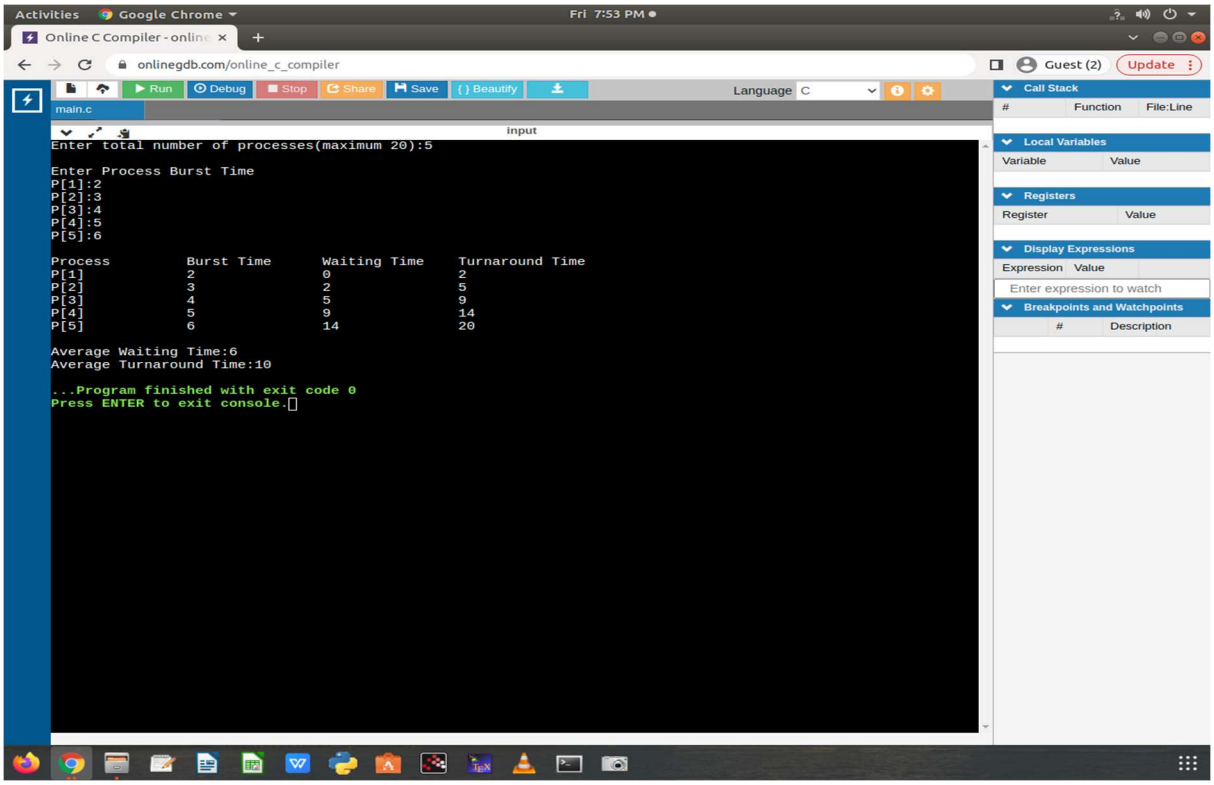
These algorithms schedule processes in the order in which the shortest job is done first. It has a minimum average waiting time.

RR (PREEMPTIVE) :

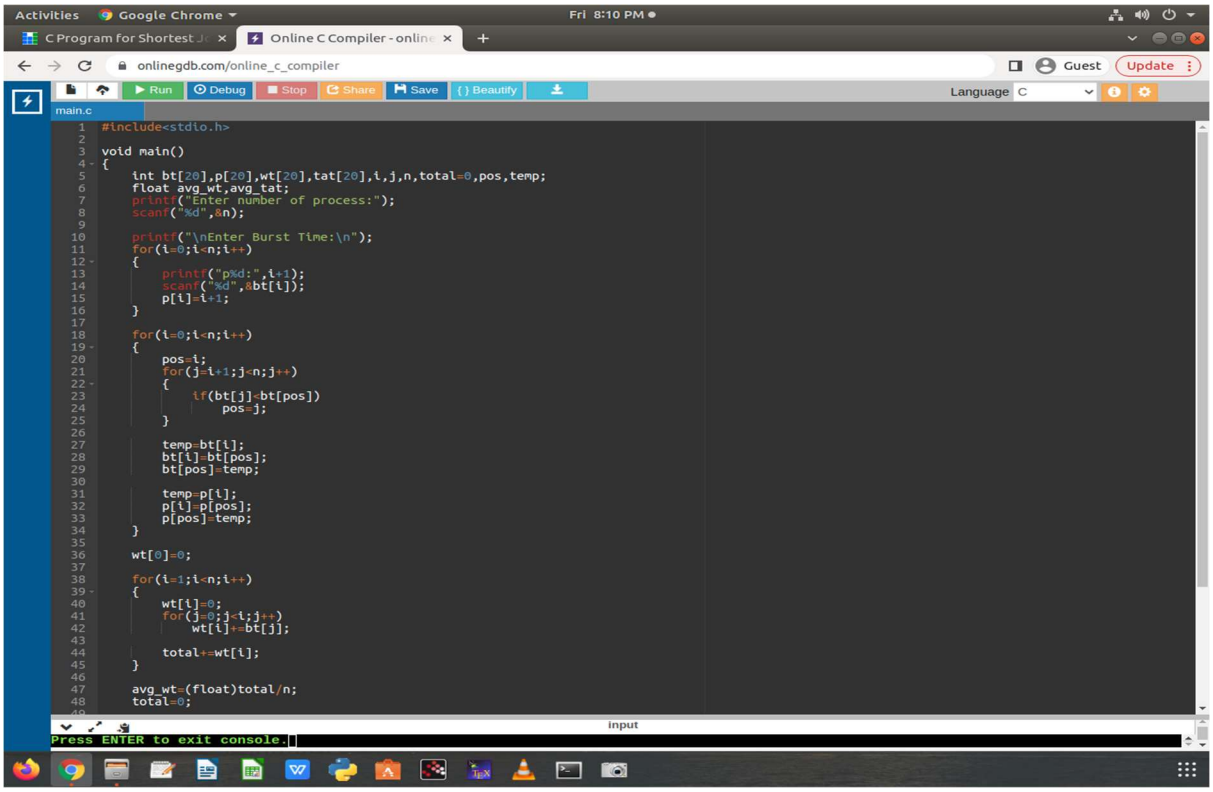
Round Robin Scheduling is a scheduling algorithm used by the system to schedule CPU utilization. This is a preemptive algorithm. There exist a fixed time slice associated with each request called the quantum. The job scheduler saves the progress of the job that is being executed currently and moves to the next job present in the queue when a particular process is executed for a given time quantum.

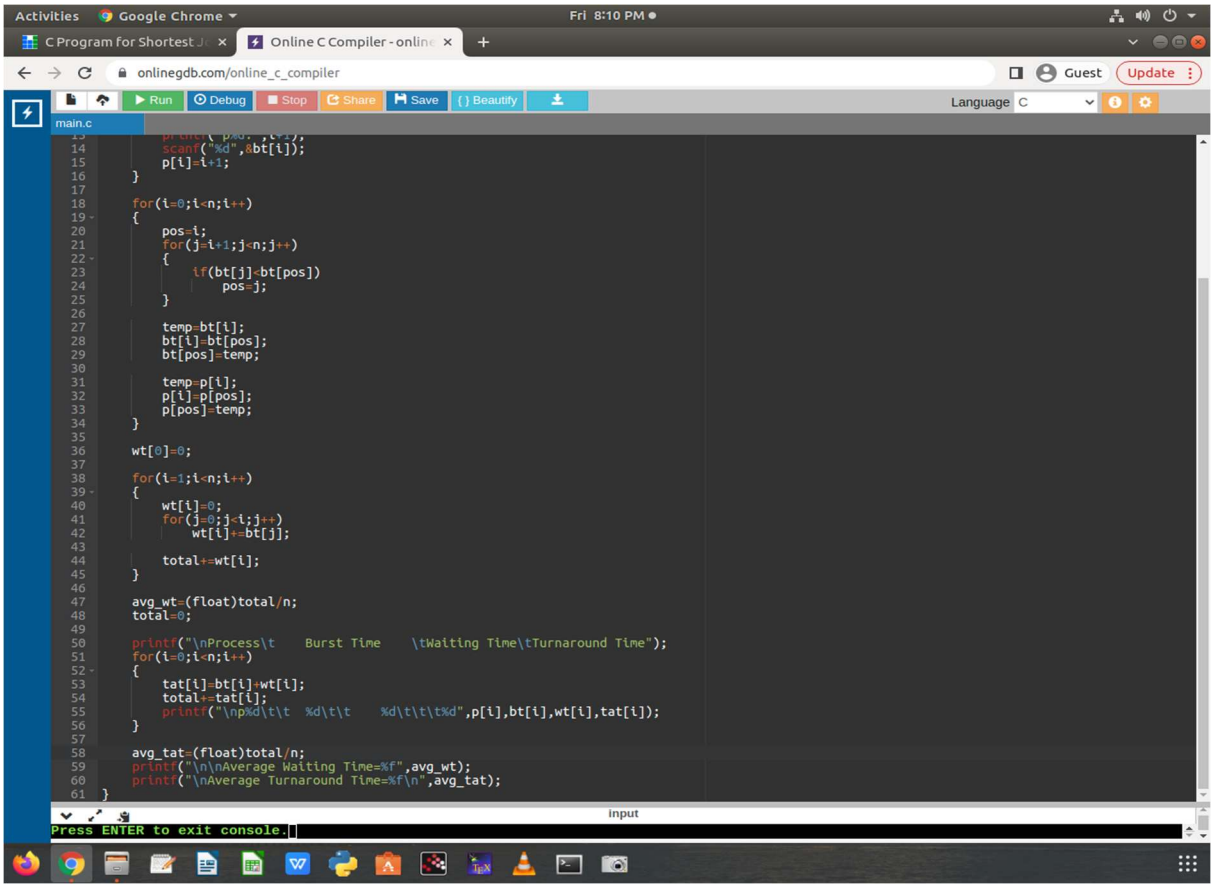
FCFS OUTPUT:

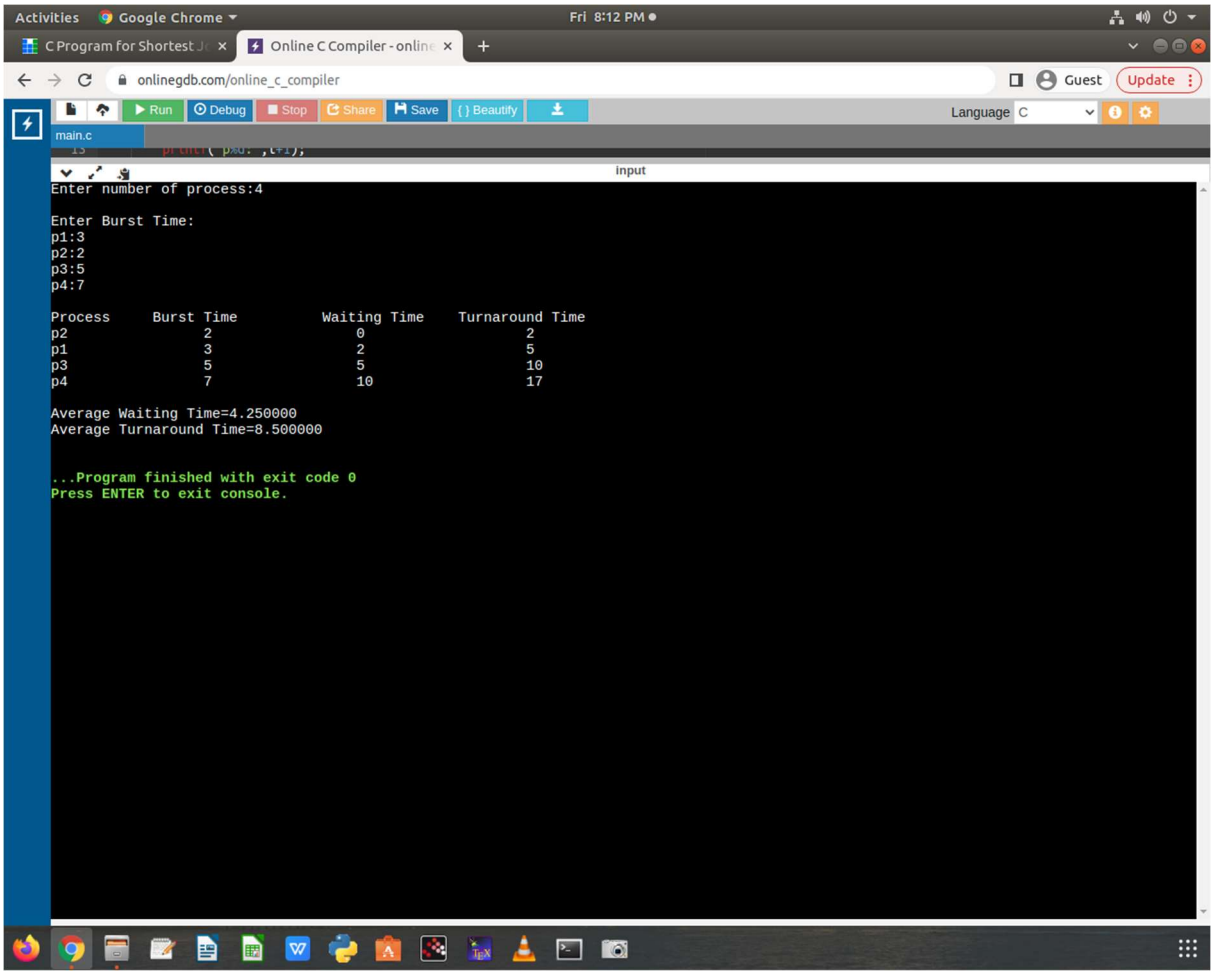

Activities
Google Chrome
Fri 7:53 PM
Online C Compiler - online
onlinegdb.com/online_c_compiler
Guest (2)
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
#include<stdio.h>
int main()
{
int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\ Time\tTurnarountWaitingd Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
avwt+=wt[i];
avtat+=tat[i];
printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
return 0;
}
Call Stack
Function File:Line
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Expression Value
Enter expression to watch
Breakpoints and Watchpoints
Description
input
Press ENTER to exit console.


Activities
Google Chrome
Fri 7:53 PM
Online C Compiler - online
onlinegdb.com/online_c_compiler
Guest (2)
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
input
Enter total number of processes(maximum 20):5
Enter Process Burst Time
P[1]:2
P[2]:3
P[3]:4
P[4]:5
P[5]:6
Process Burst Time Waiting Time Turnaround Time
P[1] 2 0 2
P[2] 3 2 5
P[3] 4 5 9
P[4] 5 9 14
P[5] 6 14 20
Average Waiting Time:6
Average Turnaround Time:10
...Program finished with exit code 0
Press ENTER to exit console.
Call Stack
Function File:Line
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Expression Value
Enter expression to watch
Breakpoints and Watchpoints
Description

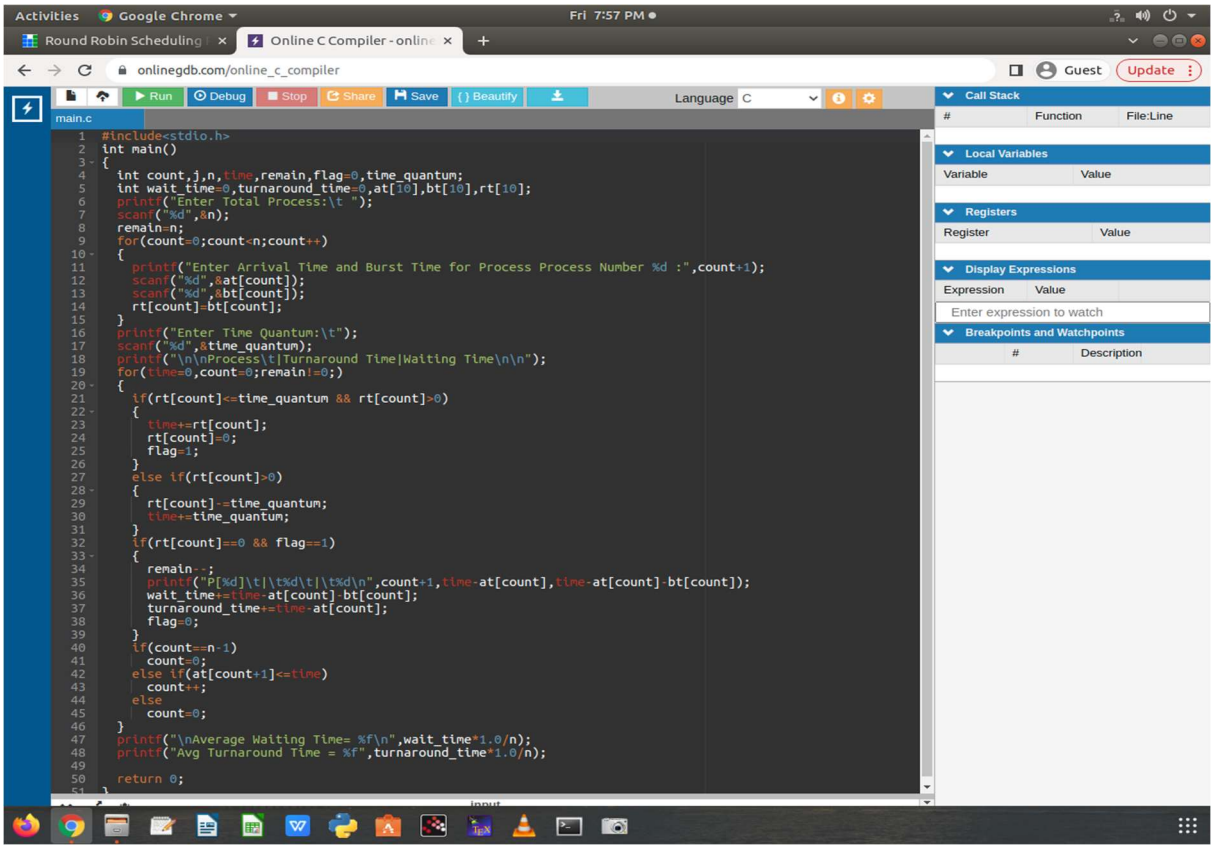
SJF OUTPUT:

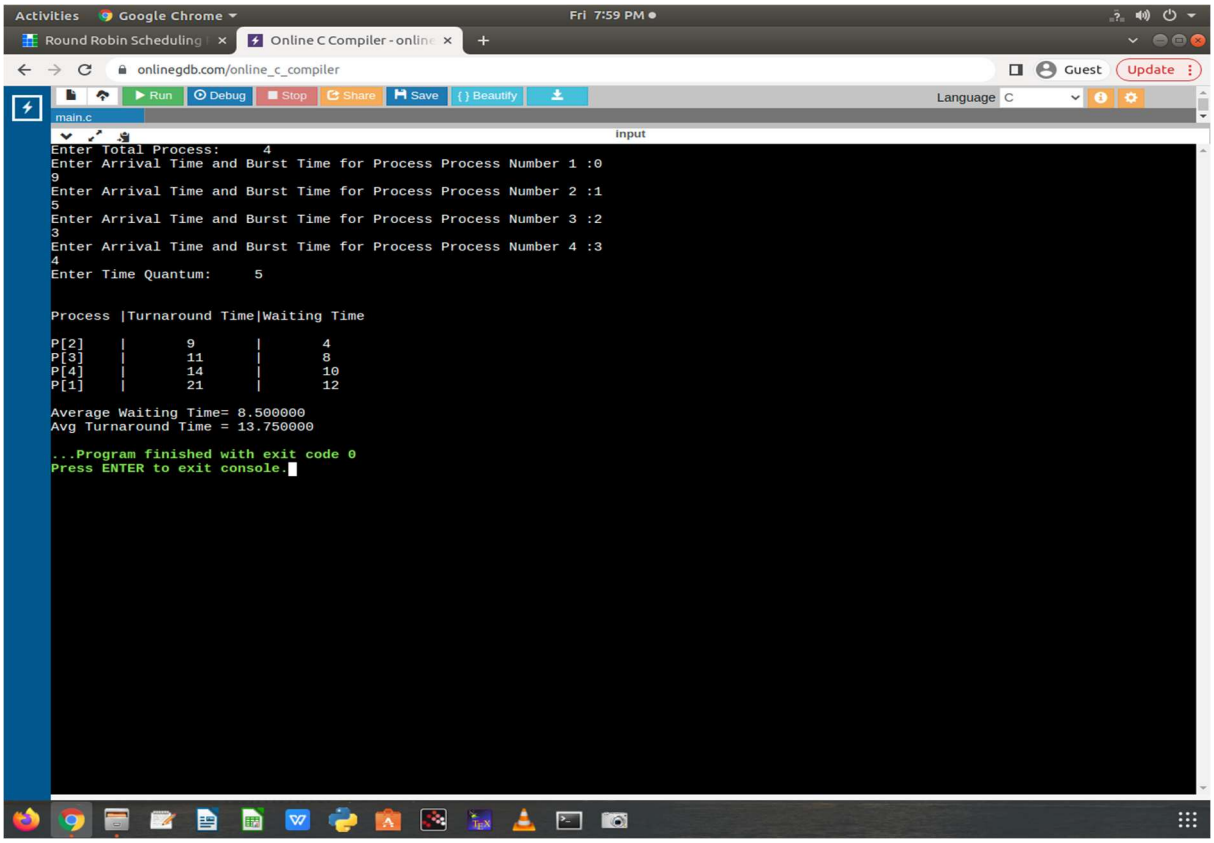

Activities
Google Chrome
Fri 8:10 PM
C Program for Shortest J
Online C Compiler - onlin
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
#include<stdio.h>
void main()
{
int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);
printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;
}
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
input
Press ENTER to exit console.


Activities
Google Chrome
Fri 8:10 PM
C Program for Shortest J
Online C Compiler - onlin
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
scanf("%d",&bt[i]);
p[i]=i+1;
}
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
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=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
input
Press ENTER to exit console.


Activities
Google Chrome
Fri 8:12 PM
C Program for Shortest J
Online C Compiler - online
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
input
Enter number of process:4
Enter Burst Time:
p1:3
p2:2
p3:5
p4:7
Process Burst Time Waiting Time Turnaround Time
p2 2 0 2
p1 3 2 5
p3 5 5 10
p4 7 10 17
Average Waiting Time=4.250000
Average Turnaround Time=8.500000
...Program finished with exit code 0
Press ENTER to exit console.

RR OUTPUT:


Activities
Google Chrome
Fri 7:57 PM
Round Robin Scheduling
Online C Compiler - online
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
#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;
}
Call Stack
Function File:Line
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Expression Value
Enter expression to watch
Breakpoints and Watchpoints
Description
input


Activities
Google Chrome
Fri 7:59 PM
Round Robin Scheduling
Online C Compiler - online
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
input
Enter Total Process: 4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
9
Enter Arrival Time and Burst Time for Process Process Number 2 :1
5
Enter Arrival Time and Burst Time for Process Process Number 3 :2
3
Enter Arrival Time and Burst Time for Process Process Number 4 :3
4
Enter Time Quantum: 5
Process |Turnaround Time|Waiting Time
P[2] | 9 | 4
P[3] | 11 | 8
P[4] | 14 | 10
P[1] | 21 | 12
Average Waiting Time= 8.500000
Avg Turnaround Time = 13.750000
...Program finished with exit code 0
Press ENTER to exit console.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	05
DOP	25/03/2022
DOS	26/03/2022

Experiment No. 5

Aim: Write a program to demonstrate the concept of dynamic partitioning placement algorithm i.e., BEST FIT, FIRST FIT, WORST FIT.

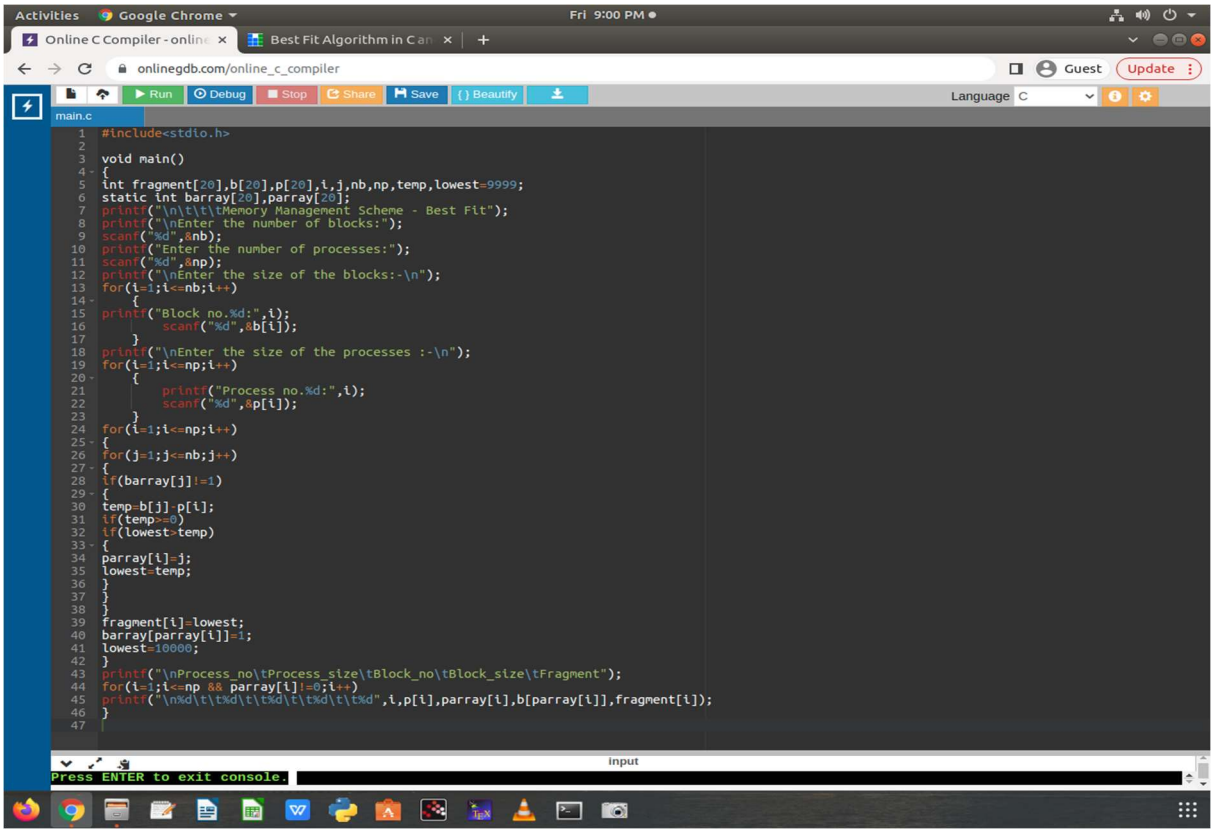
Theory:

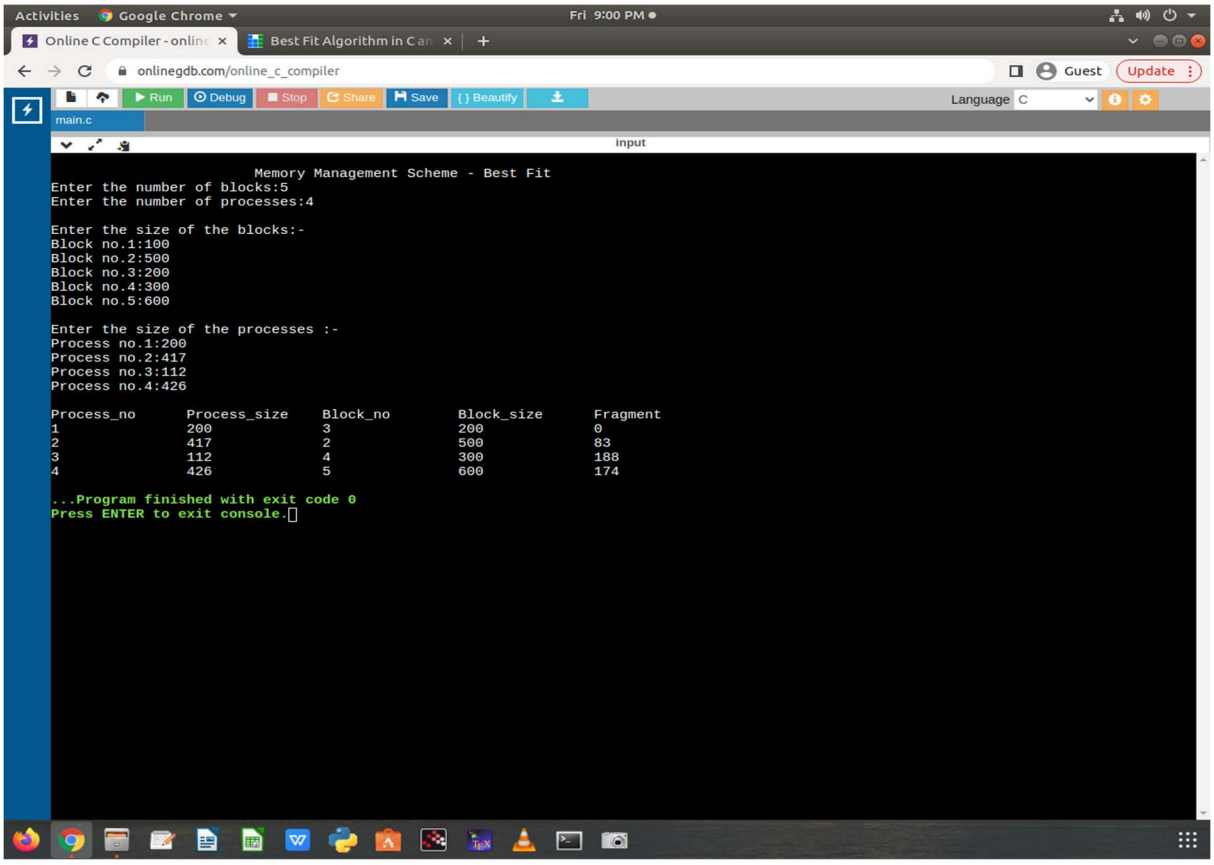
Best fit- Best fit uses the best memory block based on the Process memory request. In best fit implementation the algorithm first selects the smallest block which can adequately fulfill the memory request by the respective process.

First fit- The first fit memory allocation scheme checks the empty memory block in a sequential manner. This means that the memory block found empty at the first attempt is checked for size. If the size is not less than the required size, it is allocated.

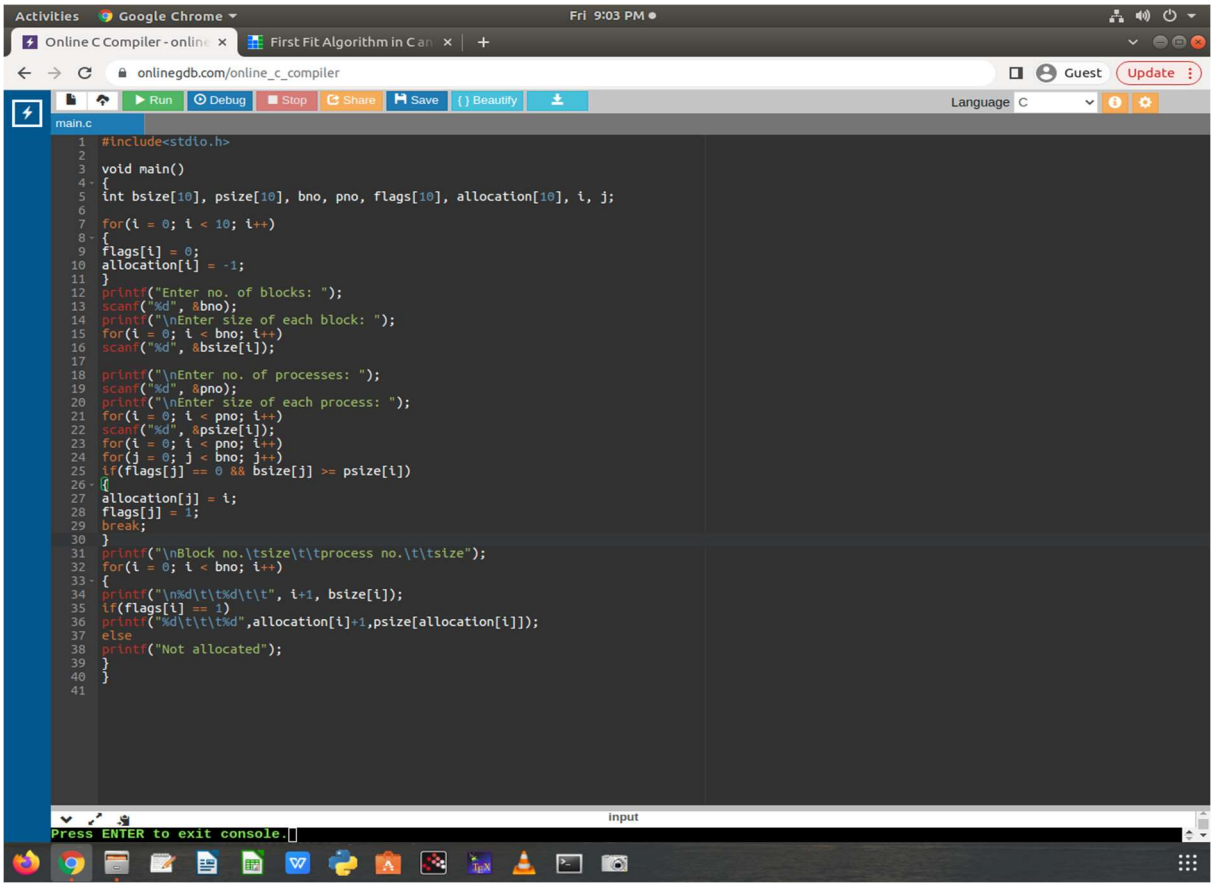
Worst fit- An empty block is assigned to the processes as soon as the CPU identifies it. The scheme is also said as the worst fit memory management scheme as sometimes a process is allocated a memory block that is much larger than the actual demand resulting in a huge amount of wasted memory.

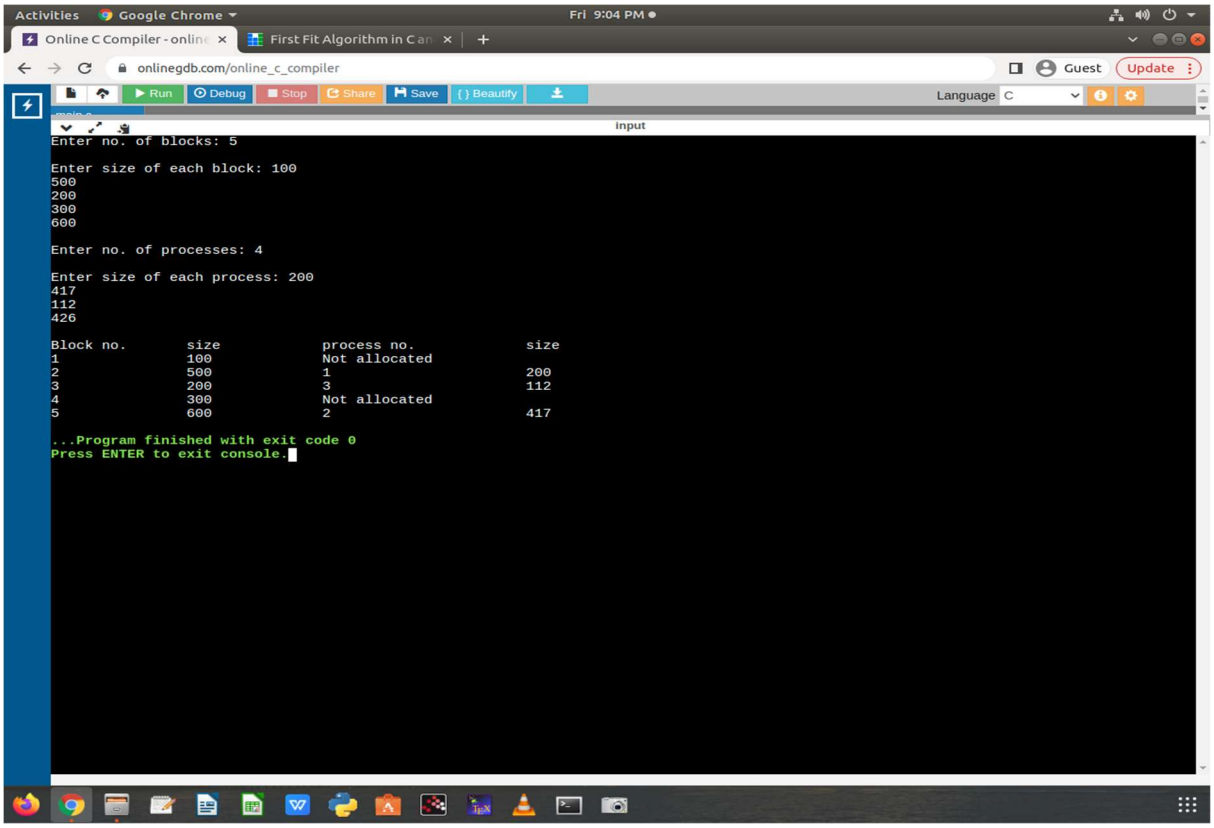
BEST FIT OUTPUT:



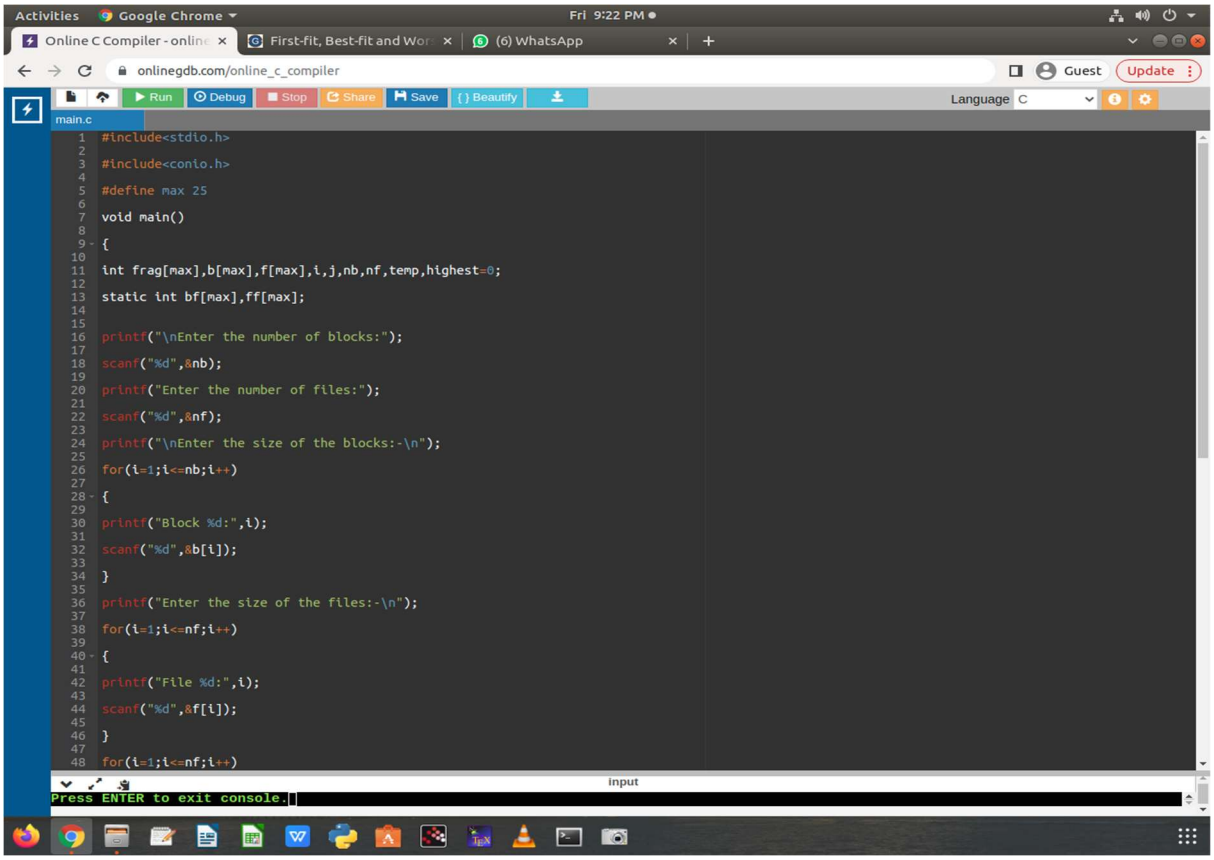


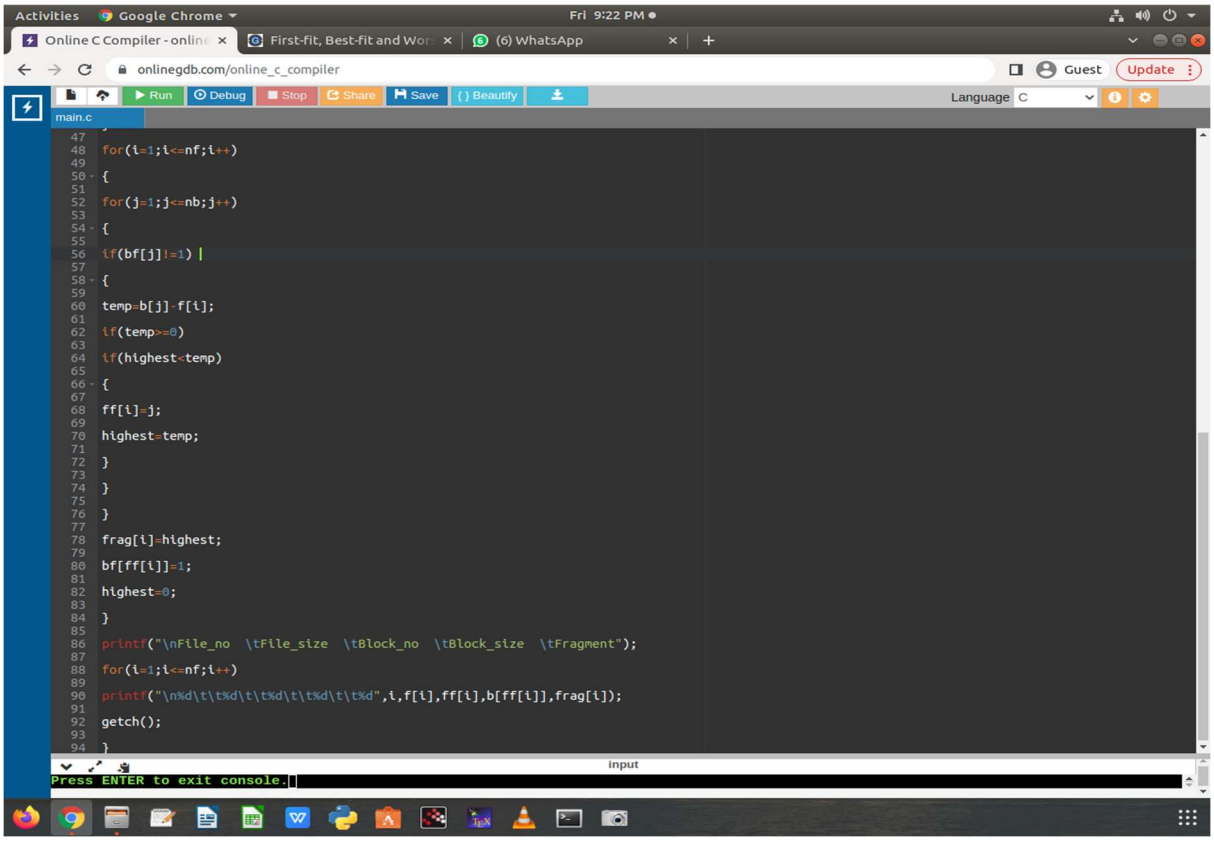
FIRST FIT OUTPUT:

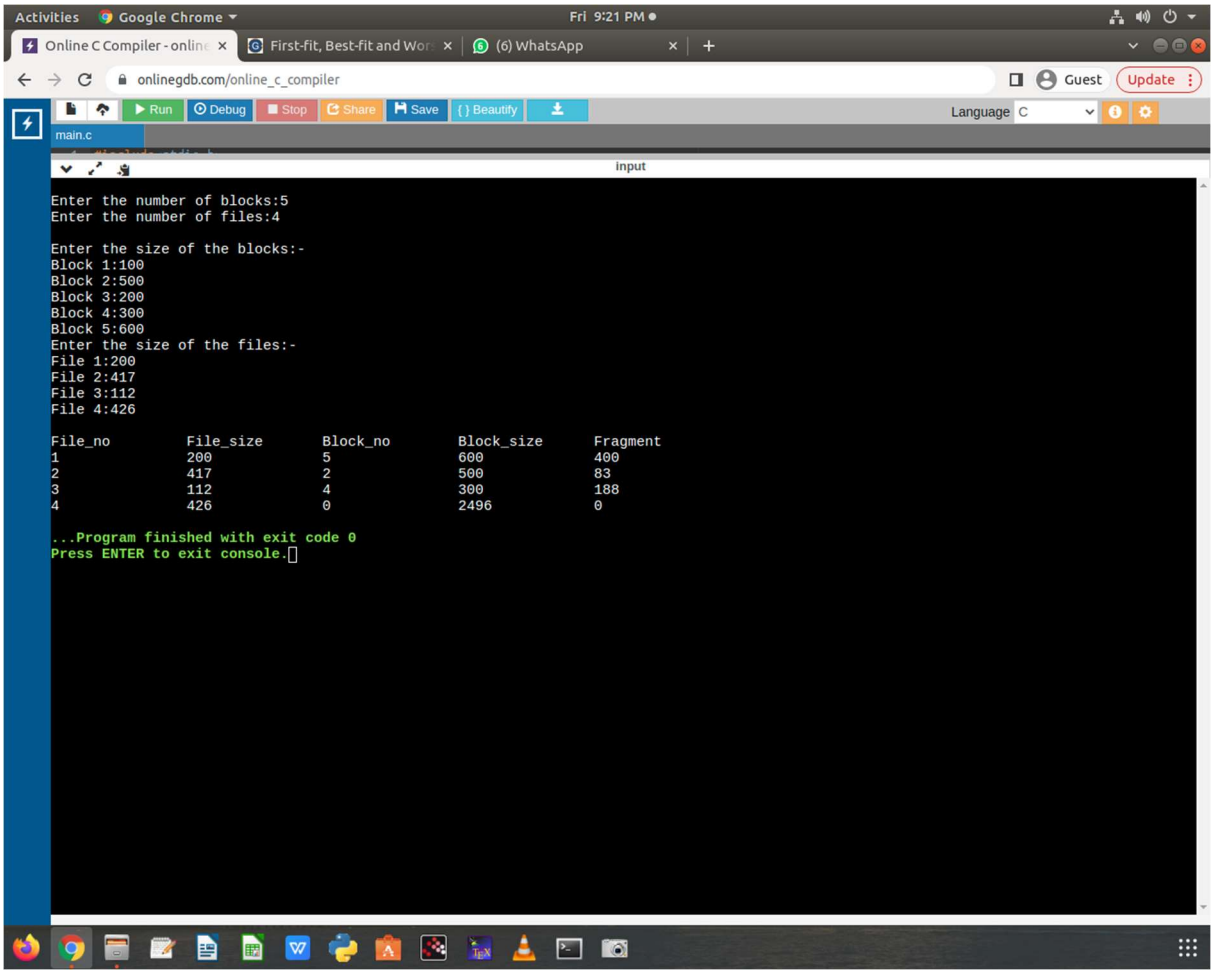




WORST FIT OUTPUT:






Activities
Google Chrome
Fri 9:21 PM
Online C Compiler - online
First-fit, Best-fit and Wor
(6) WhatsApp
onlinegdb.com/online_c_compiler
Guest
Update
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
input
Enter the number of blocks:5
Enter the number of files:4
Enter the size of the blocks:-
Block 1:100
Block 2:500
Block 3:200
Block 4:300
Block 5:600
Enter the size of the files:-
File 1:200
File 2:417
File 3:112
File 4:426
File_no File_size Block_no Block_size Fragment
1 200 5 600 400
2 417 2 500 83
3 112 4 300 188
4 426 0 2496 0
...Program finished with exit code 0
Press ENTER to exit console.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	06
DOP	14/04/2022
DOS	14/04/2022

LTCE

Experiment No. :- 6

Aim:- Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm.

Theory :-

It is a Banker's algorithm used to avoid deadlock and allocate resources safely to each processes in the computer system. The 's state' examines all possible tests on activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The Banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation process.

Similarly, it works in an operating system when a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their sources, counting them and always delay. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a systems. Therefore it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

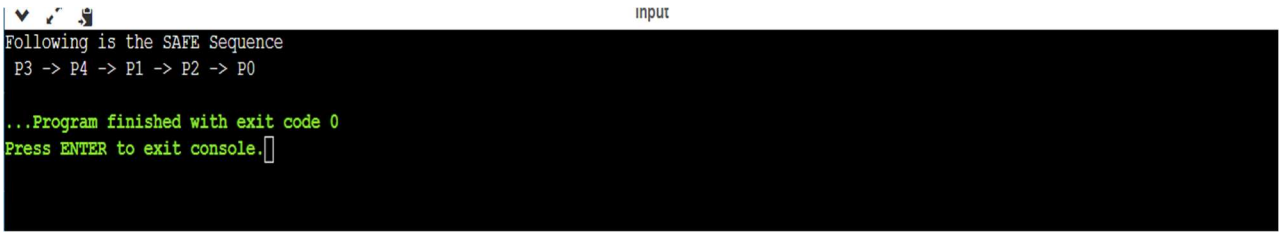
BANKER'S ALGORITHM:

```
#include <stdio.h>
int main()
{


        int n, m, i, j, k;
        n = 5;
        m = 3;
        int alloc[5][3] = { { 0, 1, 0 },
                                                { 2, 0, 0 },
                                                { 3, 0, 2 },
                                                { 2, 1, 1 },
                                                { 0, 2, 2 } };

        int max[5][3] = { { 7, 5, 3 },
                                        { 3, 2, 2 },
                                        { 9, 0, 2 },
                                        { 2, 2, 2 },
                                        { 4, 3, 3 } };

        int avail[3] = { 3, 1, 2 };

        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
```

```
}
int y = 0;
for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
                if (f[i] == 0) {

                        int flag = 0;
                        for (j = 0; j < m; j++) {
                                if (need[i][j] > avail[j]){
                                        flag = 1;
                                        break;
                                }
                        }

                        if (flag == 0) {
                                ans[ind++] = i;
                                for (y = 0; y < m; y++)
                                        avail[y] += alloc[i][y];
                                f[i] = 1;
                        }
                }
        }
}

int flag = 1;

for(int i=0;i<n;i++)
{
if(f[i]==0)
{
        flag=0;
```

```
                printf("The following system is not safe");
                break;
        }
        }

        if(flag==1)
        {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
        }


        return (0);
}
```

OUTPUT:



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	07
DOP	14/04/2022
DOS	14/04/2022

LTCE

Experiment No. :- 7

Aim :- Write a program in c to demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU, Optimal.

Theory :-

A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page fault happens in case of page fault, operating system might have to replace one of the existing pages with newly needed page.

① FIFO :

This is smallest page replacement algorithm in this algorithm, the operating system keep tracks of all pages in the memory in a queue the oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected.

② ~~LRU [Least Recently Used]~~ Optimal Page Replacement :-

In operating systems, whenever a new page is referred and not present in memory, page fault occur and operating system replaces one of the existing pages with newly needed page. Different page replacement algorithm suggest different ways to decide which page to replace.

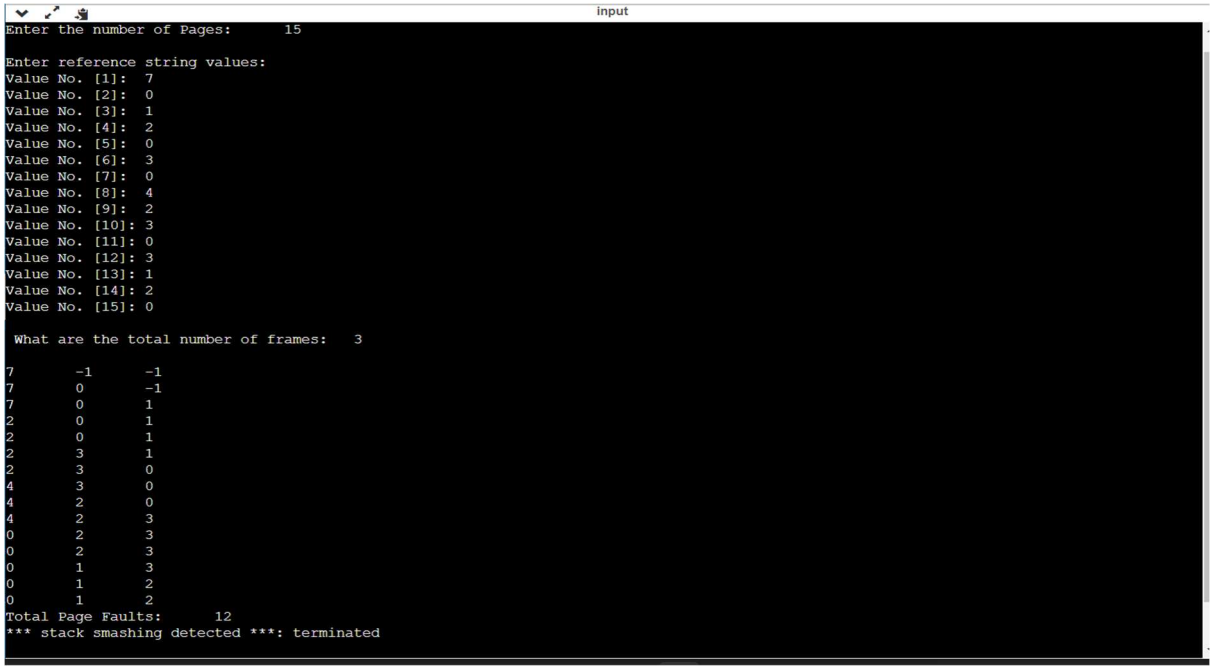
③ Least Recently Used [LRU]

In least recently used [LRU] algorithm is a Greedy algorithm where the page to be replaced in least recently used. The idea is based on locality of reference, the least recently page is not likely.

1. FIFO:

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
  printf("Value No. [%d]:\t", m + 1);
  scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
  scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
 temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
 s = 0;
 for(n = 0; n < frames; n++)
  {
    if(referenceString[m] == temp[n])
      {
        s++;
        pageFaults--;
```

```
      }
  }
  pageFaults++;
  if((pageFaults <= frames) && (s == 0))
    {
     temp[m] = referenceString[m];
    }
  else if(s == 0)
    {
     temp[(pageFaults - 1) % frames] = referenceString[m];
    }
    printf("\n");
    for(n = 0; n < frames; n++)
    {
      printf("%d\t", temp[n]);
    }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```



2. OPTIMAL PAGE REPLACEMENT:

```
#include<stdio.h>

int main()

{

   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos,
max, faults = 0;

   printf("Enter number of frames: ");

   scanf("%d", &no_of_frames);


   printf("Enter number of pages: ");

   scanf("%d", &no_of_pages);


   printf("Enter page 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]){

               flag1 = flag2 = 1;

               break;

          }

      }
```

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

if(flag2 == 0){
 flag3 =0;

   for(j = 0; j < no_of_frames; ++j){
    temp[j] = -1;

    for(k = i + 1; k < no_of_pages; ++k){
    if(frames[j] == pages[k]){
    temp[j] = k;
    break;
    }
    }
   }

   for(j = 0; j < no_of_frames; ++j){
    if(temp[j] == -1){
    pos = j;
    flag3 = 1;
    break;
```

```
            }
           }

           if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
       }

       printf("\n");

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

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

   return 0;
}
```


input
Enter number of frames: 3
Enter number of pages: 15
Enter page reference string: 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
2 4 3
2 4 3
2 4 3
2 0 3
2 0 3
2 0 1
2 0 1
2 0 1
Total Page Faults = 8
...Program finished with exit code 0
Press ENTER to exit console.

3. LRU [LEAST RECENTLY USED]:

```
#include<stdio.h>


int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;


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

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}


int main()

{

   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], 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("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	08
DOP	14/04/2022
DOS	14/04/2022

LTCE

Experiment No:- 8

Aim:- To create a child process using fork system call. Obtain process ID of both child and parent using getpid & getpid system call.

Theory:-

Forks: Fork() is a system call function which can generate child process from parent main process. Using some conditions we can generate as many child process as need.

Parent Process:- All the processes are created when a process executes the fork() system call except the start up process. The process that executes the fork() system call is the parent process. Parent process is the one that creates a child process using a fork() system call.

Child Process:- A child process is created as a copy of it parent process using a fork() system call. A child process may also be known as sub process or a subtask.

LTCE

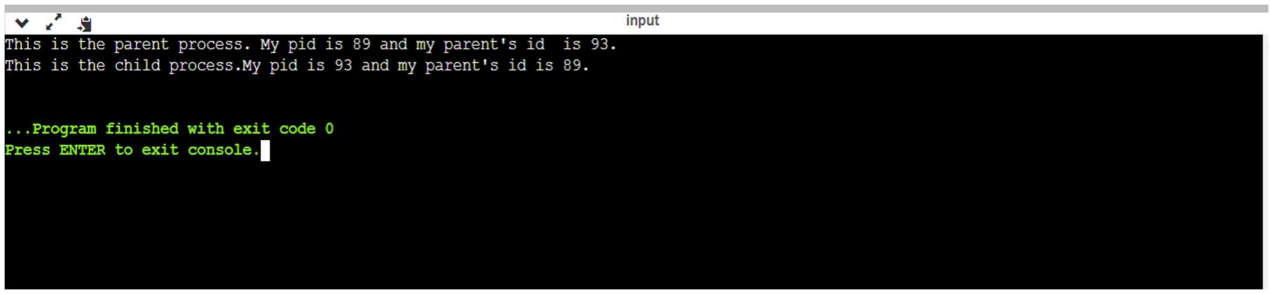
Code :-

```
#include <stdlib.h>
int fork ();
int getpid();
int getppid();

int main ()
{
  int pid = fork();
  if (pid == 0)
  {
    printf("This is child process. My pid is %d and my
    parent's Id is %d.\n", getpid(), getppid());
  }
  else
  {
    printf(" This is parent process. My pid is %d and my
    parent's Id is %d.\n", getpid(), pid);
  }
  return 0;
}
```

SYSTEM CALL:

```
#include <stdio.h>

#include <stdlib.h>


int fork();

int getpid();

int getppid();


int main()

{

        int pid = fork();



        if(pid == 0){

        printf("This is the child process.My pid is %d and my parent's id is %d.\n",getpid(),getppid());

        }else{

           printf("This is the parent process. My pid is %d and my parent's id  is %d.\n",getpid(),pid);


        }

        return 0;

}
```

OUTPUT:



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	09
DOP	16/04/2022
DOS	16/04/2022

LTCE

Aim:- Write a program in C to do disk scheduling - FCFS, SCAN, C-SCAN

Theory:- Disk scheduling is done by operating system to schedule I/O request arriving for the disk. Disk scheduling is also known as I/O scheduling.

① FCFS Algorithm:-

FCFS is the simplest disk scheduling algorithm. As the name sugests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation call requests are serviced sequentially but generally, it does not provide the fastest service

② SSTF:-

Shortest seek time first (SSTF) algorithm select the disk I/O request ~~which request~~ requires the least disk arm movement from its current position regardless of the direction. It reduces seek time as compared to FCFS. It allows the head to move to the closest track in the service queue.

③ SCAN:-

In SCAN disk scheduling algorithm, head starts from one end of the disk and moves start from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. so, this algorithm as a result, this requests at the midrange are ~~served~~ serviced more and those arriving behind the disk arm will have to wait.

LTCE

(4) CSCAN:.

The circular SCAN [C-SCAN] scheduling algorithm is a modified version of the SCAN disk scheduling algorithm that deals with the inefficiency of the SCAN algorithm by servicing the requests more uniformly Like SCAN [elevator Algorithm] C-SCAN moves the head from one end servicing all the requests to the other end.
This is also known as "Circular Elevator Algorithm".

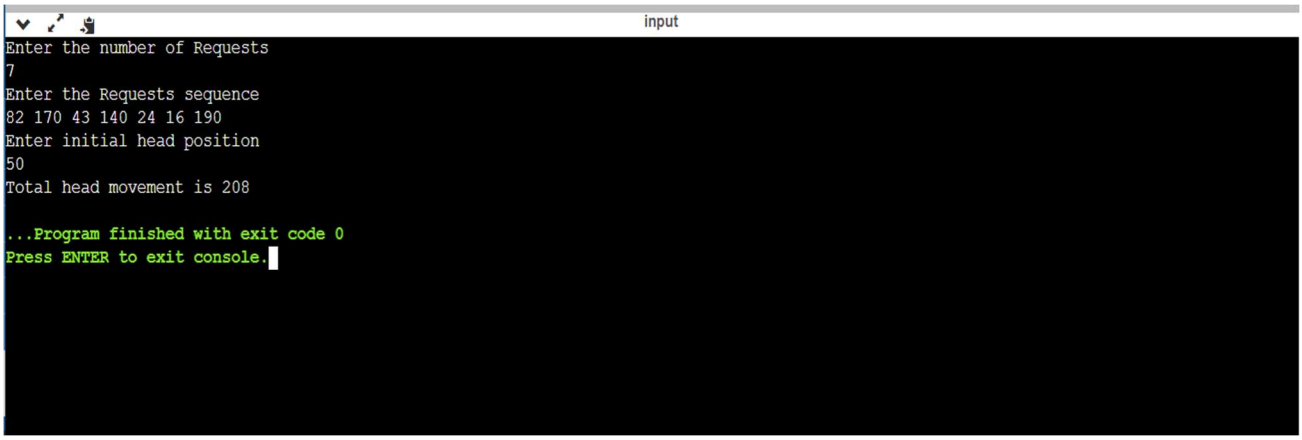
1. FCFS:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   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 FCFS disk scheduling

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   printf("Total head moment is %d",TotalHeadMoment);
   return 0;
}
```

2. 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

   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];
```

```
        RQ[index]=1000;
        count++;
    }
    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```



3. SCAN:

```
#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 Scan disk scheduling

   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);
    initial = size-1;
    for(i=index-1;i>=0;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);
        initial =0;
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
    }
    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

input

```
Enter the number of Requests
7
Enter the Requests sequence
82 170 43 140 24 16 190
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 332

...Program finished with exit code 0
Press ENTER to exit console.
```

4. 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

   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;
}
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

S.E/SEM IV/CBCGS/AIML Academic Year: 2021-22

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML51
SUBJECT	OPERATING SYSTEM LAB
COURSE CODE	CSL403
PRACTICAL NO.	10
DOP	16/04/2022
DOS	16/04/2022

LTCE

Aim: Write a C program to implement a solution to the producer consumer problem through Semaphore.

Theory:- We have a buffer of fixed size. A buffer can produce an item and can replace in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the ~~can~~ critical section. Producer Consumer problem is a classical Synchronization problem. We can solve this problem by using Semaphores. A semaphore is an integer ~~value~~ variable that can be accessed only through two standard operations: wait() & Signal(). The wait operation reduces the value of semaphore by 1 and the signal() operations increases its value by 1

Semaphores are of two types:-

1. Binary Semaphore: This is similar to mutex lock but not the same thing. It can have only two values - 0 & 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
2. Counting Semphore:- Its value can ~~change~~ range over an instructed domain. It is used to control access to a resource that has multiple instances.

CODE:

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

int mutex=1,full=0,empty=3,x=0;
int main()
{
   int n;
   void producer();
   void consumer();
   int wait(int);
   int signal(int);
   printf("\n1.Producer\n2.Consumer\n3.Exit");
   while(1)
   {
      printf("\nEnter your choice:");
      scanf("%d",&n);
      switch(n)
      {
         case 1:   if((mutex==1)&&(empty!=0))
                      producer();
                   else
                      printf("Buffer is full!!");
                   break;
         case 2:   if((mutex==1)&&(full!=0))
                      consumer();
                   else
                      printf("Buffer is empty!!");
                   break;
         case 3:
                   exit(0);
```

```
                break;
        }
    }
    return 0;
}
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return(++s);
}
void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex);
}
void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}
```

OUTPUT:

CONCLUSION:

IN THIS EXPERIMENT WE HAVE STUDIED TO IMPLEMENT A SOLUTION TO THE PRODEUCER CONSUMER PROBLEM THROUGH SEMAPHORE.

Name :- Singh Sudham Dharmendra

Branch :- CSE (AI & ML)

Roll no :- AIML 51

Subject :- Operating System

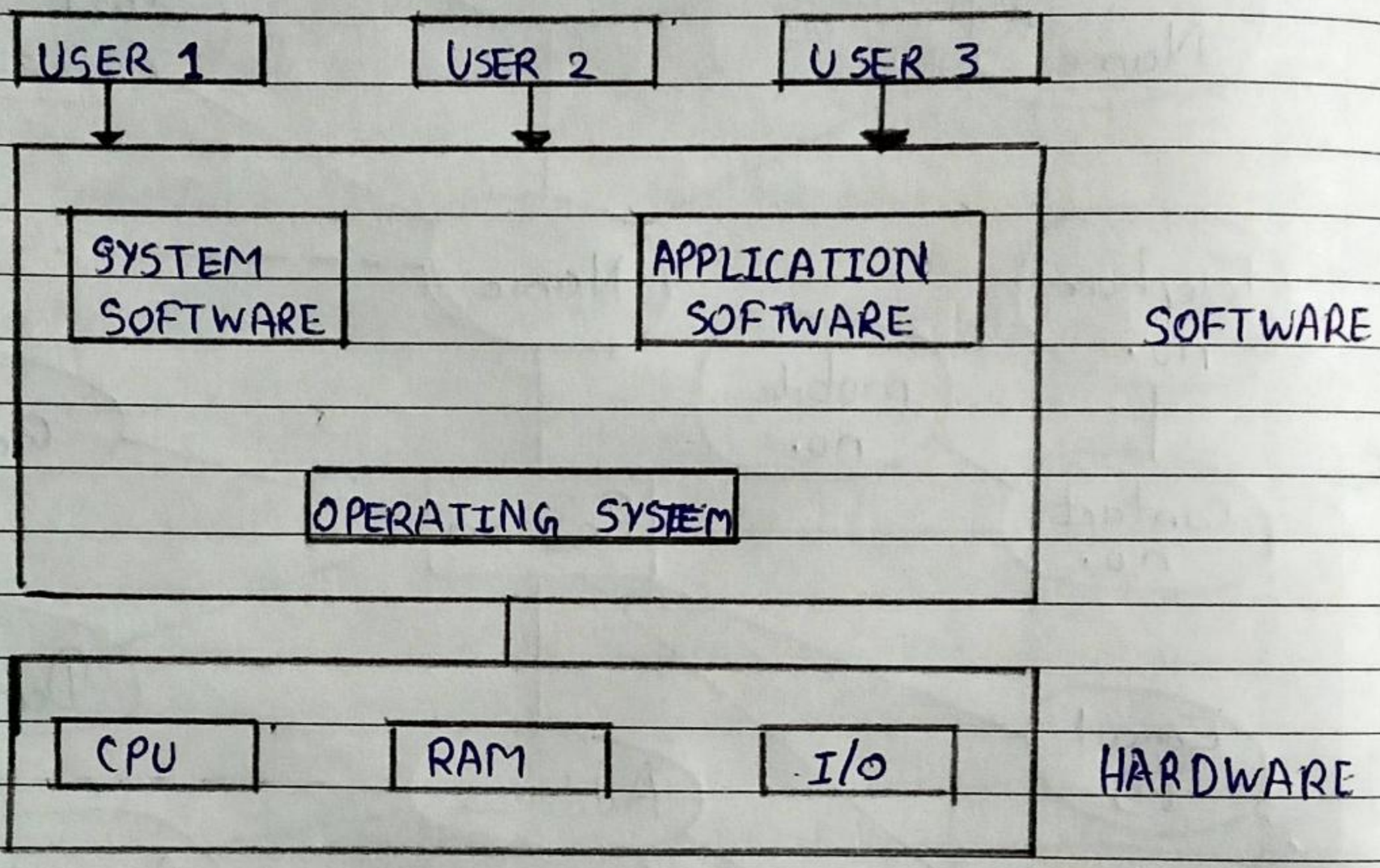
Topic :- Assignment No. 1

Date of Submission :- 03/02/2022

Q.1 What is Operating System ? Explain Objective & Functions of OS

→

Definition :- Operating System is interface between user and hardware . It is an software used for communication between user and hardware and also controls executions of application programs. It is called "resource management".



Objectives :-

There are 3 objective of Operating System.

1. Convenient :- Because of Operating System it is easy and convenient to use computer.

2. Efficiency :- Because of Operating System of computer increase we can work efficiently.

3. Ability to Evolve :- If we add more modules in our computer then also our computer work it, does not get damaged.

Functions:-

Operating Systems have varius functions mentioned below:-

(1) Process Management :-
(i) Creation, Execution, Deletion of user & system process.
(ii) Control the execution of user application.
(iii) helps in scheduling process.

(2) Memory Mangement:-
(i) It allocates primary as well as secondary memory to user and system and system process.
(ii) Reclaims allocated memory from all process that have finished its execution.
(iii) Once used block become free OS allocates it again to processes.
(iv) Monitoring & keeping track of how much memory used by process

(3) File Management :-
(i) Creation & deletion of files and directories
(ii) It allocates storage space to the files by using different file allocation methods.
(iii) It keeps back up of files and offeres security of files.

(4) Device Manggement:-
(i) Device drives are open, closed and written by Os.
(ii) keeps an eye on device drivex, communicate, control & monitor device drivex.

(5) Protection & Security :-
(i) The resaurce of system are protected by Os
(ii) keeps ~~an eye on~~ back-up of data.
(iii) Lock the system of security purpose by password.

Q.2 → Differentiate between monolithic & microkernel.

Monolithic kernel	Micro kernel
① If virtually entire operating System code is executed in kernel mode, then it is a monolithic kernel program that runs in a single address space.	① In micro kernel set of modules for managing hardware is kept, which can uniformly well be executed in user mode. A small microkernel contains only that must execute in kernel mode. It is second part of operating system.
② There is same address space for user mode as well as kernel mode.	② There is different address space for user mode as well as kernel mode.
③ It has large space as compared to microkernel	③ It has small space as compared to monolithic kerne
④ Execution speed is faster than micro kerne.	④ Execution speed slower than monolithic kernel.
⑤ If one service crashes whole operating system falls	⑤ If one service crashes whole operating system do not ~~crash~~, it does not affect working of other part micro kernel.
⑥ kernel calls function directly for communication	⑥ Communication is done through message passing

Q.3 Explain PCB and its role.

→ Process control Block (PCB) is data structure used by operating system to keep track on process. Any process is defined by its PCB.

① All information associated with process is kept in PCB. There is separate PCB for each process.

② Traffic controller module keeps a track on status of process. PCB's of processes which are in same state (ready, waiting, executing etc) are linked together giving a specific name to list such as ready list.

③ If many processes wait for device the PCB's of all these processes are linked together in chain waiting for that devices.

④ If the processes are available in that device chain, then again it is placed back to ready state to request that device.

POINTER		CURRENT STATE
PROCESS ID		
PROGRAM COUNTER		
REGISTER		
EVENT INFORMATION		
LIST OF OPEN FILES		
.....		

PROCESS CONTROL BLOCK

a) Pointer: The field points to other process PCB. The scheduling list is maintained by pointer.

b) Current State: Currently process can be in any of state from new, ready, executing, waiting, etc.

c) Process ID: Identification number of the process. Operating System assign this number to process to distinguish it from other process

d) Priority: Different process can have different priority. Priority field indicate priority of process.

e) Program Counter : The counter indicates address of next instruction to be executed for this process.

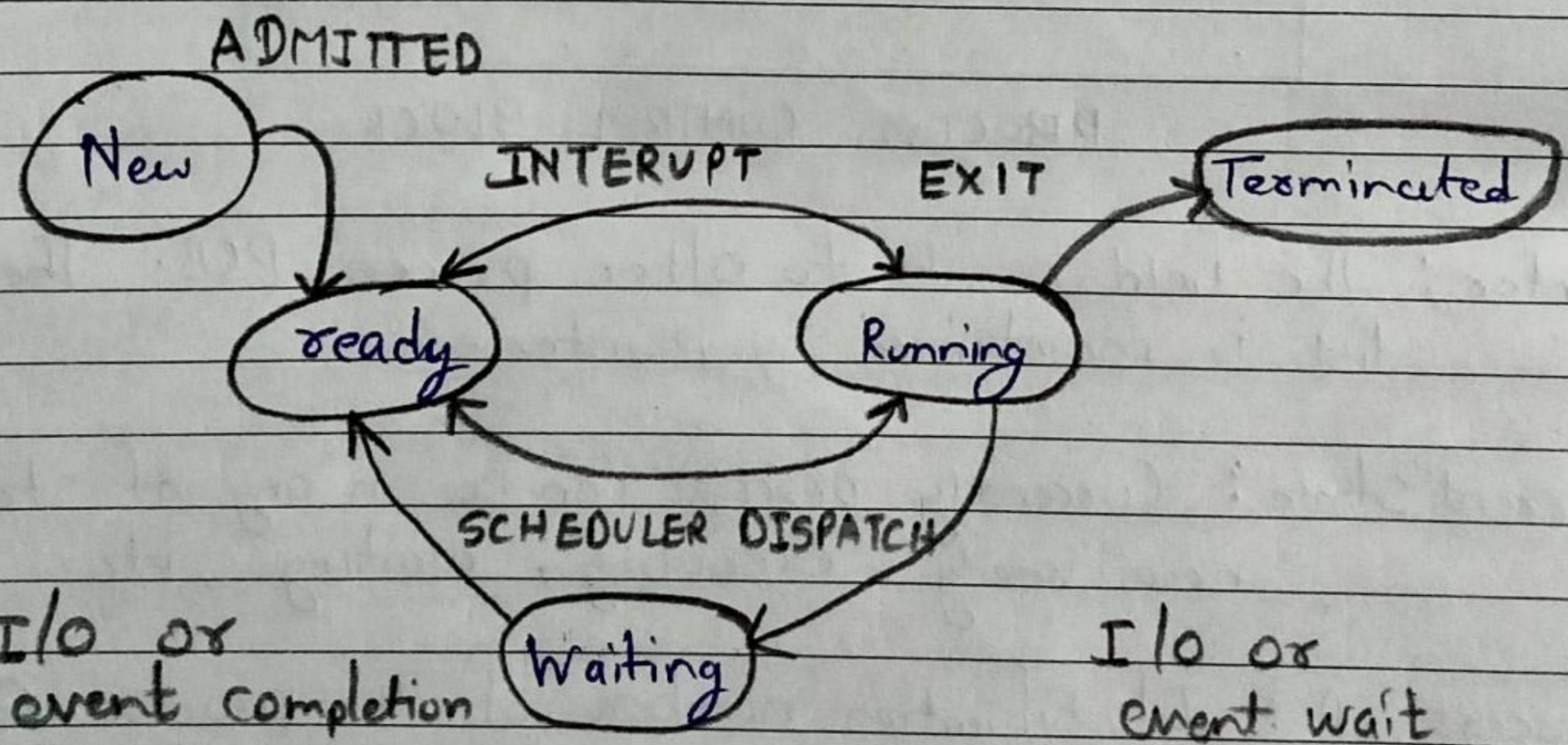
f) Registers : The registers ~~can~~ vary in number and type, depending on computer architecture. They include accumulators, index & registers, stack pointer and general purpose register.

g) Event information : For a process in blocked state this field contains information concerning event for which process is waiting.

h) List of Open files: Files opening, reading, writing, closing is done by process. After creation of process, hardware registers and flags are set as per the details supplied by generally placed on stack and pointer to respective stack frame is stored in PCB.

Q.4 Explain process State diagram.

→



Process can have one of following five states at a time:

1. New State : A process that first has been created but has not yet been admitted to pool of execution process by operating system. Every new operation which is requested to system is known as new born process

2. Ready State : When process is ready to execute but he is waiting for CPU to execute then thus is called as ready state. After completion of input and output process will be on ready state means process will wait for processor to execute.

3. Running State : The process that is currently being executed. when the executed by CPU. then thus is called as running process and when a process is running then this will also provide us some outputs on Screen.

4. Waiting or blocked State : A process that cannot execute until some event occurs or an I/o completion when a process is waiting for some input and output operations then this is called as waiting state and in process is not under execution instead process is stored out of memory and when user will provide input and then this will again be on ready state.

5. Terminated State : After completion of process the process will be automatically terminated by CPU. so this is also called as terminated state of process. After executing compile process, the processor will also deallocate memory which is allocated to process. So this called as terminated process.

Q. 5 ~~What~~ Write the advantages & disadvantages of FCFS, SJF, ROUND ROBIN algo.

→

① First-Come First-Served algorithm :- (FCFS)

Advantages :-

(i) More predictable than others schemes since it offers time.
(ii) Code for FCFS scheduling is simple to write and understand.

Disadvantages :-
(i) Short jobs (process) may have to wait for long time.
(ii) Important jobs (with higher priority) have to wait.
(iii) Cannot guarantee good response time.
(iv) lower CPU & device utilization.

(2) Shortest Job First (SJF) :-

Advantages :-
(i) Shortest jobs are favoured.
(ii) It is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

Disadvantages :-
(i) SJF may cause starvation, if shortest processes keep coming. This can be solved by aging.
(ii) It cannot be implemented at the level of short term CPU scheduling.

(3) Round Robin algorithm (RR) :-

Advantages :-
(i) Every process gets an equal share of the CPU.
(ii) RR is cyclic in nature, so there is no starvation.

Disadvantages :-
(i) Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.
(ii) Average waiting time under the RR policy is often long.

LTCE

Name :- Singh Sudham Dharmendra

Branch :- CSE (AI & ML)

Roll no :- AIML 51

Subject :- Operating System

Topic :- Assignment No. 2

Date of Submission :- 13/03/2022

Q.1) Explain paging hardware with TLB along with protection bits in page table.

→ (1) The TLB (translation look aside buffer) is similar to page table. important as most used pages are stored in TLB

(2) First pages are searched in TLB if page is found then it proceeds further else if searches pages in page table. TLB reduce execution time.

(3) TLB is a piece of very fast, associative memory, capable of searching many areas of memory simultaneously.

(4) This means that many tables entries can be reached at the same time for a ~~legal~~ logical page entry. This type of memory is very expensive which means that not much of it is used. Memory management unit (MMUs) usually use TLBs with between 64 & 1024 entries.

(5) A new page table causes all the entries stored in the TLB to become useless. When a new page table is used for example, on a context switch the TLB entries must be erased to make sure the new process's logical address space maps to the old process's physical address space.

Protection bit:-

0	4	Valid
1		Invalid
2	6	Valid
3		Invalid
4		Invalid
5	9	Valid
6		Invalid

(1) With all the paging & replacement that goes on, it might be easy to forget that there needs to be protective barriers thrown up around memory pages.

(2) We need to prevent code from straying beyond the boundaries of its pages our scheme must embrace the swapping & paging ideas we have developed'.

(3) Protection in a paged environment mug on physical-memory frames. Since every memory refrence goes through the page table to reference frames. Since we can add protection bits for frames and store them in page table. These bits can give certain properties for frames: read-only or read-write.

(4) When logical-to-physical translation is taking place, the nature and owner of the request is also analysed.

Q.2) Explain the effect of page frame size on performance of page replacement algo..

-> (1) The number of frames is equal to the size of memory divided by the page-size. So a increase in page size means a decrease in the number of available frames.

(2) Having a fewer frames will increase the numbers of page faults because of the lower freedom in replacement choice.

(3) Large pages would also waste space by internal fragmentation

(4) On-the other hand a larger size page size would draw in more memory per faults, so the number of fault may decrease if there is limited contention.

(5) larger pages also reduces the number of TLB misses.

An important hardware design decision is the size of page to be used. There are several factors to consider!-

• Internal fragmentation:-

① Clearly, the smaller to the page size, the lesser is amount of internal fragmentation to optimize the use of main memory, we would like to reduce internal fragmentation.

② On the other hand, smaller the page, the greater is the number of pages required per process which could mean that some portion of pages tables of active processes must be in virtual memory, not in main memory.

③ This eventually leads to double page fault for a single reference of memory.

Rate at which page fault occurs:-

①: If the page size is very small, the ordinary a large number of pages will be available, in main memory for process, which after some time will contain portions of process near recent references leading to low page fault rate.

② As the size of page is increased, each individual page will contain locations further and further from any particular recent reference.

③ This, the effect of the principle of locality is weakend and the page fault rate begins to rise.

④ Eventually, however the page fault rate will begin to fall as the size of page approaches the size of the entire process.

Q.3): What is thrashing / Explain.

→ ① A process should have some minimum numbers of frames to support active pages, which are in memory.
② It helps to reduce the number of page faults. If these numbers of frames are not available to the process then it will quickly page fault.
③ To handle this page fault, it is necessary to replace the existing page from memory.
④ Since all the pages of the process are active, it will also need in future. so any replaced page will cause page fault again & again.
⑤ Since in paging, pages are transferred between main memory and disk, this has an enormous overhead.
⑥ Because of this frequent movement of pages between main memory and disk, system throughout reduces.
⑦ This frequent paging activity causing the reduction in system throughput called as thrashing.
⑧ Although many processes are in memory; due to thrashing CPU utilization goes low.
⑨ When operating system monitors the CPU utilization, it introduces new process in memory to increase the degree of multiprogramming.
⑩ Now it is needed that the existing pages should be replaced for this new process.
⑪ If global page replacement algorithm is used, it replaces the pages of other process and allocates frames to this newly introduced process. So other processes whose pages are replaced are also cause page faults.
⑫ All these faulting processes go in wait state & waits for paging device. In this case again CPU utilization goes low.

(13) When CPU scheduler observes the low CPU utilization, it introduces new processes again in system to improve degree of multiprogramming.

(14) Again the waiting queue of paging device becomes long causing further drop in CPU utilization. In this way page fault rate further increases & CPU utilization decreases.

(15) There is no actual work getting done & processes spend time only in paging.

(16) This thrashing can be limited by using local page replacement algorithm instead of global page replacement algorithm.

(17) Local page replacement algorithm replaces the pages of same process instead of other process. Therefore frames of the other processes will not be taken & these processes will not thrash.

Q.4] Explain virtual memory & demand paging.

→ (i) Virtual Memory:-

① There are many cases where entire program is not needed in main memory at a time.

② In many cases even though entire program is needed, it may not all be needed at the same time.

③ Application programs always get the feeling of availability of contiguous working address space due to the idea of virtual memory.

④ Actually, this working memory can be physically fragmented and may even overflow on to disk storage.

⑤ This technique makes programming of large applications easier and use real physical memory more efficiently than those without virtual memory.

- Virtual memory permits execution of larger size programs although smaller size physical memory is available.
- It means larger size programs than available physical memory can complete the execution.
- Virtual memory concept separate the user logical memory from actual physical memory.
- This separation offers very large virtual memory to programmers although actual physical memory is having very small size.

(ii) Demand Paging :-

① A demand paging is paging system with swapping where pages are brought in main memory from secondary storage on demand.

② When it is needed to execute a process, memory manager swap it into memory.

③ Instead of swapping the entire process into memory, demand paging allows to swap in only those pages which are required for execution at that time.

④ There will be failure in accessing the page, if it is not presently mapped to a frame. This will cause a page fault which is a special type of interrupt.

⑤ To handle this interrupt, disk operation is initiated by OS handler to read the required page in memory.

⑥ After this mapping of page to free frame is carried out. During all this operation the process remains blocked.

⑦ As soon as page becomes available, the blocked process during disk operation is now unblocked, and placed on the ready queue.

⑧ When scheduler schedules this process, it restarts its execution with the instruction that caused the page fault.

(9) The execution will continue untill all the instructions are in memory. This approach of fetching pages as they are needed for execution is called demand paging.

(10) Before swapping in the process in memory, the pages program makes sure about which pages will be used before the process is swapped out again.

(11) The complete process is not swapped in. Instead the pager brings only those required pages into memory.

(12) This would restrict the swapping in of pages in memory which are not needed for execution.

(13) It saves swap time and the amount of physical memory needed

(14) In demand paging, valid and invalid bits are used to differentiate between those pages that are in memory and those pages that are on the disk.

(15) Demand paging keep more processes in memory than the sum of their memory needs ensuring CPU utilization as high as possible.

(16) Continuous allocation of memory to the pages is not necessary.

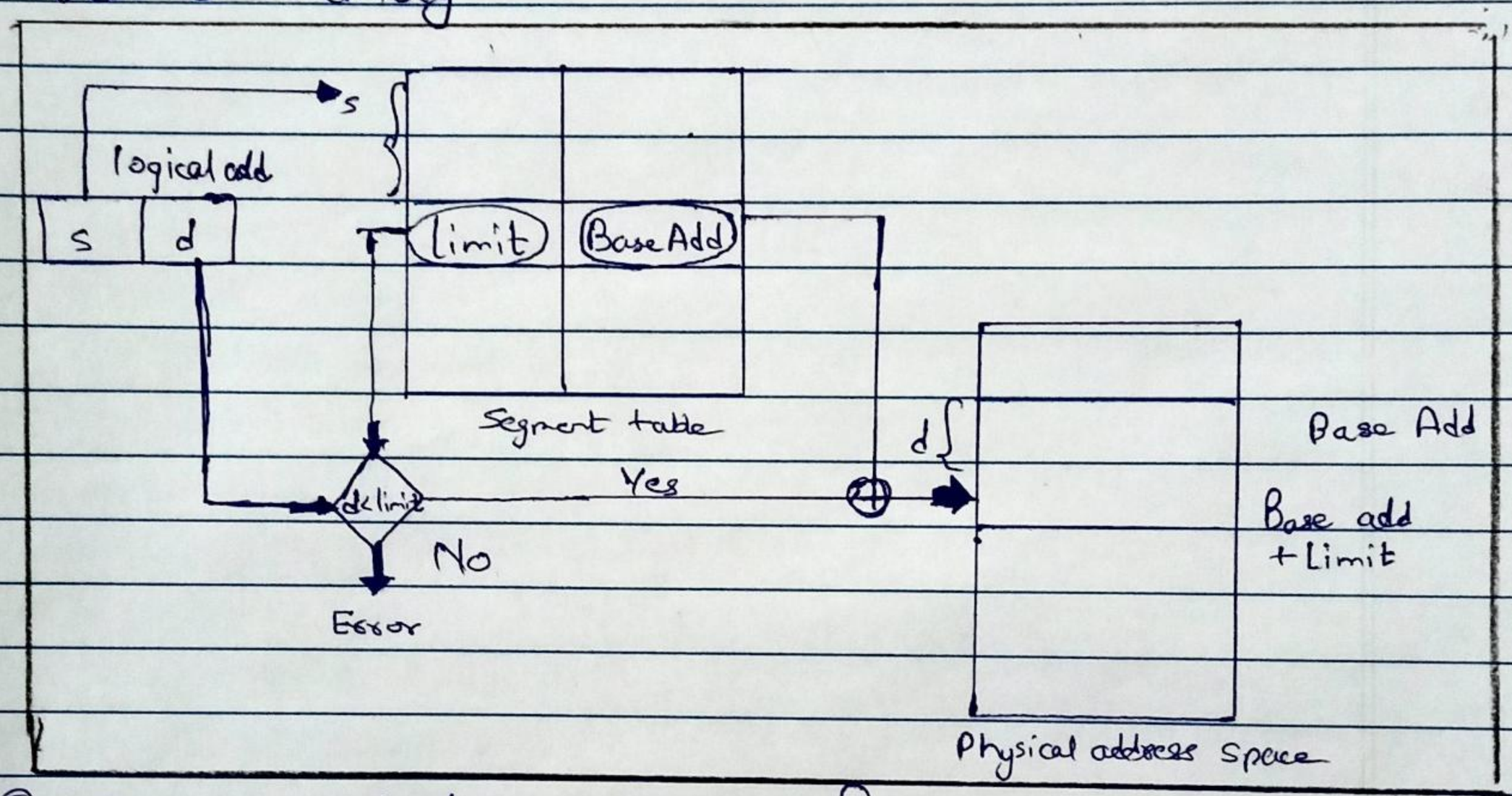
(17) The reason is, any page can be mapped into any available page frame.

(18) The page table maintain the information regarding mapping of page to page fram. Due to this, it offers the impression of having one contiguous block of memory.

(D.5) Explain the concept of segmentation with hardware support.

→ (1) Segmentation is a ~~vertu~~ virtual process that creates variable sized address spaces in computer storage for related data called ~~segmentation~~.

(2) It is a memory management technique which supports user's views of memory.



(3) The mapping is done with help of segment table. Each entry of segment table has base and limits.

(4) The segment base contains starting physical address, where resides in memory whereas limit specifies length of the segment.

(5) The Segment number is used as index of for segment table.

(6) The offset must be between 0 & limits.

(7) If it is not less than limit then it is trapped (addressing the error)

(8) If it is less than limit, then add it to segment base to produce address in physical memory.