GRACE COLLEGE OF ENGINEERING

Mullakadu -628005

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CS8461 --OPERATING SYSTEMS LABORATORY

LAB MANUAL

ODD SEMESTER

YEAR/SEM: II/IV

LIST OF EXPERIMENTS

1. Basics of UNIX commands
2. Write programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir
3. Write C programs to simulate UNIX commands like cp, ls, grep, etc.
4. Shell Programming
5. Write C programs to implement the various CPU Scheduling Algorithms
6. Implementation of Semaphores
7. Implementation of Shared memory and IPC
8. Bankers Algorithm for Deadlock Avoidance
9. Implementation of Deadlock Detection Algorithm
10. Write C program to implement Threading & Synchronization Applications
11. Implementation of Paging Technique of Memory Management
12. Implementation of the following Page Replacement Algorithms a) FIFO b) LRU c) LFU
13. Implementation of the following File Allocation Strategies
a) Indexed b) Linked

EX.NO:7) Implementation of Shared memory and IPC

AIM:-

To write a C Program to implement shared memory and inter process communication.

Algorithm:

Step 1: Start the program
Step 2: Create two files one as writer.c and other as reader.c
Step 3: Obtain the required data through char datatypes.
Step 4: Now run the writer.c file.
Step 5: Enter the Write data.
Step 6: Now run the reader.c file.
Step 7: It will print the Write data which was entered while executing the writer.c file.
Step 8: Stop the execution.

Program Coding:-

writer.c:

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
   // ftok to generate unique key
   key_t key = ftok("shmfile",65);

   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);

   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);

   cout<<"Write Data : ";
   gets(str);
```

```
    printf("Data written in memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    return 0;
}
```

Reader.c:

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    printf("Data read from memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

Output:

Running writer.c:
Write Data: IPC through Shared Memory
Data written in memory: IPC through Shared Memory
Running reader.c:
Data read from memory: IPC through Shared Memory

Result:-

Thus the program was executed successfully.

EX.NO: 8) Bankers Algorithm for Deadlock Avoidance

Aim:-

To write a C program to implement bankers algorithm for dead lock avoidance.

Algorithm:-

Step 1: Start the Program.

Step 2: Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize:Work=Available
Finish[i] = false; for i=1,2,3,4,…..n.

Step3: Find an i such that both a) Finish[i] = false & b) Needi<=Work
If no such i exists goto Step(4).

Step 3: Work = Work +Allocation[i]
Finish[i] = true goto Step(3).

Step 4: if Finish[i] = trye for all I then the system is in a safe state.

Step 4: Stop the execution.

Program Coding:-

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

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

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

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

   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++) {
      f[k] = 0;
   }
   int need[n][m];
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   }
   int y = 0;
   for (k = 0; k < 5; k++) {
```

```
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                         break;
                    }
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);

    return (0);

}
```

Output:-

Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2

Result:-

Thus the program executed successfully.

EX.NO: 9) Implementation of Deadlock Detection Algorithm

Aim:-

To write a C program to implement Deadlock Detection Algorithm.

Algorithm:-

Step 1: Start the program
Step 2: Mark each process that has a row in the allocation matrix of all zeros.
Step 3: Initialize a temporary vector W to equal the Available vector.
Step 4: Find an indexi such that processi is currently unmarked and the ith row of Q is less than or equal to W.
Step 5: If such a row is found, mark processi and add the corresponding row of the allocation matrix to W . That is, setWk = Wk + Aik, for 1 … k … m . Return to step 4.
Step 5: Terminate the Program.

Program Coding:-

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
```

```
printf("\nEnter the request matrix:");

for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];

}
}

//marking processes with zero allocation

for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
      if(alloc[i][j]==0)
        count++;
      else
        break;
    }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail
```

```
for(j=0;j<nr;j++)
   w[j]=avail[j];

//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
  for(j=0;j<nr;j++)
   {
     if(request[i][j]<=w[j])
       canbeprocessed=1;
     else
        {
        canbeprocessed=0;
        break;
         }
    }
if(canbeprocessed)
{
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}

//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;


if(deadlock)
printf("\n Deadlock detected");
else
```

```
printf("\n No Deadlock possible");
}
```

Output:-

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix: 0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix: 1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

Deadlock Detected.
```

Result:

Thus the program executed successfully.

EX.NO: 10) Write C program to implement Threading & Synchronization Applications

Aim:-

To write a C program to implement Threading & Synchronization Applications

Algorithm:-

Step 1: Start the program
Step 2: Job 1 is Started and runs until it's completed.
Step 3: Job 2 is Started after completing job 1.
Step 4: Job 2 also is Completed.
Step 5: Terminate the program.

Program Coding:-

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg)
{
unsigned long i = 0;
counter += 1;
printf("\n Job %d started\n", counter);

for(i=0; i<(0xFFFFFFFF);i++); printf("\n Job %d finished\n", counter); return
NULL;
}
int main(void)
{
int i =0; int err; while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err!= 0)
printf ("\ncan't create thread :[%s]", strerror(err));
```

```
i++;
}
pthread_join(tid[0],NULL);
pthread_join(tid[0],NULL);
return 0;
}
```

Output:-

Job 1 started

Job 1 finished

Job 2 started

Job 2 finished

Result:-

Thus the program executed successfully.

EX.NO: 11) Implementation of Paging Technique of Memory Management

Aim:-

To write a C program to implement Paging Technique of Memory Management.

Algorithm:-

Step 1: Start the Program
Step 2: Read all the necessary input from the keyboard.
Step 3: Pages – Logical memory is broken into fixed – sized blocks.
Step 4: Frames – Physical memory is broken into fixed – sized blocks.
Step 5: Calculate the physical address using the following
Physical address = (Frame Number * Frame size) + offset.
Step 6: Display the physical address.
Step 7: Terminate the program.

Program Coding:-

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

main()
{
int np,ps,i;
int *sa;
clrscr();
printf("enter how many pages\n");
scanf("%d",&np);
printf("enter the page size \n");
scanf("%d",&ps);
sa=(int*)malloc(2*np);
for(i=0;i<np;i++)
{
sa[i]=(int)malloc(ps);
printf("page%d\t address %u\n",i+1,sa[i]);
}
getch();
}
```

OUTPUT:

```
Enter how many pages: 5
Enter the page size: 4
Page1 Address: 1894
Page2 Address: 1902
Page3 Address: 1910
Page4 Address: 1918
Page5 Address: 1926
```

Result:-

Thus the program executed successfully.

EX.NO: 12) Implementation of the following Page Replacement Algorithms a) FIFO b) LRU c) LFU

(A) FIFO

Aim:-

To write a C program to implement Page replacement algorithm FIFO.

Algorithm:-

Step 1: Start the program.
Step 2: Declare the size with respect to page length.
Step 3: Check the need of replacement from the page to memory.
Step 4: Check the need of replacement from old page to new page in memory.
Step 5: Form a queue to hold all pages.
Step 6: Insert the page require memory into the queue.
Step 7: Check for bad replacement and page fault.
Step 8: Get the number of processes to be inserted.
Step 9: Display the values.
Step 10: Terminate the program.

Program Coding:-

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
```

```
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
}
                        printf("\n");
}
                printf("Page Fault Is %d",count);
                return 0;
}
```

Output:-

```
ENTER THE NUMBER OF PAGES:  20
ENTER THE PAGE NUMBER :      7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
ENTER THE NUMBER OF FRAMES :3
    ref string      page frames
7           7     -1    -1
0           7     0     -1
1           7     0     1
2           2     0     1
0
3           2     3     1
0           2     3     0
4           4     3     0
2           4     2     0
3           4     2     3
0           0     2     3
3
2
1           0     1     3
```

2 0 1 2
0
1
7 7 1 2
0 7 0 2
1 7 0 1
Page Fault Is 15

(B) LRU

Aim:-

To write a C program to implement LRU page replacement algorithm.

Algorithm:-

Step 1: Start the program.
Step 2: Declare the size.
Step 3: Get the number of pages to be inserted.
Step 4: Get the value.
Step 5: Declare counter and stack.
Step 6: Select the least recently used page by counter value.
Step 7: Stack them according to the selection.
Step 8: Display the values.
Step 9: Terminate the program.

Program Coding:-

```
#include<stdio.h>
main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
```

```
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
                                for(r=0;r<f;r++)
                                {
                                        c2[r]=0;
                                        for(j=i-1;j<n;j--)
                                        {
                                        if(q[r]!=p[j])
                                        c2[r]++;
                                        else
                                        break;
                                }
                        }
                        for(r=0;r<f;r++)
                         b[r]=c2[r];
                        for(r=0;r<f;r++)
                        {
```

```
                        for(j=r;j<f;j++)
                        {
                                if(b[r]<b[j])
                                {
                                        t=b[r];
                                        b[r]=b[j];
                                        b[j]=t;
                                }
                        }
                }
                for(r=0;r<f;r++)
                {
                        if(c2[r]==b[0])
                        q[r]=p[i];
                        printf("\t%d",q[r]);
                }
                printf("\n");
        }
    }
}
printf("\nThe no of page faults is %d",c);
}
```

Output:-

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3
        7
        7       5
        7       5       9
        4       5       9
        4       3       9
        4       3       7
        9       3       7
        9       6       7
        9       6       2
        1       6       2
```

The no of page faults is 10

(C) LFU

Aim:-

To write a C program to implement LFU page replacement algorithm.

Algorithm:-

Step 1: Start the program.
Step 2: Declare the size.
Step 3: Get the number of pages to be inserted.
Step 4: Get the value.
Step 5: Declare counter and stack.
Step 6: Select the least frequently used page by counter value.
Step 7: Stack them according to the selection.
Step 8: Display the values.
Step 9: Terminate the program.

Program Coding:-

```
#include<stdio.h>

int main()
{
    int total_frames, total_pages, hit = 0;
    int pages[25], frame[10], arr[25], time[25];
    int m, n, page, flag, k, minimum_time, temp;
    printf("Enter Total Number of Pages:\t");
    scanf("%d", &total_pages);
    printf("Enter Total Number of Frames:\t");
    scanf("%d", &total_frames);
    for(m = 0; m < total_frames; m++)
    {
        frame[m] = -1;
    }
    for(m = 0; m < 25; m++)
    {
        arr[m] = 0;
    }
```

```
printf("Enter Values of Reference String\n");
for(m = 0; m < total_pages; m++)
{
    printf("Enter Value No.[%d]:\t", m + 1);
    scanf("%d", &pages[m]);
}
printf("\n");
for(m = 0; m < total_pages; m++)
{
    arr[pages[m]]++;
    time[pages[m]] = m;
    flag = 1;
    k = frame[0];
    for(n = 0; n < total_frames; n++)
    {
        if(frame[n] == -1 || frame[n] == pages[m])
        {
            if(frame[n] != -1)
            {
                hit++;
            }
            flag = 0;
            frame[n] = pages[m];
            break;
        }
        if(arr[k] > arr[frame[n]])
        {
            k = frame[n];
        }
    }
    if(flag)
    {
        minimum_time = 25;
        for(n = 0; n < total_frames; n++)
        {
            if(arr[frame[n]] == arr[k] && time[frame[n]] < minimum_time)
            {
                temp = n;
                minimum_time = time[frame[n]];
            }
```

```
            }
            arr[frame[temp]] = 0;
            frame[temp] = pages[m];
        }
        for(n = 0; n < total_frames; n++)
        {
            printf("%d\t", frame[n]);
        }
        printf("\n");
    }
    printf("Page Hit:\t%d\n", hit);
    return 0;
}
```

Output:-

```
Enter Total number of pages: 5
Enter Total number of frames: 4
Enter value of reference string.
Enter Value No. [1]: 5
Enter Value No. [2]: 3
Enter Value No. [3]: 1
Enter Value No. [4]: 2
Enter Value No. [5]: 4

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

Page Hit: 0
```

Result:-

Thus the Program executed successfully.

EX.NO:13) Implementation of the following File Allocation Strategies

(A) Indexed

Aim:-

To write a C program to implement Indexed file allocation.

Algorithm:-

Step 1: Start the program
Step 2: Let n be the size of the buffer.
Step 3: Check if there are any producer.
Step 4: If yes check whether the buffer is full.
Step 5: If the buffer is full the producer has to wait.
Step 6: Check there is any consumer. If yes check whether the buffer is empty.
Step 7: If no the consumer consumes them from the buffer.
Step 8: If the buffer is empty, the consumer has to wait.
Step 9: Repeat checking for the producer and consumer till required.
Step 10: Terminate the program.

Program Coding:-

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n",
ind);
scanf("%d",&n);
```

```
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

Output:-

Enter the index block: 5

Enter no of blocks needed and no of files for the index 5 on the disk:
4
1 2 3 4
Allocated
File Indexed
5 ----------→ 1:1
5----------→2:1
5----------→3:1
5----------→4:1

Do you want to enter more file(Yes-1/No-0): 1
Enter the index Block: 4
4 index is already allocated
Enter the index block: 6
Enter no of blocks needed and no of files for the index 6 on the disk:
2
7 8
Allocated
File Indexed
6----------→7:1
6----------→8:1
Do you want to enter more file(Yes-1/No-0): 0

(B) Linked

Aim:-

To write a C program to implement Linked File Allocation.

Algorithm:-

Step 1: Start the program.
Step 2: Create a queue to hold all pages in memory.
Step 3: When the page is required replace the page at the head of the queue.
Step 4: Now the new page is inserted at the tail of the queue.
Step 5: Create a stack.
Step 6: When the page fault occurs replace page present at the bottom if the stack.
Step 7: Terminate the program.

Program Code:-

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
```

```
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

Output:-

```
Enter how many block already allocated : 3
Enter blocks already allocated: 1 3 5
Enter the index starting block and length: 2 2
2------→1
3 Block is already allocated
4------→1
Do you want to enter more file(Yes-1/No-0): 0
```