DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PRACTICAL RECORD

CS6413

OPERATING SYSTEM LAB

NAME : ____________________________

REGISTER NO : ____________________________

SEMESTER : ____________________________

LIST OF EXPERIMENTS

(Implement the following on LINUX or other Unix like platform. Use C for high level language implementation)

1. Write programs using the following system calls of UNIX operating system:
fork, exec, getpid, exit, wait, close, stat, opendir, readdir
2. Write programs using the I/O system calls of UNIX operating system (open, read, write, etc)
3. Write C programs to simulate UNIX commands like ls, grep, etc.
4. Given the list of processes, their CPU burst times and arrival times, display/print the Gantt chart for FCFS and SJF. For each of the scheduling policies, compute and print the average waiting time and average turnaround time. (2 sessions)
5. Given the list of processes, their CPU burst times and arrival times, display/print the Gantt chart for Priority and Round robin. For each of the scheduling policies, compute and print the average waiting time and average turnaround time. (2 sessions)
6. Developing Application using Inter Process communication (using shared memory, pipes or message queues)
7. Implement the Producer – Consumer problem using semaphores (using UNIX system calls).
8. Implement some memory management schemes – I
9. Implement some memory management schemes – II
10. Implement any file allocation technique (Linked, Indexed or Contiguous)

Example for exercises 8 & 9 :

Free space is maintained as a linked list of nodes with each node having the starting byte address and the ending byte address of a free block. Each memory request consists of the process-id and the amount of storage space required in bytes. Allocated memory space is again maintained as a linked list of nodes with each node having the process-id, starting byte address and the ending byte address of the allocated space. When a process finishes (taken as input) the appropriate node from the allocated list should be deleted and this free disk space should be added to the free space list. [Care should be taken to merge contiguous free blocks into one single block. This results in deleting more than one node from the free space list and changing the start and end address in the appropriate node]. For allocation use first fit, worst fit and best fit.

EX. NO: 1(a) Implementation of process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

AIM:

To write a program for implementing process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

ALGORITHM:

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID
4. execvp() system call used to execute that command given on that command line argument
5. execlp() system call used to execute specified command.
6. Open the directory at specified in command line input.
7. Display the directory contents.
8. Stop the program.

PROGRAM:

```
#include<stdio.h>
main(int arc,char*ar[])

{
        int pid;
        char s[100];
        pid=fork();
        if(pid<0)

        printf("error");

        else if(pid>0)

        {
                wait(NULL);
```

```
		printf("\n Parent Process:\n");
		printf("\n\tParent Process id:%d\t\n",getpid());
		execlp("cat","cat",ar[1],(char*)0);
		error("can't execute cat %s,",ar[1]);

	}
	else

	{
		printf("\nChild process:");
		printf("\n\tChildprocess parent id:\t %d",getppid());
		sprintf(s,"\n\tChild process id :\t%d",getpid());
		write(1,s,strlen(s));
		printf(" ");
		printf(" ");
		printf(" ");
		execvp(ar[2],&ar[2]);
		error("can't execute %s",ar[2]);

	}
}
```

OUTPUT:

```
[root@localhost ~]# ./a.out tst date
Child process:
Child process id :
3137 Sat Apr 10 02:45:32 IST 2010
Parent Process:
Parent Process id:3136
sd
dsaASD[root@localhost ~]# cat tst
sd
dsaASD
```

RESULT:

Thus the program for process management was written and successfully executed.

EX NO:1 (b) Implementation of directory management using the following system calls of UNIX operating system: opendir, readdir.

AIM:

To write a program for implementing Directory management using the following system calls of UNIX operating system: opendir, readdir.

ALGORITHM:

1.Start the program.
2. Open the directory at specified in command line input.
3. Display the directory contents.
4. Stop the program.

PROGRAM:

```
#include<sys/types.h>
#include<dirent.h>
#include<stdio.h>
main(int c,char* arg[])
{

	DIR *d;

	struct dirent *r;
	int i=0;
	d=opendir(arg[1]);
	printf("\n\t NAME OF ITEM \n");
	while((r=readdir(d)) != NULL)
	{

		printf("\t %s \n",r->d_name);
		i=i+1;
	}
	printf("\n TOTAL NUMBER OF ITEM IN THAT DIRECTORY IS
	%d \n",i);
}
```

OUTPUT:

```
[root@localhost ~]# cc dr.c
[root@localhost ~]# ./a.out lab_print

NAME OF ITEM
pri_output.doc
sjf_output.doc
fcfs_output.doc
rr_output.doc

ipc_pipe_output.doc

pro_con_prob_output.doc

TOTAL NUMBER OF ITEM IN THAT DIRECTORY IS 8
```

RESULT:

Thus the program for directory management was written and successfully executed.

EX NO:2 Write a program using the I/O system calls of UNIX operating system (open,read, write, etc)

AIM:

To write a program using the I/O system calls of UNIX operating system (open, read, write, etc)

ALGORITHM:

1. Start the program.
2. Read the input from user specified file.
3. Write the content of the file to newly created file.
4. Show the file properties (access time, modified time, & etc,.)
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
main(int ag,char*arg[])

{
        char buf[100];
        struct stat s;
        int fd1,fd2,n;
        fd1=open(arg[1],0);
        fd2=creat(arg[2],0777);
        stat(arg[2],&s);
        if(fd2==-1)

        printf("ERROR IN CREATION");
        while((n=read(fd1,buf,sizeof(buf)))>0)
       {

                if(write(fd2,buf,n)!=n)
```

```
            {
                close(fd1);
                close(fd2);

            }
        }
        printf("\t\n UID FOR FILE.......>%d \n FILE ACCESS TIME.....>%s \n FILE
        MODIFIED TIME........>%s \n FILE I-NODE NUMBER......>%d \n
        PERMISSION FOR FILE.....>%o\n\n",s.st_uid,ctime
        (&s.st_atime),ctime(&s.st_mtime),s.st_mode);

        close(fd1);
        close(fd2);
}
```

OUTPUT:

```
[root@localhost ~]# cc iosys.c
[root@localhost ~]# ./a.out
UID FOR FILE.......>0
FILE ACCESS TIME.....>Thu Apr 8 01:23:54 2011
FILE MODIFIED TIME........>Thu Apr 8 01:23:54 2011
FILE I-NODE NUMBER......>33261
PERMISSION FOR FILE.....>1001101014
```

RESULT:

Thus the program using the I/O system calls was written and successfully executed.

EX NO:3 Write a programs to simulate UNIX commands like ls, grep, etc.

AIM:

To write a program to simulate UNIX commands like ls, grep, etc.

ALGORITHM:

1. Start the program.
2. Read the input through command line.
3. Open the specified file.
4. Options (c & i) are performed.
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
main(int ag,char* arg[])

{
        char buf[200],line[200];
        int i,j,n,fd1,count=0,opt;
        if(ag==4)
        {

                fd1=open(arg[3],0);
                if(strcmp(arg[1],"-c")==0)
                opt=2;
                if(strcmp(arg[1],"-i")==0)
                opt=3;
        }
        else if(ag==3)
        {
                fd1=open(arg[2],0);
                opt=1;
        }
        if(fd1==-1)
        printf("error in opening");
```

```
j=0;
switch(opt)
{
            case 1:
            while((n=read(fd1,buf,sizeof(line)))>0)
            {
                        for(i=0;i<n;i++,j++)

                    {
                            if(buf[i]!='\n') line[j]=buf[i];
                            else
                            {

                                        line[j]='\n';
                                        if(strstr(line,arg[1])!=0)
                                        write(1,line,j+1);
                              }
                      }
           }
           break;
           case 2:
           while((n=read(fd1,buf,sizeof(line)))>0)
           {
                   for(i=0;i<n;i++,j++)
                   {
                            if(buf[i]!='\n') line[j]=buf[i];
                            else
                            {
                                        line[j]='\n';
                                        if(strstr(line,arg[2])!=0)
                                        count=count+1;
                                        j=-1;
                              }
                   }
         }
         printf("%d \n",count);
         break;
```

```
            case 3:
            while((n=read(fd1,buf,sizeof(line)))>0)
            {
                for(i=0;i<n;i++,j++)
                {
                    if(buf[i]!='\n') line[j]=buf[i];
                    else
                    {
                        line[j]='\n';
                        if(strcasestr(line,arg[2])!=0)
                        write(1,line,j+1);
                        j=-1;
                    }
                }
            }
            break;
        }
        close(fd1);
}
```

OUTPUT:

[root@localhost ~]# cat tst sd
dsaASD[root@localhost ~]# ./a.out -i a tst
aA[root@localhost ~]# ./a.out -c a tst 1[root@localhost ~]# ./a.out -c A tst
1[root@localhost ~]# ./a.out -c sd tst 1[root@localhost ~]# ./a.out -c s tst 2

RESULT:

Thus the program to stimulate the UNIX commands was written and successfully executed.

EX NO:4 (a) Write a program for implementing the FCFS Scheduling algorithm

AIM:

To write a program for implementing FCFS scheduling algorithm.

ALGORITHM:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Select the process that first arrived in the ready queue
5. Make the average waiting the length of next process.
6. Start with the first process from it's selection as above and let other process to be in queue.
7. Calculate the total number of burst time.
8. Display the values.
9. Stop the process.

PROGRAM:

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

	float avgwt,avgtt;

	char pname[10][10],c[10][10];
	int wt[10],tt[10],bt[10],at[10],t,q,i,n,sum=0,sbt=0,ttime,j,ss=0;
	printf("\n\n Enter the number of processes: ");
	scanf("%d",&n);
	printf("\n\n Enter the NAME , BURST TIME and ARRIVAL TIME of the
	process");
	for(i=0;i<n;i++)
	{
```

```
            printf("\n\n NAME : ");
            scanf("%s",&pname[i]);
            printf("\n\n BURST TIME : ");
            scanf("%d",&bt[i]);
            printf("\n\n ARRIVAL TIME : ");
            scanf("%d",&at[i]);

}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
            if(at[i]>at[j])

                        {
                                     t=at[i];
                                    at[i]=at[j];
                                    at[j]=t;
                                    q=bt[i];
                                    bt[i]=bt[j];
                                    bt[j]=q;
                                    strcpy(c[i],pname[i]);

                                   Strcpy(pname[i],pname[j]);

                                    strcpy(pname[j],c[i]);
                          }

 }
 wt[0]=0;
 for(i=0;i<n;i++)

 {
              wt[i+1]=wt[i]+bt[i];
              sum=sum+(wt[i]-at[i]);
              sbt=sbt+(wt[i+1]-at[i]);
              tt[i]=wt[i]+bt[i];
              ss=ss+bt[i];

   }
```

```
 avgwt=(float) sum/n;

avgtt=(float)sbt/n;
printf("\n\n Average waiting time = %f",avgwt);
printf("\n\n Average turn-around time = %f",avgtt);
printf("\n\n GANTT CHART\n");
for(i=0;i<n;i++)
printf("|\t%s\t",pname[i]);
printf("\n");
for(i=0;i<n;i++)
printf("%d\t\t",wt[i]);
printf("%d\n",ss);
printf("\n");
}
```

OUTPUT:

```
[root@localhost ~]# ./a.out
Enter the number of processes: 4
Enter the NAME , BURST TIME and ARRIVAL TIME of the process
NAME : p1
BURST TIME : 4
ARRIVAL TIME : 0

NAME : p2
BURST TIME : 9
ARRIVAL TIME : 2

NAME : p3
BURST TIME : 8
ARRIVAL TIME : 4

NAME : p4
BURST TIME : 3
ARRIVAL TIME : 3
```

Average waiting time = 6.000000
Average turn-around time = 12.000000
GANTT CHART

```
 |   p1   |   p2   |   p4  |  p3

   0         4        13        16    24
```

RESULT:

Thus the program for implementing FCFs scheduling algorithm was written and successfully executed.

EX NO:4 (b) Write a program for simulation of SJF Scheduling algorithm

AIM:

To write a program for implementing SJF scheduling algorithm

ALGORITHM:

1.Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Select the process which have shortest burst will execute first.
5. If two process have same burst length then FCFS scheduling algorithm used.
6. Make the average waiting the length of next process.
7. Start with the first process from it’s selection as above and let other process to be in queue.
8. Calculate the total number of burst time.
9. Display the values.
10. Stop the process.

PROGRAM:

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

	float avgwt,avgtt;

	char pname[10][10],c[10][10];
	int wt[10],tt[10],bt[10],at[10],t,q,i,n,sum=0,sbt=0,ttime,j,ss=0;
	printf("\n\n Enter the number of processes: ");
	scanf("%d",&n);
	printf("\n\n Enter the NAME, BURSTTIME, and ARRIVALTIME of the
	processes ");
	for(i=0;i<n;i++)
	{

		printf("\n\n NAME : ");
		scanf("%s",&pname[i]);
		printf("\n\n BURST TIME : ");
```

```
          scanf("%d",&bt[i]);
          printf("\n\n ARRIVAL TIME : ");
          scanf("%d",&at[i]);

}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
          if(at[i]==at[j])
          if(bt[i]>bt[j])

          {
                    t=at[i];
                  at[i]=at[j];
                  at[j]=t;
                  q=bt[i];
                  bt[i]=bt[j];
                  bt[j]=q;
                  strcpy(c[i],pname[i]);
                  strcpy(pname[i],pname[j]);

                 strcpy(pname[j],c[i]);
          }
         if(at[i]!=at[j])
         if(bt[i]>bt[j])

        {
                  t=at[i];
                 at[i]=at[j];
                 at[j]=t;
                 q=bt[i];
                 bt[i]=bt[j];
                 bt[j]=q;
                 strcpy(c[i],pname[i]);
                 strcpy(pname[i],pname[j]);
                 strcpy(pname[j],c[i]);

         }
```

```
 }
 wt[0]=0;
 for(i=0;i<n;i++)

{
        wt[i+1]=wt[i]+bt[i];
        sum=sum+(wt[i]-at[i]);
       sbt=sbt+(wt[i+1]-at[i]);
       tt[i]=wt[i]+bt[i];
       ss=ss+bt[i];

 }
 printf("\n\n GANTT CHART");

 printf("\n\n---------------------------------------------------------------------- \n");
for(i=0;i<n;i++)
printf("|\t%s\t",pname[i]);
printf("\n----------------------------------------------------------------------\n");
for(i=0;i<n;i++)
printf("%d\t\t",wt[i]);
printf("%d\n",ss);

printf("\n-----------------------------------------------------------------------");
printf("\n\n Total WAITING TIME = %d ",sum);
printf("\n\n Total TURNAROUND TIME = %d ",sbt);
avgwt=(float)sum/n;
avgtt=(float)sbt/n;
printf("\n\n Average WAITING TIME = %f ",avgwt);
printf("\n\n Average TURNAROUND TIME = %f ",avgtt);

}
```

OUTPUT:

Enter the number of processes: 5
Enter the NAME, BURSTTIME, and ARRIVALTIME of the processes

NAME : p0
BURST TIME : 2
ARRIVAL TIME : 0

NAME : p1
BURST TIME : 4
ARRIVAL TIME : 0

NAME : p2
BURST TIME : 5
ARRIVAL TIME : 0

NAME : p3
BURST TIME : 6
ARRIVAL TIME : 0

NAME : p4
BURST TIME : 8
ARRIVAL TIME : 0

GANTT CHART

```
-----------------------------------------------------------------------
| p0 | p1 | p2 | p3 | p4
-----------------------------------------------------------------------
0   2     6      11   17   25
```

Total WAITING TIME = 36
Total TURNAROUND TIME = 61
Average WAITING TIME = 7.200000

RESULT:

Thus the program for implementing FCFS scheduling algorithm was written and successfully executed.

EX NO:5 (a) Write a program for implementing the round robin Scheduling algorithm

AIM:

To write a program for implement the round robin scheduling algorithm

ALGORITHM:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Get the value.
5. Set the time sharing system with preemption.
6. Define quantum is defined from 10 to 100ms.
7. Declare the queue as a circular.
8. Make the CPU scheduler goes around the ready queue allocating CPU to each

   process for the time interval specified.
9. Make the CPU scheduler picks the first process and sets time to interrupt after quantum expired dispatches the process.
10. If the process has burst less than the time quantum than the process releases the CPU.

PROGRAM:

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

	int pt[10][10],a[10][10],at[10],pname[10][10],i,j,n,k=0,q,sum=0;

	float avg;
	printf("\n\n Enter the number of processes : ");
	scanf("%d",&n);
	for(i=0;i<10;i++)
	{

		for(j=0;j<10;j++)
```

```
{
pt[i][j]=0;
a[i][j]=0;
}

}
for(i=0;i<n;i++)

{
j=0;
printf("\n\n Enter the process time for process %d : ",i+1);
scanf("%d",&pt[i][j]);

}
printf("\n\n Enter the time slice : ");

scanf("%d",&q);
printf("\n\n");
for(j=0;j<10;j++)
{

for(i=0;i<n;i++)

{
a[2*j][i]=k;
if((pt[i][j]<=q)&&(pt[i][j]!=0))

{

pt[i][j+1]=0;
printf(" %d P%d %d\n",k,i+1,k+pt[i][j]);
k+=pt[i][j];
a[2*j+1][i]=k;

}
else if(pt[i][j]!=0)

{
pt[i][j+1]=pt[i][j]-q;
```

```
        printf(" %d P%d %d\n",k,i+1,(k+q));
            k+=q;
            a[2*j+1][i]=k;

        }
        else

        {
            a[2*j][i]=0;
            a[2*j+1][i]=0;

        }
    }
}
for(i=0;i<n;i++)

sum+=a[0][i];
for(i=0;i<n;i++)
{

    for(j=1;j<10;j++)
    {
        if((a[j][i]!=0)&&(a[j+1][i]!=0)&&((j+1)%2==0))
        sum+=((a[j+1][i]-a[j][i]));
    }
}
avg=(float)sum/n;

printf("\n\n Average waiting time = %f msec",avg);
sum=avg=0;
for(j=0;j<n;j++)

{
     i=1;
     while(a[i][j]!=0)
     i+=1;
    sum+=a[i-1][j];

}
```

```
	avg=(float)sum/n;
	printf("\n\n Average turnaround time = %f msec\n\n",avg);
}
```

OUTPUT:

[root@localhost ~]# ./a.out
Enter the number of processes : 4

Enter the process time for process 1 : 8 Enter the process time for process 2 : 3 Enter the process time for process 3 : 6 Enter the process time for process 4 : 1

Enter the time slice : 2

0 P1 2
2 P2 4
4 P3 6
6 P4 7
7 P1 9
9 P2 10
10 P3 12
12 P1 14
14 P3 16
16 P1 18

Average waiting time = 8.250000 msec
Average turnaround time = 12.750000 msec

RESULT:

Thus the program for implementing RR scheduling algorithm was written and successfully executed.

EX NO:5 (b) Write a program for implementing the Priority scheduling algorithm

AIM:

To write a program for implement the priority scheduling algorithm.

ALGORITHM:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Get the priority for each process and value
5. start with the higher priority process from it's initial position let other process to be queue.
6. calculate the total number of burst time.
7. Display the values
8. Stop the process.

PROGRAM:

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

	float avgwt,avgtt;

	char pname[10][10],c[10][10];
	int wt[10],tt[10],bt[10],pt[10],t,q,i,n,sum=0,sbt=0,ttime,j,ss=10;
	printf("\n\n Enter the number of processes : ");
	scanf("%d",&n);
	printf("\n\n Enter the NAME and BURSTTIME ");
	for(i=0;i<n;i++)
	{

		printf("\n\n NAME : ");
		scanf("%s",&pname[i]);
		printf("\n\n BURSTTIME : ");
		scanf("%d",&bt[i]);

	}
```

```
printf("\n\n Enter the priorities of the processes ");
for(i=0;i<n;i++)

{
        printf("\n\n Priority of process%d : ",i+1);
       scanf("%d",&pt[i]);

}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
       if(pt[i]>pt[j])

       {
              t=pt[i];
              pt[i]=pt[j];
              pt[j]=t;
              q=bt[i];

             bt[i]=bt[j];
             bt[j]=q;
             strcpy(c[i],pname[i]);
             strcpy(pname[i],pname[j]);
             strcpy(pname[j],c[i]);

       }
}
wt[0]=0;
for(i=0;i<n;i++)

{
      wt[i+1]=wt[i]+bt[i];
      sum=sum+wt[i];
      sbt=sbt+wt[i+1];
      tt[i]=wt[i]+bt[i];
     ss=ss+bt[i];

}
printf("\n\n GANTT CHART");
```

```
printf("\n------------------------------------------------------------------\n");
for(i=0;i<n;i++)
printf("|\t%s\t",pname[i]);
printf("\n------------------------------------------------------------------\n");
for(i=0;i<n;i++)
printf("%d\t\t",wt[i]);
printf("%d\n",ss);
printf("\n------------------------------------------------------------------\n");
printf("\n\n Total WAITING TIME of the process = %d",sum);
printf("\n\n Total TURNAROUND TIME of the process = %d",sbt);
avgwt=(float)sum/n;
avgtt=(float)sbt/n;
printf("\n\n Average WAITING TIME of the process = %f",avgwt);
printf("\n\n Average TURNAROUND TIME of the process = %f",avgtt);

}
```

OUTPUT:

```
[root@localhost ~]# ./a.out
Enter the number of processes : 4
Enter the NAME and BURSTTIME

NAME : p1
BURSTTIME : 8
NAME : p2
BURSTTIME : 3
NAME : p3
BURSTTIME : 6
NAME : p4
BURSTTIME : 1

Enter the priorities of the processes
Priority of process1 : 1
Priority of process2 : 5
Priority of process3 : 2
Priority of process4 : 4
```

GANTT CHART

```
------------------------------------------------------
| p1 | p3 | p4 | p2
------------------------------------------------------
0   8     14    15  28
```

Total WAITING TIME of the process = 37
Total TURNAROUND TIME of the process = 55
Average WAITING TIME of the process = 9.250000
Average TURNAROUND TIME of the process = 13.750000

RESULT:

Thus the program for implementing FCFS scheduling algorithm was written and successfully executed.

EX NO: 6 . Developing Application using Inter Process communication (using shared memory, pipes or message queues)

AIM:

To write a program for developing Application using Inter Process communication with pipes.

ALGORITHM:

1. Start the program.
2. Read the input from parent process and perform in child process.
3. Write the date in parent process and read it in child process.
4. Fibonacci Series was performed in child process.
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/uio.h>
#include<sys/types.h>
main()
{

    int pid,pfd[2],n,a,b,c;

    if(pipe(pfd)==-1)

   {
        printf("\nError in pipe connection\n");
        exit(1);

   }
   pid=fork();
   if(pid>0)

  {
        printf("\nParent Process");\
        printf("\n\n\tFibonacci Series");
```

```
            printf("\nEnter the limit for the series:");
            scanf("%d",&n);
            close(pfd[0]);
            write(pfd[1],&n,sizeof(n));
            close(pfd[1]);
            exit(0);

      }
      else

    {
          close(pfd[1]);
          read(pfd[0],&n,sizeof(n));
          printf("\nChild Process");
          a=0;

          b=1;
          close(pfd[0]);
          printf("\nFibonacci Series is:");
          printf("\n\n%d\n%d",a,b);
          while(n>2)

        {
            c=a+b;

            printf("\n%d",c);
            a=b;
            b=c;
            n--;

        }

    }
}
```

OUTPUT:

```
[root@localhost ~]# ./a.out
Parent Process
Fibonacci Series
Enter the limit for the series:5
```

Child Process
Fibonacci Series is:
01123

RESULT:

Thus the program for developing Application using Inter Process communication with pipes is written and successfully executed.

EX NO: 7 Implement the Producer – Consumer problem using semaphores (using UNIX system calls).

AIM:

To write a program for Implement the Producer – Consumer problem using semaphores (using UNIX system calls).

ALGORITHM:

1. Start the process
2. Initialize buffer size
3. Consumer enters, before that producer buffer was not empty.
4. Producer enters, before check consumer consumes the buffer.
5. Stop the process.

PROGRAM:

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()

{
	int n;
	void producer();
	void consumer();
	int wait(int);
	int signal(int);
	printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
	while(1)
	{

		printf("\nENTER YOUR CHOICE\n");
```

```
                scanf("%d",&n);
                switch(n)
                {

                        case 1:

                        if((mutex==1)&&(empty!=0))
                        producer();

                        else
                        printf("BUFFER IS FULL");
                        break;

                        case 2:
                        if((mutex==1)&&(full!=0))
                        consumer();

                        else
                        printf("BUFFER IS EMPTY");
                        break;

                        case 3:
                        exit(0);
                        break;

                }
        }
}
int wait(int s)
{
        return(--s);
}
int signal(int s)
{
        return(++s);
}
void producer()
```

```
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nproducer produces the item%d",x);
        mutex=signal(mutex);

}
void consumer()

{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\n consumer consumes item%d",x);
        x--;
        mutex=signal(mutex);

}
```

OUTPUT:

```
[root@localhost ~]# ./a.out
1.PRODUCER
2.CONSUMER
3.EXIT

ENTER YOUR CHOICE
1producer produces the item1
ENTER YOUR CHOICE
1producer produces the item2
ENTER YOUR CHOICE
2consumer consumes item2
ENTER YOUR CHOICE
2consumer consumes item1
ENTER YOUR CHOICE
2BUFFER IS EMPTY
```

ENTER YOUR CHOICE
3

RESULT:

Thus the program for Implement the Producer – Consumer problem using semaphores (using UNIX system calls) was written and successfully executed.

EX NO: 8   Implement first fit algorithm for memory management

AIM:

To write a program to implement first fit algorithm for memory management.

ALGORITHM

1. Start the process.
2. Declare the size.
3. Get the number of processes to be inserted.
4. Allocate the first hole that is big enough searching.
5. Start at the beginning of the set of holes.
6. If not start at the hole that is sharing the pervious first fit search end.
7. If large enough then stop searching in the procedure.
8. Display the values.
9. Stop the process.

PROGRAM:

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
struct allocate
{

	int pid;

	int st_add;
	int end_add;
	struct allocate *next;
};

struct free_list
{
	int st_add;
```

```
        int end_add;
        struct free_list *next;
};

struct process
{
        int pid;
        int size;
};

struct process pro[10];
struct free_list *flist=NULL;
struct allocate *alot=NULL;

void display_alot(struct allocate *temp_a)
{
        printf("\n\n allocated list:");

        printf("\n================");
        while(temp_a!=NULL)
        {

                printf("\n process:%d st_add:%d end_add:%d ",temp_a->pid,

                temp_a- >st_add,temp_a->end_add);
                temp_a=temp_a->next;

        }

}

void display_free(struct free_list *temp_f)

{
        printf("\n\n free list:");

        printf("\n=================");
        while(temp_f!=NULL)
        {
```

```
			printf("\n st_add:%d end_add:%d",temp_f->st_add,

			temp_f->end_add);

			temp_f=temp_f->next;
		}
}
void insert(int p)
{
		struct free_list *temp_f;

		struct allocate *temp_a,*pre_a;
		int i,n;
		do
		{

			srand((unsigned int)time((time_t*)NULL));

			n=rand()%5;
		}

		while(n==0);
		printf("\n\n no. of process:%d",n);
		for(i=0;i<n;i++)
		{

			pro[i].pid=i+p;
			do
			{

				pro[i].size=rand()%300;

			}
			while(pro[i].size==0);
		}
		for(i=0;i<n;i++)
		{
			printf("\n\n process to be inserted:%d size:%d",pro[i].pid,pro[i].size);
```

```
temp_f=flist;
temp_a=alot;
while(temp_f!=NULL && temp_f->end_add-temp_f->st_add

<   pro[i].size)
{

        temp_f=temp_f->next;

}
if(temp_f!=NULL)
{
        pre_a=(struct allocate*)malloc(sizeof(struct allocate));

        pre_a->st_add=temp_f->st_add;
        pre_a->end_add=temp_f->st_add=temp_f->st_add+pro[i].size;
        pre_a->pid=pro[i].pid;
        if(temp_a==NULL)
        {

                alot=pre_a;

                pre_a->next=NULL;

        }

        else
        {

                        while(temp_a->next!=NULL)
                        {

                                temp_a=temp_a->next;
                        }

                temp_a->next=pre_a;

                pre_a->next=NULL;
        }
}
```

```
		else

		printf("\n there is not enough space");
		display_alot(alot);
		display_free(flist);
		getch();
	}

}

void main()

{
	int no,n,i,nod,ndpid;

	struct process pro[10];
	struct free_list *temp_f,*free_alot,*pre_f;
	struct allocate *temp_a,*pre_a;
	clrscr();
	alot=NULL;
	flist=(struct free_list*)malloc(sizeof(struct free_list));
	flist->st_add=0;
	flist->end_add=1024;
	flist->next=NULL;
	insert(0);
	do
	{

		srand((unsigned int)time((time_t*)NULL));

		nod=rand()%2;
	}

	while(nod==0);
	printf("\n\n no.of process deletion:%d",nod);
	for(i=0;i<nod;i++)
	{

		printf("\n\n\n process to be deleted:");
```

```
 scanf("%d",&ndpid);
 temp_a=alot;
 temp_f=flist;
 while(temp_a!=NULL && temp_a->pid!=ndpid)
 {

        pre_a=temp_a;

        temp_a=temp_a->next;
}
 if(temp_a!=NULL)
 {
        if(alot==temp_a)

        alot=temp_a->next;
        else
        pre_a->next=temp_a->next;
        pre_f=NULL;
        while(temp_f!=NULL && temp_f->st_add < temp_a->st_add)
        {

                pre_f=temp_f;

                temp_f=temp_f->next;
        }
        if(pre_f!=NULL && pre_f->end_add==temp_a->st_add)

        pre_f->end_add=temp_a->end_add;
        else if(pre_f!=NULL&&temp_f!=NULL&&

        temp_f- >st_add==temp_a->end_add)
        temp_f->st_add=temp_a->st_add;
        else
        {

                free_alot=(struct free_list*)malloc(sizeof

                (struct   free_list));
```

```
                        free_alot->st_add=temp_a->st_add;
                        free_alot->end_add=temp_a->end_add;
                        if(pre_f!=NULL)
                        pre_f->next=free_alot;
                        free_alot->next=temp_f;
                        if(flist==temp_f)

                        {
                                flist=free_alot;
                        }
                }
                free(temp_a);
        }
        else printf("\n process not in memory");

        temp_f=flist;
        while(temp_f!=NULL)
        {

                if(temp_f->end_add==temp_f->next->st_add)

                {
                        temp_f->end_add=temp_f->next->end_add;
                        temp_f->next=temp_f->next->next;

                }

                temp_f=temp_f->next;
        }

        display_alot(alot);
        display_free(flist);

        getch();
    }
    insert(10);
}
```

OUTPUT:

```
no. of process:3
process to be inserted:0 size:120
allocated list:
================
process:0 st_add:0 end_add:120
free list:

=================
st_add:120 end_add:1024
process to be inserted:1 size:185

allocated list:

================
process:0 st_add:0 end_add:120
process:1 st_add:120 end_add:305

free list:
=================

st_add:305 end_add:1024
process to be inserted:2 size:246
allocated list:

================
process:0 st_add:0 end_add:120
process:1 st_add:120 end_add:305
process:2 st_add:305 end_add:551

free list:
=================
st_add:551 end_add:1024
no.of process deletion:1
process to be deleted:0
allocated list:
```

```
================
 process:1 st_add:120 end_add:305
 process:2 st_add:305 end_add:551

 free list:
 =================
 st_add:0 end_add:120

 st_add:551 end_add:1024
 no. of process:3
 process to be inserted:10 size:195
 allocated list:

================
 process:1 st_add:120 end_add:305
 process:2 st_add:305 end_add:551
 process:10 st_add:551 end_add:746

 free list:

=================
 st_add:0 end_add:120
 st_add:746 end_add:1024

 process to be inserted:11 size:96
 allocated list:

================
 process:1 st_add:120 end_add:305
 process:2 st_add:305 end_add:551
 process:10 st_add:551 end_add:746
 process:11 st_add:0 end_add:96

 free list:

=================
 st_add:96 end_add:120
 st_add:746 end_add:1024
```

```
process to be inserted:12 size:148
allocated list:

================
process:1 st_add:120 end_add:305
process:2 st_add:305 end_add:551
process:10 st_add:551 end_add:746
process:11 st_add:0 end_add:96
process:12 st_add:746 end_add:894

free list:

=================
st_add:96 end_add:120
st_add:894 end_add:1024
```

<u>RESULT</u>:

Thus the program for Implement first fit algorithm for memory management is written and successfully executed.

EX NO: 9 (a) Implement Best fit algorithm for memory management

AIM :

To write a program for Implement best fit memory management

ALGORITHM:

1.Start the program.
2. Declare the size.
3. Get the number of processes to be inserted.
4. Allocate the best hole that is small enough searching.
5. Start at the best of the set of holes.
6. If not start at the hole that is sharing the pervious best fit search end.
7. Compare the hole in the list.
8. If small enough then stop searching in the procedure.
9. Display the values.
10. Stop the program.

PROGRAM:

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
struct allocate
{

	int pid;

	int st_add;
	int end_add;
	struct allocate *next;
};

struct free_list
{
	int st_add;
```

```
	int end_add;
	struct free_list *next;
};

struct process
{
	int pid;
	int size;
};

struct process pro[10];
struct free_list *flist=NULL;
struct allocate *alot=NULL;

void display_alot(struct allocate *temp_a)
{

	printf("\n\n allocated list:");
	printf("\n================");
	while(temp_a!=NULL)
	{

		printf("\n process:%d st_add:%d end_add:%d ",temp_a->pid,

		temp_a->st_add,temp_a->end_add);
		temp_a=temp_a->next;
	}

}

void display_free(struct free_list *temp_f)

{
	printf("\n\n free list:");

	printf("\n=================");
	while(temp_f!=NULL)
	{
```

```
            printf("\n st_add:%d end_add:%d",temp_f->st_add,

            temp_f->end_add);

            temp_f=temp_f->next;
        }
}
void insert(int p)
{
        struct free_list *temp_f;

        struct allocate *temp_a,*pre_a;
        int i,n;
        do
        {

            srand((unsigned int)time((time_t*)NULL));

            n=rand()%10;
        }
        while(n==0);
        printf("\n\n no. of process:%d",n);
        for(i=0;i<n;i++)
        {
            pro[i].pid=i+p;
            do
            {
                pro[i].size=rand()%550;
            }
            while(pro[i].size==0);
        }
        for(i=0;i<n;i++)
        {
            printf("\n\n process to be inserted:%d size:%d",pro[i].pid,pro[i].size);

            temp_f=flist;
            temp_a=alot;
            while(temp_f!=NULL && temp_f->end_add-temp_f->
```

```
st_add < pro[i].size)
{

        temp_f=temp_f->next;

}
if(temp_f!=NULL)
{
        pre_a=(struct allocate*)malloc(sizeof(struct allocate));

        pre_a->st_add=temp_f->st_add;
        pre_a->end_add=temp_f->st_add=temp_f->st_add+pro[i].size;
        pre_a->pid=pro[i].pid;
        if(temp_a==NULL)
        {

                alot=pre_a;

                pre_a->next=NULL;

        }

        else
        {

                while(temp_a->next!=NULL)
                {

                        temp_a=temp_a->next;

                }
                temp_a->next=pre_a;
                pre_a->next=NULL;
        }
}
else

printf("\n there is not enough space");
display_alot(alot);
```

```
        display_free(flist);
        getch();
    }

}

void bestfit(int p)

{
    struct free_list *temp_f,*enough_hole;

    struct allocate *temp_a,*pre_a;
    int i,n,rem_space;
    do
    {

        srand((unsigned int)time((time_t*)NULL));

        n=rand()%10;
    }

    while(n==0);
    printf("\n no of processes:%d",n);
    for(i=0;i<n;i++)
    {

        pro[i].pid=i+p;

        do

        {

            pro[i].size=rand()%200;
        }

        while(pro[i].size==0);
    }

    for(i=0;i<n;i++)
    {
```

```
printf("\n process to be inserted:%d size:%d",pro[i].pid,pro[i].size);

temp_f=flist;
temp_a=alot;
enough_hole=NULL;
rem_space=1024;
while(temp_f!=NULL)
{

    if(temp_f->end_add - temp_f->st_add >= pro[i].size)

    {

        if(temp_f->end_add - temp_f->

        st_add -     pro[i].size<rem_space)
        {

            rem_space=temp_f->end_add - temp_f->

            st_add - pro[i].size;
            enough_hole=temp_f;

        }
    }
    temp_f=temp_f->next;
}
if(enough_hole!=NULL)
{
    pre_a=(struct allocate*)malloc(sizeof(struct allocate));

    pre_a->st_add=enough_hole->st_add;
    pre_a->end_add=enough_hole->st_add=

    enough_hole->st_add + pro[i].size;
    pre_a->pid=pro[i].pid;
    if(temp_a==NULL)
    {
```

```
                    alot=pre_a;

                    pre_a->next=NULL;
                }
                else
                {
                    while(temp_1->next!=NULL)

                    {

                        temp_a=temp_a->next;
                    }

                    temp_a->next=pre_a;
                    pre_a->next=NULL;

                }
            }
            else

            printf("\n there is not enough space");
            display_alot(alot);
            display_free(flist);
            getch();
        }

}

void main()

{
        int no,n,i,nod,ndpid;

        struct process pro[10];
        struct free_list *temp_f,*free_alot,*pre_f;
        struct allocate *temp_a,*pre_a;
        clrscr();
        alot=NULL;
        flist=(struct free_list*)malloc(sizeof(struct free_list));
```

```
flist->st_add=0;
flist->end_add=1024;
flist->next=NULL;
insert(0);
do
{

    srand((unsigned int)time((time_t*)NULL));

    nod=rand()%5;
}
while(nod==0);

printf("\n\n no.of process deletion:%d",nod);
for(i=0;i<nod;i++)
{

    printf("\n\n\n process to be deleted:");

    scanf("%d",&ndpid);
    temp_a=alot;
    temp_f=flist;
    while(temp_a!=NULL && temp_a->pid!=ndpid)
    {

        pre_a=temp_a;

        temp_a=temp_a->next;
    }
    if(temp_a!=NULL)
    {
        if(alot==temp_a)

        alot=temp_a->next;
        else
        pre_a->next=temp_a->next;
        pre_f=NULL;
        while(temp_f!=NULL && temp_f->st_add < temp_a->st_add)
        {
```

```
            pre_f=temp_f;

            temp_f=temp_f->next;
        }
        if(pre_f!=NULL && pre_f->end_add==temp_a->st_add)

        pre_f->end_add=temp_a->end_add;
        else if(pre_f!=NULL&&temp_f!=NULL&&

        temp_f->st_add==temp_a->end_add)
        temp_f->st_add=temp_a->st_add;
        else
        {

            free_alot=(struct free_list*)malloc(sizeof

            (struct free_list));

            free_alot->st_add=temp_a->st_add;
            free_alot->end_add=temp_a->end_add;

            if(pre_f!=NULL)
            pre_f->next=free_alot;
            free_alot->next=temp_f;
            if(flist==temp_f)
            {

                flist=free_alot;

            }
        }
        free(temp_a);
    }
    else printf("\n process not in memory");

    temp_f=flist;
    while(temp_f!=NULL)
    {
```

```
                    if(temp_f->end_add==temp_f->next->st_add)

                    {
                            temp_f->end_add=temp_f->next->end_add;
                            temp_f->next=temp_f->next->next;

                    }

                    temp_f=temp_f->next;
            }

            display_alot(alot);
            display_free(flist);

            getch();
        }
        bestfit(10);
}
```

OUTPUT:

```
no. of process:7
process to be inserted:0 size:351
allocated list:

================
process:0 st_add:0 end_add:351
free list:
=================

st_add:351 end_add:1024
process to be inserted:1 size:466
allocated list:

================
process:0 st_add:0 end_add:351
process:1 st_add:351 end_add:817

free list:
```

```
=================
st_add:817 end_add:1024
process to be inserted:2 size:337
there is not enough space
allocated list:

================
process:0 st_add:0 end_add:351
process:1 st_add:351 end_add:817

free list:
=================
st_add:817 end_add:1024
process to be inserted:3 size:410
there is not enough space
allocated list:

================
process:0 st_add:0 end_add:351
process:1 st_add:351 end_add:817

free list:
=================
st_add:817 end_add:1024
process to be inserted:4 size:542
there is not enough space
allocated list:

================
process:0 st_add:0 end_add:351
process:1 st_add:351 end_add:817

free list:
=================
st_add:817 end_add:1024
process to be inserted:5 size:547
there is not enough space
allocated list:
```

```
================
 process:0 st_add:0 end_add:351
 process:1 st_add:351 end_add:817

 free list:
 =================

 st_add:817 end_add:1024
 process to be inserted:6 size:89
 allocated list:

================
 process:0 st_add:0 end_add:351
 process:1 st_add:351 end_add:817
 process:6 st_add:817 end_add:906

 free list:
 =================

 st_add:906 end_add:1024
 no.of process deletion:4
 process to be deleted:0
 allocated list:

================
 process:1 st_add:351 end_add:817
 process:6 st_add:817 end_add:906

 free list:
 =================
 st_add:0 end_add:351
 st_add:906 end_add:1024
 process to be deleted:2
 process not in memory
 allocated list:

================
 process:1 st_add:351 end_add:817
 process:6 st_add:817 end_add:906
```

free list:

=================
st_add:0 end_add:351
st_add:906 end_add:1024

process to be deleted:3
process not in memory
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906

free list:

=================
st_add:0 end_add:351
st_add:906 end_add:1024

process to be deleted:4
process not in memory
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906

free list:

=================
st_add:0 end_add:351
st_add:906 end_add:1024

no of processes:9
process to be inserted:10 size:155
allocated list:
================
process:1 st_add:351 end_add:817

process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
free list:

=================
st_add:155 end_add:351
st_add:906 end_add:1024

process to be inserted:11 size:43
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155

process:11 st_add:906 end_add:949
free list:

=================
st_add:155 end_add:351
st_add:949 end_add:1024

process to be inserted:12 size:188
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343

free list:

=================
st_add:343 end_add:351
st_add:949 end_add:1024

process to be inserted:13 size:7
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350

free list:

=================
st_add:350 end_add:351
st_add:949 end_add:1024

process to be inserted:14 size:160
there is not enough space
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350

free list:

=================
st_add:350 end_add:351
st_add:949 end_add:1024
process to be inserted:15 size:100
there is not enough space

allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350

free list:

=================
st_add:350 end_add:351
st_add:949 end_add:1024

process to be inserted:16 size:198
there is not enough space
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906

process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350

free list:

=================
st_add:350 end_add:351
st_add:949 end_add:1024

process to be inserted:17 size:51
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906

```
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350
process:17 st_add:949 end_add:1000

free list:

=================
st_add:350 end_add:351
st_add:1000 end_add:1024

process to be inserted:18 size:42
there is not enough space
allocated list:

================
process:1 st_add:351 end_add:817
process:6 st_add:817 end_add:906
process:10 st_add:0 end_add:155
process:11 st_add:906 end_add:949
process:12 st_add:155 end_add:343
process:13 st_add:343 end_add:350
process:17 st_add:949 end_add:1000

free list:

=================
st_add:350 end_add:351
st_add:1000 end_add:1024
```

RESULT:

Thus the program for Implement best fit algorithm for memory management is written and successfully executed.

EX NO: 9 (b) Implement worst fit algorithm for memory management

AIM :

To write a program for Implement worst fit memory management.

ALGORITHM:

1. Start the program.
2. Declare the size.
3. Get the number of processes to be inserted.
4. Allocate the first hole that is small enough searching.
5. If small enough then stop searching in the procedure.
6. Display the values.
7. Stop the program.

PROGRAM:

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
struct allocate
{

        int pid; int st_add; int end_add; struct allocate *next;
};
 struct free_list
 {
        int st_add; int end_add; struct free_list *next;
};
 struct process
 {
        int pid; int size;
 };
```

```
struct process pro[10];
struct free_list *flist=NULL;
struct allocate *alot=NULL;

void display_alot(struct allocate *temp_a)
{
	printf("\n\n allocated list:");

	printf("\n================");
	while(temp_a!=NULL)
	{

		printf("\n process:%d st_add:%d end_add:%d ",temp_a->pid,

		temp_a->st_add,temp_a->end_add);
		temp_a=temp_a->next;
	}

}

void display_free(struct free_list *temp_f)

{

	printf("\n\n free list:");
	printf("\n=================");
	while(temp_f!=NULL)
	{

		printf("\n st_add:%d end_add:%d",temp_f->st_add,temp_f->end_add);

		temp_f=temp_f->next;
	}
}
void insert(int p)
{
	struct free_list *temp_f;
```

```
struct allocate *temp_a,*pre_a;
int i,n;
do
{

	srand((unsigned int)time((time_t*)NULL));

	n=rand()%10;
}
while(n==0);

printf("\n\n no. of process:%d",n);
for(i=0;i<n;i++)
{

	pro[i].pid=i+p;

	do
	{
		pro[i].size=rand()%550;
	}
	while(pro[i].size==0);

}

for(i=0;i<n;i++)
{

	printf("\n\n process to be inserted:%d size:%d",pro[i].pid,pro[i].size);
	temp_f=flist;

	temp_a=alot;
	while(temp_f!=NULL && temp_f->end_add-temp_f->st_add < pro[i].size)
	{

		temp_f=temp_f->next;

	}
	if(temp_f!=NULL)
```

```
        {
            pre_a=(struct allocate*)malloc(sizeof(struct allocate));

            pre_a->st_add=temp_f->st_add;
            pre_a->end_add=temp_f->st_add=temp_f->st_add+pro[i].size;
            pre_a->pid=pro[i].pid;
            if(temp_a==NULL)
            {

                alot=pre_a;

                pre_a->next=NULL;

            }

            else
            {

                while(temp_a->next!=NULL)
                {

                    temp_a=temp_a->next;
                }

                temp_a->next=pre_a;
                pre_a->next=NULL;

            }
        }
        else

        printf("\n there is not enough space");
        display_alot(alot);
        display_free(flist);
        getch();
    }

}

void worstfit(int p)
```

```
{
        struct free_list *temp_f,*big_hole;

        struct allocate *temp_a,*pre_a;
        int i,n;
        do
        {

        srand((unsigned int)time((time_t*)NULL));

        n=rand()%10;
        }

        while(n==0);
        printf("\n no.of process:%d",n);
        for(i=0;i<n;i++)
        {

                pro[i].pid=i+p;

                do

                {

                        pro[i].size=rand()%250;
                }

                while(pro[i].size==0);
        }

        for(i=0;i<n;i++)
        {

                printf("\n process to be inserted: %d size :%d",pro[i].pid,pro[i].size);
                temp_f=flist;

                temp_a=alot;
                big_hole=NULL;
                while(temp_f!=NULL)
                {
```

```
        if(temp_f->end_add-temp_f->st_add>=pro[i].size)

        {
                if(big_hole==NULL)
                big_hole=temp_f;
                else if(temp_f->end_add - temp_f->st_add > big_hole
                ->  end_add - big_hole-> st_add)
                big_hole=temp_f;

        }

        temp_f=temp_f->next;
}

if(big_hole!= NULL)
{

        pre_a=(struct allocate*) malloc (sizeof(struct allocate));

        pre_a->st_add=big_hole->st_add;
        pre_a->end_add=big_hole->st_add=big_hole->st_add + pro[i].size;
        pre_a->pid=pro[i].pid;
        if(temp_a==NULL)
        {

                alot=pre_a;

                pre_a->next=NULL;
        }
        else
        {
                while(temp_a->next!=NULL)

                temp_a=temp_a->next;
                temp_a->next= pre_a;
                pre_a->next=NULL;
        }

}
```

```c
        else

        printf("\n there is not enough space");
        display_alot(alot);
        display_free(flist);
        getch();
    }

}

void main()

{

    int no,n,i,nod,ndpid;
    struct process pro[10];
    struct free_list *temp_f,*free_alot,*pre_f;
    struct allocate *temp_a,*pre_a;
    clrscr();
    alot=NULL;
    flist=(struct free_list*)malloc(sizeof(struct free_list));
    flist->st_add=0;
    flist->end_add=1024;
    flist->next=NULL;
    insert(0);
    do
    {

        srand((unsigned int)time((time_t*)NULL));

        nod=rand()%5;
    }

    while(nod==0);
    printf("\n\n no.of process deletion:%d",nod);
    for(i=0;i<nod;i++)
    {

        printf("\n\n\n process to be deleted:");
```

```
scanf("%d",&ndpid);
temp_a=alot;
temp_f=flist;
while(temp_a!=NULL && temp_a->pid!=ndpid)
{

        pre_a=temp_a;

        temp_a=temp_a->next;

}

if(temp_a!=NULL)
{

        if(alot==temp_a)
        alot=temp_a->next;

        else

        pre_a->next=temp_a->next;
        pre_f=NULL;
        while(temp_f!=NULL && temp_f->st_add < temp_a->st_add)
        {

                pre_f=temp_f;

                temp_f=temp_f->next;
        }
        if(pre_f!=NULL && pre_f->end_add==temp_a->st_add)

        pre_f->end_add=temp_a->end_add;
        else if(pre_f!=NULL&&temp_f!=NULL&&

        temp_f->st_add==temp_a->end_add)
        temp_f->st_add=temp_a->st_add;
        else
        {

                free_alot=(struct free_list*)malloc(sizeof(struct free_list));
```

```
                    free_alot->st_add=temp_a->st_add;
                    free_alot->end_add=temp_a->end_add;
                    if(pre_f!=NULL)
                    pre_f->next=free_alot;
                    free_alot->next=temp_f;
                    if(flist==temp_f)
                    {

                        flist=free_alot;

                    }
                }
                free(temp_a);
            }
            else printf("\n process not in memory");

            temp_f=flist;
            while(temp_f!=NULL)
            {

                if(temp_f->end_add==temp_f->next->st_add)

                {
                    temp_f->end_add=temp_f->next->end_add;
                    temp_f->next=temp_f->next->next;

                }

                temp_f=temp_f->next;
            }

            display_alot(alot);
            display_free(flist);

            getch();
        }
        worstfit(10);
}
```

OUTPUT:

```
no. of process:9
 process to be inserted:0 size:21
 allocated list:

 ================
 process:0 st_add:0 end_add:21
 free list:
 =================

 st_add:21 end_add:1024
 process to be inserted:1 size:469
 allocated list:

================
 process:0 st_add:0 end_add:21
 process:1 st_add:21 end_add:490

 free list:
 =================

 st_add:490 end_add:1024
 process to be inserted:2 size:30
 allocated list:

================
 process:0 st_add:0 end_add:21
 process:1 st_add:21 end_add:490
 process:2 st_add:490 end_add:520

 free list:
 =================

 st_add:520 end_add:1024
 process to be inserted:3 size:74
 allocated list:
```

```
================
 process:0 st_add:0 end_add:21
 process:1 st_add:21 end_add:490
 process:2 st_add:490 end_add:520
 process:3 st_add:520 end_add:594

 free list:
 =================

 st_add:594 end_add:1024
 process to be inserted:4 size:182
 allocated list:

================
 process:0 st_add:0 end_add:21
 process:1 st_add:21 end_add:490
 process:2 st_add:490 end_add:520
 process:3 st_add:520 end_add:594
 process:4 st_add:594 end_add:776

 free list:
 =================

 st_add:776 end_add:1024
 process to be inserted:5 size:100
 allocated list:

================
 process:0 st_add:0 end_add:21
 process:1 st_add:21 end_add:490
 process:2 st_add:490 end_add:520
 process:3 st_add:520 end_add:594
 process:4 st_add:594 end_add:776
 process:5 st_add:776 end_add:876

 free list:
 =================
 st_add:876 end_add:1024
 process to be inserted:6 size:183
```

there is not enough space
allocated list:

=================
process:0 st_add:0 end_add:21
process:1 st_add:21 end_add:490
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876

free list:
==================
st_add:876 end_add:1024
process to be inserted:7 size:411
there is not enough space
allocated list:

=================
process:0 st_add:0 end_add:21
process:1 st_add:21 end_add:490
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876

free list:
==================
st_add:876 end_add:1024
process to be inserted:8 size:292
there is not enough space
allocated list:

=================
process:0 st_add:0 end_add:21
process:1 st_add:21 end_add:490
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594

```
 process:4 st_add:594 end_add:776
 process:5 st_add:776 end_add:876

 free list:
 =================

 st_add:876 end_add:1024
 no.of process deletion:2
 process to be deleted:0
 allocated list:

================
 process:1 st_add:21 end_add:490
 process:2 st_add:490 end_add:520
 process:3 st_add:520 end_add:594
 process:4 st_add:594 end_add:776
 process:5 st_add:776 end_add:876

 free list:

=================
 st_add:0 end_add:21
 st_add:876 end_add:1024

 process to be deleted:1
 allocated list:

================
 process:2 st_add:490 end_add:520
 process:3 st_add:520 end_add:594
 process:4 st_add:594 end_add:776
 process:5 st_add:776 end_add:876

 free list:

=================
 st_add:0 end_add:490
 st_add:876 end_add:1024
```

no.of process:6
process to be inserted: 10 size :105
allocated list:

================
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105

free list:

=================
st_add:105 end_add:490
st_add:876 end_add:1024

process to be inserted: 11 size :93
allocated list:

================
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105
process:11 st_add:105 end_add:198

free list:

=================
st_add:198 end_add:490
st_add:876 end_add:1024

process to be inserted: 12 size :60
allocated list:

================
process:2 st_add:490 end_add:520

process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105
process:11 st_add:105 end_add:198
process:12 st_add:198 end_add:258

free list:

=================
st_add:258 end_add:490
st_add:876 end_add:1024

process to be inserted: 13 size :103
allocated list:

================
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105
process:11 st_add:105 end_add:198
process:12 st_add:198 end_add:258
process:13 st_add:258 end_add:361

free list:

=================
st_add:361 end_add:490
st_add:876 end_add:1024

process to be inserted: 14 size :72
allocated list:

================
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776

process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105
process:11 st_add:105 end_add:198

process:12 st_add:198 end_add:258 process:13 st_add:258 end_add:361 process:14 st_add:876 end_add:948

free list:

=================
st_add:361 end_add:490
st_add:948 end_add:1024

process to be inserted: 15 size :17
allocated list:

================
process:2 st_add:490 end_add:520
process:3 st_add:520 end_add:594
process:4 st_add:594 end_add:776
process:5 st_add:776 end_add:876
process:10 st_add:0 end_add:105
process:11 st_add:105 end_add:198
process:12 st_add:198 end_add:258
process:13 st_add:258 end_add:361
process:14 st_add:876 end_add:948
process:15 st_add:361 end_add:378

free list:

=================
st_add:378 end_add:490
st_add:948 end_add:1024

RESULT:

Thus the program for Implement best fit algorithm for memory management is written and successfully executed.

EX NO: 10 Implement Contiguous file allocation technique.

AIM :

To write a program for implement the contiguous file allocation technique.

ALGORITHM:

1. Start the program
2. Declare the size
3. Get the number of files to be inserted.
4. Get the capacity of each file.
5. Get the starting address.
6. The file is allocated in memory
7. The file is not allocated if the contiguous memory is not available.
8. Display the result
9. Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
	int nf, fc[20], mb[100], i, j, k, fb[100], fs[20], mc=0;
	clrscr();
	printf("\nEnter the number of files: ");
	scanf("%d",&nf);
	for(i=0;i<nf;i++)
	{
		printf("\nEnter the capacity of file %d: ",i+1);
		scanf("%d",&fc[i]);
		printf("\nEnter the starting address of file %d: ",i+1);
		scanf("%d",&fs[i]);
	}
	printf("\n---CONTIGUOUS FILE ALLOCATION---\n");
	for(i=0;i<100;i++)
	fb[i]=1;
	for(i=0;i<nf;i++)
	{
		j=fs[i];
```

```
			{
				if(fb[j]==1)
				{
					for(k=j;k<(j+fc[i]);k++)
					{
						if(fb[k]==1)
						mc++;
					}
					if(mc==fc[i])
					{
						for(k=fs[i];k<(fs[i]+fc[i]);k++)
						{
							fb[k]=0;
						}
						printf("\nFile %d allocated in memory %d to
					%d...",i+1,fs[i],fs[i]+fc[i]-1);
					}
				}
				else
				printf("\nFile %d not allocated since %d contiguous memory not
			available from %d...",i+1,fc[i],fs[i]);
			}
			mc=0;
		}
}
```

OUTPUT:

```
Enter the number of files: 3
Enter the capacity of file 1: 21
Enter the starting address of file 1: 21
Enter the capacity of file 2: 24
Enter the starting address of file 2: 36
Enter the capacity of file 3: 2
Enter the starting address of file 3: 54
---CONTIGUOUS FILE ALLOCATION---
File 1 allocated in memory 21 to 41...
File 2 not allocated since 24 contiguous memory not available from 36...
File 3 allocated in memory 54 to 55...
```

RESULT:

Thus the program for contiguous file allocation technique was written and successfully executed.