KAVIYATRI BAHINABAI CHAUDHARI,
NORTH MAHARASHTRA UNIVERSITY, JALGAON

NES's

GANGAMAI COLLEGE OF ENGINEERING
ISO 9001:2008

COMPUTER ENGINEERING DEPARTMENT

Laboratory Manuals

Class: T.E. Computer (60-40 Pattern). **Semester:** VI

Subject: Operating System Laboratory.

Academic Year:2023-24.

Institute Vision

- Empowering first generation engineers to excel in technical education based on human values

Institute Mission

- To import affordable and quality education in order to meet needs of industry and to achieve excellence in teaching learning process.
- To achieve excellence in application oriented research inselected area of Technology to contribute to thedevelopment of the region.
- To collaborate with industries to promote innovation capabilities of budding engineers.
- To develop responsible citizens to awareness and acceptance of ethical values.
- To build a support system of all stakeholders to develop the institute.

DEPARTMENT OF COMPUTER ENGINEERING

Vision

- Enriching computer students through quality & value education, developing globally computing computer engineers.

Mission

- To create globally competent students having ability to design, develop and test the software's in coordination with latest technology.
- To facilitate continuous teaching, learning process and collaborate local province state, national and international education for research.
- To interact industry expertise for academic & research.
- To impart ethical and social values among students.
- To develop skills set of our graduates so that they can be competent with software industry.

A Laboratory Manual

For

Operating System – Lab

For the Bachelor of Engineering in the Computer Engineering

TE Computer

Semester – VI

2023-2024

Name: ……….

Roll No:…………………………………………………Batch:……………………………………………………………………

PRN NO:……….

Nagaon Education Society's
GANGAMAI COLLEGE OF ENGINEERING,
NAGAON,DHULE-05
Computer Department

CERTIFICATE

This is to certify that

Mr./Mr.____

Having Roll No. _

Of VI Semester for the course Bachelor of Computer Engineering

Of the institute **Gangamai College of Engineering, Nagaon, Dhule**,has

completed the term work satisfactorily of the subject

Engineering Operating System - Lab

for the academic year 2023 – 2024

as prescribed in the cirriculum

Date:___________ **PRN No:___________________**

Place: Nagaon, Dhule Exam Seat No: _______________

Subject Teacher **HOD** **Principal**

Seal of the Institute

PRACTICAL-COURSE OUTCOMES

COURSE OUTCOMES(COs)

1. Apply concept of file handling and process scheduling.
2. Identify problems of deadlock and semaphore.
3. Apply concept of memory management.
4. Design a file allocation and organization techniques.
5. Solve disk scheduling algorithms.

Expt No.	**Name of Experiment**	**Page No.**	**Starting Date**	**Ending Date**	**Remark**
1.	Write a C program for File Handling for following File handling operation				
2.	Write a C program to implement CPU Scheduling algorithms.				
3.	Write a C program for Procedure consumer problem using Semaphores.				
4.	Write a C program to Memory Allocation Technique				
5.	Write a C Program to Sequencial file Allocation Stratergy				
6.	Write a C program to Single Level directory file organization technique.				
7.	Write a C program to FEFO Page Replacement algorithm				
8.	Write a C program to FCFS Disk Scheduling algorithm				

DEPARTMENT OF COMPUTER ENGINEERING

Objectives

1. To understand and implement concept of file handling and process scheduling.
2. To study problems of deadlock and semaphore and provide practical solutions to it.
3. To understand the memory management concept and its implementation.
4. To study and implement file allocation and organization techniques.
5. To understand and implement disk scheduling algorithms.

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:__________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _ /_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 1

TITLE: - File handling operations

File Operations

In C, you can perform four major operations on files, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Writing information to a file
4. Reading from a file
5. Closing a file

Creating a new file / Opening an existing file

Opening a file is performed using the fopen() function defined in the stdio.h header file.

The syntax for opening a file in standard I/O is

FILE *fp;

fp = fopen ("file_name", "mode");

For example,

fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","r+");

File Mode	**Description**
r	Open a file for reading. If a file is in reading mode, then no data is deleted if a file is already present on a system.
w	Open a file for writing. If a file is in writing mode, then a new file is created if a file doesn't exist at all. If a file is already present on a system, then all the data inside the file is truncated, and it is opened for writing purposes.
a	Open a file in append mode. If a file is in append mode, then the file is opened. The content within the file doesn't change.
r+	open for reading and writing from beginning
w+	open for reading and writing, overwriting a file
a+	open for reading and writing, appending to file

Writing to a File

In C, when you write to a file, newline characters '\n' must be explicitly added.

The stdio library offers the necessary functions to write to a file:

1. fputc(char, file_pointer): It writes a character to the file pointed to by file_pointer.

2. fputs(str, file_pointer): It writes a string to the file pointed to by file_pointer.
3. fprintf(file_pointer, str, variable_lists): It prints a string to the file pointed to by file_pointer.

The program below shows how to perform writing to a file:

fputc() Function:

```
#include <stdio.h>
int main()
{
      int i;
      FILE * fptr;
      char fn[50];
      char str[] = "Guru99 Rocks\n";
      fptr = fopen("fputc_test.txt", "w"); // "w" defines "writing mode"for
      (i = 0; str[i] != '\n'; i++) {
         /* write to file using fputc() function */
         fputc(str[i], fptr);
      }
      fclose(fptr);
      return 0;
   }
```

fputs () Function:

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

{
      FILE * fp;
      fp = fopen("fputs_test.txt", "w+");
      fputs("This is Guru99 Tutorial on fputs,",
      fp);fputs("We don't need to use for loop\n",
      fp);fputs("Easier than fputc function\n", fp);
      fclose(fp);
      return (0);
   }
```

fprintf()Function:

```
#include <stdio.h>
  int main() {
    FILE *fptr;
    fptr = fopen("fprintf_test.txt", "w"); // "w" defines "writing mode"
    /* write to file */
    fprintf(fptr, "Learning C with Guru99\n");
    fclose(fptr);
    return 0;
  }
```

Reading data from a File

There are three different functions dedicated to reading data from a file

1. fgetc(file_pointer): It returns the next character from the file pointed to by the file pointer. When the end of the file has been reached, the EOF is sent back.
2. fgets(buffer, n, file_pointer): It reads n-1 characters from the file and stores the string in a buffer in which the NULL character '\0' is appended as the last character.
3. fscanf(file_pointer, conversion_specifiers, variable_adresses): It is used to parse and analyze data. It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifiers. Keep in mind that as with scanf, fscanf stops reading a string when space or newline is encountered.

The following program demonstrates reading from fputs_test.txt file using fgets(), fscanf() and fgetc ()functions respectively :

```
#include <stdio.h>
int main() {
    FILE *
    file_pointer;char
    buffer[30], c;
    file_pointer = fopen("fprintf_test.txt", "r");
    printf("----read a line --- \n");
    fgets(buffer, 50, file_pointer);
    printf("%s\n", buffer);
    printf("----read and parse data--- \n");
    file_pointer = fopen("fprintf_test.txt", "r"); //reset the pointer
    char str1[10], str2[2], str3[20], str4[2];
    fscanf(file_pointer, "%s %s %s %s", str1, str2, str3, str4);
    printf("Read String1 |%s|\n", str1);
    printf("Read String2 |%s|\n", str2);
    printf("Read  String3  |%s|\n",  str3);
    printf("Read  String4  |%s|\n",  str4);
    printf(" read the entire file \n");
    file_pointer = fopen("fprintf_test.txt", "r"); //reset the pointerwhile
    ((c = getc(file_pointer)) != EOF) printf("%c", c);
    fclose(file_pointer);
    return 0;
  }
```

4. Close a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using the fclose() function.

fclose(fptr);

Here, fptr is a file pointer associated with the file to be closed.

RESULT:

In this way handling the file operations practical was performed created, open, read, write operations performed

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

    FILE *fp; /* file pointer*/char
    fName[20];

    printf("\nEnter file name to create :");
    scanf("%s",fName);

    /*creating (open) a file*/
    fp=fopen(fName,"w");
    /*check file created or not*/
    if(fp==NULL)
    {
        printf("File does not created!!!");exit(0);
        /*exit from program*/
    }

    printf("File created successfully.");
    /*writting into file*/
    putc('K',fp);
    putc('A',fp);
    putc('R',fp);
    putc('T',fp);
    putc('I',fp);
    putc('K',fp);


    printf("\nData written successfully.");
    fclose(fp);

    /*again open file to read data*/
    fp=fopen(fName,"r");
    if(fp==NULL)
    {
        printf("\nCan't open file!!!");
        exit(0);
    }

    printf("Contents of file is :\n");
    printf("%c",getc(fp));
    printf("%c",getc(fp));
    printf("%c",getc(fp));
      printf("%c",getc(fp));
    printf("%c",getc(fp));
    printf("%c",getc(fp));
      printf("%c",getc(fp));

    fclose(fp);
    return 0;
}
```

```
OUTUPUT :
Enter file name to create :KartikFile
created successfully.
Data written successfully.Contents of file is :
KARTIK
```

Question:

1) **What is File Handling ?**
2) **Explain get function?**
3) **What are the various File Operations?**
4) **What are the different Accessing Methods of a File?**
5) **What are the operations that can be performed on a Directory?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _/_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 2

TITLE: - FCFS CPU scheduling algorithm

AIM: - Write a C program to simulate the FCFS CPU scheduling algorithms to find turnaround time and waiting time.

Theory:

First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

First Come First Serve, is just like FIFO(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.

This is used in Batch Systems.

It's easy to understand and implement programmatically, using a Queue data structure, where a new process enters through the tail of the queue, and the scheduler selects process from the head of the queue.

A perfect real life example of FCFS scheduling is buying tickets at ticket counter.

Formulae:

1. Completion Time: Time at which process completes its execution.

2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time. Waiting Time = Turn Around Time – Burst Time

Calculating Average Waiting Time

For every scheduling algorithm, Average waiting time is a crucial parameter to judge it's performance.

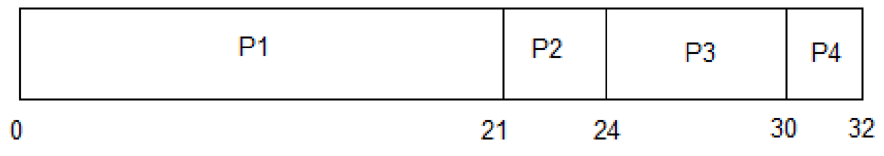
AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given processes, first **P1** will be provided with the CPU resources

- Hence, waiting time for **P1** will be 0
- **P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**,which will be (21 + 3) ms = 24 ms.
- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

Pesudocode:

```
1-    Input the processes along with their burst time (bt).
2-    Find waiting time (wt) for all processes.
3-    As first process that comes need not to wait so 4-
      waiting time for process 1 will be 0 i.e. wt[0] =
0.5-  Find waiting time for all other processes i.e. for
         a.    process i ->
         b.    wt[i] = bt[i-1] + wt[i-1] .
6-    Find turnaround time = waiting_time +
burst_time7-  for all processes.
8-    Find average waiting time =
                    1. total_waiting_time / no_of_processes.
9-    Similarly,  find average turnaround time =
                    1. total_turn_around_time / no_of_processes.
```

RESULT:

In this way the FCFS CPU scheduling algorithms to find turnaround time was implemented

**Program: Write a C program to simulate the FCFS CPU scheduling algorithms to find turnaround time and
waiting time.**

```
#include<stdio.h>
#include<conio.h
> main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ",
i);scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
}
```

Output

```
Enter the number of processes -- 4

Enter Burst Time for Process 0 --

12 Enter Burst Time for Process 1 --

23 Enter Burst Time for Process 2 –

02
```

```
Enter Burst Time for Process 3 -- 10
            PROCESS          BURST TIME        WAITING TIME   TURNAROUND TIME


              P0               12                0                 12
              P1               23                12                35
              P2               2                 35                37
              P3               10                37                47
Average Waiting Time -- 21.000000
Average Turnaround Time --
32.750000
```

Question:

1) **What is CPU Scheduling?**
2) **Explain FCFS & SJF algorithm ?**
3) **Define AWT & ATAT?**
4) **What is Priority Scheduling?**
5) **What is Round Robin algo?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _/_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 3

TITLE: - Producer-Consumer problem using semaphores

AIM: - Write a C program to simulate producer-consumer problem using semaphores.

Theory:

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.

What's the problem here?

The following are the problems that might occur in the Producer-Consumer:

- The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.

- The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.

- The producer and consumer should not access the buffer at the same time.

What's the solution?

The above three problems can be solved with the help of semaphores. In the producer-consumer problem, we use three semaphore variables:

Semaphore S: This semaphore variable is used to achieve mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access the shared buffer at a particular time. This variable is set to 1 initially.

Semaphore E: This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.

Semaphore F: This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

By using the above three semaphore variables and by using the wait() and signal() function, we can solve our problem(the wait() function decreases the semaphore variable by 1 and the signal() function increases the semaphore variable by 1).

The following is the pseudo-code for the producer:

```
void producer( )
   {while(T) {
      produce ( )
      wait (E)
      wait (S)
      append ( )
      signal (S)
      signal (F)
   }
}
```

The above code can be summarized as:

while() is used to produce data, again and again, if it wishes to produce, again and again.

produce() function is called to produce data by the producer.

wait(E) will reduce the value of the semaphore variable "E" by one i.e. when the producer produces something then there is a decrease in the value of the empty space in the buffer. If the buffer is full i.e. the vale of the semaphore variable "E" is "0", then the program will stop its execution and no production will be done.

wait(S) is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.

append() function is used to append the newly produced data in the buffer.

signal(s) is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the production is done and the append operation is also done.

signal(F) is used to increase the semaphore variable "F" by one because after adding the data into the buffer, one space is filled in the buffer and the variable "F" must be updated.

This is how we solve the produce part of the producer-consumer problem. Now, let's see the consumer solution.

The following is the code for the consumer:

```
void consumer( )
   {while(T) {
      wait(F)
      wait(S)
      take( )
      signal(S
      )
      signal(E
      )use( )
   }
}
```

The above code can be summarized as:

while() is used to consume data, again and again, if it wishes to consume, again and again.

wait(F) is used to decrease the semaphore variable "F" by one because if some data is consumed by the consumer then the variable "F" must be decreased by one.

wait(S) is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.

take() function is used to take the data from the buffer by the consumer.

signal(S) is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the consumption is done and the take operation is also done.

signal(E) is used to increase the semaphore variable "E" by one because after taking the data fromthe buffer, one space is freed from the buffer and the variable "E" must be increased.

use() is a function that is used to use the data taken from the buffer by the process to do some operation.

RESULT:

In this way the producer – consumer problem was solved using semaphores

Program: - Write a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<conio.h
>
 void main()
{
intbuffer[10], bufsize, in, out, produce, consume, choice=0;in
= 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n 1. Produce  \t 2. Consume \t 3. Exit");
printf("\n Enter your choice: ");
scanf("%d",
&choice);
switch(choice) {
case                              1:
if((in+1)%bufsize==out)
printf("\nBuffer  is  Full");
else
{
printf("\nEnter  thevalue:  ");
scanf("%d",       &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2: if(in == out)
printf("\nBuffer is
Empty");else
{
consume = buffer[out];
printf("\nThe consumedvalue is %d", consume);
out = (out+1)%bufsize;
}
break;
} } }
```

****Output****

```
1. Produce          2. Consume          3.
  ExitEnter your  choice:  2

Buffer is  Empty
  1. Produce          2. Consume          3.
  ExitEnter your  choice:  1
```

```
Enter the value: 18

 1. Produce           2. Consume           3.
 ExitEnter your choice: 3
```

Question:

1) **What is Consumer Producer Problem?**
2) **What is a Semaphore in C ?**
3) **What is race condition in C ?**
4) **How many Semaphores is used in Consumer Producer?**
5) **What is thread in OS?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:_________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _/_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date**

EXPERIMENT NO. 4

TITLE: - Memory Allocation Technique.

AIM: - Write a C program to simulate the First Fit contiguous memory allocation technique.

Theory:

First Fit Memory Allocation Technique

This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it's size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

Advantages of First-Fit Memory Allocation:

It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.

Disadvantages of Fist-Fit Memory Allocation :

It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory iswasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

Example:

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	200 K	J1	20 K	Busy	180 K
30457	30 K			Free	30
300875	700 K	J2	200 K	Busy	500 K
809567	50 K	J4	50 K	Busy	None
Total available :	980 K	Total used :	270 K		710 K

As illustrated above, the system assigns J1 the nearest partition in the memory. As a result, there is no partition with sufficient space is available for J3 and it is placed in the waiting list.

RESULT:

Hence the First Fit contiguous memory allocation technique was implemented using C.

Program: Write a C program to simulate the First Fit contiguous memory allocation technique.

```
#include<stdio.h>
#include<conio.h>
#define max 25
void 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]);
getch();
}
```

****Output****

```
Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:4

Enter the size of the blocks:-
Block 1:500
Block 2:800
Block 3:200
Enter the size of the files :-
File 1:1000
File 2:300
File 3:800
File 4:600

File_no:      File_size      Block_no:      Block_size:      Fragement
                  :
1             1000           0              7343200          -800
2             300            1              500              200
3             800            2              800              0
4             600            0              7343200          -400
```

Question:

1) **What is Contiguous memory allocation?**
2) **Explain first fit Technique ?**
3) **What is Memory Management in OS ?**
4) **What is Non Contiguous Allocation?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _ /_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 5

TITLE: - Sequential File allocation strategies

AIM: - Write a C program to simulate the File Sequential allocation strategies.

Theory:

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main diskspace or file allocation methods.

1. Sequential Allocation
2. Linked Allocation
3. Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

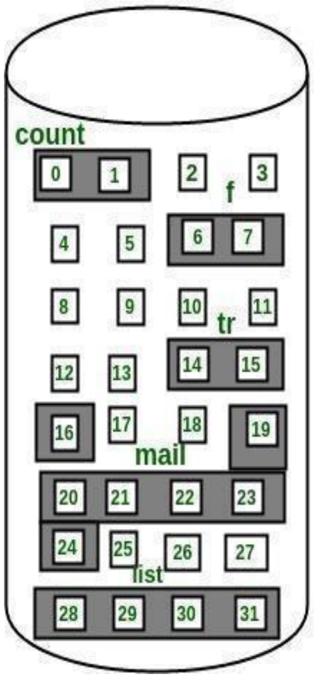
1. Sequential Allocation

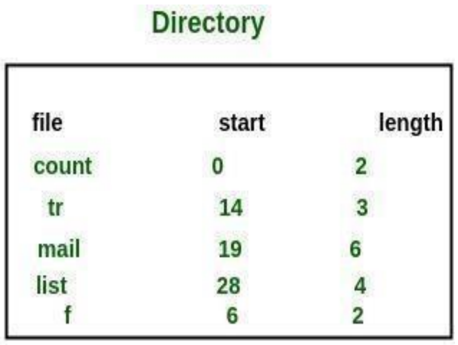
In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

Address of starting block
Length of the allocated portion.

The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.




Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Advantages:

1. Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).

2. This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

1. This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

2. Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Algorithm for Sequential File Allocation:

Step 1: Start the program.
Step 2: Get the number of memory partition and their sizes.
Step 3: Get the number of processes and values of block size for each process.
Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.
Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.
Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.
Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.
Step 8: Stop the program.

RESULT:

Sequential File allocation strategies were simulated using C.

Program: Write a C program to simulate the File Sequential allocation strategies

```
#include<stdio.h>
#include<conio.h
>struct fileTable
{
char name[20];
int sb, nob;
}ft[30];
void main()
{
int i, j, n;
char s[20];
printf("Enter no of files
:");scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d
:",i+1);scanf("%s",ft[i].name);
printf("Enter starting block of file %d
:",i+1);scanf("%d",&ft[i].sb);
printf("Enter no of blocks in file %d
:",i+1);scanf("%d",&ft[i].nob);
}
printf("\nEnter the file name to be searched-- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s,
ft[i].name)==0)break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");
printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].sb+j);
}
getch();
}
```

Output

```
Enter no of files :4

Enter file name 1 :dir
Enter starting block of file 1 :0
Enter no of blocks in file 1 :12

Enter file name 2 :lib
Enter starting block of file 2
:4 Enter no of blocks in file 2
:5

Enter file name 3 :list
Enter starting block of file 3 :7
Enter no of blocks in file 3 :3

Enter file name 4 :sub
Enter starting block of file 4 :7

Enter no of blocks in file 4 :9

Enter the file name to be searched-- lib

FILE NAME START BLOCK NO OF BLOCKS BLOCKS

OCCUPIED lib    4                5        4, 5, 6, 7,

8,
```

Question:

1) **What is Sequntial File Allocation?**
2) **Why do we use the Sequential File Allocation method in the operating system?**

3) **What is Memory Management in OS ?**
4) **Explain Different File allocation strategies?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _ /_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _ _	**Grade**: _ _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 6

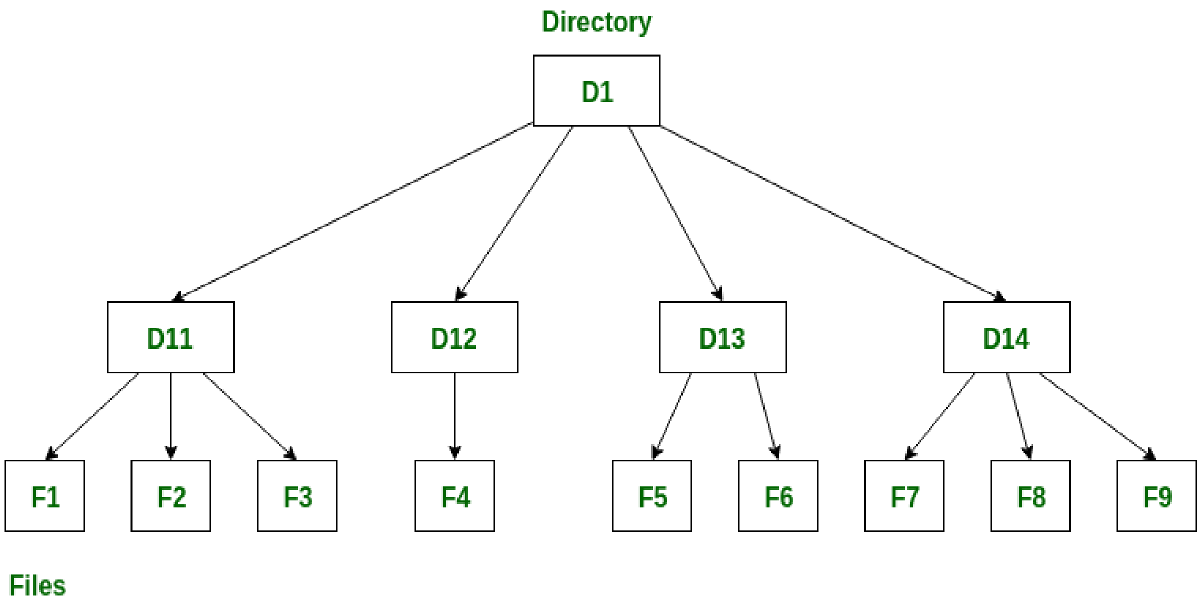
TITLE: - Single level directory file organization technique

AIM: - Write a C program to simulate the Single level directory file organization technique

Theory:

Structures of Directory

A directory is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.

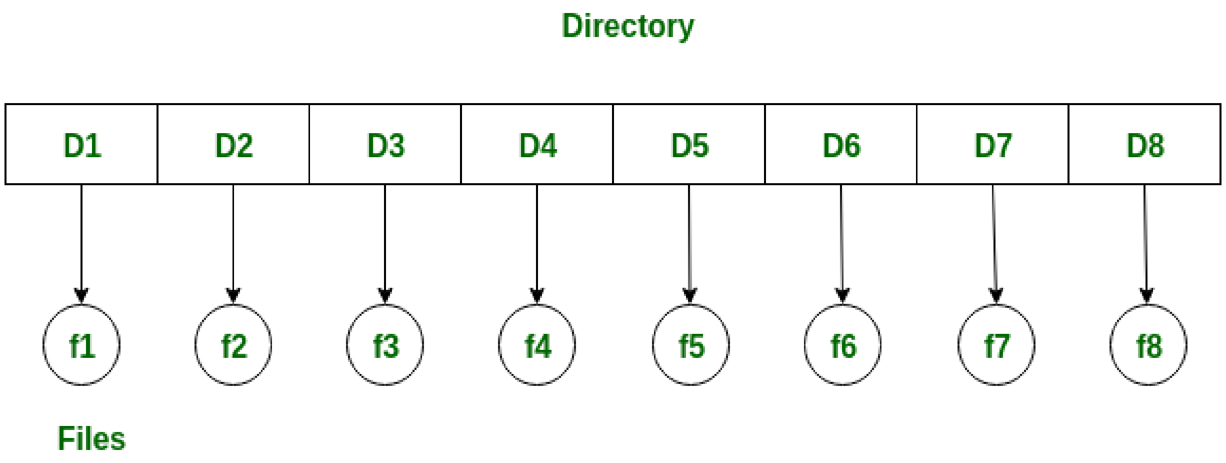


There are several logical structures of a directory, these are given below.

Single-level directory –

Single level directory is simplest directory structure.In it all files are contained in same directory which make it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name . if two users call their dataset test, then the unique name rule violated.



Advantages:

- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if the directory is large.
- In this can not group the same type of files together.

Algorithm for Single Level Directory Structure:

Step 1: Start

Step 2: Initialize values gd=DETECT,gm,count,i,j,mid,cir_x;

```
Initialize character array fname[10][20];
Step 3: Initialize graph function as
Initgraph(& gd, &gm," c:/tc/bgi");
Clear device();
Step 4:set back ground color with setbkcolor();Step
5:read number of files in variable  count. Step 6:if
check i<count
Step    7:    for    i=0    &
i<counti          increment;
Cleardevice();
setbkcolor(GREEN);
read file name;
setfillstyle(1,MAGENTA); Step 8:
mid=640/count; cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);  settextstyle(1,1);
outtextxy(320,125,"rootdirectory");
setcolor(BLUE);
i++;
Step 9:for
j=0&&j<=i&&cir_x+=midj
increment; line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[i]); Step
10: End
```

Result:

C program to simulate the Single level directory file organization technique was implemented

Program: Write a C program to simulate the Single level directory file organization technique

```
#include<stdio.h>
#include<conio.h>
#include<string.h
>
#include<stdlib.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void 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 \n4. Display Files\t5. Exit\nEnter your choice --
");
scanf("%d",&ch)
;switch(ch)
{
case 1: printf("\nEnter thename of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++
;break;
case 2: printf("\nEnter thename 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 thename 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);
}
}
getch();
}
```

Output

```
Enter name of directory -- os


1. Create File   2. Delete File   3. Search File
4. Display Files             5.
ExitEnter your choice ---- 1

Enter the name of the file -- notes


1. Create File   2. Delete File   3. Search File
4. Display Files             5.
ExitEnter your choice ---- 1

Enter the name of the file -- reference books


1. Create File   2. Delete File   3. Search File
4. Display Files             5. Exit
```

```
Enter your choice -- 1

Enter the name of the file -- programs


1. Create File   2. Delete File   3. Search File
4. Display Files            5.
ExitEnter your choice ----1

Enter the name of the file -- output


1. Create File   2. Delete File   3. Search File
4. Display Files            5.
ExitEnter your choice ----2

Enter the name of the file -- output
File output is deleted

1. Create File   2. Delete File   3. Search File
4. Display Files            5.
ExitEnter your choice ----3

Enter the name of the file -- programs
File programs is found

1. Create File   2. Delete File   3. Search File
4. Display Files            5.
ExitEnter your choice ----4

The Files are --            notes       referenc books    programs

1. Create File   2. Delete File   3. Search File
4. Display Files            5.
ExitEnter your choice ----5
```

Question:

1) **What is Directory in OS ?**
2) **Explain different operations on Directory ?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name: ________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _ /_ _/20_ _
Div: _ _ _ , **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 7

TITLE: - FIFO page replacement algorithm

AIM: - Write a C program to simulate FIFO page replacement algorithm

Theory:

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

First In First Out (FIFO) page replacement algorithm –

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example

Consider page reference string 1, 3, 0, 3, 5, 6 and 3 page slots.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 Page Faults.

when 3 comes, it is already in memory so —> 0 Page Faults.

Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault.
Finally 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault.

So total page faults = 5.

Procedure:

```
1- Start traversing the pages.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the pages in the
      queue to perform FIFO.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Remove the first page from the queue
        as it was the first to be entered in
        the memory
     b) Replace the first page in the queue with
        the current page in the string.
     c) Store current page in the queue.
     d) Increment page faults.
2. Return page faults.
```

RESULT: FIFO page replacement algorithm was studied and implemented in C.

Program: Write a C program to simulate FIFO page replacement algorithm

```
#include<stdio.h>
#include<conio.h
>main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter thereference  string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no.of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i+
+)m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(m[k]==rs[i]
)break;
}
if(k==f)
{
m[count++]=rs[i]
;pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f
)count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
}
```

Output

```
Enter the length of reference string -- 5

  Enter the reference string -- 12 5 23 6 9
Enter no. of frames -- 4

The Page Replacement Process is --
          12        -1        -1        -1        PF No. 1
          12        5         -1        -1        PF No. 2
          12        5         23        -1        PF No. 3
          12        5         23        6         PF No. 4
          9         5         23        6         PF No. 5

The number of Page Faults using FIFO are 5
```

Question:

1) **What is mean Page Replacement algo?**
2) **Explain FIFO & Optimal Algo ?**
3) **Define LRU Algo**
4) **What is Page Fault in Paging?**

GANGAMAI COLLEGE OF ENGINEERING, NAGAON

Department of Computer Engineering

Name:________________________	**Date of Performance**:_ _ /_ _/20_ _
Class: TE Computer	**Date of Completion:**_ _ /_ _/20_ _
Div: _ _ _, **Batch**: _ _ _ _ _ _	**Grade**: _ _ _ _ _
Roll No:_ _ _ _ _	
Subject: OS Laboratory	**Sign. of Teacher with Date:**

EXPERIMENT NO. 8

TITLE: - FCFS disk scheduling algorithm

AIM: - Write a C program to simulate FCFS disk scheduling algorithm

Theory:

Disk Scheduling

As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss some important terms related to disk scheduling.

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write requestwill be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access theR/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,
Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.

5. Go to step 2 until all tracks in request array have not been serviced.

Example:

Input:

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

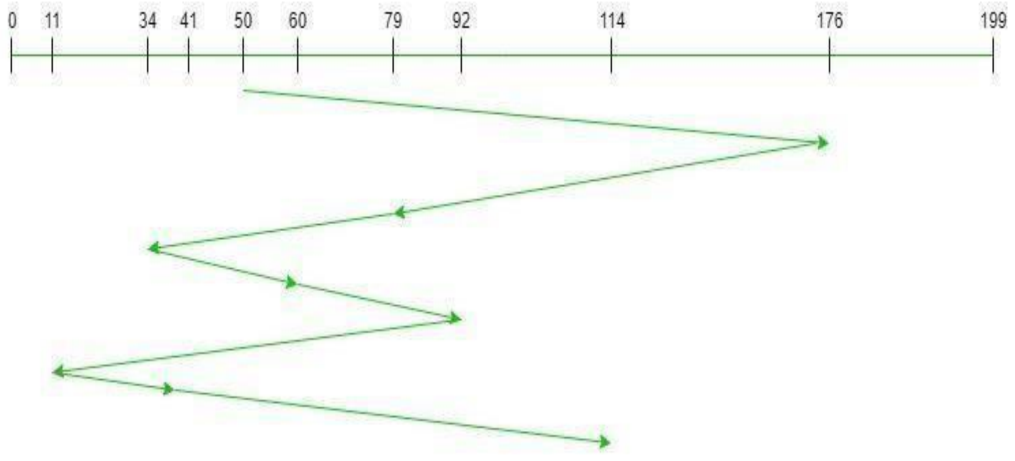
Output:

Total number of seek operations = 510

Seek Sequence is

176, 79, 34, 60, 92, 11, 41, 114

The following chart shows the sequence in which requested tracks are serviced using FCFS.



Therefore, the total seeks count is calculated as:

= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41)

= 510

RESULT: FCFS disk scheduling algorithm was studied and implemented using C

Program: Write a C program to simulate FCFS disk scheduling algorithm

```
#include<stdio.h>
void main()
{
int queue[20],n,head,i,j,k,seek=0,max,diff;
float aver;
//clrscr();
printf("enter the max range of disk");
scanf("%d",&max);
printf("enter the size of queue  request");
scanf("%d",&n);
printf("enter the queue");
for(i=1;i<=n;i++)
{scanf("%d",&queue[i]);}
printf("enter the initial head position");
scanf("%d",&head);
queue[0]=head;
for(j=0;j<=n-
1;j++)
{
diff=abs(queue[j+1]-queue[j]);
seek+=diff;
printf("move is from %d to %d with seek %d\n",queue[j],queue[j+1],diff);
}
printf("total seek time is%d\n",seek);
aver=seek/(float)n;
printf("avrage seek time is %f\n",aver);
//getch();
}
```

Output

```
enter the max range of disk 800
enter the size of queue request 4
enter the queue 110 342 432 456
enter the initial head position 80
move is from 80 to 110 with seek 30
move is from 110 to 342 with seek 232
move is from 342 to 432 with seek 90
move is from 432 to 456 with seek 24
total seek time is376
avrage seek time is 94.000000

Process returned 30 (0x1E)    execution time : 26.550 s
Press any key to continue.
```

Question:

1) **What is Disk Scheduling?**
2) **Explain FCFS & SJF algorithm ?**
3) **Define AWT & ATAT?**
4) **What is FCFS Disk Scheduling?**