Parul® University

NAAC ACCREDITED UNIVERSITY A++

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

OPERATING SYSTEM LABORATORY(303105252)

4TH SEMESTER

COMPUTER SCIENCE AND ENGINEERING

Laboratory Manual

CERTIFICATE

This is to certify that

Mr./Ms.....

..... with enrolment no. ..... has successfully completed his/her laboratory experiments in the OPERATING SYSTEM (303105252) from the department of ……

..... during the academic year .....

Date of Submission:..... Staff In charge:.....

Head Of Department:.....

TABLE OF CONTENT

Sr. No	Experiment Title	Page No		Date of Start	Date of Completion	Sign	Marks (out of 10)
		To	From				
1	Study of Basic commands of Linux.						
2	2.1Study the basics of shell programming. Write Shell Script for Following. 2.2 Adding of Two Numbers. 2.3 Swap Two Variables without Using Third Variable. 2.4 Average Of 3 Numbers. 2.5 Calculate Factorial Of Given Number. 2.6 To validate the entered date. (eg. Date format is: dd-mm-yyyy)						
3	3.1 Write a shell script to check if the entered string is palindrome or not. 3.2 Write a shell script to Print an Array.						
4	Write a C program to create a child process						
5	Finding out biggest number from given three numbers supplied as command line arguments						
6	6.1 Printing the patterns using a for loop. 6.2 Write Shell Script to Print Pyramid.						
7	7.1 Shell script to determine whether given file exist or not. 7.2 Write a Shell Script that prints the even number up to the number given by the user.						
8	Write a program for process creation using C. (Use of gcc compiler).						
9	Implementation of FCFS &Round Robin Algorithm.						
10	Implementation of Banker's Algorithm						

Practical – 1

1. ls (List all the files and directories)

Description: - Display the list information about files.

Example: -

```
ubuntu@ubuntulinux:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
```

2. pwd (Print working directory)

Description: - Print name of the working/current directory.

Example: -

```
ubuntu@ubuntulinux:~$ pwd
/home/ubuntu
```

3. mkdir (make directories)

Description: - Create directory, if they do not already exist.

Example: -

```
ubuntu@ubuntulinux:~$ mkdir OS
ubuntu@ubuntulinux:~$ ls
Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
```

4. rmdir (Remove empty directories)

Description: - Remove the directories, if they are empty.

Example: -

```
ubuntu@ubuntulinux:~$ ls
COMA  Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
ubuntu@ubuntulinux:~$ rmdir COMA
ubuntu@ubuntulinux:~$ ls
Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
```

5. cat (Create files)

Description: - Concatenate files and print on the standard output. **Example: -**

```
ubuntu@ubuntulinux:~$ cat >ch_1
1. Process
2. Operating system
3. types
4. Multiprocesser
5. Multiprogramming
```

```
ubuntu@ubuntulinux:~$ cat ch_1
1. Process
2. Operating system
3. types
4. Multiprocesser
5. Multiprogramming
```

6. cp (Copy files and directories)

Description: - Copy one file data to another file. **Example: -**

```
ubuntu@ubuntulinux:~$ cp ch_1 ch_2
ubuntu@ubuntulinux:~$ cat ch_2
1. Process
2. Operating system
3. types
4. Multiprocesser
5. Multiprogramming
```

7. rm (Remove files or directory)

Description: - Removes each specified file.

- By default, it does not remove directory.

Example: -

```
ubuntu@ubuntulinux:~$ ls
ch  ch_1  ch_2  Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
ubuntu@ubuntulinux:~$ rm ch
ubuntu@ubuntulinux:~$ ls
ch_1  ch_2  Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
```

8. mv (move (rename) files)

Description: - Rename source or move source to directory.

Example: -

```
ubuntu@ubuntulinux:~$ mv ch_1 ch_2
ubuntu@ubuntulinux:~$ ls
ch_2  Desktop  Documents  Downloads  Music  OS  Pictures  Public  snap  Templates  Videos
```

9. uname (Print system information) Description:

- Print certain system information. **Example: -**

10. head (Output the first part of files)

Description: - Print the first 10 lines of each file to standard output.

Example: -

```
ubuntu@ubuntulinux:~$ uname
Linux
```

```
ubuntu@ubuntulinux:~$ head simple
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
ubuntu@ubuntulinux:~$ head -4 simple
line 1
line 2
line 3
line 4
```

11. tail (Output the last part of files)

Description: - Print the last 10 lines of each file to standard output. **Example: -**

```
ubuntu@ubuntulinux:~$ tail simple
line 6
line 7
line 8
line 9
line 10
line 11
line 12
line 13
line 14
line 15
ubuntu@ubuntulinux:~$ tail -4 simple
line 12
line 13
line 14
line 15
```

12. grep (Print lines matching a pattern)

Description: - grep searches for the pattern in each file. Use to find specific string in series of output.

Example: -

```
ubuntu@ubuntulinux:~$ grep "i" simple
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
line 11
line 12
line 13
line 14
line 15
```

13. history (history library)

Description: - Shows a list of previously executed commands, including their line numbers. **Example: -**

```
ubuntu@ubuntulinux:~$ history
    1  man mv
    2  man head
    3  man tail
    4  man history
    5  man clear
    6  man echo
    7  man cal
    8  man chmod
    9  man cd
   10  mkdir rinkal
   11  cd rinkal
   12  cd
   13  man hostname
   14  man ps
   15  man df
   16  locate
   17  ls
   18  pwd
   19  mkdir OS
```

14. clear (Clear the terminal screen)

Description: - Clear the terminal screen, providing a clean slate for new commands or output.

Example: -

```
ubuntu@ubuntulinux:~$ clear
```

15. Echo (Display a line of text)

Description: - echo the string to standard output. **Example: -**

```
ubuntu@ubuntulinux:~$ echo "Hello World"
Hello World
```

16. man (manuals)

Description: - an interface to the online reference manuals.

Example: -

```
PWD(1)                                                  User Commands

NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd [OPTION]...

DESCRIPTION
       Print the full filename of the current working directory.

       -L, --logical
              use PWD from environment, even if it contains symlinks

       -P, --physical
              avoid all symlinks
```

17. cd (Change directory)

Description: - The cd command is used to change the current working directory. Without any arguments, it takes you to your home directory. **Example: -**

```
ubuntu@ubuntulinux:~$ cd OS
ubuntu@ubuntulinux:~/OS$
```

18. Whoami (Print effective user I'd)

Description: - Print the user name associated with the current effective use I'd. **Example: -**

```
ubuntu@ubuntulinux:~$ whoami
ubuntu
```

19. Sort (Sort lines of text files)

Description: - Sorts the lines in unsorted text alphabetically and prints the result. **Example: -**

```
ubuntu@ubuntulinux:~$ sort simple
line 1
line 10
line 11
line 12
line 13
line 14
line 15
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
```

20. hostname (System hostname)

Description: - Show or set the system's hostname. **Example: -**

```
ubuntu@ubuntulinux:~$ hostname
ubuntulinux
```

21. ps (Process status)

Description: - Display information about active processes.

Example: -

```
ubuntu@ubuntulinux:~$ ps
    PID TTY          TIME CMD
  32637 pts/0    00:00:00 bash
  34552 pts/0    00:00:00 ps
```

22. df (Disk free)

Description: - Report file system disk space usage.

Example: -

```
ubuntu@ubuntulinux:~$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
tmpfs             300996     1628    299368   1% /run
/dev/sda3       20028636 13524140   5461760  72% /
tmpfs            1504976        0   1504976   0% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
/dev/sda2         524252     6220    518032   2% /boot/efi
tmpfs             300992      184    300808   1% /run/user/1000
```

23. find (Find files)

Description: - Search for all files with a .txt extension in the specified directory and its sub-directories.

Example: -

```
ubuntu@ubuntulinux:~$ find simple
simple
```

24. time (execution time)

Description: - Display the information about resources used by command. If command exits with non-zero status that's means time display a warning message and the exit status.

Example: -

```
ubuntu@ubuntulinux:~$ time

real    0m0.000s
user    0m0.000s
sys     0m0.000s
```

25.chmod (Change mode)

Description: - Change file or directory mode.

Example: -

```
ubuntu@ubuntulinux:~$ ls -l
total 48
-rw-rw-r-- 1 ubuntu ubuntu    79 Dec 29 18:07 ch_2
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Desktop
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Documents
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Downloads
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Music
drwxrwxr-x 2 ubuntu ubuntu  4096 Dec 28 23:37 OS
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Pictures
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Public
-rw-rw-r-- 1 ubuntu ubuntu   111 Dec 29 18:38 simple
drwx------ 4 ubuntu ubuntu  4096 Dec 29 19:01 snap
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Templates
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Videos
ubuntu@ubuntulinux:~$ chmod 666 simple
ubuntu@ubuntulinux:~$ ls -l
total 48
-rw-rw-r-- 1 ubuntu ubuntu    79 Dec 29 18:07 ch_2
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Desktop
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Documents
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Downloads
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Music
drwxrwxr-x 2 ubuntu ubuntu  4096 Dec 28 23:37 OS
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Pictures
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Public
-rw-rw-rw- 1 ubuntu ubuntu   111 Dec 29 18:38 simple
drwx------ 4 ubuntu ubuntu  4096 Dec 29 19:01 snap
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Templates
drwxr-xr-x 2 ubuntu ubuntu  4096 Dec 27 23:27 Videos
ubuntu@ubuntulinux:~$
```

Practical – 2

AIM: Study the basics of shell programming.

THEORY: Basics programs of shell scripting are as follows:

The steps in creating a shell script:

1. Create a file using a vi editor name script file with extension .sh Ex. Addition.sh
2. Then press i (i=insert mode)
3. Start the script with #!/bin/bash
4. Write your code
5. Then press esc
6. :x and Enter or :wq! (! = save and exit)
7. Change the mode execution
8. Run – bash filename.sh or ./filename.sh

Programs:

Program 1: Write Hello World using shell script.

Code:

```
#!/bin/sh
echo "Hello World"
echo 'Hello World'
echo Hello World
```

Output:

```
ubuntu@ubuntulinux:~$ vi Hello.sh
ubuntu@ubuntulinux:~$ bash Hello.sh
Hello World
Hello World
Hello World
ubuntu@ubuntulinux:~$
```

Program 2: Write a shell script program to add two numbers.

Code:

```
#!/bin/sh
echo Enter the first number a
read a
echo Enter the second number b
read b
sum=$(($a+$b))
echo Sum is $sum
~
```

Output:

```
ubuntu@ubuntulinux:~$ vi add.sh
ubuntu@ubuntulinux:~$ bash add.sh
Enter the first number a
10
Enter the second number b
20
Sum is 30
ubuntu@ubuntulinux:~$
```

Program 3: Swap two variables without using third variable.

Code:

```
#!/bin/sh
echo Enter the first number a
read a
echo Enter the second number b
read b
a=$((a+b))
b=$((a-b))
a=$((a-b))
echo After swapping first number a = $a and second number b = $b
~
~
```

Output:

```
ubuntu@ubuntulinux:~$ vi swap.sh
ubuntu@ubuntulinux:~$ bash swap.sh
Enter the first number a
10
Enter the second number b
20
After swapping first number a = 20 and second number b = 10
ubuntu@ubuntulinux:~$
```

Program 4: Find average of three numbers.

Code:

```
#!/bin/sh
echo Enter the first number a
read a
echo Enter the second number b
read b
echo Enter the third number c
read c
sum=$(($a+$b+$c))
average=$((sum/3))
echo Average of three number is $average
```

Output:

```
ubuntu@ubuntulinux:~$ vi average.sh
ubuntu@ubuntulinux:~$ bash average.sh
Enter the first number a
10
Enter the second number b
20
Enter the third number c
30
Average of three number is 20
ubuntu@ubuntulinux:~$
```

Program 5: Calculate factorial of given number.

Code:

```
#!/bash/sh
echo Enter number
read num
fact=1
for((i=2;i<=num;i++))
{
  fact=$((fact*i))
}
echo Factorial is $fact
```

Output:

```
ubuntu@ubuntulinux:~$ vi factorial.sh
ubuntu@ubuntulinux:~$ bash factorial.sh
Enter number
5
Factorial is 120
ubuntu@ubuntulinux:~$
```

Practical – 3

Program 1: Write a shell script to check entered string is palindrome or not.

Code:

```
#!/bin/sh
echo "Enter the string"
read string
reverse=" "
len=${#string}
for (( i=$len-1; i>=0; i-- ))
do
        reverse="$reverse${string:$i:1}"
done
if [ $string == $reverse ]
then
        echo "$String is Palindrome"
else
        echo "$String is not Palindrome"
fi
```

Output:

```
ubuntu@ubuntulinux:~$ vi palindrome.sh
ubuntu@ubuntulinux:~$ bash palindrome.sh
Enter the string
mom
 is Palindrome
ubuntu@ubuntulinux:~$ bash palindrome.sh
Enter the string
system
 is not Palindrome
ubuntu@ubuntulinux:~$
```

Program 2: Write a shell script to Print an Array.

Code:

```
#!/bin/sh
echo "Enter array"
read arr
echo "array list:- { ${arr[@]} }"
```

Output:

```
ubuntu@ubuntulinux:~$ vi array.sh
ubuntu@ubuntulinux:~$ bash array.sh
Enter array
Apple,Orange,Banana,Cherry
array list:- { Apple,Orange,Banana,Cherry }
ubuntu@ubuntulinux:~$
```

Practical – 4

1. Write a c program to create a child process.

```c
#include <unistd.h>
#include <stdlib.h>

int main() {
    pid_t p = fork();
    if (p < 0) {
        perror("fork failed");
        exit(1);
    } else if (p == 0) {

        printf("Hello from Child! Process ID (PID): %d\n", getpid());
    } else {

        printf("Hello from Parent! Process ID (PID): %d\n", getpid());
    }
    return 0;
}

```

Output :-

```
/tmp/USn5o314sN.o
Hello from Parent! Process ID (PID): 7773
Hello from Child! Process ID (PID): 7774
```

Practical – 5

AIM: Finding out biggest number from given three numbers supplied as command line arguments

```
Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ read n1
15

Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ read n2
2

Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ read n3
77

Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
> then
> echo "$n1 is greater"
> elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
> then
> echo "$n2 is greater"
> else
> echo "$n3 is greater"
> fi
77 is greater
```

Practical – 6

AIM: Printing the patterns using a for loop.

- **Write Shell Script to Print Pyramid.**

```
Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ p=6

Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ for((row=1;row<=p;row++))
> do
> for((s=row;s<=p;s++))
> do
> echo -ne " "
> done
> for((j=1;j<=row;j++))
> do
> echo -ne "$j"
> done
> echo
> done
     1
    12
   123
  1234
 12345
123456
```

Practical – 7

1 Shell script to determine whether given file exist or not.

```
Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ if [ -f "Ansh.txt" ];
> then
> echo "File exists"
> else
> echo "File does not exist"
> fi
File does not exist
```

2 Write a Shell Script that prints the even number up to the number given by the user.

```
Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ read number
15

Sandeep Kedia@DESKTOP-3OOD7CD MINGW64 /d/
$ for i in $(seq 1 $number); do
> if [ $((i % 2)) -eq 0 ]; then
> echo $i
> fi
> done
2
4
6
8
10
12
14
```

Practical – 8

AIM: Write a program for process creation using C. (Use of gcc compiler).

Practical – 9

AIM: Implementation of FCFS &Round Robin

Algorithm.

➢ // TO CALCULATE AVERAGE WAITING TIME USING FIRST COME FIRST SERVE SCHEDULING

- **CODE :**

```
#include <stdio.h>
int main()
{
int p[10], at[10], bt[10], ct[10], tat[10], wt[10], i, j, temp = 0, n;
float awt = 0, atat = 0;
printf("ENTER THE NO. OF PROCESSES:");
scanf("%d", &n);
printf("ENTER %d
PROCESS:", n); 11                for (i =
0; i < n; i++)
{
scanf("%d",
&p[i]); 14 }
15 printf("ENTER %d ARRIVAL TIME:", n); 16 for (i
= 0; i < n; i++)
{
scanf("%d",
&at[i]); 19 }
20            printf("ENTER %d BURST
TIME:", n); 21 for (i = 0; i < n; i++)
{
scanf("%d",
&bt[i]); 24 }
25          // sorting at,bt, and process according to at 26      for (i = 0; i < n;
i++)
{
for (j = 0; j < (n - i); j++)
{
if (at[j] > at[j + 1])
{
temp = p[j + 1]; 33  p[j + 1] = p[j];
p[j] = temp;
temp = at[j + 1];
at[j + 1] = at[j];
at[j] = temp;
temp = bt[j + 1];
bt[j + 1] = bt[j];
bt[j] = temp;
}
}
}
/* calculating 1st ct */ 45       ct[0] = at[0] + bt[0];
46          /* calculating 2 to n ct */ 47 for (i = 1;
i < n; i++)
{
```

```
43. // when proess is ideal in between i and i+1
44. temp = 0;
45. if (ct[i - 1] < at[i])
46. {
47. temp = at[i] -
48. ct[i - 1]; 54 }
49. 55                          ct[i] = ct[i - 1] + bt[i]
50. + temp; 56 }
51. /* calculating tat and wt */
52. printf("\nP\t A.T\t B.T\t C.T\t
53. TAT\t WT"); 59 for (i = 0; i < n; i++)
54. {
55. tat[i] = ct[i] - at[i];
56. wt[i] = tat[i] - bt[i];
57. atat += tat[i]; 64      awt +=
58. wt[i]; 65 } 66          atat = atat / n;

59. awt = awt / n;
60. for (i = 0; i < n; i++)
61. {
62. printf("\nP%d\t %d\t %d\t %d \t %d \t %d", p[i], at[i], bt[i], ct[i],
63. tat[i], wt[i]);
64. }
65. printf("\n\nAVERAGE TURN AROUND TIME IS : %f", atat);
66. 74
67. printf("\nAVERAGE WAITING TIME IS : %f", awt);
68. return 0;
69. }
```

```
ENTER THE NO. OF PROCESSES:5
ENTER 5 PROCESS:1 2 3 4 5
ENTER 5 ARRIVAL TIME:0 2 4 6 8
ENTER 5 BURST TIME:2 4 6 7 9

P       A.T     B.T     C.T     TAT     WT
P1      0       2       2       2       0
P0      0       0       2       2       2
P2      2       4       6       4       0
P3      4       6       12      8       2
P4      6       7       19      13      6

AVERAGE TURN AROUND TIME IS : 5.800000
AVERAGE WAITING TIME IS : 2.000000
```

➢ // TO CALCULATE AVERAGE WAITING TIME USING ROUND ROBIN SCHEDULING

- CODE:

```
#include<stdio.h>
#include<conio.h>
 void main() 7 {
// initlialize the variable name
int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
float avg_wt, avg_tat;
printf("TOTAL NO. OF PROCESSES: ");
scanf("%d", &NOP);
y = NOP; // Assign the number of process to variable y 14
// Use loop to enter the details of the process like Arrival ime and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\nENTER THE ARRIVAL TIME AND BURST TIME : [%d]\n", i+1);
printf("ARRIVAL TIME IS : "); // Accept arrival time
scanf("%d", &at[i]);
printf("\nBURST TIME IS : "); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("ENTER QUANTUM TIME(MAX. TIME) : ");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n P \t\t BT \t\t TAT \t\t WT ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
sum = sum + temp[i];
temp[i] = 0;
count=1;
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - quant;
sum = sum + quant;
if(temp[i]==0 && count==1)
{
y--; //decrement the process no.
printf("\nP[%d]\t\t %d\t\t %d\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i ]);
wt = wt+sum-at[i]-bt[i];
tat = tat+sum-at[i];
count =0;
if(i==NOP-1)
{
i=0;
}
else if(at[i+1]<=sum)
{
i++;
}
else {
i=0;
}
```

```
55. }
56. // represents the average waiting time
57. avg_wt = wt * 1.0/NOP;
58. printf("\n\nAVERAGE WAITING TIME : \t%f", avg_wt);
59. getch();
60. }
```

```
TOTAL NO. OF PROCESSES: 4

ENTER THE ARRIVAL TIME AND BURST TIME : [1]
ARRIVAL TIME IS : 0

BURST TIME IS : 7

ENTER THE ARRIVAL TIME AND BURST TIME : [2]
ARRIVAL TIME IS : 1

BURST TIME IS : 4

ENTER THE ARRIVAL TIME AND BURST TIME : [3]
ARRIVAL TIME IS : 2

BURST TIME IS : 3

ENTER THE ARRIVAL TIME AND BURST TIME : [4]
ARRIVAL TIME IS : 3

BURST TIME IS : 1
ENTER QUANTUM TIME(MAX. TIME) : 3

 P               BT              TAT             WT
P[3]             3               7               4
P[4]             1               7               6
P[2]             4               13              9
P[1]             7               15              8

AVERAGE WAITING TIME :   6.750000
```

Practical – 10

AIM: Implementation of Banker's Algorithm

- Banker's Algorithm
- ✓ CODE :

```
#include <stdio.h>
int main()
{
// P0, P1, P2, P3, P4 are the Process names here


int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                    { 2, 0, 0 }, // P1
                    { 3, 0, 2 }, // P2
                    { 2, 1, 1 }, // P3
                    { 0, 0, 2 } }; // P4


int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                  { 3, 2, 2 }, // P1
                  { 9, 0, 2 }, // P2
                  { 2, 2, 2 }, // P3
                  { 4, 3, 3 } }; // P4


int avail[3] = { 3, 3, 2 }; // Available Resources


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);
}
```

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
Process returned 0 (0x0)   execution time : 0.310 s
Press any key to continue.
```