Ex No :8

IMPLEMENTATION OF THREADING

AIM:

To write a c program to implement Threading and Synchronization Applications.

ALGORITHM:

Step 1: Start the process
Step 2: Declare process thread, thread-id.
Step 3: Read the process thread and thread state.
Step 4: Check the process thread equals to thread-id by using if condition.
Step 5: Check the error state of the thread.
Step 6: Display the completed thread process.
Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* trythis(void *arg)
{
        unsigned long i = 0;
        counter += 1;
        printf("\n Job %d has Started.....\n", counter);
        for(i=0; i<(0xFFFFFFFF);i++);
        printf("\n Job %d has Finished.....\n", counter);
        return NULL;
}
int main(void)
{
        int i = 0;
        int error;
        while(i < 2)
        {
                error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
                if (error != 0)
                printf("\nThread can't be created : [%s]", strerror(error));
                i++;
        }
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
return 0;
```

OUTPUT:

RESULT:

Ex No :9

IMPLEMENTATION OF PAGING TECHNIQUE

AIM:

To write a c program to implement Paging technique for memory management.

ALGORITHM:

Step 1: Start the process
Step 2: Declare page number, page table, frame number and process size.
Step 3: Read the process size, total number of pages
Step 4: Read the relative address
Step 5: Calculate the physical address
Step 6: Display the address
Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
main()
{
        int memsize=15;
        int pagesize,nofpage;
        int p[100];
        int frameno,offset;
        int logadd,phyadd;
        int i;
        int choice=0;
        printf("\nYour Memory Size is %d ",memsize);
        printf("\nEnter Page Size:");
        scanf("%d",&pagesize);
        nofpage=memsize/pagesize;
        for(i=0;i<nofpage;i++)
        {
                printf("\nEnter the Frame of Page%d:",i+1);
                scanf("%d",&p[i]);
        }
        do
        {
                printf("\nEnter a logical address:");
                scanf("%d",&logadd);
                frameno=logadd/pagesize;
                offset=logadd%pagesize;
                phyadd=(p[frameno]*pagesize)+offset;
                printf("\nPhysical address is:%d",phyadd);
                printf("\nDo you want to continue(1/0)?:");
                scanf("%d",&choice);
                }while(choice==1);
        }
```

OUTPUT:

RESULT:

Ex No :10(a)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – FIRST BIT

AIM:

To write a C program for implementation memory allocation methods for fixed partition using first fit.

ALGORITHM:

Step 1: Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp , highest=0, bf[max],ff[max] .
Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check highest<temp,if so assign ff[i]=j,highest=temp
Step 6: Assign frag[i]=highest, bf[ff[i]]=1,highest=0
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program.

PROGRAM:Step 9

```
#include<stdio.h>
#define max 25
main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp;
        static int bf[max],ff[max];
        printf("\n\tMemory Management Scheme - First Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of files:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
        {
                printf("Block %d:",i);
                scanf("%d",&b[i]);
        }
        printf("Enter the size of the files :-\n");
        for(i=1;i<=nf;i++)
        {
                printf("File %d:",i);
                scanf("%d",&f[i]);
        }
        for(i=1;i<=nf;i++)
        {
                for(j=1;j<=nb;j++)
                {
                        if(bf[j]!=1)
```

```
                {
                        temp=b[j]-f[i];
                        if(temp>=0)
                        {
                                ff[i]=j;
                                break;
                        }
                }
        }
        frag[i]=temp;
        bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

OUTPUT:

RESULT:

Ex No :10(b)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – WORST BIT

AIM:

To write a C program for implementation memory allocation methods for fixed partition using worst fit.

ALGORITHM:

Step 1: Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp , highest=0, bf[max],ff[max] .
Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check temp>=0,if so assign ff[i]=j break the for loop.
Step 6: Assign frag[i]=temp,bf[ff[i]]=1;
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program

PROGRAM:

```
#include<stdio.h>
#define max 25
main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
        static int bf[max],ff[max];
        printf("\n\t Memory Management Scheme - Worst Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of files:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
{
        printf("Block %d:",i);
        scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
        printf("File %d:",i);
        scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
        for(j=1;j<=nb;j++)
```

```
            {
                if(bf[j]!=1) //if bf[j] is not allocated
                {
                    temp=b[j]-f[i];
                    if(temp>=0)
                    if(highest<temp)
                    {
                        ff[i]=j;
                        highest=temp;
                    }
                }
            }
        frag[i]=highest;
        bf[ff[i]]=1;
        highest=0;
        }
        printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
        for(i=1;i<=nf;i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
        }
```

OUTPUT:

RESULT:

Ex No :10(c)

IMPLEMENTATION OF MEMORY ALLOCATION METHODS – BEST BIT

AIM:

To write a C program for implementation memory allocation methods for fixed partition using best fit.

ALGORITHM:

Step 1: Define the max as 25.
Step 2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp , highest=0, bf[max],ff[max] .
Step 3: Get the number of blocks,files,size of the blocks using for loop.
Step 4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
Step 5: Check lowest>temp,if so assign ff[i]=j,highest=temp
Step 6: Assign frag[i]=lowest, bf[ff[i]]=1,lowest=10000
Step 7: Repeat step 4 to step 6.
Step 8: Print file no,size,block no,size and fragment.
Step 9: Stop the program.

PROGRAM:

```
#include<stdio.h>
#define max 25
main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
        static int bf[max],ff[max];
        printf("\n\tMemory Management Scheme - Best Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of files:");
        scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
        {
                printf("Block %d:",i);
                scanf("%d",&b[i]);
        }
        printf("Enter the size of the files :-\n");
        for(i=1;i<=nf;i++)
        {
                printf("File %d:",i);
                scanf("%d",&f[i]);
        }
        for(i=1;i<=nf;i++)
        {
```

```
                        for(j=1;j<=nb;j++)
                        {
                                if(bf[j]!=1)
                                {
                                        temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;
lowest=temp;
}       }       }
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

OUTPUT:

RESULT:

Ex No :11(a)

IMPLEMENTATION OF FIFO PAGE REPLACEMENT ALGORITHM

AIM:

To write a C program for implementation of FIFO page replacement algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the necessary variables.
Step 3: Enter the number of frames.
Step 4: Enter the reference string end with zero.
Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen.
Step 6: When a page is brought into memory, it is inserted at the tail of the queue.
Step 7: Initially all the three frames are empty.
Step 8: The page fault range increases as the no of allocated frames also increases.
Step 9: Print the total number of page faults.
Step 10: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
        int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("FIFO Page Replacement \n");
        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:

RESULT:

Ex No :11(b)

IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM

AIM:

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

ALGORITHM:

Step 1: Start the process
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 the selection.
Step 8: Display the values
Step 9: Stop the process

PROGRAM:

```
#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("LRU Page Replacement \n");
        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]);
                                   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:

RESULT:

Ex No :11(c)

IMPLEMENTATION OF OPTIMAL PAGE REPLACEMENT ALGORITHM

AIM:

To write C program to implement optimal page replacement algorithm.

ALGORITHM:

Step 1: Start the process
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 the selection.
Step 8: Display the values
Step 9: Stop the process

PROGRAM:

```
#include<stdio.h>
 int main()
{
        int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1,
        flag2, flag3, i, j, k, pos, max, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter page reference string: ");
for(i = 0; i < no_of_pages; ++i){
scanf("%d", &pages[i]);
}
for(i = 0; i < no_of_frames; ++i){
        frames[i] = -1;
}
for(i = 0; i < no_of_pages; ++i){
        flag1 = flag2 = 0;
        for(j = 0; j < no_of_frames; ++j){
                if(frames[j] == pages[i]){
                        flag1 = flag2 = 1;
                        break;
                }
        }
        if(flag1 == 0){
                for(j = 0; j < no_of_frames; ++j){
                        if(frames[j] == -1){
                                faults++;
                                frames[j] = pages[i];
```

```
                        flag2 = 1;
                        break;
                    }
                }
            }
            if(flag2 == 0){
                flag3 =0;
                for(j = 0; j < no_of_frames; ++j){
                    temp[j] = -1;
                    for(k = i + 1; k < no_of_pages; ++k){
                        if(frames[j] == pages[k]){
                            temp[j] = k;
                            break;
                        }
                    }
                }
            for(j = 0; j < no_of_frames; ++j){
                if(temp[j] == -1){
                    pos = j;
                    flag3 = 1;
                    break;
                }
            }
            if(flag3 ==0){
                max = temp[0];
                pos = 0;
                for(j = 1; j < no_of_frames; ++j)
                {
                    if(temp[j] > max)
                    {
                        max = temp[j];
                        pos = j;
                    }
                }
            }
            frames[pos] = pages[i];
            faults++;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
    }
    }
    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
    }
```

OUTPUT:

RESULT:

Ex No :12(a)

IMPLEMENTATION OF FILE ORGANIZATION -SINGLE LEVEL DIRECTORY

AIM:

To write C program to organize the file using single level directory.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the count, file name, graphical interface.
Step 3: Read the number of files
Step 4: Read the file name
Step 5: Declare the root directory
Step 6: Using the file eclipse function define the files in a single level
Step 7: Display the files
Step 8: Stop the program

PROGRAM:

```
#include<stdio.h>
struct
{
        char dname[10],fname[10][10];
        int fcnt;
}dir;
main()
{
        int i,ch;
        char f[30];
        dir.fcnt = 0;
        printf("\nEnter name of directory -- ");
        scanf("%s", dir.dname);
        while(1)
        {
                printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4.
                Display Files\t5. Exit\n Enter your choice -- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter the name of the file -- ");
                        scanf("%s",dir.fname[dir.fcnt]);
                        dir.fcnt++;
                        break;
                        case 2: printf("\nEnter the name of the file -- ");
                        scanf("%s",f);
                        for(i=0;i<dir.fcnt;i++)
                        {
                                if(strcmp(f, dir.fname[i])==0)
```

```
                                {
                                        printf("File %s is deleted ",f);
                                strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
                                break;
                        }
                }
                if(i==dir.fcnt)
                printf("File %s not found",f);
                else
                dir.fcnt--;
                break;
                case 3: printf("\nEnter the name of the file -- ");
                scanf("%s",f);
                for(i=0;i<dir.fcnt;i++)
                {
                        if(strcmp(f, dir.fname[i])==0)
                        {
                                printf("File %s is found ", f);
                                break;
                        }
                }
                if(i==dir.fcnt)
                        printf("File %s not found",f);
                        break;
                case 4: if(dir.fcnt==0)
                printf("\nDirectory Empty");
                else
                {
                        printf("\nThe Files are -- ");
                        for(i=0;i<dir.fcnt;i++)
                        printf("\t%s",dir.fname[i]);
                }
                break;
                default: exit(0);
                }
        }
    }
```

OUTPUT:

RESULT:

Ex No :12(b)

IMPLEMENTATION OF FILE ORGANIZATION -TWO LEVEL DIRECTORY

AIM:

To write C program to organize the file using two level directory.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the count, file name, graphical interface.
Step 3: Read the number of files
Step 4: Read the file name
Step 5: Declare the root directory
Step 6: Using the file eclipse function define the files in a single level
Step 7: Display the files
Step 8: Stop the program

PROGRAM:

```
#include<stdio.h>
struct
{
        char dname[10],fname[10][10];
        int fcnt;
}dir[10];
main()
{
        int i,ch,dcnt,k;
        char f[30], d[30];
        dcnt=0;
        while(1)
        {
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
        printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice -- ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\nEnter name of directory -- ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt=0;
                dcnt++;
                printf("Directory created");
                break;
                case 2: printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                        if(strcmp(d,dir[i].dname)==0)
                        {
```

```
                printf("Enter name of the file -- ");
                scanf("%s",dir[i].fname[dir[i].fcnt]);
        dir[i].fcnt++;
        printf("File created");
        break;
}
if(i==dcnt)
        printf("Directory %s not found",d);
        break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
        if(strcmp(d,dir[i].dname)==0)
        {
                printf("Enter name of the file -- ");
                scanf("%s",f);
                for(k=0;k<dir[i].fcnt;k++)
                {
                if(strcmp(f, dir[i].fname[k])==0)
                {
                        printf("File %s is deleted ",f);
                        dir[i].fcnt--;
                        strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                        goto jmp;
                        }
                }
                printf("File %s not found",f);
                goto jmp;
        }
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
        if(strcmp(d,dir[i].dname)==0)
        {
                printf("Enter the name of the file -- ");
                scanf("%s",f);
                for(k=0;k<dir[i].fcnt;k++)
                {
                        if(strcmp(f, dir[i].fname[k])==0)
                        {
                                printf("File %s is found ",f);
                                goto jmp1;
                        }
```

```
                                }
                                printf("File %s not found",f);
                        goto jmp1;
                        }
                }
                printf("Directory %s not found",d);
                jmp1: break;
                case 5: if(dcnt==0)
                printf("\nNo Directory's ");
                else
                {
                        printf("\nDirectory\tFiles");
                        for(i=0;i<dcnt;i++)
                        {
                                printf("\n%s\t\t",dir[i].dname);
                                for(k=0;k<dir[i].fcnt;k++)
                                printf("\t%s",dir[i].fname[k]);
                        }
                }
        break;
        default:
        exit(0);
        }
        }
        }
```

OUTPUT:

RESULT:

Ex No :13(a)

IMPLEMENTATION OF SEQUENTIAL FILE ALLOCATION

AIM:

To write a C program for sequential file allocation for the student information.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of records user want to store in the system.
Step 3: Using Standard Library function open the file to write the data into the file.
Step 4: Store the entered information in the system.
Step 5: Using do..While statement and switch case to create the options such as 1-DISPLAY, 2.SEARCH, 3.EXIT.
Step 6: Close the file using fclose() function.
Step 7: Process it and display the result.
Step 8: Stop the program.

PROGRAM:

```
#include < stdio.h>
main()
{
        int f[50], i, st, len, j, c, k, count = 0;
        for(i=0;i<50;i++)
        f[i]=0;
        printf("Files Allocated are : \n");
        x: count=0;
        printf("Enter starting block and length of files: ");
        scanf("%d%d", &st,&len);
        for(k=st;k<(st+len);k++)
        if(f[k]==0)
                count++;
                if(len==count)
                {
                        for(j=st;j<(st+len);j++)
                        if(f[j]==0)
                        {
                                f[j]=1;
                                printf("%d\t%d\n",j,f[j]);
                         }
                        if(j!=(st+len-1))
                        printf(" The file is allocated to disk\n");
                }
        else
        printf(" The file is not allocated \n");
        printf("Do you want to enter more file(Yes - 1/No - 0)");
        scanf("%d", &c);
```

```
if(c==1)
goto  x;
else
exit();
}
```

OUTPUT:

RESULT:

Ex No :13(b)

IMPLEMENTATION OF LINKED FILE ALLOCATION

AIM:

To write a C program for implementation of linked file allocation.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of records user want to store in the system.
Step 3: Using Standard Library function open the file to write the data into the file.
Step 4: Store the entered information in the system.
Step 5: Using do..While statement and switch case to create the options such as 1-DISPLAY, 2.SEARCH, 3.EXIT.
Step 6: Close the file using fclose() function.
Step 7: Process it and display the result.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        int f[50], p,i, st, len, j, c, k, a;
        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);
        }
```

OUTPUT:

RESULT:

Ex No :13(c)

IMPLEMENTATION OF INDEXED FILE ALLOCATION

AIM:

To write a C program for implementation of indexed file allocation.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of records user want to store in the system.
Step 3: Using Standard Library function open the file to write the data into the file.
Step 4: Store the entered information in the system.
Step 5: Using do..While statement and switch case to create the options such as 1-DISPLAY, 2.SEARCH, 3.EXIT.
Step 6: Close the file using fclose() function.
Step 7: Process it and display the result.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
        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);
      }
```

OUTPUT:

RESULT: