

OS Lab Question Set Programs

Computer Science and Engineering

Sona College OF Technology



DISCLAIMER :

- ✓ Itha Code laam Correct ah Thappa nuh Engaluku theriyaathu, Engaluku therinja vara pootrukoom.
- ✓ With Output ooda thaa Programs Pootrukoom, In-case ungaluku lab la Output varalana athuku naanga porupillai.

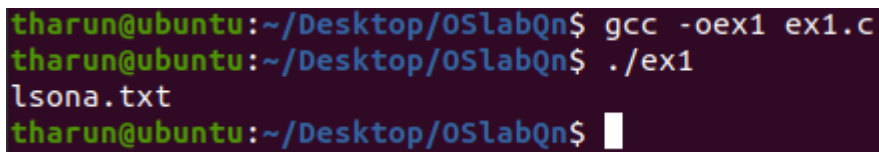
- ✓ We Don't know whether the below Codes are Correct or not, these are as per our knowledge.
- ✓ And the Codes given below are with Output..., We are not responsible, In-case if you didn't get the Output in the Lab.

1. Write a C program to simulate the 'ls' command that will display all the files starting with a letter 'l'.

Program :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    system("ls l*");
    return 0;
}
```

Output :



```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex1 ex1.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex1
lsona.txt
tharun@ubuntu:~/Desktop/OSlabQn$
```

2. Write a C program to simulate the 'grep' command that will count the number of occurrence of a give word in a file.

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUFFER_SIZE 1000
int countOccurrences(FILE *fptr, const char *word);
int main()
{
    FILE *fptr;
    char path[100];
    char word[50];
    int wCount;
    printf("Enter file path: ");
    scanf("%s", path);
    printf("Enter word to search in file: ");
    scanf("%s", word);
```

```

    fptr = fopen(path, "r");
    if (fptr == NULL)
    {
        printf("Unable to open file.\n");
        printf("Please check you have read/write previleges.\n");
        exit(EXIT_FAILURE);
    }
    wCount = countOccurrences(fptr, word);
    printf("%s' is found %d times in file.", word, wCount);
    fclose(fptr);
    return 0;
}

```

```

int countOccurrences(FILE *fptr, const char *word)
{
    char str[BUFFER_SIZE];
    char *pos;
    int index, count;
    count = 0;
    while ((fgets(str, BUFFER_SIZE, fptr)) != NULL)
    {
        index = 0;
        while ((pos = strstr(str + index, word)) != NULL)
        {
            index = (pos - str) + 1;

            count++;
        }
    }
    return count;
}

```

Output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -ogrep grep.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./grep
Enter file path: data.txt
Enter word to search in file: OS
'OS' is found 2 times in file.
tharun@ubuntu:~/Desktop/OSlabQn$
```

3. Write a C program to display the number of words in each file in the current working directory.

Program :

```
#include <dirent.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
int main(void){
    DIR *d;//Stores all sub directory and file information
    struct dirent *dir;//dir stores individual file / directory details
    char *filename;//to store the filename
    char *command = "/bin/wc";
    char *arg1 = "-w";
    int i=0;
    d=opendir(".");
    if(d){
        while((dir=readdir(d))!=NULL){
            if(dir->d_type==DT_REG){
                //Retrive the file name and pass it as third option
                filename = (char *) malloc(sizeof(char)*strlen(dir->d_name));
                strncat(filename,dir->d_name,strlen(dir->d_name));
                //create a new child process and execute execl call
                if(fork()==0){
                    execl(command,command,arg1,filename,NULL);
                }
                i++;
            }
        }
        closedir(d);
    }
    return 0;
}
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex3 ex3.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex3
65 ex9.sh
7 ex14.sh
139 ex18
83 ex3.c
tharun@ubuntu:~/Desktop/OSlabQn$ 93 ex12.c
67 test.c
62 ex17.c
11 ex1.c
93 ex12
49 ex13.sh
98 ex8.sh
69 ex16.c
103 data.txt
172 ex4.c
105 file.txt
8 cse.txt
374 ex18.c
24 ex15.sh
85 ex17
8 lsona.txt
58 test
56 ex3
49 ex1
69 ex16
89 grep
77 ex5
11 ex7.sh
31 ex6.c
34 ex5.c
232 grep.c
64 ex6
77 ex4
```

4. Write a C program to list the files in the specified directory.

Program :

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
void tree(char *basePath, const int root);
int main()
{
```

```

    char path[100];
    printf("Enter path to list files: ");
    scanf("%s", path);
    tree(path, 0);
    return 0;
}
void tree(char *basePath, const int root)
{
    int i;
    char path[1000];
    struct dirent *dp;
    DIR *dir = opendir(basePath);
    if (!dir)
        return;
    while ((dp = readdir(dir)) != NULL)
    {
        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
        {
            for (i=0; i<root; i++)
            {
                if (i%2 == 0 || i == 0)
                    printf("%c", 179);
                else
                    printf(" ");
            }
            printf("%c%c%s\n", 195, 196, dp->d_name);
            strcpy(path, basePath);
            strcat(path, "/");
            strcat(path, dp->d_name);
            tree(path, root + 2);
        }
    }
    closedir(dir);
}

```

Output :

```

tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex4 ex4.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex4
Enter path to list files: sona
♦♦data.txt
♦♦test.c
♦♦test
♦♦grep
♦♦grep.c
♦♦ex4.c
tharun@ubuntu:~/Desktop/OSlabQn$

```

5. Write a C program to create a specified number of child processes (given through command line) and display the pid of child processes along with its parent's pid.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
    int n;
    printf("Enter the number of processes to be created\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        if (fork()==0)
        {
            printf("\t[child] pid %d from[parents] pid %d\n",getpid(),getppid());
            exit(0);
        }
    }
    return 0;
}
```

Output :



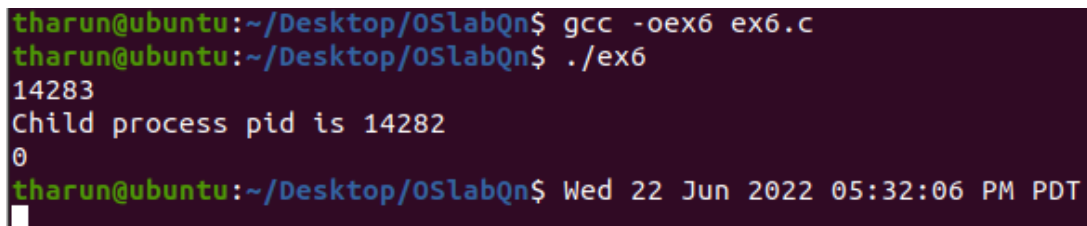
```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex5 ex5.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex5
Enter the number of processes to be created
15
    [child] pid 14189 from[parents] pid 14188
    [child] pid 14190 from[parents] pid 14188
    [child] pid 14191 from[parents] pid 14188
    [child] pid 14192 from[parents] pid 14188
    [child] pid 14194 from[parents] pid 14188
    [child] pid 14193 from[parents] pid 14188
    [child] pid 14195 from[parents] pid 14188
    [child] pid 14197 from[parents] pid 14188
    [child] pid 14196 from[parents] pid 14188
    [child] pid 14198 from[parents] pid 14188
    [child] pid 14199 from[parents] pid 14188
    [child] pid 14201 from[parents] pid 14188
    [child] pid 14200 from[parents] pid 14188
    [child] pid 14202 from[parents] pid 14188
    [child] pid 14203 from[parents] pid 1532
tharun@ubuntu:~/Desktop/OSlabQn$
```


6. Write a C program using fork() and execv() system calls to spawn a child and the child should print the date and the parent should print the child's pid and status.

Program :

```
#include<stdio.h>
#include<unistd.h>
int main(){
    pid_t pid;
    char*const paramList[]={"/bin/date",NULL};
    pid=fork();
    printf("%d\n",pid);
    if(pid==-1){
        printf("Error is creating child process\n");
    }
    else if(pid==0){
        execv("/bin/date",paramList);
    }
    else{
        printf("Child process pid is %d\n",getpid());
    }
    return 0;
}
```

Output :



```
ttharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex6 ex6.c
ttharun@ubuntu:~/Desktop/OSlabQn$ ./ex6
14283
Child process pid is 14282
0
ttharun@ubuntu:~/Desktop/OSlabQn$ Wed 22 Jun 2022 05:32:06 PM PDT
```

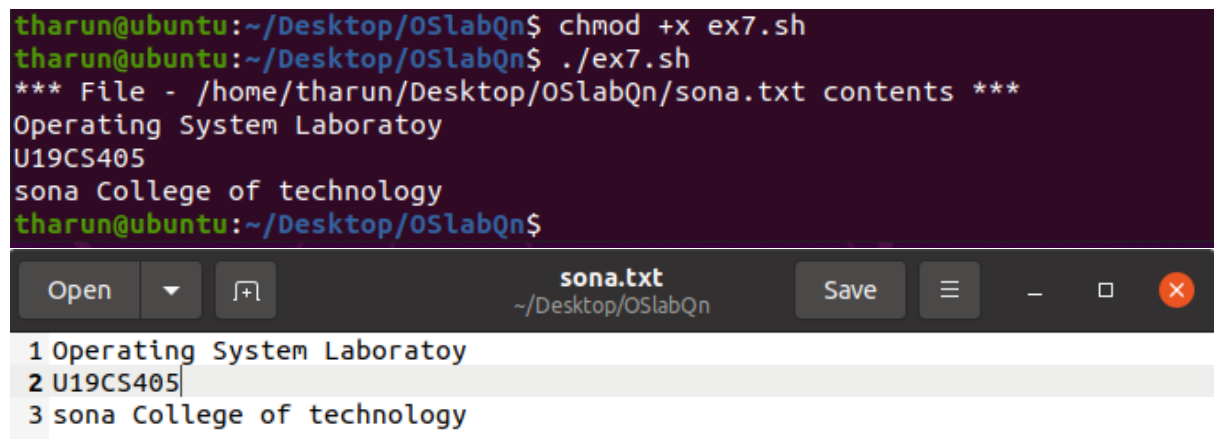
7. Write a shell script to display the content of file named sonacse.txt in the current working directory or in the subdirectories.

Program :

```
#!/bin/bash
FILE="/home/tharun/Desktop/OSlabQn/sona.txt"
echo "*** File - $FILE contents ***"
cat $FILE
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex7.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex7.sh
*** File - /home/tharun/Desktop/OSlabQn/sona.txt contents ***
Operating System Laboratoy
U19CS405
sona College of technology
tharun@ubuntu:~/Desktop/OSlabQn$
```



The screenshot shows a terminal window with a dark background. The user 'tharun' is at the 'ubuntu' machine in the directory '~/Desktop/OSlabQn'. They run 'chmod +x ex7.sh' and then './ex7.sh'. The script outputs the contents of '/home/tharun/Desktop/OSlabQn/sona.txt', which are: 'Operating System Laboratoy', 'U19CS405', and 'sona College of technology'. Below the terminal, a file editor window titled 'sona.txt' is open, showing the same three lines of text.

8. Write a shell script which accepts two filenames from command line, copies the first file to the second and then displays it

Program :

```
echo -n "Enter soruce file name : "
read src
echo -n "Enter target file name : "
read targ

if [ ! -f $src ]
then
    echo "File $src does not exists"
    exit 1
elif [ -f $targ ]
then
    echo "File $targ exist, cannot overwrite"
    exit 2
fi
cp $src $targ
status=$?
if [ $status -eq 0 ]
then
    echo 'File copied successfully'
else
    echo 'Problem copying file'
fi
cat $src
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex8.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex8.sh
Enter soruce file name : sona.txt
Enter target file name : cse.txt
File copied successfully
Operating System Laboratoy
U19CS405
sona College of technology
tharun@ubuntu:~/Desktop/OSlabQn$
```

9. Create a file named sample.txt with 20 lines and do the following using shell script.
- Count the number of words in the file.
 - Display the first and last five lines of the contents.
 - Find the occurrence of any word.
 - Display the contents of the file.

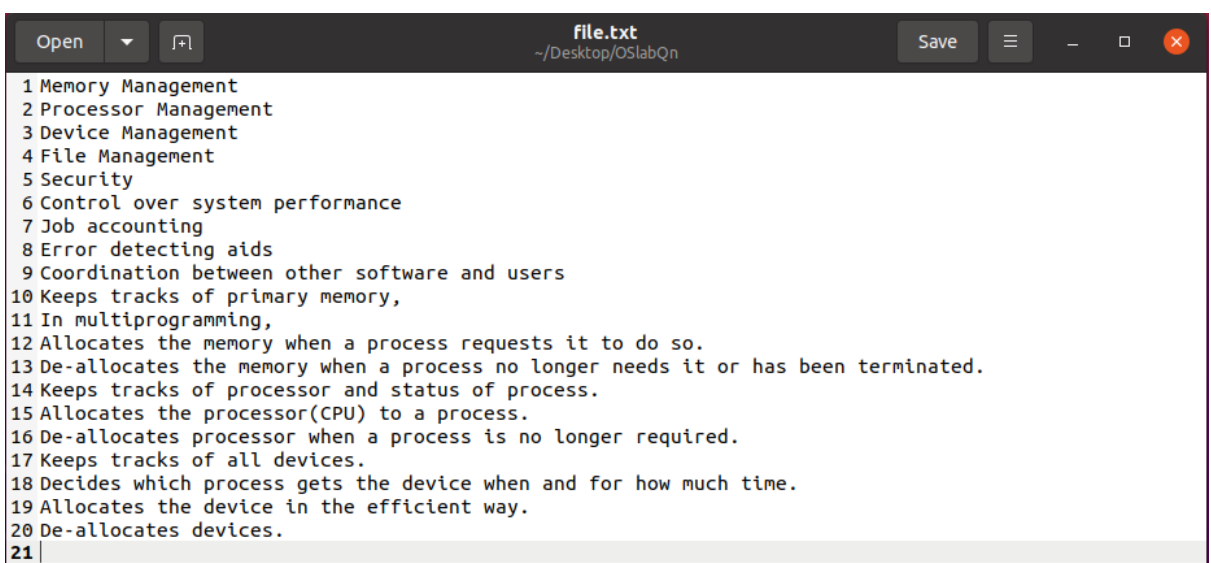
Program :

```
#!/usr/bin/bash
echo -n "Enter soruce file name : "
read src
now=`wc --word < $src`
echo "-----"
echo "1.Number of words: $now"
echo "-----"
echo "2.printing the starting and last 5 lines of file"
head -5 $src
echo "-----"
tail -5 $src
echo "-----"
echo "3.Occerence of the file"
egrep -c '\<Management\>' $src
echo "-----"
echo "4.Displaying the contents of the file"
cat $src
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex9.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex9.sh
Enter source file name : file.txt
-----
1.Number of words: 105
-----
2.printing the starting and last 5 lines of file
Memory Management
Processor Management
Device Management
File Management
Security
-----
Keeps tracks of all devices.
Decides which process gets the device when and for how much time.
Allocates the device in the efficient way.
De-allocates devices.
-----
3.Occurrence of the file
4
-----
4.Displaying the contents of the file
Memory Management
Processor Management
Device Management
File Management
Security
Control over system performance
Job accounting
Error detecting aids
Coordination between other software and users
Keeps tracks of primary memory,
In multiprogramming,
Allocates the memory when a process requests it to do so.
De-allocates the memory when a process no longer needs it or has been terminated.
Keeps tracks of processor and status of process.
Allocates the processor(CPU) to a process.
De-allocates processor when a process is no longer required.
Keeps tracks of all devices.
Decides which process gets the device when and for how much time.
Allocates the device in the efficient way.
De-allocates devices.

tharun@ubuntu:~/Desktop/OSlabQn$
```



```
Open file.txt ~/Desktop/OSlabQn Save
1 Memory Management
2 Processor Management
3 Device Management
4 File Management
5 Security
6 Control over system performance
7 Job accounting
8 Error detecting aids
9 Coordination between other software and users
10 Keeps tracks of primary memory,
11 In multiprogramming,
12 Allocates the memory when a process requests it to do so.
13 De-allocates the memory when a process no longer needs it or has been terminated.
14 Keeps tracks of processor and status of process.
15 Allocates the processor(CPU) to a process.
16 De-allocates processor when a process is no longer required.
17 Keeps tracks of all devices.
18 Decides which process gets the device when and for how much time.
19 Allocates the device in the efficient way.
20 De-allocates devices.
21
```

10. Write a C program to create a specified number of child processes (given through command line) and display the pid of child processes along with its parent's pid.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
    int n;
    printf("Enter the number of processes to be created\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        if (fork()==0)
        {
            printf("\t[child] pid %d from[parents] pid %d\n",getpid(),getppid());
            exit(0);
        }
    }
    return 0;
}
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex5 ex5.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex5
Enter the number of processes to be created
15
\t[child] pid 14189 from[parents] pid 14188
\t[child] pid 14190 from[parents] pid 14188
\t[child] pid 14191 from[parents] pid 14188
\t[child] pid 14192 from[parents] pid 14188
\t[child] pid 14194 from[parents] pid 14188
\t[child] pid 14193 from[parents] pid 14188
\t[child] pid 14195 from[parents] pid 14188
\t[child] pid 14197 from[parents] pid 14188
\t[child] pid 14196 from[parents] pid 14188
\t[child] pid 14198 from[parents] pid 14188
\t[child] pid 14199 from[parents] pid 14188
\t[child] pid 14201 from[parents] pid 14188
\t[child] pid 14200 from[parents] pid 14188
\t[child] pid 14202 from[parents] pid 14188
\t[child] pid 14203 from[parents] pid 1532
tharun@ubuntu:~/Desktop/OSlabQn$
```

11. Write a C program to create a specified number of child processes (given through command line) and display the pid of child processes along with its parent's pid.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
    int n;
    printf("Enter the number of processes to be created\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        if (fork()==0)
        {
            printf("\t[child] pid %d from[parents] pid %d\n",getpid(),getppid());
            exit(0);
        }
    }
    return 0;
}
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex5 ex5.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex5
Enter the number of processes to be created
15
\t[child] pid 14189 from[parents] pid 14188
\t[child] pid 14190 from[parents] pid 14188
\t[child] pid 14191 from[parents] pid 14188
\t[child] pid 14192 from[parents] pid 14188
\t[child] pid 14194 from[parents] pid 14188
\t[child] pid 14193 from[parents] pid 14188
\t[child] pid 14195 from[parents] pid 14188
\t[child] pid 14197 from[parents] pid 14188
\t[child] pid 14196 from[parents] pid 14188
\t[child] pid 14198 from[parents] pid 14188
\t[child] pid 14199 from[parents] pid 14188
\t[child] pid 14201 from[parents] pid 14188
\t[child] pid 14200 from[parents] pid 14188
\t[child] pid 14202 from[parents] pid 14188
\t[child] pid 14203 from[parents] pid 1532
tharun@ubuntu:~/Desktop/OSlabQn$
```

12. Write a C program to simulate the ls and grep commands along with its variations.

Program :

```
#include<stdio.h>
#include<dirent.h>
#include <stdlib.h>
int main()
{
    struct dirent *d;
    char filename[100], c;
    DIR *dr;
    FILE *fptr;
    dr = opendir(".");
    if(dr!=NULL)
    {
        printf("***Listing the Files & Folders of the Current Directory***\n\n");
        for(d=readdir(dr); d!=NULL; d=readdir(dr))
        {
            printf("%s\n", d->d_name);
        }
        closedir(dr);
    }
    else
        printf("\nError occurred while opening the current directory!\n");
    printf("\nEnter the filename to open:\n");
    scanf("%s", filename);
    fptr = fopen(filename, "r");
    if (fptr == NULL)
    {
        printf("\nFile not Available or Cannot open file \n");
        exit(0);
    }
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf ("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    return 0;
}
```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex12
***Listing the Files & Folders of the Current Directory***

ex9.sh
ex18
ex14.sh
data.txt
.
ex8.sh
ex3.c
ex12.c
test.c
ex17.c
ex7.sh
ex12
test
ex6
ex5.c
ex13.sh
ex16.c
ex5
ex16
ex18.c
ex15.sh
grep
cse.txt
..
grep.c
file.txt
ex3
sona.txt
ex6.c
ex4
ex17
ex1
ex4.c
sona
ex1.c

Enter the filename to open:
sona.txt
Operating System Laboratoy
U19CS405
sona College of technology
tharun@ubuntu:~/Desktop/OSlabQn$
```

13. Write a shell script to display the message “Good Morning / Good Afternoon / Good Evening” depending upon the time the user logs in.

Program :

```
h=$(date +"%H")
if [ $h -gt 6 -a $h -le 12 ]
then
echo Good Morning
elif [ $h -gt 12 -a $h -le 16 ]
then
echo Good Afternoon
elif [ $h -gt 16 -a $h -le 20 ]
then
echo Good Evening
else
echo Good Night
fi
```

Output :

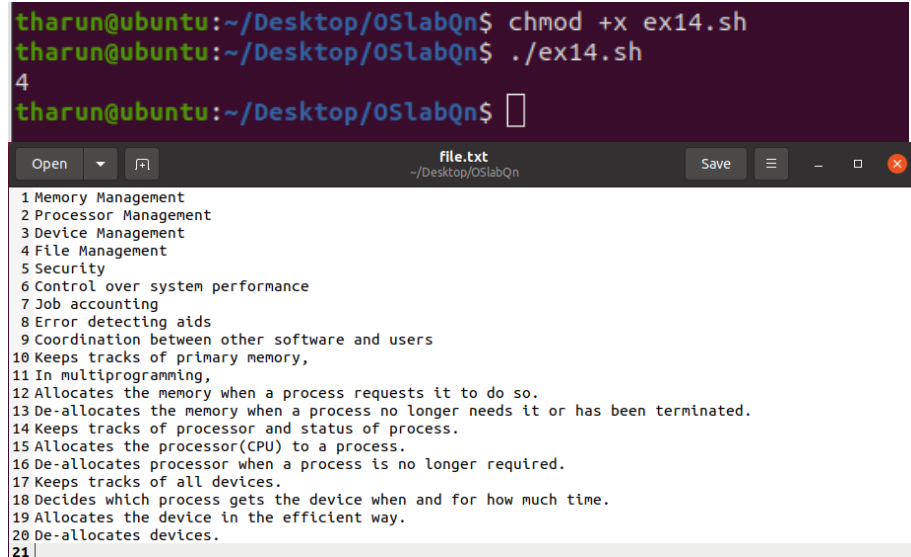
```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex13.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex13.sh
Good Morning
tharun@ubuntu:~/Desktop/OSlabQn$
```


14. Write a shell script to find the number of occurrences of a given word in a word file.

Program :

```
egrep -o 'Management' file.txt | wc -l
```

Output :



The screenshot shows a terminal window and a file named 'file.txt'. The terminal shows the following commands and output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex14.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex14.sh
4
tharun@ubuntu:~/Desktop/OSlabQn$
```

The file 'file.txt' contains the following text:

```
1 Memory Management
2 Processor Management
3 Device Management
4 File Management
5 Security
6 Control over system performance
7 Job accounting
8 Error detecting aids
9 Coordination between other software and users
10 Keeps tracks of primary memory,
11 In multiprogramming,
12 Allocates the memory when a process requests it to do so.
13 De-allocates the memory when a process no longer needs it or has been terminated.
14 Keeps tracks of processor and status of process.
15 Allocates the processor(CPU) to a process.
16 De-allocates processor when a process is no longer required.
17 Keeps tracks of all devices.
18 Decides which process gets the device when and for how much time.
19 Allocates the device in the efficient way.
20 De-allocates devices.
21
```

15. Write a script that would take command line input a number and a word. It then prints the word n times, one word per line.

Program :

```
echo -n "Enter the Word : "
read wd
echo -n "Enter the Number : "
read num
```

```
for (( i=1;i<=$num; i++))
do
    echo $wd
```

```
done
```

Output :



The screenshot shows a terminal window with the following commands and output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ chmod +x ex15.sh
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex15.sh
Enter the Word :Tharun
Enter the Number :5
Tharun
Tharun
Tharun
Tharun
Tharun
tharun@ubuntu:~/Desktop/OSlabQn$
```

16. Write a C program to find whether a file in the current working directory and a file in the immediate subdirectory exist with the same name. If so, remove the duplicate one.

Program :

```
#include <dirent.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
int main(void){
    DIR *d,*td;//Stores all sub directory and file information
    struct dirent *dir,*tempdir;//dir stores individual file / directory details
    char *filename="sona";//to store the filename
    char *fname="sona.txt";
    char *command="/bin/cat";
    char *command2="/bin/rm";
    //char *arg1 = "-w";
    int tf=0;
    int i=0;
    d=opendir(".");
    if(d){
        while((dir=readdir(d))!=NULL) {
            if(dir->d_type==DT_REG && strcmp(fname,dir->d_name)==0) {
                if(tf!=1){
                    system("cat sona.txt");
                    tf=1;
                    break;
                }
            }
        }
        closedir(d);
    }
    l1:
    printf("Start\n\n");
    d=opendir(".");
    if(d){
        while((dir=readdir(d))!=NULL){
            if(dir->d_type==DT_DIR && strcmp(".",dir->d_name)!=0 && strcmp("..",dir->d_name)!=0 ){
                td = opendir(dir->d_name);
                while((tempdir=readdir(td))!=NULL){
                    if(tempdir->d_type==DT_REG && strcmp(fname,tempdir->d_name)==0) {
                        if(tf!=1){
                            chdir(dir->d_name);
                            system("cat sona.txt");
                            chdir("..");
                        }
                    }
                }
            }
        }
    }
}
```

```

tf=1;
break;
}else{
chdir(dir->d_name);
printf("\nDeteting file from %s",dir->d_name);
system("rm sona.txt");
chdir("..");
tf=1;
break;
}
}
}
closedir(td);
}
}
closedir(d);
}
return 0;
}

```

Output :

```

tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex16 ex16.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex16
Operating System Laboratoy
U19CS405
sona College of technology
Start

Deteting file from sona
tharun@ubuntu:~/Desktop/OSlabQn$ █

```

17. Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the FIFO replacement algorithm?

Assume three frames. Remember that all frames are initially empty, so your first unique pages will all cost one fault each. Next, consider no. of free frame to be four and then find the page fault. Will it suffer from belady's anomaly?

Program :

```

#include<stdio.h>
int n, nf;
int in[100];
int p[50];
int hit = 0;
int i,j,k;

```

```

int pgfaultcnt=0;
void getData(){
printf("\nEnter length of page reference sequence : ");
scanf("%d", &n);
printf("\nEnter the page reference sequence:");
for (i = 0; i < n; i++) {
scanf("%d", &in[i]);
}
printf("\nEnter no of frames: ");
scanf("%d", &nf);
}
void initialize()
{
pgfaultcnt=0;
for(i=0; i<nf; i++)
p[i]=9999;
}
int isHit(int data)
{
hit=0;
for(j=0; j<nf; j++)
{
if(p[j]==data)
{
hit=1;
break;
}
}
return hit;
}
void dispPages()
{
for (k=0; k<nf; k++)
{
if(p[k]!=9999)
printf(" %d",p[k]);
}
}
void dispPgFaultCnt()
{
printf("\nTotal no of page faults:%d",pgfaultcnt);
}
void fifo()
{
initialize();
for(i=0; i<n; i++)
{
printf("\nFor %d :",in[i]);

```

```

    if(isHit(in[i])==0)
    {
        for(k=0; k<nf-1; k++)
            p[k]=p[k+1];
        p[k]=in[i];
        pgfaultcnt++;
        dispPages();
    }
    else
        printf("No page fault");
    }
    dispPgFaultCnt();
}

void main(){
    getData();
    fifo();
}#include<stdio.h>
int n, nf;
int in[100];
int p[50];
int hit = 0;
int i,j,k;
int pgfaultcnt=0;
void getData(){
    printf("\nEnter length of page reference sequence : ");
    scanf("%d", &n);
    printf("\nEnter the page reference sequence:");
    for (i = 0; i < n; i++) {
        scanf("%d", &in[i]);
    }
    printf("\nEnter no of frames: ");
    scanf("%d", &nf);
}

void initialize()
{
    pgfaultcnt=0;
    for(i=0; i<nf; i++)
        p[i]=9999;
}

int isHit(int data)
{
    hit=0;
    for(j=0; j<nf; j++)
    {
        if(p[j]==data)
        {
            hit=1;
            break;

```

```

    }
    }
    return hit;
}
void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }
}
void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}
void fifo()
{
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);
        if(isHit(in[i])==0)
        {
            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];
            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}
void main(){
    getData();
    fifo();
}

```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex17 ex17.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex17

Enter length of page reference sequence : 20

Enter the page reference sequence:1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

Enter no of frames: 3

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 2 3 4
For 2 :No page fault
For 1 : 3 4 1
For 5 : 4 1 5
For 6 : 1 5 6
For 2 : 5 6 2
For 1 : 6 2 1
For 2 :No page fault
For 3 : 2 1 3
For 7 : 1 3 7
For 6 : 3 7 6
For 3 :No page fault
For 2 : 7 6 2
For 1 : 6 2 1
For 2 :No page fault
For 3 : 2 1 3
For 6 : 1 3 6
tharun@ubuntu:~/Desktop/OSlabQn$
```

18. Given the rack partitions of rice storage, each of 100kg, 500kg, 200kg, 300kg and 600kg (in order) in a super market, the supervisor has to place the rice bags of 212kg, 417kg, 112kg and 426 kg (in order). Write an algorithm to help the supervisor for the effective utilization of rack partitions. Write a program for all the memory management schemes.

Program :

```
#include<stdio.h>
#include<stdlib.h>
typedef struct process{
    int psize;
    int pflag;
}process;
typedef struct block{
    int bsize;
    int bflag;
}block;
void accept(process p[],block b[],int *n,int *m){
    int i,j;
    printf("\n Enter no. of blocks : ");
    scanf("%d",m);
```

```

        for(i=0;i<*m;i++){
            printf(" Enter size of block[%d] : ",i);
            scanf("%d",&b[i].bsize);
        }
        printf("\n Enter no. of processes : ");
        scanf("%d",n);
        for(i=0;i<*n;i++){
            printf(" Enter size of block[%d] : ",i);
            scanf("%d",&p[i].psize);
        }
    }
    void re_init(process p[],block b[],int n,int m){
        int i;
        for(i=0;i<n;i++){
            p[i].pflag=0;
        }
        for(i=0;i<m;i++){
            b[i].bflag=0;
        }
    }
    void first_fit(process p[],block b[],int n,int m){
        int i,j,in_frag=0,ex_frag=0;
        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && p[i].pflag ==
0){
                    b[j].bflag = p[i].pflag = 1;
                    in_frag += b[j].bsize - p[i].psize;
                    printf("\n P[%d]\t-\tB[%d]",i,j);
                    break;
                }
            }
            if(p[i].pflag == 0)
                printf("\n P[%d]\t-\tUnassigned",i);
        }
        printf("\n\n Total internal fragmentation : %d",in_frag );
        for(j=0;j<m;j++){
            if(b[j].bflag == 0)
                ex_frag+=b[j].bsize;
        }
        printf("\n Total external fragmentation : %d",ex_frag );
    }
    void best_fit(process p[],block b[],int n,int m){
        int i,j,min_frag,in_frag=0,ex_frag=0,id;
        for(i=0;i<n;i++){
            min_frag = 9999;
            id=9999;
            for(j=0;j<m;j++){
                if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && min_frag >
(b[j].bsize - p[i].psize)){

```



```

                                min_frag = b[j].bsize - p[i].psize;
                                id = j;
                            }
                        }
                    if(min_frag != 9999){
                        b[id].bflag = p[i].pflag = 1;
                        in_frag += b[id].bsize - p[i].psize;
                        printf("\n P[%d]\t-\tB[%d]",i,id);
                    }
                    if(p[i].pflag == 0)
                        printf("\n P[%d]\t-\tUnassigned",i);
                }
                printf("\n\n Total internal fragmentation : %d",in_frag );
                for(j=0;j<m;j++){
                    if(b[j].bflag == 0)
                        ex_frag+=b[j].bsize;
                }
                printf("\n Total external fragmentation : %d",ex_frag );
            }
        void worst_fit(process p[],block b[],int n,int m){
            int i,j,max_frag,in_frag=0,ex_frag=0,id;
            for(i=0;i<n;i++){
                max_frag = -1;
                id=9999;
                for(j=0;j<m;j++){
                    if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && max_frag <
(b[j].bsize - p[i].psize)){
                                max_frag = b[j].bsize - p[i].psize;
                                id = j;
                            }
                        }
                    if(max_frag != -1){
                        b[id].bflag = p[i].pflag = 1;
                        in_frag += b[id].bsize - p[i].psize;
                        printf("\n P[%d]\t-\tB[%d]",i,id);
                    }
                    if(p[i].pflag == 0)
                        printf("\n P[%d]\t-\tUnassigned",i);
                }
                printf("\n\n Total internal fragmentation : %d",in_frag );
                for(j=0;j<m;j++){
                    if(b[j].bflag == 0)
                        ex_frag+=b[j].bsize;
                }
                printf("\n Total external fragmentation : %d",ex_frag );
            }
        }
        int main(){
            int ch,n=0,m=0;

```

```

process p[10];
block b[10];
do{
    printf("\n\n SIMULATION OF FIXED PARTITIONING MEMORY
MANAGEMENT SCHEMES");
    printf("\n\n Options:");
    printf("\n 0. Accept");
    printf("\n 1. First Fit");
    printf("\n 2. Best Fit");
    printf("\n 3. Worst Fit");
    printf("\n 4. Exit");
    printf("\n Select : ");
    scanf("%d",&ch);
    switch(ch){
        case 0:
            accept(p,b,&n,&m);
            break;
        case 1:
            re_init(p,b,n,m);
            first_fit(p,b,n,m);
            break;
        case 2:
            re_init(p,b,n,m);
            best_fit(p,b,n,m);
            break;
        case 3:
            re_init(p,b,n,m);
            worst_fit(p,b,n,m);
            break;
        case 5:exit(0);
        default:
            printf("\n Invalid selection.");
    }
}while(ch != 4);
return 0;
}

```

Output :

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex18 ex18.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex18

SIMULATION OF FIXED PARTITIONING MEMORY MANAGEMENT SCHEMES

Options:
0. Accept
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Select : 0

Enter no. of blocks : 5
Enter size of block[0] : 100
Enter size of block[1] : 500
Enter size of block[2] : 200
Enter size of block[3] : 300
Enter size of block[4] : 600

Enter no. of processes : 4
Enter size of block[0] : 212
Enter size of block[1] : 417
Enter size of block[2] : 112
Enter size of block[3] : 426

SIMULATION OF FIXED PARTITIONING MEMORY MANAGEMENT SCHEMES

Options:
0. Accept
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Select : 1

P[0]   -      B[1]
P[1]   -      B[4]
P[2]   -      B[2]
P[3]   -      Unassigned

Total internal fragmentation : 559
Total external fragmentation : 400
```

```
Total internal fragmentation : 559
Total external fragmentation : 400
```

SIMULATION OF FIXED PARTITIONING MEMORY MANAGEMENT SCHEMES

```
Options:
0. Accept
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Select : 2
```

```
P[0] - B[3]
P[1] - B[1]
P[2] - B[2]
P[3] - B[4]
```

```
Total internal fragmentation : 433
Total external fragmentation : 100
```

SIMULATION OF FIXED PARTITIONING MEMORY MANAGEMENT SCHEMES

```
Options:
0. Accept
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Select : 3
```

```
P[0] - B[4]
P[1] - B[1]
P[2] - B[3]
P[3] - Unassigned
```

```
Total internal fragmentation : 659
Total external fragmentation : 300
```

SIMULATION OF FIXED PARTITIONING MEMORY MANAGEMENT SCHEMES

```
Options:
0. Accept
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Select : 4
```

```
tharun@ubuntu:~/Desktop/OSlabQn$
```

19. The HCL Company is manufacturing the computers; the company is having the fixed stock size to hold the manufactured computers. The client can place the order if stock is not empty and company cannot manufacture the computers if the stock size full. Develop an application to handle the situation.

Program :

```
#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.To Manufacture the Computer\n2.To Order the Computer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                producer();
            else
                printf("Stock are full!!");
                break;
            case 2: if((mutex==1)&&(full!=0))
                consumer();
            else
                printf("Stocks are 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("\nManufactured the Computer %d",x);
    mutex=signal(mutex);
}

void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nOrdered the Computer %d",x);
    x--;
    mutex=signal(mutex);
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex19 ex19.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex19

1.To Manufacture the Computer
2.To Order the Computer
3.Exit
Enter your choice:1

Manufactured the Computer 1
Enter your choice:1

Manufactured the Computer 2
Enter your choice:1

Manufactured the Computer 3
Enter your choice:1
Stock are full!!
Enter your choice:2

Ordered the Computer 3
Enter your choice:2

Ordered the Computer 2
Enter your choice:2

Ordered the Computer 1
Enter your choice:2
Stocks are empty!!
Enter your choice:3
tharun@ubuntu:~/Desktop/OSlabQn$ █

```

20. Five persons are in a queue to book railway tickets. There is only one counter and all of them arrived at 9.30 AM. The counter will be open by 10.00 AM. The processing time for each person to book the tickets are 10 minutes, 20 minutes, 5 minutes, 7 minutes and 13 minutes respectively. Identify the scheduling algorithm and write a C program for the same.

(FIFO)

Program :

```
#include <stdio.h>
typedef struct fcfs
{
    int process;
    int burst;
    int arrival;
    int tat;
    int wt;
}fcfs;
int sort(fcfs [], int);
int main()
{
    int n, i, temp = 0, AvTat = 0, AvWt = 0;

    printf ("Enter the number of processes: ");
    scanf ("%d", &n);
    fcfs arr[n];
    int tct[n];
    for (i = 0; i < n; i++)
    {
        arr[i].process = i;
        printf ("Enter the process %d data\n", arr[i].process);
        printf ("Enter CPU Burst: ");
        scanf ("%d", &(arr[i].burst));
        printf ("Enter the arrival time: ");
        scanf ("%d", &(arr[i].arrival));
    }
    sort(arr, n);
    printf ("Process\t\tBurst Time\tArrival Time\tTurn Around Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        tct[i] = temp + arr[i].burst;
        temp = tct[i];
        arr[i].tat = tct[i] - arr[i].arrival;
        arr[i].wt = arr[i].tat - arr[i].burst;
        AvTat = AvTat + arr[i].tat;
        AvWt = AvWt + arr[i].wt;
    }
}
```

```

        printf ("%5d\t%15d\t\t%9d\t%12d\t%12d\n", arr[i].process, arr[i].burst,
arr[i].arrival, arr[i].tat, arr[i].wt);
    }
    printf ("Average Turn Around Time: %d\nAverage Waiting Time: %d\n",
AvTat / n, AvWt / n);
    return 0;
}
int sort(fcfs arr[], int n)
{
    int i, j;
    fcfs k;

    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (arr[i].arrival > arr[j].arrival)
            {
                k = arr[i];
                arr[i] = arr[j];
                arr[j] = k;
            }
        }
    }
    return 0;
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex20 ex20.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex20
Enter the number of processes: 5
Enter the process 0 data
Enter CPU Burst: 10
Enter the arrival time: 0
Enter the process 1 data
Enter CPU Burst: 20
Enter the arrival time: 0
Enter the process 2 data
Enter CPU Burst: 5
Enter the arrival time: 0
Enter the process 3 data
Enter CPU Burst: 7
Enter the arrival time: 0
Enter the process 4 data
Enter CPU Burst: 13
Enter the arrival time: 0

```

Process	Burst Time	Arrival Time	Turn Around Time	Waiting Time
0	10	0	10	0
1	20	0	30	10
2	5	0	35	30
3	7	0	42	35
4	13	0	55	42

```

Average Turn Around Time: 34
Average Waiting Time: 23
tharun@ubuntu:~/Desktop/OSlabQn$ █

```


21. Six persons are in a queue to book railway tickets. There is only one counter and all of them arrived at 9.30 AM. The counter will be open by 10.00 AM. The processing time for each person to book the tickets are 10 minutes, 20 minutes, 5 minutes, 7 minutes, 2 minutes and 13 minutes respectively. The person with the minimum processing time has to be given preference. Identify the scheduling algorithm and write a C program for the same.

(SJF)

Program :

```
#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;

    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);
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex21 ex21.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex21
Enter number of process:6

Enter Burst Time:
p1:2
p2:5
p3:7
p4:10
p5:13
p6:20

Process      Burst Time      Waiting Time      Turnaround Time
p1            2                0                 2
p2            5                2                 7
p3            7                7                14
p4           10               14                24
p5           13               24                37
p6           20               37                57

Average Waiting Time=14.000000
Average Turnaround Time=23.500000
tharun@ubuntu:~/Desktop/OSlabQn$

```

22. Three patients are in a Hospital to visit the doctor. The first patient is suffering from high temperature, the second patient is in unconscious state and the third patient is suffering from cold and cough. The time of each patient is 10 minutes, 30 minutes and 5 minutes respectively. Identify the scheduling algorithm and write a C program for the same.

(Priority)

Program :

```
#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;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
```

```

        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }
    avg_wt=total/n;
    total=0;
    printf("\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=total/n;
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ ./ex22
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:10
Priority:2

P[2]
Burst Time:30
Priority:1

P[3]
Burst Time:5
Priority:3

Process      Burst Time      Waiting Time      Turnaround Time
P[2]          30              0                30
P[1]          10              30               40
P[3]           5              40               45

Average Waiting Time=23
Average Turnaround Time=38
tharun@ubuntu:~/Desktop/OSlabQn$ █

```

23. Group discussion is conducted for 5 students in a campus recruitment drive. The time each student had prepared is ten minutes. But the GD conductor will be giving only 3 minutes for each student. Identify the scheduling algorithm and write a C program for the same.

(Round-Robin,RR)

Program :

```
#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;
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ gedit ex23.c
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex23 ex23.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex23
Enter Total Process:      5
Enter Arrival Time and Burst Time for Process Process Number 1 :0 10
Enter Arrival Time and Burst Time for Process Process Number 2 :0 10
Enter Arrival Time and Burst Time for Process Process Number 3 :0 10
Enter Arrival Time and Burst Time for Process Process Number 4 :0 10
Enter Arrival Time and Burst Time for Process Process Number 5 :0 10
Enter Time Quantum:      3

Process |Turnaround Time|Waiting Time
P[1]    |      46      |      36
P[2]    |      47      |      37
P[3]    |      48      |      38
P[4]    |      49      |      39
P[5]    |      50      |      40

Average Waiting Time= 38.000000
Avg Turnaround Time = 48.000000tharun@ubuntu:~/Desktop/OSlabQn$

```

24. A is the producer of random numbers and share memory buffer with B. B consumes the numbers generated by A. Help B to view numbers sent by A and print the odd numbers separately.

Program :

```

#include<stdio.h>
void display(int a[], int size);
int main(){
    int size, i, a[10], odd[20];
    int Ecount = 0, Ocount = 0;
    printf("enter size of array :\n");
    scanf("%d", &size);
    printf("enter array elements:\n");
    for(i = 0; i < size; i++){
        scanf("%d", &a[i]);
    }
    for(i = 0; i < size; i++){

```

```

        if(a[i] % 2 == 0){
            even[Ecount] = a[i];
            Ecount++;
        }
        else{
            odd[Ocount] = a[i];
            Ocount++;
        }
    }
    printf("no: of elements comes under odd are = %d \n", Ocount);
    printf("The elements that are present in an odd array is : ");
    display(odd, Ocount);
    return 0;
}

void display(int a[], int size){
    int i;
    for(i = 0; i < size; i++){
        printf("%d \t ", a[i]);
    }
    printf("\n");
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ ./ex24
enter size of array :
5
enter array elements:
2
7
20
13
24
no: of elements comes under odd are = 2
The elements that are present in an odd array is : 7      13
tharun@ubuntu:~/Desktop/OSlabQn$

```

25. Write a C program that will simulate FCFS scheduling algorithm. For this algorithm, the program should compute waiting time, turnaround time of every job as well as the average waiting time and the average turnaround time. The average values should be consolidated in a table for easy comparison. You may use the following data to test your program. Using your program, consider that each context switching require 0.4 ms then find out the total context switching time.

Processes	Arrival time	CPU cycle(in ms)
1	0	6
2	3	2

3	5	1
4	9	7
5	10	5
6	12	3
7	14	4
8	16	5
9	17	7
10	19	2

Program :

```
#include<stdio.h>
void main()
{
int n,a[10],b[10],t[10],w[10],g[10],r[10],i,m;
float atat=0,awt=0,art=0;
    for(i=0;i<10;i++)
    {
        a[i]=0; b[i]=0; w[i]=0; g[i]=0;
    }
printf("\nEnter the number of process:");
    scanf("%d",&n);
printf("\nEnter the Burst times:");
    for(i=0;i<n;i++)
        scanf("%d",&b[i]);
printf("\nEnter the Arrival times:");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
g[0]=0;
    for(i=0;i<10;i++)
        g[i+1]=g[i]+b[i];
    for(i=0;i<n;i++)
    {
        w[i]=g[i]-a[i];
        t[i]=g[i+1]-a[i];
        r[i]=g[i]-a[i];
        awt=awt+w[i];
        atat=atat+t[i];
        art=art+w[i];
    }
    awt =awt/n;
    atat=atat/n;
    art=art/n;
    printf("Process\t\tArrival Time\tBurst time \tWaiting time\tTurn
around time\t Response time\n");
    for(i=0;i<n;i++)
```



```

    {

printf("\t p%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,a[i],b[i],w[i],t[i],r[i]);
    }
printf("The Average Waiting Time is %f\n",awt);
printf("The Average turn around time is %f\n",atat);
printf("The Average response time is %f\n",art);
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ ./ex25

Enter the number of process:10

Enter the Burst times:6 2 1 7 5 3 4 5 7 2

Enter the Arrival times:0 3 5 9 10 12 14 16 17 19
Process      Arrival Time    Burst time      Waiting time     Turn around time  Response time
p0           0                6               0                6                0
p1           3                2               3                5                3
p2           5                1               3                4                3
p3           9                7               0                7                0
p4          10                5               6               11                6
p5          12                3               9               12                9
p6          14                4              10               14               10
p7          16                5              12               17               12
p8          17                7              16               23               16
p9          19                2              21               23               21

The Average Waiting Time is 8.000000
The Average turn around time is 12.200000
The Average response time is 8.000000
tharun@ubuntu:~/Desktop/OSlabQn$

```

26. Write a C program that will simulate SJF scheduling algorithm. For this algorithm, the program should compute waiting time, turnaround time of every job as well as the average waiting time and the average turnaround time. The average values should be consolidated in a table for easy comparison. You may use the following data to test your program. Using your program, consider that each context switching require 0.4 ms then find out the total context switching time.

Processes	Arrival time	CPU cycle(in ms)
1	0	6
2	3	2
3	5	1
4	9	7
5	10	5
6	12	3
7	14	4
8	16	5
9	17	7
10	19	2

Program :

```
#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;
    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=(float)total/n;    //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\n\nAverage Turnaround Time=%f\n",avg_tat);
}

```

Output:

```

tharun@ubuntu:~/Desktop/OSlabQn$ gedit ex26.c
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex26 ex26.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex26
Enter number of process:10

Enter Burst Time:
p1:6
p2:2
p3:1
p4:7
p5:5
p6:3
p7:4
p8:5
p9:7
p10:2

Process      Burst Time      Waiting Time      Turnaround Time
p3           1              0                1
p2           2              1                3
p10          2              3                5
p6           3              5                8
p7           4              8               12
p5           5             12               17
p8           5             17               22
p1           6             22               28
p9           7             28               35
p4           7             35               42

Average Waiting Time=13.100000
Average Turnaround Time=17.299999
tharun@ubuntu:~/Desktop/OSlabQn$

```

27. Write a C program that will simulate round robin scheduling algorithm. For this algorithm, the program should compute waiting time, turnaround time of every job as well as the average waiting time and the average turnaround time. The average values should be consolidated in a table for easy comparison. You may use the following data to test your program. The time quantum for round robin is 4 ms and the context switching time is zero. Using your program, consider that each context switching require 0.4 ms then find out the total context switching time.

Processes	Arrival time	CPU cycle(in ms)
1	0	6
2	3	2
3	5	1
4	9	7
5	10	5
6	12	3
7	14	4
8	16	5
9	17	7
10	19	2

Program :

```
#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\n",turnaround_time*1.0/n);

return 0;
}
```

Output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex27 ex27.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex27
Enter Total Process:      10
Enter Arrival Time and Burst Time for Process Process Number 1 :0 6
Enter Arrival Time and Burst Time for Process Process Number 2 :3 2
Enter Arrival Time and Burst Time for Process Process Number 3 :5 1
Enter Arrival Time and Burst Time for Process Process Number 4 :9 7
Enter Arrival Time and Burst Time for Process Process Number 5 :10 5
Enter Arrival Time and Burst Time for Process Process Number 6 :12 3
Enter Arrival Time and Burst Time for Process Process Number 7 :14 4
Enter Arrival Time and Burst Time for Process Process Number 8 :16 5
Enter Arrival Time and Burst Time for Process Process Number 9 :17 7
Enter Arrival Time and Burst Time for Process Process Number 10 :19 2
Enter Time Quantum:      4

Process |Turnaround Time|Waiting Time
P[2]    |      3      |      1
P[3]    |      2      |      1
P[1]    |      9      |      3
P[6]    |      8      |      5
P[7]    |     10      |      6
P[10]   |     15      |     13
P[4]    |     28      |     21
P[5]    |     28      |     23
P[8]    |     23      |     18
P[9]    |     25      |     18

Average Waiting Time= 10.900000
Avg Turnaround Time = 15.100000
tharun@ubuntu:~/Desktop/OSlabQn$
```

28. Consider that there are three processes in ready queue and 9 free memory frame and processes has the following page reference string:

P1:1, 2, 3, 4, 2, 1

P2: 5, 6, 2, 1, 2, 3

P3: 7, 6, 3, 2, 1, 2

And use equal partitioning method and allocate initial set of frame to all the three process. How many page faults would occur for the FIFO replacement algorithm using local replacement policy? Remember that all frames are initially empty, so your first unique pages will all cost one fault each.

Program :

```
#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;
}

```

Output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex28 ex28.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex28
```

Enter the number of Pages: 6

Enter reference string values:

Value No. [1]: 1
Value No. [2]: 2
Value No. [3]: 3
Value No. [4]: 4
Value No. [5]: 2
Value No. [6]: 1

What are the total number of frames: 9

1	-1	-1	-1	-1	-1	-1	-1	-1
1	2	-1	-1	-1	-1	-1	-1	-1
1	2	3	-1	-1	-1	-1	-1	-1
1	2	3	4	-1	-1	-1	-1	-1
1	2	3	4	-1	-1	-1	-1	-1
1	2	3	4	-1	-1	-1	-1	-1

Total Page Faults: 4

```
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex28
```

Enter the number of Pages: 6

Enter reference string values:

Value No. [1]: 5
Value No. [2]: 6
Value No. [3]: 2
Value No. [4]: 1
Value No. [5]: 2
Value No. [6]: 3

What are the total number of frames: 9

5	-1	-1	-1	-1	-1	-1	-1	-1
5	6	-1	-1	-1	-1	-1	-1	-1
5	6	2	-1	-1	-1	-1	-1	-1
5	6	2	1	-1	-1	-1	-1	-1
5	6	2	1	-1	-1	-1	-1	-1
5	6	2	1	-1	3	-1	-1	-1

Total Page Faults: 5

```
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex28
```

Enter the number of Pages: 6

Enter reference string values:

Value No. [1]: 7
Value No. [2]: 6
Value No. [3]: 3
Value No. [4]: 2
Value No. [5]: 1
Value No. [6]: 2

What are the total number of frames: 9

7	-1	-1	-1	-1	-1	-1	-1	-1
7	6	-1	-1	-1	-1	-1	-1	-1
7	6	3	-1	-1	-1	-1	-1	-1
7	6	3	2	-1	-1	-1	-1	-1
7	6	3	2	1	-1	-1	-1	-1
7	6	3	2	1	-1	-1	-1	-1

Total Page Faults: 5

29. Consider that there are three processes in ready queue and 9 free memory frame and processes has the following page reference string:

P1: 11, 12, 13, 14, 12, 11

P2: 15, 16, 12, 11, 12, 13

P3: 17, 16, 13, 12, 11, 12

And use equal partitioning method and allocate initial set of frame to all the three process. How many page faults would occur for the LRU replacement algorithm using local replacement policy? Remember that all frames are initially empty, so your first unique pages will all cost one fault each.

Program :

```
#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\n", faults);
return 0;
}

```

Output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex29 ex29.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex29
Enter number of frames: 9
Enter number of pages: 6
Enter reference string: 11 12 13 14 12 11

11      -1      -1      -1      -1      -1      -1      -1      -1
11      12      -1      -1      -1      -1      -1      -1      -1
11      12      13      -1      -1      -1      -1      -1      -1
11      12      13      14      -1      -1      -1      -1      -1
11      12      13      14      -1      -1      -1      -1      -1
11      12      13      14      -1      -1      -1      -1      -1

Total Page Faults = 4
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex29
Enter number of frames: 9
Enter number of pages: 6
Enter reference string: 15 16 12 11 12 13

15      -1      -1      -1      -1      -1      -1      -1      -1
15      16      -1      -1      -1      -1      -1      -1      -1
15      16      12      -1      -1      -1      -1      -1      -1
15      16      12      11      -1      -1      -1      -1      -1
15      16      12      11      -1      -1      -1      -1      -1
15      16      12      11      13      -1      -1      -1      -1

Total Page Faults = 5
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex29
Enter number of frames: 9
Enter number of pages: 6
Enter reference string: 17 16 13 12 11 12

17      -1      -1      -1      -1      -1      -1      -1      -1
17      16      -1      -1      -1      -1      -1      -1      -1
17      16      13      -1      -1      -1      -1      -1      -1
17      16      13      12      -1      -1      -1      -1      -1
17      16      13      12      11      -1      -1      -1      -1
17      16      13      12      11      -1      -1      -1      -1

Total Page Faults = 5
tharun@ubuntu:~/Desktop/OSlabQn$ █
```

30. The Chip manufacturing Company is manufacturing the chips; the company is having the fixed stock size to hold the manufactured chips. The client can place the order if stock is not empty and company cannot manufacture the computers if the stock size full. Develop an application to handle the situation.

Program :

```
#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.To Manufacture the Chips\n2.To Order the Chips\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                    producer();
                    else
                    printf("Stock are full!!");
                    break;
            case 2: if((mutex==1)&&(full!=0))
                    consumer();
                    else
                    printf("Stocks are 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("\nManufactured the Chips %d",x);
mutex=signal(mutex);
}
```

```
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\nOrdered the Chips %d",x);
x--;
mutex=signal(mutex);
}
```

Output:

```
tharun@ubuntu:~/Desktop/OSlabQn$ gedit ex30.c
tharun@ubuntu:~/Desktop/OSlabQn$ gcc -oex30 ex30.c
tharun@ubuntu:~/Desktop/OSlabQn$ ./ex30

1.To Manufacture the Chips
2.To Order the Chips
3.Exit
Enter your choice:1

Manufactured the Chips 1
Enter your choice:1

Manufactured the Chips 2
Enter your choice:1

Manufactured the Chips 3
Enter your choice:1
Stock are full!!
Enter your choice:2

Ordered the Chips 3
Enter your choice:2

Ordered the Chips 2
Enter your choice:2

Ordered the Chips 1
Enter your choice:2
Stocks are empty!!
Enter your choice:3
tharun@ubuntu:~/Desktop/OSlabQn$
```