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

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

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h> int
main(){
char str[50];
char *path = "/bin/sh";
//char *arg[] = {path,"-c","grep -o -i 's' p1.c | wc -l",NULL};
printf("Enter your pattern: ");
scanf("%s",str); char
*command1 = "grep -o -i "; char
*command2 = " p1.c | "; char
*command3 = "wc -w";
char *query ;
query = (char *) malloc (sizeof(char) *
(strlen(command1)+strlen(command2)+strlen(str)+strlen(command3)));
strncat(query,command1,sizeof(command1));
strncat(query,str,sizeof(str));
strncat(query,command2,sizeof(command2));
strncat(query,command3,sizeof(command3));
printf("%s",query); char *arg[] = {path,"-
c",query,NULL};
execv(path,arg); //system(query);
return 0;
}
```

3. 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;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(){
char dirname_in[100];
char *dname;
char *p="/";
printf("Enter your directory name :");
scanf("%s",dirname_in); dname = (char *) malloc
(sizeof(char)*strlen(dirname_in)+1);
strncat(dname,p,strlen(p));
strncat(dname,dirname_in,strlen(dirname_in)); char
*command = "/bin/ls"; char *arr[]={command,"-
l",NULL}; execv(command,arr);
return 0;
}
```

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;
}
```

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;
}
```

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
```

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

```
cp $s1 $s2
cat $s2
```

9. Create a file named sample.txt with 20 lines and do the following using shell script
   i) Count the number of words in the file.
   ii) Display the first and last five lines of the contents
   iii) Find the occurrence of any word iv) Display the contents of the file.
   TEXT FILE NAME SHOULD BE : sample.txt

```
echo "1. Number of words in a file :";
wc -w sample.txt
```

```
echo "2. Display First and last 5 line: ";
echo "First 5 lines: "; head -5
sample.txt echo "Last 5 lines: ";
tail -5 sample.txt
echo "Occurence of the word 's' :";
grep -o -i s sample.txt | wc -w
echo "Display the content of the file: "; cat
sample.txt
```

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

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

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. also, child process should execute user specified
program

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

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

```
#include <stdio.h>
#include <stdlib.h>
int main(){
printf("Using ls command to print all text files in a directory: \n");
printf("ls *.txt\n");
system("ls *.txt");
printf("\n\nUsing ls command to display all content in long format: \n");
printf("ls -l\n");
system("ls -l");
printf("\n\nUsing ls command to show hidden files: \n");
printf("ls -a\n");
system("ls -a");
printf("\n\nUsing ls command to show files sort by date and time: \n");
printf("ls -t\n");
system("ls -t");
printf("\n\nUsing ls command to show all files sort by size: \n");
printf("ls -s\n");
system("ls -s");
printf("\n\nUsing grep command to print number of occurence of a pattern in a file: \n");
printf("grep -i -n s p2.c\n");
system("grep -i -n s p2.c");
printf("\n\nUsing egrep command to print lines that contain exactly two consecutive 'm': \n");
printf("egrep m{2} p2.c\n");
system("egrep m{2} p2.c");
printf("\n\nUsing fgrep command to print occurence of a pattern in a file: \n");
printf("fgrep -i include p2.c\n");
system("fgrep -i include p2.c");
printf("\n\nUsing rgrep command to print all files, recursively for a string 'command': \n");
printf("rgrep -i command *\n");
system("rgrep -i command *");
return 0;
}
```

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
```

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
```

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

```
echo "Enter File Name :" read f
echo "Enter a word :" read
w
echo "\n Number of occurence of the word is: ";
grep -o -i "$w" "$f" | wc –w
```

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
```

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;
}
```

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?

```
#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];
  1. p[k]=in[i];
     pgfaultcnt++;
     dispPages();
     }
     else
     printf("No page fault");
     }
     dispPgFaultCnt();
     }
     void main(){
     getData(); fifo();
     }
```

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

```
#include <stdio.h>
#include <stdlib.h> void
sort(int a[],int n){
for(int i=0;i<n-1;i++){
int s=a[i]; int pos=i;
for(int j=i+1;j<n;j++){
if(a[j]<s){
pos=j;
}
}
int tmp=a[pos];
a[pos]=a[i]; a[i]=tmp;
}
```

```
          }
          void First_Fit(int inp[],int Weight[] ,int n,int in){
          int dWeight[n]; for(int i=0;i<n;i++){
          dWeight[i]=Weight[i];
          }
          for(int j=0;j<in;j++){ int tf=0;
          for(int i=0;i<n;i++){
if(inp[j]<=dWeight[i]){ printf("%d
%d\n",inp[j],dWeight[i]);
dWeight[i]=0;
tf=1;
break;
}
}
if(tf==0){
printf("%d NA\n",inp[j]);
}
}
}
void Best_Fit(int inp[],int Weight[] ,int n,int in){
int dWeight[n]; for(int i=0;i<n;i++){
dWeight[i]=Weight[i];
}
for(int j=0;j<in;j++){
int tf=0; for(int i=0;i<n;i++){
if(inp[j]<=dWeight[i]){ printf("%d
%d\n",inp[j],dWeight[i]);
dWeight[i]=0;
tf=1;
break;
}
}
if(tf==0){
printf("%d NA\n",inp[j]);
}
}
}
void Worst_Fit(int inp[],int Weight[] ,int n,int in){
int dWeight[n]; for(int i=0;i<n;i++){
dWeight[i]=Weight[i];
}
for(int j=0;j<in;j++){ int tf=0;
for(int i=n-1;i>=0;i--){
if(inp[j]<=dWeight[i]){ printf("%d
%d\n",inp[j],dWeight[i]);
dWeight[i]=0;
tf=1;
break;
}
}
if(tf==0){
printf("%d NA\n",inp[j]);
}
}
}
void main(){ int n=5,inp=4;
int arr[10],unsrt[10],inVal[10];
for(int i=0;i<n;i++){ scanf("
%d",&arr[i]);
unsrt[i]=arr[i];
}
for(int i=0;i<inp;i++){
scanf(" %d",&inVal[i]);
}
int *temp=arr;
sort(arr,n);
First_Fit(inVal,unsrt,n,inp);
Best_Fit(inVal,arr,n,inp);
    1. Worst_Fit(inVal,arr,n,inp);
       }
```

19. Producer consumer hcl
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.

    PROGRAM:

    FCFS – First Come First Serve

```
//FCFS
#include<stdio.h>
int main(){
int arr[5],i,j,arr1[6],sum=0;
arr1[0]=sum;
printf("\n\tFCFS - FIRST COME FIRST SERVE\n");
for(i=0;i<5;i++){
printf("Enter burst time of P[%d] : ",i);
scanf("%d",&arr[i]);
sum=sum+arr[i];
arr1[i+1]=sum;
}
float avg;
for(j=0;j<2;j++)
{
if(j==0)
printf("\nWaiting Time \n");
else
printf("\nTurn around Time\n");
avg=0;
for(i=j;i<j+5;i++)
{
printf("P[%d]%d\n",i,arr1[i]);
avg=avg+arr1[i];
}
if(j==0)
printf("Average waiting time");
else
printf("Average turn around time");
avg=avg/5.0;
printf("%.1f\n",avg);
}
}
```

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

SRTF – Shortest Remaining Time First

```
#include<stdio.h>
int main(){
int i,j,arr[6],sum1=30;
struct pr{
int value;
int pri;
int sum;
}s[7];
printf("\n\tSRTF - Shortest Remaining Path First\n");
printf("Enter the Burst Time : \n");
for(i=0;i<6;i++){
```

```
printf("P[%d] = ",i);
scanf("%d",&s[i].value);
arr[i]=s[i].value;
}
for(i=0;i<5;i++){
for(j=0;j<5;j++){
if(arr[j]>arr[j+1]){
int temp;
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
}
for(i=0;i<6;i++){
for(j=0;j<6;j++){
if(arr[i]==s[j].value){
s[j].pri=i;
}
}
}
int k=0,z=0;
ss:
if(k==1){
sum1=32;
z=1;}
for(i=0;i<6;i++){
for(j=0;j<6;j++){
if(s[j].pri==i){
s[j].sum=sum1;
sum1=s[j].sum+arr[z];
z+=1;
}
}
}
s[j].sum=sum1;
if(k==0)
printf("\nIndividual Process' waiting time : \n");
else
printf("\nIndividual Process' turn around time : \n");
float avg=0.0;
for(j=0;j<6;j++){
printf("%d\n",s[j].sum);
avg=s[j].sum+avg;
```

```
}
avg=avg/6.0;
if(k==0)
printf("\nAverage Waiting Time : " );
else
printf("\nAverage Turn Around Time : ");
printf("%.1f\n",avg);
if(k==0){
k+=1;
goto ss;
}
}
```

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
#include<stdio.h>
struct sc{
int wait;
int pr;
int sum;
}sc[4];
int pri(int pr){
int i;
for(i=0;i<4;i++){
if(pr==sc[i].pr){
return sc[i].wait;
}
}
}
int main(){
sc[0].pr=0;
sc[0].wait=0;
int i,j,sum1,k=0,z=0;
printf("\n\tPRIORITY\n");
printf("\nEnter Burst Time : \n");
for(i = 1;i<4;i++){
printf("Burst time of P[%d] : ",i);
scanf("%d",&sc[i].wait);
}
printf("\nEnter Priority Assigned for each patient : \n");
for(i=1;i<4;i++){
printf("Priority of P[%d]",i);
scanf("%d",&sc[i].pr);
}
ss:
sum1=0;
for(i=1;i<4;i++){
for(j=1;j<4;j++){
if(i==sc[j].pr){
sc[j].sum=sum1+pri(z);
sum1=sc[j].sum;
z+=1;
}
}
}
if(k==0)
printf("\nIndividual process' waiting time\n");
else
printf("\nIndividual process' turn around time\n");
float avg=0.0;
for(i=1;i<4;i++){
printf("P[%d]\n",sc[i].sum);
avg=avg+sc[i].sum;
}
avg=avg/3.0;
if(k==0)
printf("\nAverage waiting time : ");
else
printf("\nAverage turn around time : ");
printf("%.2f\n",avg);
if(k==0){
k=1;z=1;
goto ss;
}
}
```

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

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.
Producer consumer

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
#define maxitem 5 #define
buffersize 5 sem_t
full,empty; int
buffer[buffersize]; int
    in=0,out=0; pthread_mutex_t
    mutex; void *producer(void
    *pno){ int item,i;
    for(i=0;i<maxitem;i++){
    sem_wait(&empty);
    pthread_mutex_lock(&mutex);
    item=random(); buffer[in]=item;
    printf("producer %d produces a %d at %d\n",*((int *)pno),buffer[in],i);
    in=(in+1)%buffersize; pthread_mutex_unlock(&mutex);
    sem_post(&full);
    }
    }
    void *consumer(void *cno){
    int item,i; for(i=0;i<maxitem;i++){ sem_wait(&full);
    pthread_mutex_lock(&mutex); printf("consumer %d consumes a %d at
    %d\n",*((int *)cno),buffer[out],i); out=(out+1)%buffersize;
    pthread_mutex_unlock(&mutex); sem_post(&empty);
    }
    }
    void oddprint(){
    int i=0;
    for(i=0;i<maxitem;i++){
    if(buffer[i]%2!=0){
    printf("%d\n",buffer[i]);
    }
    }
    }
    int main(){ pthread_t pro,con;
    pthread_mutex_init(&mutex,NULL);
    sem_init(&full,0,0);
    sem_init(&empty,0,1);
    int id=1;
    pthread_create(&pro,NULL,(void *)producer,(void *)&id);
    pthread_create(&con,NULL,(void *)consumer,(void *)&id);
    pthread_join(pro,NULL); pthread_join(con,NULL);
    oddprint(); sem_destroy(&empty); sem_destroy(&full);
    pthread_mutex_destroy(&mutex); return 0;
    }
```

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

```
#include<stdio.h>
int main(){
int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};
int n,sum=0;
float totalTAT=0,totalWT=0;
printf("Enter number of processes "); scanf("%d",&n);
printf("Enter arrival time and burst time for each process\n\n");
for(int i=0;i<n;i++)
{
```

```
printf("Arrival time of process[%d] ",i+1);
scanf("%d",&at[i]);
printf("Burst time of process[%d] ",i+1);
scanf("%d",&bt[i]);
printf("\n");
}
//calculate completion time of processes
for(int j=0;j<n;j++)
{
sum+=bt[j];
ct[j]+=sum;
}
//calculate turnaround time and waiting times
for(int k=0;k<n;k++)
{
tat[k]=ct[k]-at[k];
totalTAT+=tat[k];
}
for(int k=0;k<n;k++)
{
wt[k]=tat[k]-bt[k];
totalWT+=wt[k];
}
printf("Solution: \n\n");
printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");
for(int i=0;i<n;i++)
{
printf("P%d\t %d\t %d\t %d\t %d\t %d\n",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);
}
printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n); printf("Average
WT = %f\n\n",totalWT/n);
return 0;
}
```

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
```

```
#include <stdio.h>
int main()
{
int arrival_time[10], burst_time[10], temp[10]; int i,
smallest, count = 0, time, limit; double wait_time = 0,
turnaround_time = 0, end; float average_waiting_time,
average_turnaround_time; printf("nEnter the Total
Number of Processes:t"); scanf("%d", &limit);
printf("nEnter Details of %d Processesn", limit); for(i = 0;
i < limit; i++)
{
printf("nEnter Arrival Time:t");
scanf("%d", &arrival_time[i]);
printf("Enter Burst Time:t"); scanf("%d",
&burst_time[i]);
temp[i] = burst_time[i];
}
burst_time[9] = 9999;
for(time = 0; count != limit; time++)
{
smallest = 9;
for(i = 0; i < limit; i++)
{
if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
{
smallest = i;
}
}
burst_time[smallest]--;
if(burst_time[smallest] == 0)
{
count++;
end = time + 1;
wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
turnaround_time = turnaround_time + end - arrival_time[smallest];
}
}
average_waiting_time = wait_time / limit; average_turnaround_time
= turnaround_time / limit; printf("nnAverage Waiting Time:t%lfn",
average_waiting_time); printf("Average Turnaround Time:t%lfn",
average_turnaround_time); return 0;
}
```

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

```
#include<stdio.h>
#include<conio.h>
void main() { 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 number of process
in the system: "); scanf("%d", &NOP); y = NOP; for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t"); scanf("%d", &at[i]); printf(" \nBurst
time is: \t"); scanf("%d", &bt[i]); temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t"); scanf("%d",
&quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
```

```
if(temp[i] <= quant && temp[i] > 0)
{
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--;
printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\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;
}
}
avg_wt = wt * 1.0/NOP; avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
}
```

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.

```
#include <stdio.h>
#include <stdlib.h>
#define Max_len 18
#define F_size 9
int check(int arr[],int n,int i){
for(int j=0;j<n;j++){
if(arr[j]==i){
return 1;
}
}
return 0;
}
int main(){
int ready_q[Max_len],frame[F_size];
int pnum=1,index=0,num=0;
int page_fault=0;
for(int i=0;i<3;i++){
printf("Enter p%d page references:",i+1);
for(int j=0;j<Max_len/3;j++){
index=i+j*3;
//printf(" %d",index);
scanf(" %d",&num);
ready_q[index]=num;
}
}
for(int i=0;i<F_size;i++){
frame[i]=-1;
}
int ni=0,fi=0;
printf("processor\t\tFrame\t\t\tPage Fault\n");
for(int i=0;i<Max_len;i++){
printf("p%d:%d\t\t",i%3+1,ready_q[i]);
if(check(frame,F_size,ready_q[i])==0){
if(fi<F_size){
frame[fi]=ready_q[i];
fi++;
}else{
frame[ni]=ready_q[i];
ni++;
ni=ni%F_size;
}
page_fault++;
}
for(int j=0;j<F_size;j++){
if(frame[j]!=-1){
printf("%d ",frame[j]);
}else{
printf("- ");
}
}
printf("\t\t%d\n",page_fault);
}
printf("\n\tTotal Page Fault : %d",page_fault);
}
```

OUTPUT:

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.

```
#include <stdio.h>
#include <stdlib.h>
#define Max_len 18
#define F_size 9
int check(int arr[],int n,int i){
for(int j=0;j<n;j++){
if(arr[j]==i){
return 1;
}
}
return 0;
}
int main(){
int ready_q[Max_len],frame[F_size];
int pnum=1,index=0,num=0;
int page_fault=0;
for(int i=0;i<3;i++){
printf("Enter p%d page references:",i+1);
for(int j=0;j<Max_len/3;j++){
index=i+j*3;
//printf(" %d",index);
scanf(" %d",&num);
ready_q[index]=num;
}
}
for(int i=0;i<F_size;i++){
frame[i]=-1;
}
int fi=0,ni=0;
printf("processor\t\tFrame\t\t\tPage Fault\n");
for(int i=0;i<Max_len;i++){
printf("p%d:%d\t\t",i%3+1,ready_q[i]);
if(check(frame,F_size,ready_q[i])==0){
if(fi<F_size){
frame[fi]=ready_q[i];
fi++;
}else{
ni=i%F_size;
frame[ni]=ready_q[i];
fi++;
}
page_fault++;
}
for(int j=0;j<F_size;j++){
if(frame[j]!=-1){
printf("%d ",frame[j]);
}else{
printf("-- ");
}
}
printf("\t\t%d\n",page_fault);
}
printf("\n\tTotal Page Fault : %d",page_fault);
}
```

OUTPUT:

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.

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

```
/*
This program provides a solution for producer-consumer problem using mutex and semaphore.
*/
#define MaxItems 5 // Maximum items a producer can produce or a consumer can //consume
#define BufferSize 5 // Size of the buffer
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int max=MaxItems;
int bfs = BufferSize;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
int item;
for(int i = 0; i < max; i++) {
item = rand(); // Produce an random item
sem_wait(&empty); //Before storing the produced item in the buffer, producer waits on empty
value
pthread_mutex_lock(&mutex); // Producer waits on mutex to get exclusive access to the buffer
buffer[in] = item; //Producer stores the produced item into the buffer
printf("%s Produce chip ID: %d at stock position : %d\n", (char *)pno,buffer[in],in);
in = (in+1)%bfs; //Producer increments the buffer position to the next empty slot
pthread_mutex_unlock(&mutex); //Producer releases the buffer lock, signal operation is
performed
   1. sem_post(&full); //Producer performs the signal operation
      }
      }
      void *consumer(void *cno)
      {
      for(int i = 0; i < max; i++) {
      sem_wait(&full); //Consumer waits for the buffer to have at least one item
      pthread_mutex_lock(&mutex); //Consumer waits for the exclusive access to the buffer
      int item = buffer[out]; // Consumer consumes the item in the buffer
      printf("Customer Purchaced chip id : %d from stock position : %d\n",item, out);
      out = (out+1)%bfs;//Consumer increments the buffer size to point to the next full slot
      pthread_mutex_unlock(&mutex); //Consumer releases the buffer lock
      sem_post(&empty); // Consumer performs the signal operation
      }
      }
      int main()
      {
      printf("Enter fixed stock size : ");
      scanf(" %d",&bfs);
      printf("Enter number of stock : ");
      scanf(" %d",&max);
      pthread_t pro,con; // We have one producer and one consumer
      pthread_mutex_init(&mutex, NULL); // Initialize the mutex thread value
      sem_init(&empty,0,bfs); //Initialize the empty semaphore value to full size of the buffer
      sem_init(&full,0,0); //Initialize the full semaphore value to zero
      char * company_name = "Company";
      //The following statement creates the producer
      pthread_create(&pro, NULL, (void *)producer, (void *)company_name);
      //The following statement creates the consumer
      pthread_create(&con, NULL, (void *)consumer, (void *)company_name);
      // The following statement runs the producer
      pthread_join(pro, NULL);
      // The following statement runs the producer
      pthread_join(con, NULL);
      //Once the producer and consumer stop executing destroy all the threads and semaphores
      used in this
      program
      pthread_mutex_destroy(&mutex);
      sem_destroy(&empty);
      sem_destroy(&full);
      return 0;
      }
```

producer consumer

```
#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);
}
```