INDEX			
SNO	Date	Program	Page Nos
1		Write a C program to simulate the following file allocation strategies. a) Sequential	
2		Write a C program to simulate the following file allocation strategies. a) Linked	
3		Write a C program to simulate the following file allocation strategies. a) Indexed	
4		Download and install nachos operating system and experiment with it	
5		Control the number of ports opened by the operating system with a) Semaphore b) Monitors.	

PROGRAM1

AIM : Write a C program to simulate the Sequential file allocation strategy.

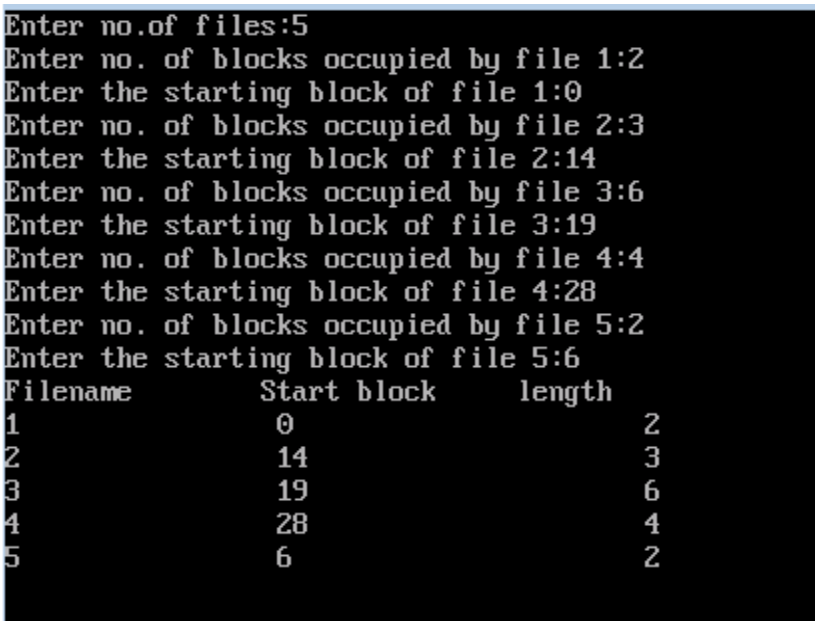
PROGRAM :

```
#include<stdio.h>
void main()
{
 int n,i,j,b[20],sb[20],f[50],flag;
printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0; i<50;i++)
     f[i]=-1;
 for(i=0;i<n;i++)
 {
        printf("Enter no. of blocks occupied by file %d:",i+1);
        scanf("%d",&b[i]);
        printf("Enter the starting block of file %d:",i+1);
        scanf("%d",&sb[i]);
        flag=0;
       for(j=0;j<b[i];j++)
      {
        if(f[sb[i]+j]!=-1)
        {
         flag=1;
         break;
        }
    }
```

```
        if (flag==1)
         {
            printf(" \nBlocks are Already Occupid\n ");
            sb[i]=-1;
         }
        else
        {
          for(j=0;j<b[i];j++)
             f[sb[i]+j]=i+1;
        }
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
  printf("%d\t\t %d\t\t\t%d\n",i+1,sb[i],b[i]);

printf("\n Storage Structure is \n");
for(i=0;i<50;i++)
{
  if(f[i]!=-1)
    printf("\nblock%d -> File%d ",i,f[i]);
}
}
```

OUTPUT :


Enter no.of files:5
Enter no. of blocks occupied by file 1:2
Enter the starting block of file 1:0
Enter no. of blocks occupied by file 2:3
Enter the starting block of file 2:14
Enter no. of blocks occupied by file 3:6
Enter the starting block of file 3:19
Enter no. of blocks occupied by file 4:4
Enter the starting block of file 4:28
Enter no. of blocks occupied by file 5:2
Enter the starting block of file 5:6
Filename Start block length
1 0 2
2 14 3
3 19 6
4 28 4
5 6 2


Storage Structure is
block0 -> File1
block1 -> File1
block6 -> File5
block7 -> File5
block14 -> File2
block15 -> File2
block16 -> File2
block19 -> File3
block20 -> File3
block21 -> File3
block22 -> File3
block23 -> File3
block24 -> File3
block28 -> File4
block29 -> File4
block30 -> File4
block31 -> File4

PROGRAM 2

AIM : Write a C program to simulate the Linked file allocation strategy.

PROGRAM :

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[50];
}f[10];
main()
{
 int i,j,k,n,p,a,len,fr[50];
 clrscr();
 for(i=0;i<50;i++)
     fr[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);
  fr[a]=1;
 }
 printf("Enter no. of files:");
```

```
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter file name:");
scanf("%s",&f[i].fname);
printf("Enter starting block:");
scanf("%d",&f[i].start);
printf("Enter no.of blocks:");
scanf("%d",&f[i].size);
len=f[i].size;
  for(j=f[i].start,k=0;j<f[i].start+len;j++)
  {
       if(fr[j]==0)
      {
       fr[j]=1;
       f[i].block[k++]=j;
      }
     else
        len++;
  }
 f[i].start=f[i].block[0];
}
printf("File\tstart\tsize\tblock\n");
for(i=0;i<n;i++)
{
       printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
       for(j=0;j<f[i].size-1;j++)
              printf("%d--->",f[i].block[j]);
       printf("%d",f[i].block[j]);
       printf("\n");
}
getch();
}
```

OUTPUT :

```
Enter how many blocks already allocated: 5
Enter blocks already allocated: 2 7 4 13 15
Enter no. of files:3
Enter file name:sec1
Enter starting block:3
Enter no.of blocks:4
Enter file name:sec2
Enter starting block:0
Enter no.of blocks:5
Enter file name:sec3
Enter starting block:7
Enter no.of blocks:7
File    start   size    block
sec1    3       4       3--->5--->6--->8
sec2    0       5       0--->1--->9--->10--->11
sec3    12      7       12--->14--->16--->17--->18--->19--->20
```

PROGRAM 3

AIM : Write a C program to simulate the Indexed file allocation strategy.

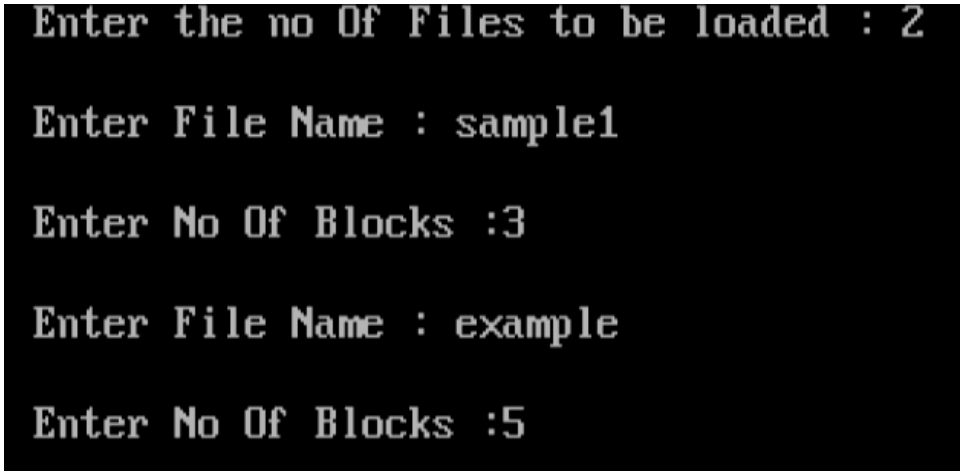
PROGRAM :

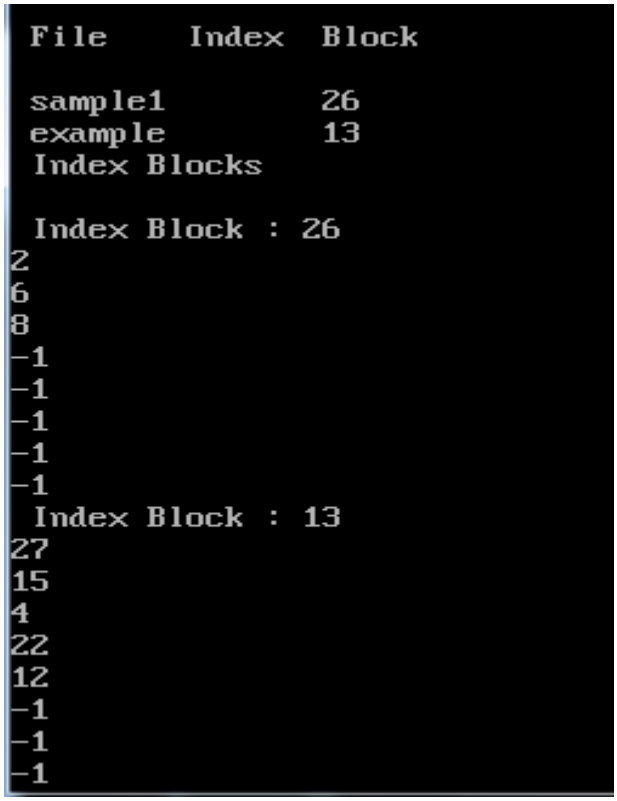
```
#include <stdio.h>
struct Dir
{
  char name[10];
  int indblk;
  int ind[8];
};
int main()
{
        struct Dir F[10];
        int FA[32], tmp, N,RI, R, len, i,j;
        clrscr();
        for(i=0; i<32;i++)
            FA[i]=-1;
        printf(" Enter the no Of Files to be loaded : ");
        scanf("%d",&N);
        for(i=0; i<N; i++)
        {
                printf("\n Enter File Name : ");
                scanf("%s",F[i].name);
                printf("\n Enter No Of Blocks :");
                scanf("%d",&len);
                while(1)
                {
                        R=rand()%32;
```

```
            if(FA[R]==-1)
            {
                FA[R]=-10;
                F[i].indblk=R;
                for(j=0; j<8;j++)
                F[i].ind[j]=-1;
                tmp=0;
                break;
            }
        }
            while(1)
            {
                R=rand()%32;
                if(FA[R]==-1)
                {
                    FA[R]=i;
                    F[i].ind[tmp]=R;
                    tmp++;
                    if(tmp==len)
                    break;
                }
            }
}
printf("\n File \t Index  Block \n");
for(i=0;i<N;i++)
printf("\n %s\t%d",F[i].name, F[i].indblk);
printf("\n Index Blocks \n");
for(i=0; i<N; i++)
{
printf("\n Index Block : %d", F[i].indblk);
for(j=0;j<8;j++)
printf("\n%d\t",F[i].ind[j]);
}
```

```
        getch();
        return 0; }
```

OUTPUT :





PROGRAM 4

AIM : Download and install nachos operating system and experiment with it.

PROCEDURE :

Nachos (Not Another Completely Heuristic Operating System) is an instructional operating system designed to run on top of a POSIX-compliant system (like Linux) as a simulation. It is typically used for educational purposes to implement OS concepts like threads, virtual memory, and file systems.

1. Prerequisites

You need a Linux environment (Ubuntu 18.04/20.04 recommended) and a C++ compiler.

```
sudo apt-get update
sudo apt-get install build-essential csh
```

2. Download and Installation

1. **Download Source Code:** Obtain the Nachos 4.0 source code (e.g., from pai4451/OS2020 GitHub or University of Waterloo).
2. **Extract the Files:**

```
tar -xvf nachos-4.0.tar.gz
```

```
cd nachos-4.0
```

3. **Compile the System:**

```
cd code
```

```
make
```

3. Running the Test Example

Nachos runs in a simulated MIPS environment. To test the installation, run the "Halt" test program:

```
cd userprog
./nachos -x ../test/halt
```

If you see a "Machine halting" message, the installation is successful.

4. Experiments and Projects

Nachos is designed to edit the kernel code. Here are common experiments:

- **Thread Management (Project 1):**
  - **Goal:** Learn how to create and manage threads.
  - **Experiment:** Modify `threads/threadtest.cc` to create multiple threads that print different messages concurrently, and use semaphores to control the order of execution.
- **System Calls (Project 2):**
  - **Goal:** Implement user-level programs (like `exec`, `join`, `exit`).
  - **Experiment:** Create a new test program in `code/test/` that prints "Hello World," compile it, and run it using the Nachos `userprog` kernel.
- **Virtual Memory (Project 3):**
  - **Goal:** Implement paging.
  - **Experiment:** Modify `userprog/addrspace.cc` to implement a page table instead of loading the entire program into memory.
- **File System (Project 4):**
  - **Goal:** Extend the file system.
  - **Experiment:** Change the file system to support files larger than the current limit or implement directories.

PROGRAM 5

To control the number of ports or resources opened by an operating system, you can use synchronization primitives to limit concurrent access. In these examples, a "port" is represented by a shared resource pool, and threads represent processes attempting to access them

The C program for controlling ports using **semaphores** utilizes sem_wait and sem_post to manage a counter. The **monitor** approach uses a while loop with pthread_cond_wait and a mutex to ensure threads only proceed when a port is available.

a) Implementation using Semaphores

A Counting Semaphore is initialized with the maximum number of available ports. When a thread wants to open a port, it performs a sem_wait (decrement). If the count is zero, the thread blocks until another thread performs a sem_post (increment) to release a port.

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

#define MAX_PORTS 3  // Limit to 3 open ports
#define TOTAL_THREADS 6

sem_t port_semaphore;

void* access_port(void* arg) {
   int id = *(int*)arg;

   printf("Thread %d: Requesting to open a port...\n", id);

   // P operation: Wait/Decrement
   sem_wait(&port_semaphore);

   printf("Thread %d: PORT OPENED successfully.\n", id);
   sleep(2); // Simulate port usage

   printf("Thread %d: Closing port...\n", id);
```

```
    // V operation: Signal/Increment
    sem_post(&port_semaphore);

    return NULL;
}

int main() {
    pthread_t threads[TOTAL_THREADS];
    int ids[TOTAL_THREADS];

    // Initialize semaphore with MAX_PORTS
    sem_init(&port_semaphore, 0, MAX_PORTS);

    for (int i = 0; i < TOTAL_THREADS; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, access_port, &ids[i]);
    }

    for (int i = 0; i < TOTAL_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    sem_destroy(&port_semaphore);
    return 0;
}
```

b) Implementation using Monitors

In C, monitors are typically simulated using **Mutexes** and **Condition Variables**. A mutex ensures mutual exclusion when checking the count of available ports, and a condition variable makes threads wait if no ports are available

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

#define MAX_PORTS 3
int available_ports = MAX_PORTS;

pthread_mutex_t lock;
pthread_cond_t port_condition;

void* open_port_monitor(void* arg) {
   int id = *(int*)arg;

   pthread_mutex_lock(&lock);

   // Wait while no ports are available
   while (available_ports == 0) {
      printf("Thread %d: No ports available. Waiting...\n", id);
      pthread_cond_wait(&port_condition, &lock);
   }

   // Acquire resource
   available_ports--;
   printf("Thread %d: PORT OPENED. (Available: %d)\n", id,
available_ports);

   pthread_mutex_unlock(&lock);

   sleep(2); // Simulate work

   pthread_mutex_lock(&lock);

   // Release resource
```

```
    available_ports++;
    printf("Thread %d: Port closed. (Available: %d)\n", id,
available_ports);

    // Signal a waiting thread
    pthread_cond_signal(&port_condition);

    pthread_mutex_unlock(&lock);

    return NULL;
}

int main() {
    pthread_t threads[6];
    int ids[6];

    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&port_condition, NULL);

    for (int i = 0; i < 6; i++) {
        ids[i] = i + 1;
 pthread_create(&threads[i], NULL, open_port_monitor, &ids[i]);
    }

    for (int i = 0; i < 6; i++) {
        pthread_join(threads[i], NULL);
    }
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&port_condition);
    return 0;
}
```