EXERCISE-2: SIMULATION OF SYSTEM COMMANDS USING SYSTEM CALLS

Objective:

To develop a C program to implement the cp, ls, grep commands (with some options) using system calls.

1) Implementing cp command

Algorithm:

1) Accept source and destination files as command line arguments
2) Open the source file in readonly mode and destination file in read write/create mode
3) If both file descriptors exist, read 100 characters from source and store to a buffer
4) Write them to the destination file
5) Else, file descriptor is invalid
6) Close the file descriptors

```c
#include <stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>

int main(int argc,char *argv[])
{
  char *buf[100];
  int src = open(argv[1],O_RDONLY,0777);
  int dest = open(argv[2],O_RDWR | O_CREAT,0777);
  if(dest>0 && src>0)
  {
    int x = read(src,&buf,100);
    write(dest,&buf,x);
    printf("COPIED\n");
  }
  else                              //if return type is neg, it is an error
  {
    printf("Invalid\n");
  }
  close(src);
  close(dest);
  return 0;
}
```

Output:

```
//creating file f1.txt

~/systemcalls$ cat>f1.txt

hi

this the file 1


//compiling and running cpy.c using command line arguments (src,destination)

~/systemcalls$ gcc cpy.c -o cpy

~/systemcalls$ ./cpy f1.txt f2.txt

COPIED


//checking contents of f2.txt

~/systemcalls$ cat f2.txt

hi

this the file 1


//when src doesn't exist

~/systemcalls$ ./cpy f1r.txt f2.txt

Invalid
```

2) Implementing cp-i command

Algorithm:

1) Accept source and destination files as command line arguments
2) Open the source file in readonly mode and destination file in read write/create mode
3) If destination file exists, prompt for allowing overwrite
4) If yes and source file descriptor exists, read 100 characters from source and store to a buffer
5) Write them to the destination file
6) Else, file descriptor is invalid
7) Close the file descriptors

```c
#include <stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>

int main(int argc,char *argv[])
{
  char *buf[100];
  int src = open(argv[1],O_RDONLY,0777);
  int dest = open(argv[2],O_RDWR | O_CREAT,0777);
  char c;
  if(dest>0)            //if file descriptor already exists
  {
    printf("Do you want to overwrite to existing file?");
    scanf("%c",&c);
  }
  if((c=='y'||c=='Y') && src>0)
  {
    int x = read(src,&buf,100);
    write(dest,&buf,x);
    printf("COPIED\n");
  }
  else if(c!='y'||c!='Y')           //overwrite disabled
  {
    printf("Cant overwrite to existing file\n");
  }
  else
  {
    printf("Invalid\n");
  }
  close(src);
  close(dest);
  return 0;
}
```

Output:

```
//create a file3

~/systemcalls$ cat>f3.txt

hi lets overwrite files

~/systemcalls$ gcc cpyi.c -o cpyi
```

```
//overwriting to existing file not allowed

~/systemcalls$ ./cpyi f3.txt f2.txt

Do you want to overwrite to existing file?n

Cant overwrite to existing file


//overwriting allowed and output verified

~/systemcalls$ ./cpyi f3.txt f2.txt

Do you want to overwrite to existing file?y

COPIED

~/systemcalls$ cat f2.txt

hi lets overwrite files
```

3) Implementing grep command

Algorithm:

1) Accept source and pattern to search as command line arguments
2) Open the source file in read only mode, if it is valid follow steps 3-4
3) Read the file character by character and store it to a character array if the scanned character is not a new line character
4) If the pattern is a substring of line array, print the line
5) Else, file descriptor is invalid
6) Close the file descriptor

```c
#include <stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>

int main(int argc,char *argv[]){
    int src = open(argv[2],O_RDONLY,0777);
    char buf,line[100];
    strcpy(argv[1],argv[1]);
```

```
    int i=0,x;
    while((x=read(src,&buf,sizeof(char))!=0)){  //read character by character
        if(buf!='\n')
        {
            line[i]=buf;
            i++;
        }
        else
        {
            line[i]='\n';
            i=0;
            if(strstr(line,argv[1]))      //check substring
            {
                printf("%s",line);
                memset(line,0,sizeof(line));
            }
        }
    }
    close(src);
    return 0;
}
```

Output:

```
~/systemcalls$ gcc grep.c -o grep

~/systemcalls$ ./grep "world" f1.txt

hello world

bye world
```

4) Implementing grep-c command

Algorithm:

1) Accept source and pattern to search as command line arguments
2) Open the source file in read only mode, if it is valid follow steps 3-4
3) Set count to 0
4) Read the file character by character and store it to a character array if the scanned character is not a new line character
5) If the pattern is a substring of line array, increment count, print count
6) Else, file descriptor is invalid
7) Close the file descriptor

```
#include <stdio.h>
#include<sys/types.h>
```

```c
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>

int main(int argc,char *argv[]){
    int src = open(argv[2],O_RDONLY,0777);
    char buf,line[100];
    strcpy(argv[1],argv[1]);
    int i=0,x,n=0;
    while((x=read(src,&buf,sizeof(char))!=0)){
        if(buf!='\n')
        {
            line[i++]=buf;
        }
        else
        {
            line[i]='\n';
            i=0;
            if(strstr(line,argv[1]))
            {
                ++n;                //increment count
            }
            memset(line,0,sizeof(line));
        }
    }
    printf("No of lines which match pattern: %d\n",n);
    close(src);
    return 0;
}
```

Output:

```
~/systemcalls$ gcc grepi.c -o grepi

~/systemcalls$ ./grepi "world" f1.txt

No of lines which match pattern: 2
```

5) Implementing ls command

Algorithm:

1) Accept source directory as command line argument
2) Open the directory

3) If descriptor is not NULL, read contents of a directory to a direct structure
4) If starting character is not '.',(hidden file) print d_name
5) Close the directory descriptor

```c
#include <stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<dirent.h>   //to import for directory

int main(int argc,char *argv[])
{
    struct dirent *d;      //to store the
    DIR *p;
    p=opendir(argv[1]);
    if(p==NULL)            //if directory is absent in the path given
    {
        printf("INVALID DIRECTORY NAME\n");
        exit(-1);
    }
    while((d=readdir(p))!=NULL)        //dir not empty condition
    {
        if(d->d_name[0]!='.'){         //to find hidden files(.)
            printf("%s\n",d->d_name);
        }
    }
    closedir(p);
    return 0;
}
```

Output:

```
//compiling and running ls using pwd as argument
~/systemcalls$ gcc ls.c -o ls
~/systemcalls$ ./ls .
main.c
f1.txt
f2.txt
f3.txt
cpy.c
cpy
cpyi
cpyi.c
grepi.c
grepi
```

```
grep.c
grep
ls.c
ls
//passing invalid directory
~/systemcalls$ ./ls hi
INVALID DIRECTORY NAME
```

6) Implementing ls –l command

Algorithm:

1) Accept source directory as command line argument
2) Open the directory
3) If descriptor is not NULL, read contents of a directory to a direct structure
4) If starting character is not '.',(hidden file), follow steps 5-7
5) Store the file details to a struct stat and use stat() function to access the file attributes
6) Print d if directory if found, else print –
7) Print r,w,x for user,group,others depending on details from stat variable (permissions)
8) Store passwords and groups into struct passwd and struct group and store the details into these structures
9) Print size of the file/directory
10) Store time to a struct time and print date, month and time using format operators
11) Print file name/directory name
12) Close the directory descriptor

```c
#include <stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<dirent.h>
#include<pwd.h>
#include<grp.h>
#include<time.h>

int main(int argc,char *argv[]){
    struct dirent *d;
    char
mon[12][4]={"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov"
,"Dec"};
    DIR *p;
    p=opendir(argv[1]);
```

```
    if(p==NULL){        //if directory is absent
        printf("INVALID DIRECTORY NAME\n");
        exit(-1);
    }
    while((d=readdir(p))!=NULL){
        if(d->d_name[0]!='.'){
            struct stat fstat;
            struct passwd *pwd;
            struct group *grp;
            struct tm dt;
            stat(d->d_name,&fstat);
            printf(S_ISDIR(fstat.st_mode)?"d":"-");
//permissions
            printf(fstat.st_mode & S_IRUSR ? "r":"-");
            printf(fstat.st_mode & S_IWUSR ? "w":"-");
            printf(fstat.st_mode & S_IXUSR ? "x":"-");
            printf(fstat.st_mode & S_IRGRP ? "r":"-");
            printf(fstat.st_mode & S_IWGRP ? "w":"-");
            printf(fstat.st_mode & S_IXGRP ? "x":"-");
            printf(fstat.st_mode & S_IROTH ? "r":"-");
            printf(fstat.st_mode & S_IWOTH ? "w":"-");
            printf(fstat.st_mode & S_IXOTH ? "x":"-");
            printf(" %ld",fstat.st_nlink);
            if((pwd=getpwuid(fstat.st_uid))!=NULL){   //username
                printf(" %-8.8s", pwd->pw_name);
            }
            else{
                printf(" %-8d",fstat.st_uid);
            }
             if((grp=getgrgid(fstat.st_gid))!=NULL){  //grpname
                printf(" %-8.8s", pwd->pw_name);
            }
            else{
                printf(" %-8d",fstat.st_uid);
            }
            printf(" %ld",fstat.st_size);
            dt=*(gmtime(&fstat.st_mtime));            //storing time
            printf(" %s %d
%d:%2d",&mon[dt.tm_mon][0],dt.tm_mday,dt.tm_hour,dt.tm_min);
            printf(" %s\n",d->d_name);
        }
    }
    closedir(p);
    return 0;
}
```

Output:

```
~/systemcalls$ gcc lsl.c -o lsl
```

```
~/systemcalls$ ./lsl .
-rw-r--r-- 1 runner   runner   349 Feb 9 11: 4 main.c
-rw-r--r-- 1 runner   runner   38 Feb 9 12:21 f1.txt
-rwxr-xr-x 1 runner   runner   24 Feb 9 12: 7 f2.txt
-rw-r--r-- 1 runner   runner   24 Feb 9 11:53 f3.txt
-rw-r--r-- 1 runner   runner   454 Feb 9 12: 7 cpy.c
-rwxr-xr-x 1 runner   runner   8520 Feb 9 12: 7 cpy
-rwxr-xr-x 1 runner   runner   8616 Feb 9 12: 8 cpyi
-rw-r--r-- 1 runner   runner   658 Feb 9 12:52 cpyi.c
-rw-r--r-- 1 runner   runner   561 Feb 9 14: 0 grepi.c
-rwxr-xr-x 1 runner   runner   8616 Feb 9 14:44 grepi
-rw-r--r-- 1 runner   runner   571 Feb 9 14:49 grep.c
-rwxr-xr-x 1 runner   runner   8616 Feb 9 14:49 grep
drwxr-xr-x 1 runner   runner   16 Feb 10 8:17 Desktop
-rw-r--r-- 1 runner   runner   496 Feb 10 8:32 ls.c
-rwxr-xr-x 1 runner   runner   8472 Feb 10 8:33 ls

~/systemcalls$ ./lsl hi
INVALID DIRECTORY NAME
```

EXERCISE 3:
IMPLEMENTATION OF CPU SCHEDULING POLICIES: FCFS AND SJF (PREEMPTIVE)

Objective:

To develop a menu driven C program to implement the CPU Scheduling Algorithms

Algorithm design

Create a structure process having variables pid,arrival time, turnaround time, burst time, completion time, waiting time, cpu first time response time and take input of number of processes as n

1. **FCFS:**

   a. Initialize variables tottat, totrt, totwt to 0.0
   b. Run a loop from 0 to n iterating through each process structure and perform steps b-e
   c. Increment the burst time of all processes in the order in which they arrive and store it in a variable sum. Completion time of each process is the sum of burst time upto that process
   d. Calculate turnaround time of each process as the difference of completion and arrival time. Increment this value to tottat
   e. Calculate waiting time as turnaround time – burst time. In case of FCFS response time is also turnaround time-burst time. Increment this value to totwt
   f. Calculate avg waiting time, avg response time, avg turnaround time by dividing total by n
   g. Print Gantt chart and table

2. **SJF Preemptive:**

   a. Run a loop from st=0 and increment it as long as count of finished processes(cnt) is not equal to no of processes(n)
   b. To compute the shortest process in that instant of time,run a loop from 0 to n to access all the process structures, if arrival time of process is greater than or equal to that instant and burst time of the process is less than the smallest process then initialize that process to smallest.
   c. Decrement burst time of the shortest process by 1

d. If the process is visited first (defined by flag=0), then store time of first execution of the process in cpuft and increment flag
e. If burst time of the shortest process is 0, increment cnt and initialize completion time of that process to st+1
f. Turnaround time of the process is calculated as completion time-arrival time
g. Waiting time of the process is tat-burst time
h. Response time is cpuft-arrival time
i. Calculate average values like previous algorithm and print Gantt chart and table

Code:

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

struct process {
    int pid,at,tat,bt,ct,wt,rt,cpuft,tempb,flg;
};

void getInput(struct process p[100],int n)
{
    for(int i=0;i<n;i++)
    {
        printf("PROCESS %d\n",(i+1));
        printf("Enter process ID: ");
        scanf("%d",&p[i].pid);
        printf("Enter process arrival time: ");
        scanf("%d",&p[i].at);
        printf("Enter process burst time: ");
        scanf("%d",&p[i].bt);
        p[i].tempb=p[i].bt;
    }
}

void printChart(struct process p[100],int n,int sum)
{
    printf("\nGANTT CHART\n");
    int j=0;
    for(int i=0;i<n;i++)
    {
        printf("-----------");
    }
    printf("\n");
    printf("|");
```

```
    for(int i=0;i<n;i++)
    {
        printf("     %d    |",p[i].pid);
    }
    printf("\n");
    for(int i=0;i<n;i++)
    {
        printf("-----------");
    }
    printf("\n");
    printf("%d          ",j);
    for(int i=0;i<n;i++)
    {
        j+=p[i].bt;
        printf("%d          ",j);
    }
}

void FCFS(struct process p[100],int n)
{
    int tq=0;
    int sum=0;
    float tottat=0.0,totwt=0.0;
    //calculate completion time
    for(int i=0;i<n;i++){
        sum+=p[i].bt;
        p[i].ct=sum;
    }

    //calculating turnaround time
    for(int i=0;i<n;i++)
    {
        p[i].tat=p[i].ct-p[i].at;
        tottat+=p[i].tat;
    }

    //calculating waiting time
    for(int i=0;i<n;i++)
    {
        p[i].wt=p[i].tat-p[i].bt;
        totwt+=p[i].wt;
    }
    //calculating average waiting time and average response time
    float avgtat,avgwt;
    avgwt=totwt/n;
    avgtat=tottat/n;
```

```
    //printing the table
    printf("\n-------------------------------------------------------------
\n");
    printf("ID\t  AT\t  BT\t  CT\t  TAT\t  WT\t  RT\n");
    printf("-------------------------------------------------------------
\n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t   %d\t   %d\t   %d\t   %d\t   %d\t
%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt,p[i].wt);
    }
    printf("-------------------------------------------------------------
\n");
    printf("AVERAGE: \t\t\t\t\t%.2f\t  %.2f\t
%.2f\n",avgtat,avgwt,avgwt);
    printChart(p,n,sum);
}


void SJFP(struct process p[100],int n)
{
    int pro[100],cnt=0,smallest,j=0;
    float avgwt,avgrt,avgtat,tottat=0.0,totwt=0.0,totrt=0.0,tottime;
    p[10].bt=p[10].tempb=9999;
    for(int i=0;i<n;i++)
    {
        p[i].flg=0;
    }
    for(int st=0;cnt!=n;st++)
    {
        smallest=10;
        for(int i=0;i<n;i++)
        {
            if(p[i].at<=st && p[i].bt<p[smallest].bt && p[i].bt>0)
            {
                smallest=i;
                if(p[i].flg==0)
                {
                    p[i].cpuft=st;
                    p[i].flg=1;
                }
            }
            pro[st]=p[smallest].pid;
            tottime=st;
        }
```

```
            p[smallest].bt--;
            if(p[smallest].bt==0)
            {
                p[smallest].ct=st+1;
                cnt++;

                //to calculate turnaround time
                p[smallest].tat=p[smallest].ct-p[smallest].at;
                tottat+=p[smallest].tat;

                //to calculate waiting time
                p[smallest].wt=p[smallest].tat-p[smallest].tempb;
                totwt+=p[smallest].wt;

                //to calculate response time
                p[smallest].rt=p[smallest].cpuft-p[smallest].at;
                totrt+=p[smallest].rt;
            }
        }
        printf("\nGANTT CHART\n");
        for(int i=0;i<tottime+1;i++)
        {
            printf("------");
        }
        printf("\n");
        printf("|");
        for(int i=0;i<tottime+1;i++)
        {
            printf("  %d  |",pro[i]);
        }
        printf("\n");
        for(int i=0;i<tottime+1;i++)
        {
            printf("------");
        }
        printf("\n");
        for(int i=0;i<tottime+2;i++)
        {
            printf("%d     ",i);
        }
        //printing the table
        printf("\n-----------------------------------------------------------
\n");
        printf("ID\t  AT\t  BT\t  CT\t  TAT\t  WT\t  RT\n");
        printf("-----------------------------------------------------------
\n");
```

```
    for(int i=0;i<n;i++)
    {
        printf("%d\t   %d\t   %d\t   %d\t   %d\t   %d\t
%d\n",p[i].pid,p[i].at,p[i].tempb,p[i].ct,p[i].tat,p[i].wt,p[i].rt);
    }
    printf("-------------------------------------------------------
\n");
    avgtat=tottat/n;
    avgwt=totwt/n;
    avgrt=totrt/n;
    printf("\nAVERAGE: \t\t\t\t\t%.2f\t  %.2f\t
%.2f\n",avgtat,avgwt,avgrt);
}



int main()
{
    int n,ch;
    char c;
    struct process p[100];
    printf("\t\t\tCPU SCHEDULING ALGORITHMS\n");
    c='y';
    do{
        printf("Enter number of processes: ");
        scanf("%d",&n);
        printf("1.FCFS\n2.SJFS\nEnter option: ");
        scanf("%d",&ch);
        getInput(p,n);
        if(ch==1)
        {
            FCFS(p,n);
        }
        else if(ch==2)
        {
            SJFP(p,n);
        }
        else
        {
            exit(0);
        }
        printf("Do you want to continue?: ");
        scanf(" %c",&c);
    }while(c=='y' || c=='Y');
    return 0;
```

}
Output:

```
            CPU SCHEDULING ALGORITHMS
Enter number of processes: 3
1.FCFS
2.SJFS
Enter option: 1
PROCESS 1
Enter process ID: 1
Enter process arrival time: 0
Enter process burst time: 5
PROCESS 2
Enter process ID: 2
Enter process arrival time: 1
Enter process burst time: 5
PROCESS 3
Enter process ID: 3
Enter process arrival time: 2
Enter process burst time: 4

-----------------------------------------------------
ID    AT      BT      CT      TAT     WT      RT
-----------------------------------------------------
1      0       5       5       5       0       0
2      1       5       10      9       4       4
3      2       4       14      12      8       8
-----------------------------------------------------
AVERAGE:                       8.67      4.00    4.00
```

```
GANTT CHART
--------------------------------
|     1    |     2    |     3    |
--------------------------------
0          5          10         14          Do you want to continue?: y
Enter number of processes: 3
1.FCFS
2.SJFS
Enter option: 2
PROCESS 1
Enter process ID: 1
Enter process arrival time: 0
Enter process burst time: 5
PROCESS 2
Enter process ID: 2
Enter process arrival time: 1
Enter process burst time: 4
PROCESS 3
Enter process ID: 3
Enter process arrival time: 2
Enter process burst time: 2

GANTT CHART
-------------------------------------------------------------
|  1  |  1  |  3  |  3  |  1  |  1  |  1  |  2  |  2  |  2  |  2  |
-------------------------------------------------------------
0     1     2     3     4     5     6     7     8     9     10    11
```

```
-----------------------------------------------------------
ID     AT       BT       CT       TAT      WT       RT
-----------------------------------------------------------
1       0        5        7        7        2        0
2       1        4        11       10       6        6
3       2        2        4        2        0        0
-----------------------------------------------------------

AVERAGE:                        6.33      2.67     2.00
Do you want to continue?: n
> s
```

EXERCISE 4
IMPLEMENTATION OF CPU SCHEDULING POLICIES PRIORITY (NON-PREEMPTIVE) AND ROUND ROBIN

Objective:

Develop a menu driven C program to implement the CPU Scheduling Algorithms Priority (Non-Preemptive) and Round Robin

Algorithm design

1) **PRIORITY SCHEDULING:**

a. Declare a structure process with pid, at, tat, bt, ct, wt, rt, flag, priority(pr)
b. Get input for the number of processes and take input for arrival time, burst time and priority of each process
c. Use two for loops to iterate one from 0 to n and j from i+1 to n. if arrival time and priority of a process j is less than i, then swap the structures i and j (to arrange processes in order of priority)
d. Set waiting time of first process to 0, turnaround time to burst time of first process
e. Iterate through every process, and calculate waiting time to waiting time + burst time. Increment this value to total waiting time
f. Calculate turnaround time as sum of waiting time of previous process and burst time of current process. Increment this value to total turnaround time
g. Calculate response time as sum of waiting time of previous waiting time and burst time of previous process. Increment this value to response time
h. Calculate average waiting time, average response time, average turnaround time as total time/process
i. Draw table and Gant chart for the process

2) **ROUND ROBIN SCHEDULING**

a. Declare a structure process with pid, at, tat, bt, ct, wt, rt, flag variable to check if process has been visited
b. Get input for the number of processes and take input for arrival time, burst time for each process and time quantum of scheduler
c. Initialise a cnt variable to keep track of processes completed and run a while loop as long as cnt!=number of processes

d. Initialize total time to 0 to keep track of total time
e. Run a for loop to iterate through all the processes, check if the process has arrived(arrival time<tottime) and if the burst time of the process is greater than 0 and initialize it as the first process
f. Declare an array called process to keep track of the process running at every instant.
g. If the process has not been visited, initialize flag to 1 and store the tottime(current time) to cpuft to mark the first time cpu allocation time of the process
h. If the burst time of the process is greater than the time quantum, then decrement burst time by time quantum and increment tottime by time quantum
i. Else increment tottime by remaining burst time and set burst time to 0. Set ct of the process to tottime and increment cnt by 1
j. Calculate turnaround time as ct-at. Increment this to tottat
k. Calculate waiting time as ct-bt. Increment this to totwt
l. Calculate response time as cpuft-at. Increment this to totrt
m. Calculate averages and tabulate the processes and represent using Gant chart

Code:

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

struct process {
    int pid,at,tat,bt,ct,wt,rt,pr,flag,cpuft;
};

void getInp(struct process p[100],int n,int c)
{
    for(int i=0;i<n;i++)
    {
        printf("PROCESS %d\n",(i+1));
        printf("Enter process ID: ");
        scanf("%d",&p[i].pid);
        printf("Enter process arrival time: ");
        scanf("%d",&p[i].at);
        printf("Enter process burst time: ");
        scanf("%d",&p[i].bt);
        if(c==1)
        {
```

```
            printf("Enter priority of the process: ");
            scanf("%d",&p[i].pr);
        }
    }
}

void printChart(struct process p[100],int n)
{
    printf("\nGANTT CHART\n");
    int j=p[0].at;
    for(int i=0;i<n;i++)
    {
        printf("-----------");
    }
    printf("\n");
    printf("|");
    for(int i=0;i<n;i++)
    {
        printf("     %d    |",p[i].pid);
    }
    printf("\n");
    for(int i=0;i<n;i++)
    {
        printf("-----------");
    }
    printf("\n");
    printf("%d          ",j);
    for(int i=0;i<n;i++)
    {
        j+=p[i].bt;
        printf("%d          ",j);
    }
}

void priorityNP(struct process p[100],int n)
{
    int first;
    float tottat=0.0,totwt=0.0,totrt=0.0,avgtat,avgwt,avgrt;
    struct process pr;
    for(int i=0;i<n;i++)
    {
        first=i;
        for(int j=i+1;j<n;j++)
        {
            if(p[j].pr<p[i].pr && p[j].at<p[i].at)
            {
```

```
                first=j;
            }
        }
        pr=p[first];
        p[first]=p[i];
        p[i]=pr;
    }
    for(int i=0;i<n;i++)
    {
        if(i==0)
        {
            //waiting time of first process is 0
            p[i].wt=p[i].rt=0;
            p[0].tat=p[0].bt-p[0].at;
            tottat+=p[0].tat;
            continue;
        }

        //calculating waiting time
        p[i].wt=p[i-1].wt+p[i-1].bt;
        totwt+=p[i].wt;

        //calculating turn around time
        p[i].tat=p[i-1].tat+p[i-1].at+p[i].bt-p[i].at;
        tottat+=p[i].tat;

        //calculating response time
        p[i].rt=p[i-1].wt+p[i-1].bt;
        totrt+=p[i].rt;
    }
    //calculating average values
    avgtat=tottat/n;
    avgwt=totwt/n;
    avgrt=totrt/n;

    //printing table
    printf("\n-------------------------------------------------------
\n");
    printf("ID\t  AT\t  BT\t  PR\t  TAT\t  WT\t  RT\n");
    printf("-------------------------------------------------------
\n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t   %d\t   %d\t   %d\t   %d\t   %d\t
%d\n",p[i].pid,p[i].at,p[i].bt,p[i].pr,p[i].tat,p[i].wt,p[i].wt);
    }
```

```
    printf("-------------------------------------------------------
\n");
    printf("AVERAGE:  \t\t\t\t\t%.2f\t  %.2f\t
%.2f\n",avgtat,avgwt,avgwt);
    printChart(p,n);
}

void RoundRobin(struct process p[100],int n)
{
    int tq,first,tempb[10],tottime=0,process[100],cnt=0,pc=0;
    float tottat=0.0,totwt=0.0,totrt=0.0,avgrt,avgwt,avgtat;
    printf("Enter time quantum: ");
    scanf("%d",&tq);
    for(int i=0;i<n;i++)
    {
        p[i].flag=0;
        tempb[i]=p[i].bt;
    }
    while(cnt!=n)
    {
        for(int i=0;i<n;i++)
        {
            if(p[i].at<=tottime && tempb[i]>0)
            {
                first=i;
                if(p[i].flag==0)
                //to check if the process has been visited earlier
                {
                    p[i].cpuft=tottime;
                    p[i].flag=1;
                }
                if(tempb[i]>tq)
                {
                    tottime+=tq;
                    tempb[i]-=tq;
                    for(int j=0;j<tq;j++)
                    {
                        process[pc++]=p[i].pid;
                    }
                }
                else  //last cycle
                {
                    for(int j=0;j<tempb[i];j++)
                    {
                        process[pc++]=p[i].pid;
                        //printf("%d",pc);
```

```
                    }
                    tottime+=tempb[i];
                    cnt++;

                    //calculating completion time
                    p[i].ct=tottime;

                    //calculating waiting time
                    p[i].wt=p[i].ct-p[i].bt;
                    totwt+=p[i].wt;

                    //calculating turnaround time
                    p[i].tat=p[i].ct-p[i].at;
                    tottat+=p[i].tat;

                    //calculating response time
                    p[i].rt=p[i].cpuft-p[i].at;
                    totrt+=p[i].rt;

                    //initialising burst time back to zero to indicate
end of process
                    tempb[i]=0;
                }
            }
        }
    }
    //printing table
    printf("\n---------------------------------------------------------
\n");
    printf("ID\t  AT\t  BT\t  CT\t  TAT\t  WT\t  RT\n");
    printf("---------------------------------------------------------
\n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t   %d\t   %d\t   %d\t   %d\t   %d\t
%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt,p[i].rt);
    }
    printf("---------------------------------------------------------
\n");
    avgtat=tottat/n;
    avgwt=totwt/n;
    avgrt=totrt/n;
    printf("\nAVERAGE: \t\t\t\t\t%.2f\t  %.2f\t
%.2f\n",avgtat,avgwt,avgrt);

    //printing Gantt chart
```

```
    printf("\nGANTT CHART\n");
    printf("\n");
    for(int i=0;i<pc;i++)
    {
        printf("----");
    }
    printf("\n");
    printf("|");
    for(int i=0;i<pc;i++)
    {
        printf(" %d |",process[i]);
    }
    printf("\n");
    for(int i=0;i<pc;i++)
    {
        printf("----");
    }
    printf("\n");
    for(int i=0;i<pc+1;i++)
    {
        printf("%d\t",i);
    }
}



int main()
{
    int n,ch;
    char c;
    struct process p[100];
    printf("\t\t\tCPU SCHEDULING ALGORITHMS\n");
    c='y';
    do{
        printf("Enter number of processes: ");
        scanf("%d",&n);
        printf("1.Non preemptive Priority scheduling\n2.Round robin
scheduling\nEnter option: ");
        scanf("%d",&ch);
        getInp(p,n,ch);
        if(ch==1)
        {
            priorityNP(p,n);
        }
        else if(ch==2)
        {
```

```
            RoundRobin(p,n);
        }
        else
        {
            exit(0);
        }
        printf("\nDo you want to continue?: ");
        scanf(" %c",&c);
    }while(c=='y' || c=='Y');
    return 0;
}
```

```
            CPU SCHEDULING ALGORITHMS
Enter number of processes: 5
1.Non preemptive Priority scheduling
2.Round robin scheduling
Enter option: 1
PROCESS 1
Enter process ID: 1
Enter process arrival time: 0
Enter process burst time: 5
Enter priority of the process: 1
PROCESS 2
Enter process ID: 2
Enter process arrival time: 4
Enter process burst time: 3
Enter priority of the process: 3
PROCESS 3
Enter process ID: 3
Enter process arrival time: 5
Enter process burst time: 7
Enter priority of the process: 4
PROCESS 4
Enter process ID: 4
Enter process arrival time: 2
Enter process burst time: 8
Enter priority of the process: 2
PROCESS 5
Enter process ID: 5
Enter process arrival time: 4
```

```
Enter process burst time: 7
Enter priority of the process: 5

------------------------------------------------------------
ID     AT       BT       PR       TAT      WT       RT
------------------------------------------------------------
1       0        5        1        5        0        0
4       2        8        2        11       5        5
2       4        3        3        12       13       13
3       5        7        4        18       16       16
5       4        7        5        26       23       23
------------------------------------------------------------
AVERAGE:                          14.40    11.40    11.40

GANTT CHART
------------------------------------------------------------
|      1     |      4     |      2     |      3     |      5     |
------------------------------------------------------------
0           5           13           16           23           30
Do you want to continue?: y
```

```
Do you want to continue?: y
Enter number of processes: 5
1.Non preemptive Priority scheduling
2.Round robin scheduling
Enter option: 2
PROCESS 1
Enter process ID: 1
Enter process arrival time: 0
Enter process burst time: 6
PROCESS 2
Enter process ID: 2
Enter process arrival time: 1
Enter process burst time: 8
PROCESS 3
Enter process ID: 3
Enter process arrival time: 3
Enter process burst time: 5
PROCESS 4
Enter process ID: 4
Enter process arrival time: 5
Enter process burst time: 9
PROCESS 5
Enter process ID: 5
Enter process arrival time: 4
Enter process burst time: 3
Enter time quantum: 2
```

ID	AT	BT	CT	TAT	WT	RT
1	0	6	21	21	15	0
2	1	8	28	27	20	1
3	3	5	24	21	19	1
4	5	9	31	26	22	1
5	4	3	19	15	16	4
AVERAGE:				22.00	18.40	1.40

GANTT CHART

1	1	2	2	3	3	4	4	5	5	1	1	2	2	3	3	4	4	5	1	1	2	2	3	4	4	2	2	4	4	4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Do you want to continue?: n

EXERCISE 5

INTER-PROCESS COMMUNICATIONS USING SHARED MEMORY

AIM:

Develop the following applications that use inter-process communication concepts using shared memory.

1.Develop an application for getting a name in parent and convert it into uppercase in child using shared memory.

2.Develop a client / server application for file transfer using shared memory.

3. Develop a client/server chat application using shared memory.

---

1) Develop an application for getting a name in parent and convert it into uppercase in child using shared memory.

ALGORITHM:

1) Use shmget() function to get a shared memory location and attach a character pointer to it(acts as parent array)
2) Attach the pointer to a the shared memory using shmat()
3) Fork the process and store the process id in pid
4) If pid>0, (parent process) accept a word from the user and store in character pointer in shared memory
5) Else (child process), create and attach the character pointer to shared memory and read from shared memory
6) Iterate through the word character by character and decrement 32(uppercase)
7) Print the array from the child process
8) Deallocate the memory space for future use

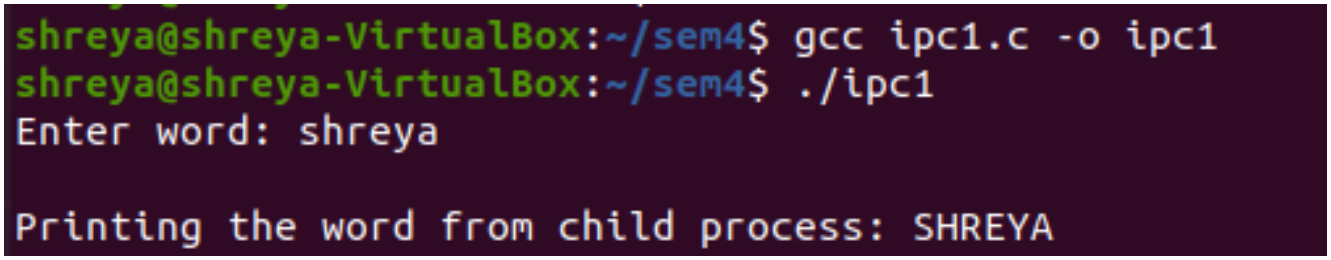
CODE:

```
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include<unistd.h>
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>


int main()
{
	int pid;
	char *word_p,*word_c;
	int id;
	id=shmget(300,50,IPC_CREAT | 00666); //creating a shared memory
using shmget
	pid=fork();
	if(pid>0) //parent process
	{
		word_p=shmat(id,NULL,0);    //attaching to shared memory
		printf("Enter word: ");
		scanf("%s",word_p);
		wait(NULL);
		shmdt(word_p);       //deallocating word_p
	}
	else //child process
	{
		sleep(3);
		word_c=shmat(id,NULL,0);  //attaching to shared mem
		int i=0;
		while(word_c[i]!='\0')  //iterating through the array
		{
			word_c[i]=word_c[i]-32; //changing to uppercase
			i++;
		}
		printf("\nPrinting the word from child process:
%s\n",word_c);
		shmdt(word_c);              //deallocating word_c
	}
	shmctl(id, IPC_RMID,NULL);  //shared memory released for reuse
}
```

OUTPUT:



2. Develop a client / server application for file transfer using shared memory.

ALGORITHM:

Client file:

1) Create a shared memory location using shmget() and attach a character pointer to it to store the file_client contents
2) Accept file name from user and store in file_client
3) Wait for server action
4) Print contents of file_client
5) Deallocate memory and free the shared memory space

Server file:

1) Create a shared memory location using shmget() and attach a character pointer to it to store the file_server contents
2) Read the shared memory and open the file in it, if file descriptor is -1, file doesn't exist
3) Else, read the file and store it to the file_server buffer
4) Wait for client action
5) Deallocate memory

CODE:

client.c

```
shreya@shreya-VirtualBox:~/sem4$ cat client.c
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include<unistd.h>
#include<stdio.h>
```

```
#include<stdlib.h>
#include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>


int main()
{
	printf("CLIENT SIDE\n");
	int id;
	char *file_client;
	id = shmget(112,100,IPC_CREAT | 00666);  //creating shared
memory
	file_client = shmat(id,NULL,0);   //attaching file_client
	printf("Enter filename: ");
	scanf(" %s",file_client);   //scan file name
	sleep(25); //wait for server
	printf("File contents:\n%s",file_client);     //print contents
	shmdt(file_client);
	shmctl(id,IPC_RMID,NULL);   //release shared memory
	return 0;
}
```

server.c

```
shreya@shreya-VirtualBox:~/sem4$ cat server.c
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>
#include<stdio.h>
#include<sys/stat.h>
#include<fcntl.h>



int main()
{
	printf("SERVER SIDE\n");
```

```
        sleep(3);
        int id;
        char *file_server;
        id = shmget(112,100,IPC_CREAT|00666);  //creating shared mem id
        file_server = shmat(id,NULL,0);
        printf("File to be transfered: %s",file_server);
        int fd = open(file_server,O_RDONLY);      //open file in read
        if(fd==-1){
                printf("\nFile doesnt exist");
        }
        else{
                int nc = read(fd,file_server,100); //read file
                file_server[nc]='\0';
                printf("\nFile read successfully\n");
                wait(NULL);
                close(fd);
                shmdt(file_server);
        }

}
```

```
shreya@shreya-VirtualBox:~/sem4$ gcc server.c -o server
shreya@shreya-VirtualBox:~/sem4$ ./server
SERVER SIDE
File to be transfered: test
File read successfully
```

```
shreya@shreya-VirtualBox:~/sem4$ gcc client.c -o client
shreya@shreya-VirtualBox:~/sem4$ ./client
CLIENT SIDE
Enter filename: test
File contents:
hello
bye
gn
world
```

3. Develop a client/server chat application using shared memory.

ALGORITHM:

CLIENT SIDE:

1) Create a shared memory location and attach a character pointer file_chat to it
2) Run a loop as long as the file_chat doesn’t contain end
3) Print server message
4) Accept client message and store in file_chat
5) Wait for server message
6) Deallocate and free shared space once end is given

USER SIDE:

1) Create a shared memory location and attach a character pointer file_chat to it
2) Run a loop as long as the file_chat doesn’t contain end
3) Wait for client message
4) Print client message
5) Accept server message and store in file_chat
6) Deallocate and free shared space once end is given

CODE:

clientchat.c

```
shreya@shreya-VirtualBox:~/sem4$ cat clientchat.c
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>

int main()
{
        int id;
        char *file_chat;
```

```
        id = shmget(200,1000,IPC_CREAT|00666);  //creating shared memid
        file_chat = shmat(id,NULL,0);
        while(strcmp(file_chat,"end")!=0)  //to end while loop
        {
                printf("SERVER: %s\n",file_chat);
                printf("YOU: ");
                scanf(" %[^\n]",file_chat);
                sleep(10);
                wait(NULL);
        }
        shmctl(id,IPC_RMID,NULL);   //releasing shared memory space
        return 0;
}
```

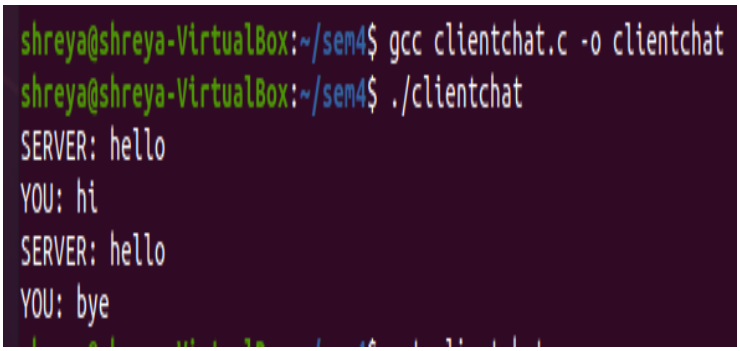
serverchat.c

```
shreya@shreya-VirtualBox:~/sem4$ cat serverchat.c
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>

int main()
{
        printf("SERVER CHAT\n");
        int id;
        char *file_chat;
        id = shmget(200,1000,IPC_CREAT|00666);
        file_chat = shmat(id,NULL,0);
        while(strcmp(file_chat,"end")!=0)
        {
                sleep(10);
                printf("CLIENT: %s\n",file_chat);
                printf("YOU: ");
                scanf(" %[^\n]",file_chat);
                wait(NULL);
        }
        shmctl(id,IPC_RMID,NULL);
        return 0;
}
```

OUTPUT:


shreya@shreya-VirtualBox:~/sem4$ gcc serverchat.c -o serverchat
shreya@shreya-VirtualBox:~/sem4$ ./serverchat
SERVER CHAT
CLIENT: hi
YOU: hello
CLIENT: bye
YOU: end
shreya@shreya-VirtualBox:~/sem4$


shreya@shreya-VirtualBox:~/sem4$ gcc clientchat.c -o clientchat
shreya@shreya-VirtualBox:~/sem4$ ./clientchat
SERVER: hello
YOU: hi
SERVER: hello
YOU: bye

EXERCISE 6

IMPLEMENTATION OF PRODUCER/CONSUMER PROBLEM USING SEMAPHORES

AIM:

To write a C program to create parent/child processes to implement the producer/consumer problem using semaphores in pthread library.

CODE:

```
shreya@shreya-VirtualBox:~/sem4$ cat semaphore1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <semaphore.h>
#include <pthread.h> // for semaphore operations
sem_init,sem_wait,sem_post
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <sys/errno.h>
#include <sys/types.h>
extern int errno;
#define SIZE 10 /* size of the shared buffer*/
#define VARSIZE 1 /* size of shared variable=1byte*/
#define INPUTSIZE 20
#define SHMPERM 0666 /* shared memory permissions */
int segid; /* id for shared memory bufer */
int empty_id;
int full_id;
int mutex_id;
char * buff;
char * input_string;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int p=0,c=0;

// Producer function

void produce()
```

```
{
	int i=0;
	while (1)
	{
		if(i>=strlen(input_string))
		{
			printf("\n Producer %d exited \n",getpid());
			wait(NULL);
			exit(1);
		}
	printf("\nProducer %d trying to aquire Semaphore Empty
\n",getpid());
	sem_wait(empty);
	printf("\nProducer %d successfully aquired Semaphore Empty
\n",getpid());
	printf("\nProducer %d trying to aquire Semaphore Mutex
\n",getpid());
	sem_wait(mutex);
	printf("\nProducer %d successfully aquired Semaphore Mutex
\n",getpid());
	buff[p]=input_string[i];
	printf("\nProducer %d Produced Item [ %c ]
\n",getpid(),input_string[i]);
	i++;
	p++;
	printf("\nItems in Buffer %d \n",p);
	sem_post(mutex);
	printf("\nProducer %d released Semaphore Mutex \n",getpid());
	sem_post(full);
	printf("\nProducer %d released Semaphore Full \n",getpid());
	sleep(2/random());
	} //while
}

// Consumer function

void consume()
{
	int i=0;
	while (1)
	{
		if(i>=strlen(input_string))
		{
			printf("\n Consumer %d exited \n",getpid());
			exit(1);
		}
```

```
                printf("\nConsumer %d trying to aquire Semaphore Full
\n",getpid());
                sem_wait(full);
                printf("\nConsumer %d successfully aquired Semaphore Full
\n",getpid());
                printf("\nConsumer %d trying to aquire Semaphore Mutex
\n",getpid());
                sem_wait(mutex);
                printf("\nConsumer %d successfully aquired Semaphore
Mutex\n",getpid());
                printf("\nConsumer %d Consumed Item [ %c ]
\n",getpid(),buff[c]);
                buff[c]=' ';
                c++;
                printf("\nItems in Buffer %d \n",strlen(input_string)-c);
                i++;
                sem_post(mutex);
                printf("\nConsumer %d released Semaphore Mutex
\n",getpid());
                sem_post(empty);
                printf("\nConsumer %d released Semaphore Empty
\n",getpid());
                sleep(1);
        }
}


//Main function

int main()
{
        int i=0;
        pid_t temp_pid;
        segid = shmget (IPC_PRIVATE, SIZE, IPC_CREAT | IPC_EXCL |
SHMPERM );
        empty_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        full_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        mutex_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        buff = shmat( segid, (char *)0, 0 );
        empty = shmat(empty_id,(char *)0,0);
        full = shmat(full_id,(char *)0,0);
        mutex = shmat(mutex_id,(char *)0,0);
        // Initializing Semaphores Empty , Full & Mutex
```

```
        sem_init(empty,1,SIZE);
        sem_init(full,1,0);
        sem_init(mutex,1,1);
        printf("\n Main Process Started \n");
        printf("\n Enter the input string (20 characters MAX) : ");
        input_string=(char *)malloc(20);
        scanf("%s",input_string);
        printf("Entered string : %s",input_string);
        temp_pid=fork();
        if(temp_pid>0) //parent
        {
               produce();
        }
        else //child
        {
               consume();
        }
        shmdt(buff);
        shmdt(empty);
        shmdt(full);
        shmdt(mutex);
        shmctl(segid, IPC_RMID, NULL);
        semctl( empty_id, 0, IPC_RMID, NULL);
        semctl( full_id, 0, IPC_RMID, NULL);
        semctl( mutex_id, 0, IPC_RMID, NULL);
        sem_destroy(empty);
        sem_destroy(full);
        sem_destroy(mutex);
        printf("\n Main process exited \n\n");
        return(0);
}
```

OUTPUT :

```
shreya@shreya-VirtualBox:~/sem4$ ./sem

 Main Process Started

 Enter the input string (20 characters MAX) : hi
Entered string : hi
Producer 6058 trying to aquire Semaphore Empty

Producer 6058 successfully aquired Semaphore Empty

Producer 6058 trying to aquire Semaphore Mutex

Producer 6058 successfully aquired Semaphore Mutex

Producer 6058 Produced Item [ h ]

Items in Buffer 1

Producer 6058 released Semaphore Mutex

Producer 6058 released Semaphore Full
Entered string : hi
Consumer 6059 trying to aquire Semaphore Full

Consumer 6059 successfully aquired Semaphore Full

Consumer 6059 trying to aquire Semaphore Mutex

Consumer 6059 successfully aquired Semaphore Mutex

Consumer 6059 Consumed Item [ h ]

Items in Buffer 1

Consumer 6059 released Semaphore Mutex

Consumer 6059 released Semaphore Empty

Producer 6058 trying to aquire Semaphore Empty

Producer 6058 successfully aquired Semaphore Empty

Producer 6058 trying to aquire Semaphore Mutex
```

```
Producer 6058 successfully aquired Semaphore Mutex

Producer 6058 Produced Item [ i ]

Items in Buffer 2

Producer 6058 released Semaphore Mutex

Producer 6058 released Semaphore Full

 Producer 6058 exited

Consumer 6059 trying to aquire Semaphore Full

Consumer 6059 successfully aquired Semaphore Full

Consumer 6059 trying to aquire Semaphore Mutex

Consumer 6059 successfully aquired Semaphore Mutex

Consumer 6059 Consumed Item [ i ]

Items in Buffer 0

Consumer 6059 released Semaphore Mutex

Consumer 6059 released Semaphore Empty

 Consumer 6059 exited
```

AIM:

Modify the program as separate client / server process programs to generate ‘N’ random numbers in producer and write them into shared memory. Consumer process should read them from shared memory and display them in terminal

CODE:

Client

```
shreya@shreya-VirtualBox:~/sem4$ cat sclient.c

#include <pthread.h> // for semaphore operations
sem_init,sem_wait,sem_post
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <sys/errno.h>
#include <sys/types.h>
#include<stdio.h>
#include<semaphore.h>
#include <unistd.h>
#include<stdlib.h>
#include<string.h>
extern int errno;
#define SIZE 5 /* size of the shared buffer*/
#define VARSIZE 1 /* size of shared variable=1byte*/
#define SHMPERM 0666 /* shared memory permissions */

int segid;    /* id for shared memory bufer */
int empty_id;
int full_id;
int mutex_id;
int shm_id;
int * buff;
char * input_string;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int c=1; // index of buffer in client side
int N;
```

```
// Consumer function
void consume()
{
	int i=0;
	while (1)
	{
		if(c==1)
		{
			sleep(3);
		}
		buff[0]=-1;
		if(i==N)  //if all elements are consumed
		{
			printf("\nConsumer exited\n");
			return;
		}
		sem_wait(full);	// decrement semaphore
		sem_wait(mutex);
		printf("\nConsumer %d Consumed Item [ %d ] in buffer
%d\n",getpid(),buff[c],c);
		buff[c]=-1;   //consumed
		c++;
		i++;
		if(c>SIZE)
		{
			c=1;
		}
		sem_post(mutex);
		sem_post(empty);
	}
}

int main()
{
	//creating shared memory id
	segid = shmget (1222, SIZE, IPC_CREAT | SHMPERM );
	empty_id=shmget(1223,sizeof(sem_t),IPC_CREAT|SHMPERM);
	full_id=shmget(1224,sizeof(sem_t),IPC_CREAT|SHMPERM);
	mutex_id=shmget(1225,sizeof(sem_t),IPC_CREAT|SHMPERM);

	//attaching to shared memory
	buff = shmat( segid, 0, 0 );
	empty = shmat(empty_id,(char *)0,0);
	full = shmat(full_id,(char *)0,0);
	mutex = shmat(mutex_id,(char *)0,0);
	// input for N
```

```
        printf("Enter N:");
        scanf("%d",&N);
        buff[0]=N;
        // initializing Semaphores Empty , Full & Mutex
        sem_init(empty,1,SIZE);
        sem_init(full,1,0);
        sem_init(mutex,1,1);

        //consumer
        consume();

        //deleting shared memory space
        shmdt(buff);
        shmdt(empty);
        shmdt(full);
        shmdt(mutex);

        //releasing space
        shmctl(segid, IPC_RMID, NULL);
        semctl( empty_id, 0, IPC_RMID, NULL);
        semctl( full_id, 0, IPC_RMID, NULL);
        semctl( mutex_id, 0, IPC_RMID, NULL);

        //destroy semaphores
        sem_destroy(empty);
        sem_destroy(full);
        sem_destroy(mutex);
        return(0);
}
```

Server

```
shreya@shreya-VirtualBox:~/sem4$ cat sserver.c

#include <pthread.h> // for semaphore operations
sem_init,sem_wait,sem_post
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <sys/errno.h>
#include <sys/types.h>
#include<stdio.h>
#include<semaphore.h>
#include <unistd.h>
```

```
#include<stdlib.h>
#include<string.h>
extern int errno;
#define SIZE 5 /* size of the shared buffer*/
#define VARSIZE 1 /* size of shared variable=1byte*/
#define SHMPERM 0666 /* shared memory permissions */
int segid;
int empty_id;
int full_id;
int mutex_id;
int shm_id;
int N;
int *buff;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int p=1; //index of buffer in server side
void produce()
{
	int i=0;
	while (1)
	{
		int n = rand();
		if(i>=N)
		{
			printf("\nProducer exited\n");
			return;
		}
		sem_wait(empty); // decrements semaphore
		sem_wait(mutex);
		buff[p]=n++;
		printf("\nProducer %d produced [ %d ] in buffer
%d\n",getpid(),buff[p],p);
		if(p==SIZE && buff[0]==N)
		{
			printf("\nProducer waits until buffer is full\n");
		}
		i++;
		p++;
		if(p>SIZE)
		{
			p=1;
		}
		sem_post(mutex);
		sem_post(full);
	}
```

```
}
int main()
{
	//creating shared memory id
	segid = shmget (1222, SIZE, IPC_CREAT | SHMPERM );
	empty_id=shmget(1223,sizeof(sem_t),IPC_CREAT|SHMPERM);
	full_id=shmget(1224,sizeof(sem_t),IPC_CREAT|SHMPERM);
	mutex_id=shmget(1225,sizeof(sem_t),IPC_CREAT|SHMPERM);

	//attaching to shared memory
	buff = shmat( segid, (char *)0, 0 );
	empty = shmat(empty_id,(char *)0,0);
	full = shmat(full_id,(char *)0,0);
	mutex = shmat(mutex_id,(char *)0,0);

	// initializing semaphores empty , full & mutex
	sem_init(empty,1,SIZE);
	sem_init(full,1,0);
	sem_init(mutex,1,1);
	buff[0]=-1;

	//sleeps as long as N is uninitialised in client
	while(buff[0]==-1)
	{
		sleep(1);
	}

	//initialising N from client
	N = buff[0];

	//producer
	produce();

	//deleting shared memory
	shmdt(buff);
	shmdt(empty);
	shmdt(full);
	shmdt(mutex);

	//releasing space
	shmctl(segid, IPC_RMID, NULL);
	semctl( empty_id, 0, IPC_RMID, NULL);
	semctl( full_id, 0, IPC_RMID, NULL);
	semctl( mutex_id, 0, IPC_RMID, NULL);

	//destroying semaphore
```

```
        sem_destroy(empty);
        sem_destroy(full);
        sem_destroy(mutex);
        return(0);
}
```

OUTPUT:

```
shreya@shreya-VirtualBox:~/sem4$ gcc sclient.c -lpthread -o semc
shreya@shreya-VirtualBox:~/sem4$ ./semc
Enter N:8

Consumer 6042 Consumed Item [ 1804289383 ] in buffer 1

Consumer 6042 Consumed Item [ 846930886 ] in buffer 2

Consumer 6042 Consumed Item [ 1681692777 ] in buffer 3

Consumer 6042 Consumed Item [ 1714636915 ] in buffer 4

Consumer 6042 Consumed Item [ 1957747793 ] in buffer 5

Consumer 6042 Consumed Item [ 424238335 ] in buffer 1

Consumer 6042 Consumed Item [ 719885386 ] in buffer 2

Consumer 6042 Consumed Item [ 1649760492 ] in buffer 3

Consumer exited
```

```
shreya@shreya-VirtualBox:~/sem4$ gcc sserver.c -lpthread -o sems
shreya@shreya-VirtualBox:~/sem4$ ./sems

Producer 6039 produced [ 1804289383 ] in buffer 1

Producer 6039 produced [ 846930886 ] in buffer 2

Producer 6039 produced [ 1681692777 ] in buffer 3

Producer 6039 produced [ 1714636915 ] in buffer 4

Producer 6039 produced [ 1957747793 ] in buffer 5

Producer waits until buffer is full

Producer 6039 produced [ 424238335 ] in buffer 1

Producer 6039 produced [ 719885386 ] in buffer 2

Producer 6039 produced [ 1649760492 ] in buffer 3

Producer exited
```

EXERCISE 7

IMPLEMENTATION OF BANKER'S ALGORITHM (DEADLOCK)

AIM:

Develop a C program to implement the banker's algorithm for deadlock avoidance.

CODE:

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

struct banker
{
    int no_proc,no_res;
    int process[10],need[10][10],avail[10],max[10][10],alloc[10][10];
    char res[10];
};

struct banker readInput(struct banker b)
{
    //entering number and process ID's
    printf("Enter no of processes: ");
    scanf("%d",&b.no_proc);
    for(int i=0;i<b.no_proc;i++)
    {
        printf("Enter process id %d: ",(i+1));
        scanf("%d",&b.process[i]);
    }

    //accepting number and names of the resource type
    printf("Enter no of resources: ");
    scanf("%d",&b.no_res);
    for(int i=0;i<b.no_res;i++)
    {
        printf("Enter resource type: ");
        scanf(" %c",&b.res[i]);
    }

    //accepting no of instances
    printf("Enter number of instances of each type of resource\n");
    for(int i=0;i<b.no_res;i++)
    {
        printf("Enter no of instances for %c: ",b.res[i]);
        scanf("%d",&b.avail[i]);
    }
```

```
    //accepting max requirement table
    printf("MAXIMUM REQUIREMENTS\n");
    for(int i=0;i<b.no_proc;i++)
    {
        printf("PROCESS %d\n",(i+1));
        for(int j=0;j<b.no_res;j++)
        {
            printf("Enter maximum requirement of resource %c: ",b.res[j]);
            scanf("%d",&b.max[i][j]);
        }
        printf("\n");
    }

    //accepting resource alloc table
    printf("ALLOCATED INSTANCES\n");
    for(int i=0;i<b.no_proc;i++)
    {
        printf("PROCESS %d\n",(i+1));
        for(int j=0;j<b.no_res;j++)
        {
            printf("Enter allocated instances of resource %c: ",b.res[j]);
            scanf("%d",&b.alloc[i][j]);
        }
        printf("\n");
    }

    //calculating need matrix
    for(int i=0;i<b.no_proc;i++)
    {
        for(int j=0;j<b.no_res;j++)
        {
            b.need[i][j] = b.max[i][j]-b.alloc[i][j];
        }
    }
    return b;
}

void printData(struct banker b)
{
    printf("\t\t\t\tALLOC\t\t\t\tMAX\t\t\t\t\tNEED\t\t\t\tAVAIL\n");
    printf("\t\t\t\t");

    //printing process types
    for(int j=0;j<4;j++)
    {
        for(int i=0;i<b.no_res;i++)
        {
            printf("%c ",b.res[i]);
        }
```

```
        printf("\t\t\t\t");
    }
    printf("\n");

    //printing each line of the table ie,process id,alloc,max,need,available
instances
    for(int i=0;i<b.no_proc;i++)
    {
        printf("P%d\t\t\t\t\t",(b.process[i]));
        for(int j=0;j<b.no_res;j++)
        {
            printf("%d ",b.alloc[i][j]);
        }
        printf("\t\t\t\t");
        for(int j=0;j<b.no_res;j++)
        {
            printf("%d ",b.max[i][j]);
        }
        printf("\t\t\t\t");
        for(int j=0;j<b.no_res;j++)
        {
            printf("%d ",b.need[i][j]);
        }
        printf("\t\t\t\t");
        for(int j=0;j<b.no_res;j++)
        {
            if(i==0)
            {
                printf("%d ",b.avail[j]);
            }
        }
        printf("\n");
    }

}


void checkSystemState(struct banker b)
{
    int status[10],sa=0,count=0,safe[10];

    //initialising status to 0
    for(int i=0;i<b.no_proc;i++)
    {
        status[i] = 0;
    }

    for(int i=0;i<b.no_proc;i++)
    {
        for(int j=0;j<b.no_proc;j++)
```

```
        {
            if(status[j]==0) //not completed
            {
                int flag = 0;
                for(int k=0;k<b.no_res;k++)
                {
                    //if need is greater than availability
                    if(b.need[j][k] > b.avail[k])
                    {
                        flag = 1;
                        break;
                    }
                }
                if(flag==0)
                {
                    //increment entry to safety array
                    safe[sa++] = j;
                    for(int k=0;k<b.no_res;k++)
                    {
                        //increment allocate to availability
                        b.avail[k] += b.alloc[j][k];
                    }
                    status[j] = 1;
                    count++; //keeps track of completed processes
                }
            }
        }
    }
    if(count==b.no_proc)
    {
        //printing safe sequence
        printf("\nSAFE SEQUENCE\n");
        for(int i=0;i<sa;i++)
        {
            if(i==sa-1)
            {
                printf("P%d",safe[i]);
                return;
            }
            printf("P%d -> ",safe[i]);
        }
        printf("\n");
    }
    else
    {
        printf("\nUNSAFE SEQUENCE\n");
    }
}

void resourceRequest(struct banker b)
```

```
{
    int p,rv[10],status[10],sa=0,count=0,safe[10],f=0;
    printf("\nRESOURCE REQUEST\n");
    printf("Enter proces for which request has been made: ");
    scanf("%d",&p);
    printf("Enter the resource request vector: ");
    for(int i=0;i<b.no_res;i++)
    {
        printf("Enter request for resource %c by process %d:
",b.res[i],b.process[i]);
        scanf("%d",&rv[i]);
    }
    for(int i=0;i<b.no_proc;i++)
    {
        status[i] = 0;
    }
    for(int i=0;i<b.no_res;i++)
    {
        //if request is lesser than availability and request is less than
need matrix entry for process p
        if(rv[i]<=b.avail[i] && rv[i]<=b.need[p][i])
        {
            //update need of process p to request vector
            b.need[p][i] = rv[i];
            f = 1;
        }
    }
    if(f==1)
    {
        //running safety algorithm on updated matrix
        for(int i=0;i<b.no_proc;i++)
        {
            for(int j=0;j<b.no_proc;j++)
            {
                if(status[j]==0)
                {
                    int flag = 0;
                    for(int k=0;k<b.no_res;k++)
                    {
                        if(b.need[j][k] > b.avail[k])
                        {
                            flag = 1;
                            break;
                        }
                    }
                    if(flag==0)
                    {
                        safe[sa++] = j;
                        for(int k=0;k<b.no_res;k++)
                        {
```

```
                            b.avail[k] += b.alloc[j][k];
                        }
                        status[j] = 1;
                        count++;
                    }
                }
            }
        }
    }
    if(count==b.no_proc)
    {
        printf("\nSAFE SEQUENCE\n");
        for(int i=0;i<sa;i++)
        {
            if(i==sa-1)
            {
                printf("P%d",safe[i]);
                return;
            }
            printf("P%d -> ",safe[i]);
        }
        printf("\nRESOURCE REQUEST FROM P%d CAN BE GRANTED\n",p);
    }
    else
    {
        printf("\nUNSAFE SEQUENCE\n");
    }
}


int main(void)
{
    struct banker b;
    int ch;
    char c;
    printf("\t\t\tBANKERS ALGORITHM\n");
    do
    {
        printf("1. READ DATA\n2. PRINT DATA\n3. CHECK SYSTEM DATE\n4.
RESOURCE REQUEST\n5. EXIT\nEnter option: ");
        scanf("%d",&ch);
        if(ch==1)
        {
            b = readInput(b);
        }
        else if(ch==2)
        {
            printData(b);
        }
        else if(ch==3)
```

```
        {
            checkSystemState(b);
        }
        else if(ch==4)
        {
            resourceRequest(b);
        }
        else if(ch==5)
        {
            exit(0);
        }
        printf("\nDo you want to continue?: ");
        scanf(" %c",&c);
    }while(c=='y' || c=='Y');
    return 0;
}
```

OUTPUT:

```
~/bankersalg$ ./banker
            BANKERS ALGORITHM
1. READ DATA
2. PRINT DATA
3. CHECK SYSTEM DATE
4. RESOURCE REQUEST
5. EXIT
Enter option:
1
Enter no of processes: 5
Enter process id 1: 0
Enter process id 2: 1
Enter process id 3: 2
Enter process id 4: 3
Enter process id 5: 4
Enter no of resources: 3
Enter resource type: A
Enter resource type: B
Enter resource type: C
Enter number of instances of each type of resource
Enter no of instances for A: 3
Enter no of instances for B: 3
Enter no of instances for C: 2
MAXIMUM REQUIREMENTS
PROCESS 1
Enter maximum requirement of resource A: 7
Enter maximum requirement of resource B: 5
Enter maximum requirement of resource C: 3
```

```
PROCESS 2
Enter maximum requirement of resource A: 3
Enter maximum requirement of resource B: 2
Enter maximum requirement of resource C: 2

PROCESS 3
Enter maximum requirement of resource A: 9
Enter maximum requirement of resource B: 0
Enter maximum requirement of resource C: 2

PROCESS 4
Enter maximum requirement of resource A: 2
Enter maximum requirement of resource B: 2
Enter maximum requirement of resource C: 2

PROCESS 5
Enter maximum requirement of resource A: 4
Enter maximum requirement of resource B: 3
3
Enter maximum requirement of resource C:
ALLOCATED INSTANCES
PROCESS 1
Enter allocated instances of resource A: 0
Enter allocated instances of resource B: 1
Enter allocated instances of resource C: 0

PROCESS 2
Enter allocated instances of resource A: 2
```

```
PROCESS 2
Enter allocated instances of resource A: 2
Enter allocated instances of resource B: 0
Enter allocated instances of resource C: 0

PROCESS 3
Enter allocated instances of resource A: 3
Enter allocated instances of resource B: 0
Enter allocated instances of resource C: 2

PROCESS 4
Enter allocated instances of resource A: 2
Enter allocated instances of resource B: 1
Enter allocated instances of resource C: 1

PROCESS 5
Enter allocated instances of resource A: 0
Enter allocated instances of resource B: 0
Enter allocated instances of resource C: 2


Do you want to continue?: Y
1. READ DATA
2. PRINT DATA
3. CHECK SYSTEM DATE
4. RESOURCE REQUEST
5. EXIT
```

```
Enter option: 2
               ALLOC          MAX            NEED           AVAIL
               A B C          A B C          A B C          A B C
P0             0 1 0          7 5 3          7 4 3          3 3 2
P1             2 0 0          3 2 2          1 2 2
P2             3 0 2          9 0 2          6 0 0
P3             2 1 1          2 2 2          0 1 1
P4             0 0 2          4 3 3          4 3 1

Do you want to continue?: Y
1. READ DATA
2. PRINT DATA
3. CHECK SYSTEM DATE
4. RESOURCE REQUEST
5. EXIT
Enter option: 3

SAFE SEQUENCE
P1 -> P3 -> P4 -> P0 -> P2
Do you want to continue?: Y
1. READ DATA
2. PRINT DATA
3. CHECK SYSTEM DATE
4. RESOURCE REQUEST
5. EXIT
Enter option: 4

RESOURCE REQUEST
```

```
Enter option: 4

RESOURCE REQUEST
Enter proces for which request has been made: 1
Enter the resource request vector: Enter request for resource A by process 0: 0
Enter request for resource B by process 1: 1
Enter request for resource C by process 2: 0

SAFE SEQUENCE
P1 -> P3 -> P4 -> P0 -> P2
Do you want to continue?: Y
```

```
1. READ DATA
2. PRINT DATA
3. CHECK SYSTEM DATE
4. RESOURCE REQUEST
5. EXIT
Enter option: 4

RESOURCE REQUEST
Enter proces for which request has been made: 1
Enter the resource request vector: Enter request for resource A by process 0: 4
Enter request for resource B by process 1: 4
Enter request for resource C by process 2: 4

UNSAFE SEQUENCE

Do you want to continue?: N
```

EXERCISE 8

IMPLEMENTATION OF MEMORY MANAGEMENT ALGORITHMS

AIM:

To develop a C program to implement first, best and worst fit memory allocation algorithms

CODE:

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

struct allocated_node
{
    int start_add,end_add,size;
    char status[3];
    struct allocated_node *next;
};

struct free_node
{
    int start_add,end_add,size;
    char status;
    struct free_node *next;
};

struct memory_request
{
    char pid[3];
    int size;
};


void OpMenu(int c,struct free_node *headfn,struct allocated_node *headan,int
n);
void insertFree(struct free_node *temp,struct free_node * headfn);
void insertAlloc(struct allocated_node *temp,struct allocated_node * headan);
void dispFree(struct free_node *headfn,int n);
void dispAlloc(struct allocated_node *headan,int n);
void readFree(struct free_node *headfn,struct allocated_node *headan,int n);
void AlgMenu();
void FirstFit(struct memory_request mr,struct allocated_node *headan,struct f
ree_node *headfn);
```

```
void BestFit(struct memory_request mr,struct allocated_node *headan,struct fr
ee_node *headfn);
void WorstFit(struct memory_request mr,struct allocated_node *headan,struct f
ree_node *headfn);
void deallocate(struct allocated_node *headan);
void Coalesce(struct allocated_node *headan,struct free_node *headfn);
void dispPhysical(struct allocated_node *headan,struct free_node *headfn,int
n);

void OpMenu(int c,struct free_node *headfn,struct allocated_node *headan,int
n)
{
    int ch;
    struct memory_request mr;
    char choice;
    while(1)
    {
        printf("\n1.Entry / Allocate\n2.Exit / Deallocate\n3.Display\n4.Coale
scing of Holes\n5. Back to Algorithm\nEnter choice: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
            printf("Enter process id :");
            scanf("%s",mr.pid);
            printf("Enter size needed : ");
            scanf("%d",&mr.size);
            mr.pid[2]='\0';
            if(c == 1)
            {
                FirstFit(mr,headan,headfn);
            }
            else if(c == 2)
            {
                BestFit(mr,headan,headfn);
            }
            else if(c == 3)
            {
                WorstFit(mr,headan,headfn);
            }
        }
        else if(ch == 2)
        {
            deallocate(headan);
            printf("\nDEALLOCATED\n");
        }
        else if(ch == 3)
        {
            dispPhysical(headan,headfn,n);
        }
        else if(ch == 4)
```

```
        {
            Coalesce(headan,headfn);
            printf("\nCOALESCED\n");
        }
        else if(ch == 5)
        {
            AlgMenu(headfn,headan,n);
        }
        printf("\n");
        dispFree(headfn,n);
        dispAlloc(headan,n);
    }
}

void AlgMenu(struct free_node *headfn,struct allocated_node *headan,int n)
{
    int ch;
    printf("\nMemory Allocation Algorithm\n1.First Fit\n2.Best Fit\n3.Worst F
it\n4.Exit from program\nEnter choice: ");
    scanf("%d",&ch);
    if(ch==1)
    {
        printf("\nFIRST FIT MEMORY ALLOCATION ALGORITHM\n");
    }
    else if(ch==2)
    {
        printf("\nBEST FIT MEMORY ALLOCATION ALGORITHM\n");
    }
    else if(ch==3)
    {
        printf("\nWORST FIT MEMORY ALLOCATION ALGORITHM\n");
    }
    else if(ch==4)
    {
        exit(0);
    }
    OpMenu(ch,headfn,headan,n);
}


void insertFree(struct free_node *temp,struct free_node * headfn)
{
    struct free_node *ptr;
    ptr=headfn;
    if(ptr->next == NULL)
    {
        ptr->next = temp;
        return;
    }
    while(ptr->next != NULL && ptr->next->start_add<temp->start_add)
```

```
    {
        ptr = ptr->next;
    }
    temp->next = ptr->next;
    ptr->next = temp;
}

void insertAlloc(struct allocated_node *temp,struct allocated_node *headan)
{
    struct allocated_node *ptr;
    ptr=headan;
    if(ptr->next == NULL)
    {
        ptr->next = temp;
        return;
    }
    while(ptr->next != NULL && ptr->next->start_add<temp->start_add)
    {
        ptr = ptr->next;
    }
    temp->next = ptr->next;
    ptr->next = temp;
}


void dispFree(struct free_node *headfn,int n)
{
    printf("\nFREE POOL\n");
    struct free_node *ptr;
    ptr = headfn;
    if(ptr->next==NULL)
    {
        printf("\n----> NULL\n");
        return;
    }
    for(int i=0;i<n;i++)
    {
         printf("----------------");
    }
    printf("\n");
    while(ptr->next!=NULL)
    {
        printf("|     %c      ",ptr->next->status);
        ptr = ptr->next;
    }
    printf("|");
    printf("\n");
    for(int i=0;i<n;i++)
    {
         printf("----------------");
```

```
    }
    printf("\n");
    ptr = headfn->next;
    while(ptr!=NULL)
    {
        printf("%d     %d  ",ptr->start_add,ptr->end_add);
        ptr = ptr->next;
    }
    printf("\n");
}

void dispAlloc(struct allocated_node *headan,int n)
{
    printf("\nALLOCATED MEMORY LL\n");
    struct allocated_node *ptr;
    ptr = headan;
    if(ptr->next==NULL)
    {
        printf("----> NULL\n");
        return;
    }
    for(int i=0;i<n;i++)
    {
        printf("-----------------");
    }
    printf("\n");
    while(ptr->next!=NULL)
    {
        printf("|    %s     ",ptr->next->status);
        ptr = ptr->next;
    }
    printf("|");
    printf("\n");
    for(int i=0;i<n;i++)
    {
        printf("-----------------");
    }
    printf("\n");
    ptr = headan->next;
    while(ptr!=NULL)
    {
        printf("%d     %d  ",ptr->start_add,ptr->end_add);
        ptr = ptr->next;
    }
    printf("\n");
}

void dispPhysical(struct allocated_node *headan,struct free_node *headfn,int
n)
{
```

```
printf("\nPHYSICAL MEMORY\n");
struct allocated_node *aptr = headan->next;
struct free_node *fptr = headfn->next;
int c= 0;
for(int i=0;i<n;i++)
{
    printf("-----------------");
}
printf("\n");
while(fptr->next!=NULL || aptr->next!=NULL)
{
    if(fptr->next!=NULL && fptr->start_add<aptr->start_add)
    {
        printf("|    %c     ",fptr->status);
        fptr = fptr->next;
    }
    else if(aptr->next!=NULL && aptr->start_add<fptr->start_add)
    {
        printf("|    %s     ",aptr->status);
        aptr = aptr->next;
    }
    else if(aptr->next==NULL && fptr->next!=NULL)
    {
        printf("|    %c     ",fptr->status);
        fptr = fptr->next;
    }
    else if(fptr->next==NULL && aptr->next!=NULL)
    {
        printf("|    %s     ",aptr->status);
        aptr = aptr->next;
    }
}
printf("\n");
for(int i=0;i<n;i++)
{
    printf("-----------------");
}
printf("\n");
aptr = headan->next;
fptr = headfn->next;
while(fptr->next!=NULL || aptr->next!=NULL)
{
    if(fptr->next!=NULL && fptr->start_add<aptr->start_add)
    {
        printf("%d     %d  ",fptr->start_add,fptr->end_add);
        fptr = fptr->next;
    }
    else if(aptr->next!=NULL && aptr->start_add<fptr->start_add)
    {
        printf("%d     %d  ",aptr->start_add,aptr->end_add);
```

```
            aptr = aptr->next;
        }
        else if(aptr->next==NULL && fptr->next!=NULL)
        {
            printf("%d     %d  ",fptr->start_add,fptr->end_add);
            fptr = fptr->next;
        }
        else if(fptr->next==NULL && aptr->next!=NULL)
        {
            printf("%d     %d  ",aptr->start_add,aptr->end_add);
            aptr = aptr->next;
        }
    }
    printf("\n");
}

void readFree(struct free_node *headfn,struct allocated_node *headan,int n)
{
    for(int i=0;i<n;i++)
    {
        struct allocated_node *atemp;
        struct free_node *temp;
        temp = (struct free_node*)malloc(sizeof(struct free_node));
        atemp = (struct allocated_node*)malloc(sizeof(struct allocated_node))
;
        printf("Starting and ending address of partition %d :",(i+1));
        scanf("%d %d",&temp->start_add,&temp->end_add);
        temp->next = NULL;
        temp->size = temp->end_add - temp->start_add;
        temp->status = 'H';
        insertFree(temp,headfn);
    }
}

void FirstFit(struct memory_request mr,struct allocated_node *headan,struct f
ree_node *headfn)
{
    struct allocated_node *aptr = (struct allocated_node*)malloc(sizeof(struc
t allocated_node));
    struct free_node *fptr = headfn->next;
    while(fptr != NULL)
    {
        if(fptr->size>=mr.size)
        {
            aptr->start_add = fptr->start_add;
            aptr->end_add = fptr->start_add+mr.size;
            aptr->size = mr.size;
            strcpy(aptr->status,mr.pid);
            aptr->next = NULL;
            insertAlloc(aptr,headan);
```

```
            fptr->start_add += mr.size;
            fptr->size -= mr.size;
            fptr->status = ' ';
            return;
        }
        else
        {
            break;
        }
        fptr = fptr->next;
    }
    printf("PROCESS HAS TO WAIT\n");
}

void BestFit(struct memory_request mr,struct allocated_node *headan,struct fr
ee_node *headfn)
{
    struct allocated_node *aptr = (struct allocated_node*)malloc(sizeof(struc
t allocated_node));
    struct free_node *sptr,*fptr = headfn->next;
    int best=1000;
    while(fptr!=NULL)
    {
        if(fptr->size-mr.size < best && fptr->size>=mr.size)
        {
            best = fptr->size-mr.size;
            sptr = fptr;
        }
        fptr = fptr->next;
    }
    aptr->start_add = sptr->start_add;
    aptr->end_add = sptr->start_add+mr.size;
    aptr->size = mr.size;
    aptr->next=NULL;
    strcpy(aptr->status,mr.pid);
    insertAlloc(aptr,headan);
    sptr->status = ' ';
    sptr->start_add += mr.size;
    sptr->size -= mr.size;
    if(sptr==NULL)
    {
        printf("\nPROCESS HAS TO WAIT\n");
    }
}

void WorstFit(struct memory_request mr,struct allocated_node *headan,struct f
ree_node *headfn)
{
    struct allocated_node *aptr = (struct allocated_node*)malloc(sizeof(struc
t allocated_node));
```

```
    struct free_node *sptr,*fptr = headfn->next;
    int worst=-1000;
    while(fptr!=NULL)
    {
        if(fptr->size-mr.size > worst && fptr->size>=mr.size)
        {
            worst = fptr->size-mr.size;
            sptr = fptr;
        }
        fptr = fptr->next;
    }
    aptr->start_add = sptr->start_add;
    aptr->end_add = sptr->start_add+mr.size;
    aptr->size = mr.size;
    aptr->next=NULL;
    strcpy(aptr->status,mr.pid);
    insertAlloc(aptr,headan);
    sptr->status = ' ';
    sptr->start_add += mr.size;
    sptr->size -= mr.size;
    if(sptr==NULL)
    {
        printf("\nPROCESS HAS TO WAIT\n");
    }
}


void deallocate(struct allocated_node *headan)
{
    char id[3];
    printf("Enter process id : ");
    scanf("%s",id);
    struct allocated_node *ptr = headan->next;
    while(ptr!=NULL)
    {
        if(strcmp(ptr->status,id)==0)
        {
            strcpy(ptr->status,"H");
            return;
        }
        ptr = ptr->next;
    }
    printf("PROCESS FOR DEALLOCATION NOT FOUND\n");
}


void Coalesce(struct allocated_node *headan,struct free_node *headfn)
{
    struct allocated_node *ptr = headan->next;
```

```
    struct free_node *free = (struct free_node*)malloc(sizeof(struct free_nod
e));
    while(ptr!=NULL && ptr->next!=NULL)
    {
        if(strcmp(ptr->status,ptr->next->status)==0)
        {
            ptr->size +=ptr->next->size;
            ptr->end_add = ptr->next->end_add;
            ptr->next = ptr->next->next;
            free->start_add = ptr->start_add;
            free->end_add = ptr->end_add;
            free->size = ptr->size;
            free->status = 'H';
            free->next = NULL;
            insertFree(free,headfn);
        }
        else
        {
            ptr = ptr->next;
        }
    }
}



int main(void)
{
    int n;
    printf("Enter the Memory Representation\n");
    printf("Enter the no.of partitions in memory : ");
    scanf("%d",&n);

    struct allocated_node *headan;
    headan = (struct allocated_node*)malloc(sizeof(struct allocated_node));
    headan->next = NULL;

    struct free_node *headfn;
    headfn = (struct free_node*)malloc(sizeof(struct free_node));
    headfn->next = NULL;

    struct memory_request mr;
    readFree(headfn,headan,n);
    AlgMenu(headfn,headan,n);
    return 0;
}
```

OUTPUT :

```
Enter the Memory Representation
Enter the no.of partitions in memory : 5
Starting and ending address of partition 1 :100 110
Starting and ending address of partition 2 :110 112
Starting and ending address of partition 3 :112 117
Starting and ending address of partition 4 :117 120
Starting and ending address of partition 5 :120 125

Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit from program
Enter choice: 1

FIRST FIT MEMORY ALLOCATION ALGORITHM

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 1
Enter process id :P1
Enter size needed : 5


FREE POOL
-------------------------------------------------------------------------------------
|            |     H      |     H      |     H      |     H      |
-------------------------------------------------------------------------------------
105     110  110     112  112     117  117     120  120     125
```

```
ALLOCATED MEMORY LL
-------------------------------------------------------------------------------------
|    P1     |
-------------------------------------------------------------------------------------
100     105

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 5

Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit from program
Enter choice: 2

BEST FIT MEMORY ALLOCATION ALGORITHM

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 1
Enter process id :P2
Enter size needed : 3
```

```
FREE POOL
-------------------------------------------------------------------------------
|           |     H      |     H      |           |     H      |
-------------------------------------------------------------------------------
105     110  110     112  112     117  120     120  120     125

ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    P1     |    P2     |
-------------------------------------------------------------------------------
100     105  117     120

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 5

Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit from program
Enter choice: 3

WORST FIT MEMORY ALLOCATION ALGORITHM

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 1
Enter process id :P3
Enter size needed : 2
```

```
FREE POOL
-------------------------------------------------------------------------------
|           |     H      |     H      |           |     H      |
-------------------------------------------------------------------------------
107     110  110     112  112     117  120     120  120     125

ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    P1     |    P3     |    P2     |
-------------------------------------------------------------------------------
100     105  105     107  117     120

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 3

PHYSICAL MEMORY
-------------------------------------------------------------------------------
|    P1     |    P3     |          |    H     |    H     |
-------------------------------------------------------------------------------
100     105  105     107  107     110  110     112  112     117  120     120


FREE POOL
-------------------------------------------------------------------------------
|           |     H      |     H      |           |     H      |
-------------------------------------------------------------------------------
107     110  110     112  112     117  120     120  120     125
```

```
ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    P1     |    P3     |    P2     |
-------------------------------------------------------------------------------
100     105  105     107  117     120

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 2
Enter process id : P1

DEALLOCATED


FREE POOL
-------------------------------------------------------------------------------
|            |     H      |     H      |            |     H      |
-------------------------------------------------------------------------------
107     110  110     112  112     117  120     120  120     125

ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    H     |    P3     |    P2     |
-------------------------------------------------------------------------------
100     105  105     107  117     120

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 2
Enter process id : P3
```

```
DEALLOCATED


FREE POOL
-------------------------------------------------------------------------------
|            |     H      |     H      |            |     H      |
-------------------------------------------------------------------------------
107     110  110     112  112     117  120     120  120     125

ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    H     |    H     |    P2     |
-------------------------------------------------------------------------------
100     105  105     107  117     120

1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 4

COALESCED


FREE POOL
-------------------------------------------------------------------------------
|     H      |            |     H      |     H      |            |     H      |
-------------------------------------------------------------------------------
100     107  107     110  110     112  112     117  120     120  120     125

ALLOCATED MEMORY LL
-------------------------------------------------------------------------------
|    H     |    P2     |
-------------------------------------------------------------------------------
100     107  117     120
```

```
1.Entry / Allocate
2.Exit / Deallocate
3.Display
4.Coalescing of Holes
5. Back to Algorithm
Enter choice: 5

Memory Allocation Algorithm
1.First Fit
2.Best Fit
3.Worst Fit
4.Exit from program
Enter choice: 4
PS D:\01-College\operating systems lab>
```

EXERCISE 9

IMPLEMENTATION OF PAGING MEMORY MANAGEMENT TECHNIQUE

AIM:

To develop a C program to implement the paging technique in memory management.

CODE:

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

//structure definitions

//process structure for maintaining id,size,no of frames required by each process and its page table
struct process
{
    int id,size,pages,pt[10];
};

//page structure for maintaining free frame list and physical memory attributes
struct page
{
    int pagesize,no_ff,totsize,no_frames,fflist[20],ffind,N;
    struct process pr[10];
};

//function definitions
void dispFreeFrame(struct page *p);
void init(struct page *p);
void processRequest(struct page *p);
void deallocate(struct page *p);
void dispPageTable(struct page *p);
//function definitions


void dispFreeFrame(struct page *p)
{
    printf("\nFREE FRAME LIST\n");
    for(int i=0;i<p->ffind;i++)
    {
        if(p->fflist[i] != -1)   //marks used free frames
```

```
        {
            printf("%d ",p->fflist[i]);
        }
    }
}


void init(struct page *p)
{
    int flag=1;
    printf("\nPaging Technique\nEnter the physical memory size: ");
    scanf("%d",&p->totsize);
    printf("Enter the page size: ");
    scanf("%d",&p->pagesize);
    p->no_frames = p->totsize/p->pagesize;
    printf("Physical memory is divided into %d frames.\n",p->no_frames);
    p->no_ff = p->ffind = 0;
    srand(time(0)); //acts as seed
    int iter = rand()%p->no_frames; //to generate a random number between 0
and no_frames
    for(int i=0;i<iter;i++)
    {
        int x = rand()%p->no_frames;
        for(int j=0;j<p->ffind;j++)
        {
            if(p->fflist[j]==x) //check if the random number already exists
in the fflist
            {
                flag = 0;
                break;
            }
        }
        if(flag == 1)   //if unique
        {
            p->fflist[p->ffind] = x;
            p->ffind += 1; //incrementing index of fflist and no of free
frames
            p->no_ff += 1;
        }
    }
    printf("\nAFTER INITIALISATION\n");
    dispFreeFrame(p);
}



void processRequest(struct page *p)
{
    int j=0;
    printf("Enter number of processes: ");
```

```
    scanf("%d",&p->N);
    int k=0;
    for(int i=0;i<p->N;i++)
    {
        printf("\nEnter process ID and size in KB: ");
        scanf("%d %d",&p->pr[i].id,&p->pr[i].size);
        p->pr[i].pages = p->pr[i].size/p->pagesize;
        printf("PROCESS REQUIRES %d FRAMES.\n",p->pr[i].pages);
        if(p->pr[i].pages<=p->no_ff)    //checking if enough frames are
available
        {
            int m=0;
            p->no_ff -= p->pr[i].pages; //decrementing free frames
            for(int j=0;j<p->pr[i].pages;j++)
            {
                if(p->fflist[k] != -1)  //identifying free frame
                {
                    p->pr[i].pt[m] = p->fflist[k];  //allocating free frame
to pagetable
                    p->fflist[k] = -1;  //blocks frame
                    k++;
                    m++;
                }
                else
                {
                    m++;
                    i--;
                }
            }
            printf("\nALLOCATED\n");
            printf("\nPAGE TABLE FOR PROCESS P%d\n",p->pr[i].id);
//displaying page table for current process
            for(int j=0;j<p->pr[i].pages;j++)
            {
                printf("PAGE %d : FRAME %d\n",j,p->pr[i].pt[j]);
            }
        }
        else
        {
            p->pr[i].id = -1;   //process hasnt been allocated with frames
            printf("\nINSUFFICIENT FRAMES\n");
        }
    }
}

void deallocate(struct page *p)
{
    int id,index=-1;
    printf("Enter process ID to deallocate: ");
    scanf("%d",&id);
```

```
    for(int i=0;i<p->N;i++)
    {
        if(p->pr[i].id == id)   //if process id matches input store index and
break
        {
            index=i;
            break;
        }
    }
    if(index!=-1)
    {
        p->pr[index].id = -1;   //removes id
        for(int k=0;k<p->pr[index].pages;k++)   //iterates through page table
        {
            p->fflist[p->ffind] = p->pr[index].pt[k];   //appends to free
space
            p->pr[index].pt[k] = -1;    //initialising page table of process
to -1
            p->ffind += 1;
            p->no_ff+=1;
        }
        printf("\nDEALLOCATED\n");
    }
    else    //process doesnt exist
    {
        printf("PROCESS NOT FOUND\n");
    }
}

void dispPageTable(struct page *p)
{
    printf("\nPAGE TABLE\n");
    for(int i=0;i<p->N;i++)
    {
        int k = 0;
        if(p->pr[i].id!=-1) //if process has been deallocated/not initialised
with frames
        {
            printf("\nPAGE TABLE FOR PROCESS P%d\n",p->pr[i].id);
            for(int j=0;j<p->pr[i].pages;j++)
            {
                printf("PAGE %d : FRAME %d\n",j,p->pr[i].pt[j]);
            }
        }
    }
}

int main()
{
    struct page p;
```

```
    int ch;
    init(&p);
    while(1)
    {
        printf("\n1. Process request\n2. Deallocation\n3. Page Table display
for all input process\n4. Free Frame list display\n5. Exit\nEnter option: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
            processRequest(&p);
        }
        else if(ch == 2)
        {
            deallocate(&p);
        }
        else if(ch == 3)
        {
            dispPageTable(&p);
        }
        else if(ch == 4)
        {
            dispFreeFrame(&p);
        }
        else if(ch == 5)
        {
            exit(0);
        }
    }
    return 0;
}
```

OUTPUT

```
Paging Technique
Enter the physical memory size: 32
Enter the page size: 1
Physical memory is divided into 32 frames.

AFTER INITIALISATION

FREE FRAME LIST
0 27 15 11 29 24 30
1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 1
Enter number of processes: 4

Enter process ID and size in KB: 1 2
PROCESS REQUIRES 2 FRAMES.

ALLOCATED

PAGE TABLE FOR PROCESS P1
PAGE 0 : FRAME 0
PAGE 1 : FRAME 27

Enter process ID and size in KB: 2 2
```

```
Enter process ID and size in KB: 2 2
PROCESS REQUIRES 2 FRAMES.

ALLOCATED

PAGE TABLE FOR PROCESS P2
PAGE 0 : FRAME 15
PAGE 1 : FRAME 11

Enter process ID and size in KB: 3 1
PROCESS REQUIRES 1 FRAMES.

ALLOCATED

PAGE TABLE FOR PROCESS P3
PAGE 0 : FRAME 29

Enter process ID and size in KB: 4 5
PROCESS REQUIRES 5 FRAMES.

INSUFFICIENT FRAMES

1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 3
```

```
PAGE TABLE

PAGE TABLE FOR PROCESS P1
PAGE 0 : FRAME 0
PAGE 1 : FRAME 27

PAGE TABLE FOR PROCESS P2
PAGE 0 : FRAME 15
PAGE 1 : FRAME 11

PAGE TABLE FOR PROCESS P3
PAGE 0 : FRAME 29

1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 4

FREE FRAME LIST
24 30
1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 2
```

```
Enter process ID to deallocate: 1

DEALLOCATED

1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 4

FREE FRAME LIST
24 30 0 27
1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 3

PAGE TABLE

PAGE TABLE FOR PROCESS P2
PAGE 0 : FRAME 15
PAGE 1 : FRAME 11

PAGE TABLE FOR PROCESS P3
PAGE 0 : FRAME 29
```

```
1. Process request
2. Deallocation
3. Page Table display for all input process
4. Free Frame list display
5. Exit
Enter option: 5
~/oscode$
```

EXERCISE 10

IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHMS

AIM:

Develop a C program to implement the page replacement algorithms (FIFO, Optimal, LRU and LFU) using linked list.

CODE:

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

struct node
{
    int frame;
    struct node *next;
};

struct data
{
    int ff,pf,refstr,reference[20],table[20][20],freq[10];
};

void getInput(struct data *d)
{
    char ref[30];
    printf("Enter number of free frames: ");
    scanf("%d",&d->ff);
    printf("Enter number of frames required by the process: ");
    scanf("%d",&d->pf);
    printf("Enter reference string (Enter to terminate): ");
    scanf("%s",ref);
    d->refstr = 0;
    //converting string input to integer array
    for(int i=0;i<strlen(ref);i++)
    {
        d->reference[i] = ref[i]-'0';
        d->refstr++;
    }
}

void dispList(struct node *head,int x,int pf)
{
```

```
    struct node *ptr = head->next;
    if(ptr==NULL)
    {
        return;
    }
    printf("%d ->\t",x);
    while(ptr != NULL)
    {
        //empty frame
        if(ptr->frame == -1)
        {
            printf("-\t");
        }
        else
        {
            printf("%d\t",ptr->frame);
        }
        ptr = ptr->next;
    }
    if(pf!=-1)
    {
        printf("->\t%d",pf);
    }
    else
    {
        printf("->\t-");
    }
}

void printTable(struct data *d,int pagefaults)
{
    printf("\nTABLE\n");
    for(int i=0;i<d->pf;i++)
    {
        for(int j=1;j<=pagefaults;j++)
        {
            if(d->table[i][j] != -1)
            {
                printf("%d\t",d->table[i][j]);
            }
            else
            {
               printf("-\t");
            }
        }
        printf("\n");
    }
}

void copyList(struct node *head,struct data *d,int pagefaults)
```

```
{
    struct node *ptr = head->next;
    int i = 0;
    while(ptr != NULL)
    {
        //copying the list elements to the table
        d->table[i++][pagefaults] = ptr->frame;
        ptr = ptr->next;
    }
}

struct node* initList(struct data *d,struct node *head)
{
    struct node *new,*ptr;
    head->next = NULL;
    for(int i=0;i<d->pf;i++)
    {
        for(int j=0;j<d->ff;j++)
        {
            d->table[i][j] = -1;
        }
    }
    for(int i=0;i<d->pf;i++)
    {
        ptr = head;
        new  = (struct node*)malloc(sizeof(struct node));
        new->frame = -1;
        //initialising all empty frames to -1
        if(ptr->next == NULL)
        {
            ptr->next = new;
            new->next = NULL;
            continue;
        }
        while(ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = new;
        new->next = NULL;
    }
    return head;
}

int toReplace(struct node *head,int n)
{
    //checking if list has the incoming reference string element
    struct node *ptr = head->next;
    while(ptr != NULL)
    {
```

```
        if(ptr->frame == n)
        {
            return 1;
        }
        ptr = ptr->next;
    }
    return 0;
}

int findIndexStart(int x,struct data *d,int skip,struct node *head)
{
    //returns the first occurence of the element
    for(int i=skip;i<d->refstr;i++)
    {
        if(d->reference[i] == x)
        {
            return i;
        }
    }
    return 100000;
}

int findIndexLessUse(int x,struct data *d,int skip)
{
    //checking if the element was least used in the reference list
    for(int i=skip-1;i>=0;i--)
    {
        if(d->reference[i] == x)
        {
            return i;
        }
    }
}

int findIndexLessFreq(int x,struct data *d,int skip)
{
    //finding the element which has less frequency
    for(int i=0;i<skip;i++)
    {
        if(d->reference[i] == x)
        {
            ++d->freq[d->reference[i]];
            return d->freq[d->reference[i]];
        }
    }
}


struct node * maxPeriod(struct node *head,struct data *d,int refind)
{
```

```
    int maxm = 0;
    struct node *ptr = head->next;
    struct node *temp;
    int frame;
    while(ptr != NULL)
    {
        //finding maximum
        int a = findIndexStart(ptr->frame,d,refind,head);
        if(a > maxm)
        {
            maxm = a;
            temp = ptr;
        }
        ptr = ptr->next;
    }
    return temp;
}

struct node* lessUse(struct node *head,struct data *d,int i)
{
    int m;
    struct node *ptr = head->next;
    struct node *temp;
    int frame;
    while(ptr != NULL)
    {
        int a = findIndexLessUse(ptr->frame,d,i);
        if(a < m)
        {
            m = a;
            temp = ptr;
        }
        ptr = ptr->next;
    }
    return temp;
}

struct node* lessFreq(struct node *head,struct data *d,int i)
{
    int m = 1000;
    struct node *ptr = head->next;
    struct node *temp;
    for(int i=0;i<10;i++)
    {
        d->freq[i] = 0;
    }
    //calculating frequency of each element from the reference string
    for(int j=0;j<i;j++)
    {
        ++d->freq[d->reference[j]];
```

```
    }
    while(ptr != NULL)
    {
        //finding least frequency
        int a = d->freq[ptr->frame];
        if(a < m)
        {
            m = a;
            temp = ptr;
        }
        ptr = ptr->next;
    }
    d->freq[temp->frame] = 0;
    return temp;
}

struct node * FIFO(struct node *head,struct data *d)
{
    printf("REF\t\tMEMORY\t\t\t\tPF\n");
    printf("------------------------------------------------------------------
-------------\n");
    int cpy,pagefaults=0;
    struct node *ptr,*p;
    ptr = head->next;
    p = head->next;
    for(int i=0;i<d->refstr;i++)
    {
        cpy = 0;
        if(toReplace(head,d->reference[i])==0)
        {
            //if the frames are not full
            if(ptr->frame == -1)
            {
                ptr->frame = d->reference[i];
                pagefaults++;
            }
            else
            {
                //increment page fault and replace the necessary frame
                p->frame = d->reference[i];
                pagefaults++;
                if(p->next!=NULL)
                {
                    p = p->next;
                }
                else
                {
                    p = head->next;
                }
            }
```

```
            //replace
            cpy = 1;
        }
        //not replaced
        else
        {
            cpy = 0;
        }
        if(ptr->next!=NULL)
        {
            ptr = ptr->next;
        }
        else
        {
            ptr = head->next;
        }
        //if replace is on copy to table and display the list
        if(cpy == 1)
        {
            copyList(head,d,pagefaults);
            dispList(head,d->reference[i],pagefaults);
            printf("\n");
        }
        //display list
        else
        {
            dispList(head,d->reference[i],-1);
            printf("\n");
        }
    }
    printf("\n\nNO OF PAGE FAULTS: %d\n",pagefaults);
    printTable(d,pagefaults);
    return head;
}



struct node * Optimal(struct node *head,struct data *d)
{
    printf("REF\t\tMEMORY\t\t\t\tPF\n");
    printf("-------------------------------------------------------------------
-------------\n");
    int cpy,pagefaults=0;
    struct node *ptr,*p;
    ptr = head->next;
    p = head->next;
    for(int i=0;i<d->refstr;i++)
    {
        cpy = 0;
        if(toReplace(head,d->reference[i])==0)
```

```
        {
            if(ptr != NULL)
            {
                ptr->frame = d->reference[i];
                pagefaults++;
                ptr = ptr->next;
            }
            else
            {
                p = maxPeriod(head,d,i);
                p->frame = d->reference[i];
                pagefaults++;
            }
            cpy = 1;
        }
        else
        {
            cpy = 0;
        }
        if(cpy == 1)
        {
            copyList(head,d,pagefaults);
            dispList(head,d->reference[i],pagefaults);
            printf("\n");
        }
        else
        {
            dispList(head,d->reference[i],-1);
            printf("\n");
        }
    }
    printf("\n\nNO OF PAGE FAULTS: %d\n",pagefaults);
    printTable(d,pagefaults);
    return head;
}

struct node* LRU(struct node *head,struct data *d)
{
    printf("REF\t\tMEMORY\t\t\t\tPF\n");
    printf("-------------------------------------------------------------------
-------------\n");
    int cpy,pagefaults=0;
    struct node *ptr,*p;
    ptr = head->next;
    for(int i=0;i<d->refstr;i++)
    {
        cpy = 0;
        if(toReplace(head,d->reference[i])==0)
        {
            if(ptr != NULL)
```

```
            {
                //if frames are not full
                ptr->frame = d->reference[i];
                pagefaults++;
                ptr = ptr->next;
            }
            else
            {
                //increment page faults and replace the required frame
                ptr = lessUse(head,d,i);
                ptr->frame = d->reference[i];
                pagefaults++;
            }
            cpy = 1;
        }
        else
        {
            cpy = 0;
        }
        //if replace is on copy to table and display it
        if(cpy == 1)
        {
            copyList(head,d,pagefaults);
            dispList(head,d->reference[i],pagefaults);
            printf("\n");
        }
        //display the list
        else
        {
            dispList(head,d->reference[i],-1);
            printf("\n");
        }
    }
    printf("\n\nNO OF PAGE FAULTS: %d\n",pagefaults);
    printTable(d,pagefaults);
    return head;
}

struct node* LFU(struct node *head,struct data *d)
{
    printf("REF\t\tMEMORY\t\t\t\tPF\n");
    printf("------------------------------------------------------------------
-------------\n");
    int cpy,pagefaults=0;
    struct node *ptr,*p;
    ptr = head->next;
    p = head->next;
    for(int i=0;i<d->refstr;i++)
    {
        cpy = 0;
```

```
        //if element does not have the be replaced
        if(toReplace(head,d->reference[i])==0)
        {
            if(ptr != NULL)
            {
                //if frames are not full
                ptr->frame = d->reference[i];
                pagefaults++;
                ptr = ptr->next;
            }
            else
            {
                //increment page fault and replace frame
                p = lessFreq(head,d,i);
                p->frame = d->reference[i];
                pagefaults++;
            }
            cpy = 1;
        }
        //continue
        else
        {
            cpy = 0;
        }
        if(cpy == 1)
        {
            copyList(head,d,pagefaults);
            dispList(head,d->reference[i],pagefaults);
            printf("\n");
        }
        else
        {
            dispList(head,d->reference[i],-1);
            printf("\n");
        }
    }
    printf("\n\nNO OF PAGE FAULTS: %d\n",pagefaults);
    printTable(d,pagefaults);
    return head;
}


int main()
{
    struct node *head;
    int ch;
    head = (struct node*)malloc(sizeof(struct node));
    struct data d;
    while(1)
    {
```

```
        printf("\nPAGE REPLACEMENT ALGORITHMS\n1. READ_INPUT\n2. FIFO\n3.
OPTIMAL\n4. LRU\n5. LFU\n6. EXIT\nEnter your option: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
            getInput(&d);
        }
        else if(ch == 2)
        {
            head = initList(&d,head);
            printf("\n");
            head = FIFO(head,&d);
        }
        else if(ch == 3)
        {
            head = initList(&d,head);
            printf("\n");
            head = Optimal(head,&d);
        }
        else if(ch == 4)
        {
            head = initList(&d,head);
            printf("\n");
            head = LRU(head,&d);
        }
        else if(ch == 5)
        {
            head = initList(&d,head);
            printf("\n");
            head = LFU(head,&d);
        }
        else if(ch == 6)
        {
            exit(0);
        }
        else
        {
            printf("\nEnter valid option!\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
PAGE REPLACEMENT ALGORITHMS
1. READ_INPUT
2. FIFO
3. OPTIMAL
4. LRU
5. LFU
6. EXIT
Enter your option: 1
Enter number of free frames: 10
Enter number of frames required by the process: 4
Enter reference string (Enter to terminate): 70120304230321201701

PAGE REPLACEMENT ALGORITHMS
1. READ_INPUT
2. FIFO
3. OPTIMAL
4. LRU
5. LFU
6. EXIT
Enter your option: 2
```

```
REF     MEMORY              PF
-------------------------------------------------------------------------------
7 ->    7   -   -   -   ->  1
0 ->    7   0   -   -   ->  2
1 ->    7   0   1   -   ->  3
2 ->    7   0   1   2   ->  4
0 ->    7   0   1   2   ->  -
3 ->    3   0   1   2   ->  5
0 ->    3   0   1   2   ->  -
4 ->    3   4   1   2   ->  6
2 ->    3   4   1   2   ->  -
3 ->    3   4   1   2   ->  -
0 ->    3   4   0   2   ->  7
3 ->    3   4   0   2   ->  -
2 ->    3   4   0   2   ->  -
1 ->    3   4   0   1   ->  8
2 ->    2   4   0   1   ->  9
0 ->    2   4   0   1   ->  -
1 ->    2   4   0   1   ->  -
7 ->    2   7   0   1   ->  10
0 ->    2   7   0   1   ->  -
1 ->    2   7   0   1   ->  -


NO OF PAGE FAULTS: 10
```

```
TABLE
7   7   7   7   3   3   3   3   2   2
-   0   0   0   0   4   4   4   4   7
-   -   1   1   1   1   0   0   0   0
-   -   -   2   2   2   2   1   1   1

PAGE REPLACEMENT ALGORITHMS
1. READ_INPUT
2. FIFO
3. OPTIMAL
4. LRU
5. LFU
6. EXIT
Enter your option: 3
```

```
 REF     MEMORY              PF
 ------------------------------------------------------------------------------
 7 ->    7   -   -   -   ->  1
 0 ->    7   0   -   -   ->  2
 1 ->    7   0   1   -   ->  3
 2 ->    7   0   1   2   ->  4
 0 ->    7   0   1   2   ->  -
 3 ->    3   0   1   2   ->  5
 0 ->    3   0   1   2   ->  -
 4 ->    3   0   4   2   ->  6
 2 ->    3   0   4   2   ->  -
 3 ->    3   0   4   2   ->  -
 0 ->    3   0   4   2   ->  -
 3 ->    3   0   4   2   ->  -
 2 ->    3   0   4   2   ->  -
 1 ->    1   0   4   2   ->  7
 2 ->    1   0   4   2   ->  -
 0 ->    1   0   4   2   ->  -
 1 ->    1   0   4   2   ->  -
 7 ->    1   0   7   2   ->  8
 0 ->    1   0   7   2   ->  -
 1 ->    1   0   7   2   ->  -


 NO OF PAGE FAULTS: 8
```

```
 TABLE
 7   7   7   7   3   3   1   1
 -   0   0   0   0   0   0   0
 -   -   1   1   1   4   4   7
 -   -   -   2   2   2   2   2

 PAGE REPLACEMENT ALGORITHMS
 1. READ_INPUT
 2. FIFO
 3. OPTIMAL
 4. LRU
 5. LFU
 6. EXIT
 Enter your option: 4
```

```
REF     MEMORY              PF
------------------------------------------------------------------------------
7 ->    7   -   -   -   ->  1
0 ->    7   0   -   -   ->  2
1 ->    7   0   1   -   ->  3
2 ->    7   0   1   2   ->  4
0 ->    7   0   1   2   ->  -
3 ->    3   0   1   2   ->  5
0 ->    3   0   1   2   ->  -
4 ->    4   0   1   2   ->  6
2 ->    4   0   1   2   ->  -
3 ->    4   3   1   2   ->  7
0 ->    4   3   0   2   ->  8
3 ->    4   3   0   2   ->  -
2 ->    4   3   0   2   ->  -
1 ->    4   3   0   1   ->  9
2 ->    2   3   0   1   ->  10
0 ->    2   3   0   1   ->  -
1 ->    2   3   0   1   ->  -
7 ->    7   3   0   1   ->  11
0 ->    7   3   0   1   ->  -
1 ->    7   3   0   1   ->  -


NO OF PAGE FAULTS: 11
```

```
TABLE
7   7   7   7   3   4   4   4   4   2   7
-   0   0   0   0   0   3   3   3   3   3
-   -   1   1   1   1   1   0   0   0   0
-   -   -   2   2   2   2   2   1   1   1

PAGE REPLACEMENT ALGORITHMS
1. READ_INPUT
2. FIFO
3. OPTIMAL
4. LRU
5. LFU
6. EXIT
Enter your option: 5
```

```
REF     MEMORY              PF
--------------------------------------------------------------------------------
7 ->    7   -   -   -   ->  1
0 ->    7   0   -   -   ->  2
1 ->    7   0   1   -   ->  3
2 ->    7   0   1   2   ->  4
0 ->    7   0   1   2   ->  -
3 ->    3   0   1   2   ->  5
0 ->    3   0   1   2   ->  -
4 ->    4   0   1   2   ->  6
2 ->    4   0   1   2   ->  -
3 ->    3   0   1   2   ->  7
0 ->    3   0   1   2   ->  -
3 ->    3   0   1   2   ->  -
2 ->    3   0   1   2   ->  -
1 ->    3   0   1   2   ->  -
2 ->    3   0   1   2   ->  -
0 ->    3   0   1   2   ->  -
1 ->    3   0   1   2   ->  -
7 ->    7   0   1   2   ->  8
0 ->    7   0   1   2   ->  -
1 ->    7   0   1   2   ->  -


NO OF PAGE FAULTS: 8
```

```
TABLE
7   7   7   7   3   4   3   7
-   0   0   0   0   0   0   0
-   -   1   1   1   1   1   1
-   -   -   2   2   2   2   2

PAGE REPLACEMENT ALGORITHMS
1. READ_INPUT
2. FIFO
3. OPTIMAL
4. LRU
5. LFU
6. EXIT
Enter your option: 6
```

EXERCISE 11

FILE ALLOCATION TECHNIQUES

AIM:

To develop a C program to implement the various file allocation techniques.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <math.h>

//linkedlist in memory
struct file_ptr
{
    int bid;
    struct file_ptr *next;
    struct file_ptr *n;
    char status[20];
};

//contains file details
struct f_details
{
    char filename[20];
    int fsize,start,end,n,fileblock[100];
};

//contains memory details
struct memory
{
    int fb,memsize,nb,nf;
    float blsize;
};

//function prototypes
```

```
void Contigous(struct memory m,struct f_details fd[10],struct file_ptr
*fhead);
void Linked(struct memory m,struct f_details fd[10],struct file_ptr
*fhead);
void Indexed(struct memory m,struct f_details fd[10],struct file_ptr
*fhead);
void initSystem(struct memory m,struct file_ptr *fhead);
int checkCont(struct file_ptr *fhead,struct memory m,struct f_details
f,int ran);
struct file_ptr * checkFree(struct file_ptr *fhead,int ran);
void setFree(struct file_ptr *fhead);


int main()
{
    struct memory m;
    struct f_details fd[10];
    struct file_ptr *fhead = (struct file_ptr*)malloc(sizeof(struct
file_ptr));
    int ch;
    fhead->next = NULL;
    //input memory details
    printf("Enter main memory size: ");
    scanf("%d",&m.memsize);
    printf("Size of each block in the disk: ");
    scanf("%f",&m.blsize);
    m.nb = m.memsize/m.blsize;
    printf("Number of files to be allocated: ");
    scanf("%d",&m.nf);
    //input file details
    for(int i=0;i<m.nf;i++)
    {
        printf("Enter name of the File%d: ",(i+1));
        scanf(" %s",fd[i].filename);
        printf("Enter size of the File%d: ",(i+1));
        scanf("%d",&fd[i].fsize);
        float x = fd[i].fsize/m.blsize;
        fd[i].n = (int)(x);
    }
    //to create data blocks in memory for file allocation
    initSystem(m,fhead);
    while(1)
    {
        printf("\nFILE ALLOCATION TECHNIQUES\n1. Contiguous\n2.
Linked\n3. Indexed\n4.Exit\nChoose the Allocation scheme: ");
        scanf("%d",&ch);
```

```
        if(ch == 1)
        {
            Contigous(m,fd,fhead);
        }
        else if(ch == 2)
        {
            Linked(m,fd,fhead);
        }
        else if(ch == 3)
        {
            Indexed(m,fd,fhead);
        }
        else if(ch == 4)
        {
            exit(0);
        }
        else
        {
            printf("\nEnter valid choice\n");
        }
    }
}

void setFree(struct file_ptr *fhead)
{
    //sets all datablocks free
    struct file_ptr *p = fhead->next;
    while(p != NULL)
    {
        strcpy(p->status,"free");
        p = p->next;
        p->next = NULL;
        p->n = NULL;
    }
}

void initSystem(struct memory m,struct file_ptr *fhead)
{
    for(int i=0;i<m.nb;i++)
    {
        struct file_ptr *temp = (struct file_ptr
*)malloc(sizeof(struct file_ptr));
        strcpy(temp->status,"free");
        temp->bid = i;
        temp->next = NULL;
        temp->n = NULL;
```

```
        //insertion of blocks into the file system
        if(fhead->next == NULL)
        {
            fhead->next = temp;
        }
        else
        {
            struct file_ptr *p = fhead;
            while(p->next != NULL)
            {
                p = p->next;
            }
            p->next = temp;
        }
    }
}


void Contigous(struct memory m,struct f_details fd[10],struct file_ptr
*fhead)
{
    int cnt = 0, flag;
    srand(time(0)); //to randomise
    for(int i=0;i<m.nb;i++)
    {
        if(cnt > m.nf) //if all processes are completed
        {
            break;
        }
        else
        {
            flag = 0;
            int ran = rand()%m.nb; //generate random number
            if(ran-fd[cnt].n>0)
            {
                flag = checkCont(fhead,m,fd[cnt],ran);
                if(flag == 1) //contiguous blocks are free
                {
                    fd[cnt].start = ran;
                    cnt++;
                }
            }
        }
    }
    //display directory structure
    printf("FILENAME\t\tSTART\tLENGTH\n");
```

```
    for(int i=0;i<m.nf;i++)
    {

printf("%s\t\t\t\t%d\t\t%d\n",fd[i].filename,fd[i].start,fd[i].n);
    }
}

struct file_ptr * checkFree(struct file_ptr *fhead,int ran)
{
    struct file_ptr *p = fhead->next;
    while(p != NULL)
    {
        //random block generated is free
        if(ran == p->bid && strcmp(p->status,"free")==0)
        {
            return p;
        }
        p = p->next;
    }
    return NULL;
}


int checkCont(struct file_ptr *fhead,struct memory m,struct f_details
f,int ran)
{
    int flag = 1;
    struct file_ptr *p = fhead->next;
    struct file_ptr *ptr;
    while(p != NULL)
    {
        if(ran == p->bid) //locate the random block
        {
            ptr = p;
            break;
        }
        p = p->next;
    }
    for(int i=0;i<f.n;i++)
    {
        if(strcmp(p->status,"free")!=0) //check if contiguous blocks
are free
        {
            flag = 0;
            return flag;
        }
```

```
            p = p->next;
        }
        if(flag == 1)
        {
            for(int i=0;i<f.n;i++)
            {
                strcpy(ptr->status,f.filename); //set block to filename
                ptr = ptr->next;
            }
        }
        return flag;
}

void Linked(struct memory m,struct f_details fd[10],struct file_ptr
*fhead)
{
    int cnt = 0,c=1,ran;
    struct file_ptr *flag,*sptr,*p;
    srand(time(0));
    for(int i=0;i<m.nb;i++)
    {
        if(cnt == m.nf) //all processes completed
        {
            break;
        }
        ran = rand()%m.nb; //generate random number
        flag = checkFree(fhead,ran);
        if(flag != NULL)
        {
            strcpy(flag->status,fd[cnt].filename);
            if(c == 1) //if its the starting block
            {
                fd[cnt].start = ran; //start
                sptr = flag;
            }
            sptr->n = flag;
            sptr = sptr->n;
            if(c == fd[cnt].n) //last block
            {
                fd[cnt].end = ran; //end
                cnt++;
                c = 0;
                sptr->n = NULL;
            }
            c++;
        }
```

```
    }
    printf("\nFILENAME\t\tSTART\tLENGTH\tEND\n");
    for(int i=0;i<m.nf;i++)
    {

printf("%s\t\t\t\t%d\t\t%d\t\t%d\n",fd[i].filename,fd[i].start,fd[i].n
,fd[i].end);
    }
    printf("\nINDIVIDUAL FILE LISTING\n");
    for(int i=0;i<m.nf;i++)
    {
        p = fhead->next;
        printf("%s\t",fd[i].filename);
        while(p != NULL)
        {
            if(fd[i].start == p->bid)
            {
                break;
            }
            p = p->next;
        }
        while(p != NULL)
        {
            printf("DATABLOCK:%d\t",p->bid);
            p = p->n;
        }
        printf("\n");
    }
}

void Indexed(struct memory m,struct f_details fd[10],struct file_ptr
*fhead)
{
    int cnt = 0,c=0,ran;
    struct file_ptr *flag,*ptr;
    srand(time(0));
    for(int i=0;i<m.nb;i++)
    {
        if(cnt == m.nf)
        {
            break;
        }
        ran = rand()%m.nb;
        flag = checkFree(fhead,ran);
        if(flag != NULL)
        {
```

```
            strcpy(flag->status,fd[cnt].filename);
            if(c == 0) //starting block ->index
            {
                fd[cnt].fileblock[0] = ran;
            }
            fd[cnt].fileblock[c] = ran;
            if(c == fd[cnt].n) //current file has been allocated
            {
                cnt++;
                c = -1;
            }
            c++;
        }
    }
    printf("\nFILENAME\t\tINDEX\n");
    for(int i=0;i<m.nf;i++)
    {
        printf("%s\t\t\t\t%d\n",fd[i].filename,fd[i].fileblock[0]);
    }
    printf("\nINDIVIDUAL FILE LISTING\n");
    for(int i=0;i<m.nf;i++)
    {
        printf("FILENAME: %s\t",fd[i].filename);
        printf("INDEX: %d\n",fd[i].fileblock[0]);
        for(int j=1;j<=fd[i].n;j++)
        {
            printf("\t\t\t\tDATABLOCK: %d\n",fd[i].fileblock[j]);
        }
        printf("\n");
    }
}
```

OUTPUT:

```
Enter main memory size: 1000
Size of each block in the disk: 10
Number of files to be allocated: 5
Enter name of the File1: f1
Enter size of the File1: 20
Enter name of the File2: f2
Enter size of the File2: 30
Enter name of the File3: f3
Enter size of the File3: 40
Enter name of the File4: f4
Enter size of the File4: 70
Enter name of the File5: f5
Enter size of the File5: 60

FILE ALLOCATION TECHNIQUES
1. Contiguous
2. Linked
3. Indexed
4.Exit
Choose the Allocation scheme: 1
FILENAME        START   LENGTH
f1              81      2
f2              10      3
f3              94      4
f4              28      7
f5              70      6
```

```
FILE ALLOCATION TECHNIQUES
1. Contiguous
2. Linked
3. Indexed
4.Exit
Choose the Allocation scheme: 2

FILENAME      START   LENGTH  END
f1            4       2       44
f2            40      3       45
f3            20      4       7
f4            98      7       42
f5            18      6       5

INDIVIDUAL FILE LISTING
f1  DATABLOCK:4 DATABLOCK:44
f2  DATABLOCK:40    DATABLOCK:26    DATABLOCK:45
f3  DATABLOCK:20    DATABLOCK:59    DATABLOCK:24    DATABLOCK:7
f4  DATABLOCK:98    DATABLOCK:99    DATABLOCK:17    DATABLOCK:89    DATABLOCK:65    DATABLOCK:36    DATABLOCK:42
f5  DATABLOCK:18    DATABLOCK:57    DATABLOCK:85    DATABLOCK:52    DATABLOCK:79    DATABLOCK:5
```

```
FILE ALLOCATION TECHNIQUES
1. Contiguous
2. Linked
3. Indexed
4.Exit
Choose the Allocation scheme: 3

FILENAME        INDEX
f1              13
f2              92
f3              8
f4              90
f5              50

INDIVIDUAL FILE LISTING
FILENAME: f1    INDEX: 13
                DATABLOCK: 0
                DATABLOCK: 25

FILENAME: f2    INDEX: 92
                DATABLOCK: 62
                DATABLOCK: 43
                DATABLOCK: 22
```

```
FILENAME: f3    INDEX: 8
                DATABLOCK: 68
                DATABLOCK: 23
                DATABLOCK: 27
                DATABLOCK: 58

FILENAME: f4    INDEX: 90
                DATABLOCK: 48
                DATABLOCK: 15
                DATABLOCK: 49
                DATABLOCK: 39
                DATABLOCK: 60
                DATABLOCK: 38
                DATABLOCK: 3

FILENAME: f5    INDEX: 50
                DATABLOCK: 67
                DATABLOCK: 91
                DATABLOCK: 46
                DATABLOCK: 76
                DATABLOCK: 16
                DATABLOCK: 77
```

```
FILE ALLOCATION TECHNIQUES
1. Contiguous
2. Linked
3. Indexed
4.Exit
Choose the Allocation scheme: 4
```

EXERCISE 12

FILE ORGANISATION TECHNIQUES

AIM:

To develop a C program to implement the following file organization techniques
a) Single level Directory
b) Hierarchical Structure

CODE:

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

struct file
{
    char filename[20];
    int start;
    int n;
};

struct sdir
{
    char name[20],parent[20];
    int nf,nd;
    struct file f[5];
};

void singleLevel(struct file s[100])
{
    srand(time(0));
    int ch,flag;
    char f[20];
    while(1)
    {
        printf("\n\n1.Create a file\n2.List the files\n3.Exit\nEnter
option: ");
```

```
        scanf("%d",&ch);
        if(ch == 1)
        {
            flag = 1;
            printf("\nEnter filename: ");
            scanf("%s",f);
            for(int i=0;i<100;i++)
            {
                if(strcmp(s[i].filename,f)==0)  //check if file
already exists
                {
                    flag = 0;
                }
            }
            if(flag == 0)
            {
                printf("\nFILE ALREADY EXISTS\n");
            }
            else
            {
                int ran = rand(); //rand number for starting address
                for(int i=0;i<100;i++)
                {
                    if(strcmp(s[i].filename,"free")==0)
                    {
                        //copy name of file and generate a start index
                        strcpy(s[i].filename,f);
                        s[i].start = ran;
                        printf("\nFILE CREATED\n");
                        break;
                    }
                }
            }
        }
        else if(ch == 2)
        {
            printf("\nContents of the root directory\n");
            printf("Filename\t\tLocation\n");
            for(int i=0;i<100;i++)
            {
                //display contents of occupied blocks
                if(strcmp(s[i].filename,"free")!=0)
                {
                    printf("%s\t\t\t\t%d\n",s[i].filename,s[i].start);
                }
            }
```

```
        }
        else
        {
            return;
        }
    }
}

void heirarchicalLevel(struct sdir sd[100])
{
    int n = 0,c;
    int i,flag;
    char name[20],f[20];
    sd[n].nd = sd[n].nf = 0;
    strcpy(sd[n].name,"root");
    while(1)
    {
        printf("\n1.Create directory\n2.Create
file\n3.List\n4.Exit\nEnter option: ");
        scanf("%d",&c);
        if(c == 1)
        {
            //select directory to which subdirectory is created
            printf("Select directory: ");
            scanf("%s",name);
            for(i=0;i<n;i++)
            {
                //checking for directory
                if(strcmp(sd[i].name,name)==0)
                {
                    break;
                }
            }
            //checking directory limit
            if(sd[i].nd<5)
            {
                printf("Enter directory name: ");
                sd[i].nd ++;  //increment no of directories
                n++;
                scanf("%s",sd[n].name);
                strcpy(sd[n].parent,name);
                sd[n].nd = sd[n].nf = 0;
                printf("Directory created\n");
            }
            else
            {
```

```
            //capacity exceeded
            printf("Directory cant be created\n");
        }
    }
    else if(c == 2)
    {
        //select directory in which file is created
        printf("Select directory: ");
        scanf("%s",name);
        for(i=0;i<n;i++)
        {
            if(strcmp(sd[i].name,name)==0)
            {
                break;
            }
        }
        //checking file limit in directories
        if(sd[i].nf<5)
        {
            printf("Enter file name: ");
            scanf("%s",f);
            flag = 1;
            for(int j=0;j<sd[i].nf;j++)
            {
                //check if file exists
                if(strcmp(sd[i].f[j].filename,f)==0)
                {
                    flag = 0;
                    printf("File exists!\n");
                }
            }
            if(flag == 1)
            {
                //copy details to directory structure
                strcpy(sd[i].f[sd[i].nf].filename,f);
                sd[i].f[sd[i].nf].start = rand();
                sd[i].nf ++;
                printf("File created\n");
            }
        }
        else
        {
            printf("File not created\n");
        }
    }
    else if(c == 3)
```

```
        {
            printf("root\n");
            //no of files
            for(int i=0;i<n;i++)
            {
                //chekcing for subdirectories
                for(int j=0;j<n;j++)
                {
                    //checking for files in subdirectories
                    if(strcmp(sd[j].parent,sd[i].name)==0)
                    {
                        printf("\t\t%s\n",sd[j].name);
                        for(int k=0;k<sd[j].nf;k++)
                        {

printf("\t\t\t\t%s\t%d\n",sd[j].f[k].filename,sd[j].f[k].start);
                        }
                    }
                }
            }
        }
        else
        {
            return;
        }
    }
}

int main()
{
    struct file s[100];
    for(int i=0;i<100;i++)
    {
        s[i].n = i;
        strcpy(s[i].filename,"free");
    }
    int ch;
    while(1)
    {
        printf("\nFile Organization techniques\n1.Single Level
Directory\n2.Tree structures directory\n3.Exit\nEnter your option: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
            singleLevel(s);
        }
```

```
        else if(ch == 2)
        {
            struct sdir sd[100];
            heirarchicalLevel(sd);
        }
        else if(ch == 3)
        {
            exit(0);
        }
    }
    return 0;
}
```

OUTPUT:

```
File Organization techniques
1.Single Level Directory
2.Tree structures directory
3.Exit
Enter your option: 1


1.Create a file
2.List the files
3.Exit
Enter option: 1

Enter filename: f1

FILE CREATED


1.Create a file
2.List the files
3.Exit
Enter option: 1

Enter filename: f2

FILE CREATED
```

```
1.Create a file
2.List the files
3.Exit
Enter option: 1

Enter filename: f3

FILE CREATED


1.Create a file
2.List the files
3.Exit
Enter option: 1

Enter filename: f4

FILE CREATED


1.Create a file
2.List the files
3.Exit
Enter option: 1

Enter filename: f1
```

```
FILE ALREADY EXISTS


1.Create a file
2.List the files
3.Exit
Enter option: 2

Contents of the root directory
Filename        Location
f1              779487813
f2              453946350
f3              1952888690
f4              79018775


1.Create a file
2.List the files
3.Exit
Enter option: 3

File Organization techniques
1.Single Level Directory
2.Tree structures directory
3.Exit
Enter your option: 3
```

```
File Organization techniques
1.Single Level Directory
2.Tree structures directory
3.Exit
Enter your option: 2

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 1
Select directory: root
Enter directory name: home
Directory created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 1
Select directory: root
Enter directory name: auth
Directory created

1.Create directory
2.Create file
3.List
4.Exit
```

```
Enter option: 1
Select directory: root
Enter directory name: other
Directory created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 1
Select directory: root
Enter directory name: user
Directory created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 1
Select directory: root
Enter directory name: pass
Directory created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 1
```

```
Select directory: root
Directory cant be created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f1
File created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f2
File created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f3
```

```
File created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f4
File created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f1
File exists!

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
Enter file name: f5
File created
```

```
1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: home
File not created

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 2
Select directory: user
Enter file name: shrey.txt
File created
```

```
1.Create directory
2.Create file
3.List
4.Exit
Enter option: 3
root
        home
                f1  1804289383
                f2  846930886
                f3  1681692777
                f4  1714636915
                f5  1957747793
        auth
        other
        user
                shrey.txt   424238335

1.Create directory
2.Create file
3.List
4.Exit
Enter option: 4
```

EXERCISE 13

THREADING APPLICATIONS

AIM:

To design a multithreaded program to determine statistical operations on a given list of numbers by employing worker threads to perform the computations

a) To find summation of n numbers

CODE:

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */
int main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of thread attributes */
    if (argc != 2)
    {
        fprintf(stderr,"usage: a.out <integer value>\n");
        return -1;
    }
    if (atoi(argv[1]) < 0)
    {
        fprintf(stderr,"%d must be >= 0 \n",atoi(argv[1]));
        return -1;
    }
    /* get the default attributes */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid,NULL);
    printf("sum = %d\n",sum);
}
/* The thread will begin control in this function */
void *runner(void *param)
{
```

```
    int i, upper = atoi(param);
    sum = 0;
    for (i = 1; i <= upper; i++)
    {
        sum += i;
    }
    pthread_exit(0);
}
```

OUTPUT:

```
~/oscode$ gcc thread.c -lpthread -o thread
~/oscode$ ./thread
usage: a.out <integer value>
~/oscode$ ./thread 5
sum = 15
```

b) To find calculate average, max and min of n numbers

CODE:

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

struct argument
{
    int argc;
    char **argv;
}cla;

int minm,maxm; /* this data is shared by the thread(s) */
float avg;
void *runner_max(void *param); /* threads call this function */
void *runner_min(void *param);
void *runner_avg(void *param);

int main(int argc, char *argv[])
{
    pthread_t tid_max; /* the thread identifier */
    pthread_t tid_min;
    pthread_t tid_avg;
```

```
    pthread_attr_t attr_max; /* set of thread attributes */
    pthread_attr_t attr_min;
    pthread_attr_t attr_avg;

    struct argument *cla;
    cla = (struct argument*)malloc(sizeof(struct argument*));
    cla->argc = argc;
    cla->argv = argv;

    if (cla->argc < 2)
    {
        fprintf(stderr,"usage: a.out <integer value>\n");
        return -1;
    }

    /* get the default attributes */
    pthread_attr_init(&attr_max);
    pthread_attr_init(&attr_min);
    pthread_attr_init(&attr_avg);

    /* create the thread */
    pthread_create(&tid_max,&attr_max,runner_max,cla);
    pthread_create(&tid_min,&attr_min,runner_min,cla);
    pthread_create(&tid_avg,&attr_avg,runner_avg,cla);

    /* wait for the thread to exit */
    pthread_join(tid_max,NULL);
    pthread_join(tid_min,NULL);
    pthread_join(tid_avg,NULL);

    printf("The average value is = %f\n",avg);
    printf("The minimum value is = %d\n",minm);
    printf("The maximum value is = %d\n",maxm);
}
/* The thread will begin control in this function */
void *runner_max(void *param)
{
    struct argument *cla = param;
    maxm = atoi(cla->argv[1]); //set first elt to max
    int i;
    for (int j = 1; j < cla->argc; j++)
    {
        i = atoi(cla->argv[j]);
        if(i > maxm) //comparing elements with max
        {
```

```
            maxm = i;
        }
    }
    pthread_exit(0);
}

void *runner_min(void *param)
{
    struct argument *cla = param;
    minm = atoi(cla->argv[1]); //set first element to min
    int i;
    for (int j = 1; j < cla->argc; j++)
    {
        i = atoi(cla->argv[j]);
        if(i < minm)
        {
            minm = i; //comparing elements with min
        }
    }
    pthread_exit(0);
}

void *runner_avg(void *param)
{
    int sum = 0; //init sum=0
    struct argument *cla = param;
    int i;
    for (int j = 1; j < cla->argc; j++) //iterate through arguments
    {
        i = atoi(cla->argv[j]);
        sum += i;
    }
    avg = sum/(cla->argc-1);
    pthread_exit(0);
}
```

OUTPUT:

```
~/oscode$ gcc thmod.c -lpthread -o thread
~/oscode$ ./thread
usage: a.out <integer value>
~/oscode$ ./thread 90 81 78 95 79 72 85
The average value is = 82.000000
The minimum value is = 72
The maximum value is = 95
```