VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018

LABORATORY MANUAL

OPERATING SYSTEMS / BCS303

Prepared by,

Dr.Jagadish Prabhu, Dr. Naveen N C, Dr. Naidila Sadashiv, Mrs. Ranjitha S R

JSSATE

BENGALURU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JSS ACADEMY OF TECHNICAL EDUCATION

Bengaluru - 560060

2023-2024

JSS ACADEMY OF TECHNICAL EDUCATION

(Approved by AICTE, New Delhi, Affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, INDIA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision and Mission of the Institution

Vision

"To be among the finest Institutions providing Engineering and Management Education empowered with research, innovation and entrepreneurship"

Mission

1. Strive towards Excellence in teaching-learning process and nurture personality development.
2. Encourage Research, Innovation & Entrepreneurship.
3. Train to uphold highest ethical standards in all activities.

Vision and Mission of the CSE Department

Vision

"To be a distinguished academic and research Department in the field of Computer Science and Engineering for enabling students to be highly competent professionals to meet global challenges."

Mission

1. Impart quality education in Computer Science and Engineering through state of-the art learning environment and committed faculty with research expertise.

2. Train students to become the most sought-after professionals in the field of Information Technology by providing them strong theoretical foundation with adequate practical training.

3. Provide a conducive environment for faculty and students to carry out research and innovation in collaboration with reputed research institutes and industry.

4. Inculcate human values and professional ethics among students to enable them to become good citizens and serve the society.

JSS ACADEMY OF TECHNICAL EDUCATION

(Approved by AICTE, New Delhi, Affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, INDIA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Outcomes	
a.	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
b.	**Problem Analysis: Identify,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c.	**Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d.	**Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
e.	**Modern Tool Usage**: Create, select and apply appropriate techniques, resources andmodern engineering and IT tools including prediction and modeling to Complex engineering activities with an under- standing of the limitations.
f.	**The Engineer and Society**: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the Consequent responsibilities relevant to professional engineering practice.
g.	**Environment and Sustainability**: Understand the impact of professional Engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
h.	**Ethics**: Apply ethical principles and commit to professional ethics and Responsibilities and norms of engineering practice.
i.	**Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
j.	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions**.**
k.	**Life-long Learning:** Recognize the need for and have the preparation and ability to engage in independent and life- long learning in the broadest context of technological change.
l.	**Project Management and Finance:** Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
Program Specific Outcomes	
a.	**PSO1**: Apply the principles of Basic Engineering Science to acquire the hardware and software aspects of Computer Science.
b.	**PSO2:** Solve the real-world problems using modelling for a specific Computer system and architecture.
c.	**PSO3**: Ability to design and develop applications using various software and hardware tools.
d.	**PSO4.** Exhibit the practical competence using broad range of programming languages.

List of Experiments

PRACTICAL COMPONENT OF IPCC OPERATING SYSTEMS			
Course Code	**BCS303**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 20 hours practicals	Total Marks	100
Credits	04	Exam Hours	03

Course Objectives:

CLO 1. To Demonstrate the need for OS and different types of OS
CLO 2. To discuss suitable techniques for management of different resources
CLO 3. To demonstrate different APIs/Commands related to processor, memory, storage and file system management.

	Note: two hours tutorial is suggested for each laboratory sessions.
	Prerequisite
	• Students should be familiarized about C installation and setting the C environment.
Sl. No.	***PART A – List of problems for which student should develop program and execute in the Laboratory***
1	Program Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
2	Program: Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
3	Program: Develop a C program to simulate producer-consumer problem using semaphores.
4	Program: Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.
5	Program: Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
6	Program: Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit

7	Program: Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU
8	Program: Simulate following File Organization Techniques a) Single level directory b) Two level directory
9	Program: Develop a C program to simulate the Linked file allocation strategies.
10	Program: Develop a C program to simulate SCAN disk scheduling algorithm.

Course Outcome (Course Skill Set)

At the end of the course the student will be able to:

CO 1. Explain the structure and functionality of operating system
CO 2. Apply appropriate CPU scheduling algorithms for the given problem.
CO 3. Analyze the various techniques for process synchronization and deadlock handling.
CO 4. Apply the various techniques for memory management
CO 5. Explain file and secondary storage management strategies.
CO 6. Describe the need for information protection mechanisms

Scheme of Continuous Internal Evaluation (CIE):

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50). The minimum passing mark for the SEE is 35% of the maximum marks (18 marks out of 50). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40%(40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (Maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**. • 25 marks for the theory component are split into **15 marks** for two Internal Assessment Test (Two Test, each of **15 marks** with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks**)
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the student shall be evaluated including viva-voce and marks shall be awarded on the same day.

the CIE marks awarded in the case of the practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all ecperiment's write-ups are added and scaled down to 15 marks.

- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks**.
- Scaled-down marks of write-up evaluation and tests added will be CIE marks for the laboratory component of IPCC for **25 marks.**
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course **(duration 03 hours)**

TEXT BOOKS (TB):

1. Java: The Complete Reference, Twelfth Edition, by Herbert Schildt, November 2021, McGraw-Hill, ISBN:9781260463422

REFERENCE BOOKS (RB):

1. Programming with Java, 6th Edition, by E Balagurusamy, Mar-2019, McGraw Hill Education, ISBN:9789353162337
2. Thinking in Java, Fourth Edition, by Bruce Eckel, Prentice Hall, 2006 (https://sd.blackball.lv/library/thinking_in_java_4th_edition.pdf)

PROGRAM :1

1. Develop a c program to implement the Process system calls (fork (), exec(), wait (), create process, terminate process)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
pid_t pid;
// Create a child process using fork()
pid = fork();
if (pid < 0)
{
// Fork failed
perror("Fork failed");
exit(1);
}
else if (pid == 0)
{
/*child process*/
printf("Child process with PID %d\n", getpid());
execlp("/bin/ls","ls",NULL);
}
else{
/*parent process*/
wait (NULL);
printf("Child Complete");
}
}
```

OUTPUT:

```
Child process with PID 13663
Child Complete
```

PROGRAM :2

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM:

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

SJF CPU SCHEDULING ALGORITHM:

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

ROUND ROBIN CPU SCHEDULING ALGORITHM:

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

PRIORITY CPU SCHEDULING ALGORITHM:

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>

void FCFS();
void SJF();
void roundrobin();
void priority();
int n, i, j, pos, temp, *bt, *wt, *tat, *p, *pt, tempn, count, terminaltime=0, initialtime, qt,     flag=0,*tempbt;
float avgwt = 0, avgtat = 0;
void main()
{
 int choice;
  for(;;)
   {
  printf("\n Enter the number of processes : ");
scanf("%d", &n);
 tempn=n;
 free(p);
 free(pt);
 free(bt);
 free(wt);
free(tat);
 free(tempbt);
  tempbt = (int*)malloc(n*sizeof(int));
  p = (int*)malloc(n*sizeof(int));
 pt = (int*)malloc(n*sizeof(int));
 bt = (int*)malloc(n*sizeof(int));
  wt = (int*)malloc(n*sizeof(int));
 tat = (int*)malloc(n*sizeof(int));

  printf("\n Enter the burst time for each process \n");
   for(i=0; i<n; i++)
 {
    printf(" Burst time for P%d : ", i);
   scanf("%d", &bt[i]);
   p[i] = i;
   tempbt[i] = bt[i];
   terminaltime += bt[i];
 }
printf("\n enter the choice");
scanf("%d",&choice);
switch(choice)
{
case 1:FCFS();
        break;
```

```
case 2: SJF();
        break;
case 3:roundrobin();
        break;
case 4:priority();
   break;
default: exit(0);
}
}
}

void FCFS()
{
   avgwt = 0, avgtat = 0;
   wt[0] = 0;
   tat[0] = bt[0];
   for(i=1; i<n; i++)
   {
      wt[i] = wt[i-1] + bt[i-1];  //waiting time[p] = waiting time[p-1] + Burst Time[p-1]
      tat[i] = wt[i] + bt[i];     //Turnaround Time = Waiting Time + Burst Time
   }

   for(i=0; i<n; i++)
   {
      avgwt += wt[i];
      avgtat += tat[i];
   }
   avgwt = avgwt/n;
   avgtat = avgtat/n;

   printf("\n PROCESS \t BURST TIME \t WAITING TIME \t TURNAROUND TIME \n");
   printf("-----------------------------------------------------------------\n");
   for(i=0; i<n; i++)
   {
      printf(" P%d \t\t %d \t\t %d \t\t %d \n", i, bt[i], wt[i], tat[i]);
   }

   printf("\n Average Waiting Time = %f \n Average Turnaround Time = %f \n", avgwt, avgtat);

   printf("\n GAANT CHART \n");
   printf("---------------\n");
   for(i=0; i<n; i++)
   {
      printf(" %d\t|| P%d ||\t%d\n", wt[i], i, tat[i]);
   }
}
void SJF()
{
avgwt = 0, avgtat = 0;
for(i=0; i<n; i++)
   {
         pos = i;
         for(j=i+1; j<n; j++)
         {
                  if(bt[j] < bt[pos])
                  {
                           pos = j;
                  }
```

```
        }
        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;

        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0;
    tat[0] = bt[0];
    for(i=1; i<n; i++)
    {
        wt[i] = wt[i-1] + bt[i-1];  //waiting time[p] = waiting time[p-1] + Burst Time[p-1]
        tat[i] = wt[i] + bt[i];     //Turnaround Time = Waiting Time + Burst Time
    }

    for(i=0; i<n; i++)
    {
        avgwt += wt[i];
        avgtat += tat[i];
    }
    avgwt = avgwt/n;
    avgtat = avgtat/n;

    printf("\n PROCESS \t BURST TIME \t WAITING TIME \t TURNAROUND TIME \n");
    printf("-----------------------------------------------------------------\n");
    for(i=0; i<n; i++)
    {
        printf(" P%d \t\t %d \t\t %d \t\t %d \n", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\n Average Waiting Time = %f \n Average Turnaround Time = %f \n", avgwt, avgtat);

    printf("\n GAANT CHART \n");
    printf("---------------\n");
    for(i=0; i<n; i++)
    {
        printf(" %d\t|| P%d ||\t%d\n", wt[i], p[i], tat[i]);
    }
}

void roundrobin()
{

    printf("\n Enter the Quantum Time : ");
    scanf("%d", &qt);
    avgwt = 0, avgtat = 0;
    wt[0] = 0;
    printf("\n GAANT CHART \n");
    printf("\n-------------\n");

    for(terminaltime=0, count=0; tempn!=0;) {
        initialtime = terminaltime;
        if(tempbt[count] <= qt && tempbt[count] > 0) {
            terminaltime += tempbt[count];
            tempbt[count] = 0;
```

```
            wt[count] = terminaltime - bt[count];
            tat[count] = wt[count] + bt[count];
            flag = 1;
        }
        else if(tempbt[count] > qt) {
            tempbt[count] -= qt;
            terminaltime += qt;
        }
        if(tempbt[count] == 0 && flag == 1) {
            tempn--;
            flag=0;
        }
        if(initialtime != terminaltime) {
            printf(" %d\t|| P%d ||\t%d\n", initialtime, count, terminaltime);
        }
        if(count == n-1)
            count = 0;
        else
            ++count;
    }

    printf("\n PROCESS \t BURST TIME \t WAITING TIME \t TURNAROUND TIME \n");
    printf("-----------------------------------------------------------------\n");
    for(i=0; i<n; i++)
    {
        printf(" P%d \t\t %d \t\t %d \t\t %d \n", i, bt[i], wt[i], tat[i]);
    }

    for(i=0; i<n; i++) {
        avgwt += wt[i];
        avgtat += tat[i];
    }
    avgwt = avgwt/n;
    avgtat = avgtat/n;

    printf("\n Average Waiting Time = %f \n Average Turnaround Time = %f \n", avgwt, avgtat);
}

void priority()
{
    avgwt = 0, avgtat = 0;
    for(i=0; i<n; i++)
    {
    printf(" Priority of P %d : ", i);
    scanf("%d", &pt[i]);
    p[i] = i;
    }
    for(i=0; i<n; i++)
    {
            pos = i;
            for(j=i+1; j<n; j++)
            {
                    if(pt[j] < pt[pos])
                    {
                            pos = j;
                    }
            }
            temp = pt[i];
```

```
        pt[i] = pt[pos];
        pt[pos] = temp;

    temp = bt[i];
    bt[i] = bt[pos];
    bt[pos] = temp;

        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
  }

  wt[0] = 0;
  tat[0] = bt[0];
  for(i=1; i<n; i++)
  {
    wt[i] = wt[i-1] + bt[i-1];  //waiting time[p] = waiting time[p-1] + Burst Time[p-1]
    tat[i] = wt[i] + bt[i];     //Turnaround Time = Waiting Time + Burst Time
  }

  for(i=0; i<n; i++)
  {
    avgwt += wt[i];
    avgtat += tat[i];
  }
  avgwt = avgwt/n;
  avgtat = avgtat/n;

  printf("\n PROCESS \t PRIORITY \t BURST TIME \t WAITING TIME \t TURNAROUND TIME \n");
  printf("-----------------------------------------------------------\n");
  for(i=0; i<n; i++)
  {
    printf(" P%d \t\t %d \t\t %d \t\t %d \t\t %d \n", p[i], pt[i], bt[i], wt[i], tat[i]);
  }

  printf("\n Average Waiting Time = %f \n Average Turnaround Time = %f \n", avgwt, avgtat);

  printf("\n GAANT CHART \n");
  printf("---------------\n");
  for(i=0; i<n; i++)
  {
    printf(" %d\t|| P%d ||\t%d\n", wt[i], p[i], tat[i]);
  }
}
```

PROGRAM :3

3 Develop a C program to simulate producer-consumer problem using semaphores.

Producer Consumer Problem: Producer consumer problem is also known as bounded buffer problem. In this problem we have two processes, producer and consumer, who share a fixed size buffer. Producer work is to produce data or items and put in buffer. Consumer work is to remove data from buffer and consume it. We have to make sure that producer do not produce data when buffer is full and consumer do not remove data when buffer is empty. The producer should go to sleep when buffer is full. Next time when consumer removes data it notifies the producer and producer starts producing data again. The consumer should go to sleep when buffer is empty. Next time when producer add data it notifies the consumer and consumer starts consuming data. This solution can be achieved using semaphores.

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

int main ()
{int n;
void producer ();
void consumer ();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while (1) {
printf("\nEnter your choice:");
 scanf("%d", &n);
    switch (n) {
        case 1: if ((mutex == 1) && (empty != 0))
         producer();
        else printf("Buffer is full!!");
        break;

    case 2: if ((mutex == 1) && (full != 0))
        consumer();
    else printf("Buffer is empty!!");
    break;
    case 3: exit(0);
    break;
    }
}
return 0;
}
int wait(int s)
{ return (--s); }

int signal(int s)
{ return (++s); }

void producer()
{ mutex = wait(mutex);
full = signal(full);
empty = wait(empty);
x++;
printf("\nProducer produces the item %d", x);
mutex = signal(mutex); }

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

OUTPUT

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
```

```
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2
Consumer consumes item 3
```

PROGRAM :4

4. Develop a C program which demonstrates inter process communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

A *pipe* is a mechanism for inter process communication; data written to the pipe by one process can be read by another process. The data is handled in a first-in, first-out (FIFO) order. The pipe has no name; it is created for one use and both ends must be inherited from the single process which created the pipe.
A *FIFO special file* is similar to a pipe, but instead of being an anonymous, temporary connection, a FIFO has a name or names like any other file. Processes open the FIFO by name in order to communicate through it.A pipe or FIFO has to be open at both ends simultaneously.

```
/*Writer Process*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
int fd;
char buf[1024];
/* create the FIFO (named pipe) */
char * myfifo = "/tmp/myfifo";
mkfifo(myfifo, 0666);
printf("Run Reader process to read the FIFO File\n");
fd = open(myfifo, O_WRONLY);
write(fd,"Hi", sizeof("Hi"));
/* write "Hi" to the FIFO */
close(fd);
unlink(myfifo); /* remove the FIFO */
return 0;
}
/* Reader Process */
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#define MAX_BUF 1024
int main()
{
int fd;
/* A temp FIFO file is not created in reader */
char *myfifo = "/tmp/myfifo";
char buf[MAX_BUF];
/* open, read, and display the message from the FIFO */
fd = open(myfifo, O_RDONLY);
read(fd, buf, MAX_BUF);
printf("Writer: %s\n", buf);
close(fd);
return 0;
```

}

OUTPUT

Run Reader process to read the FIFO File
Writer: Hi

PROGRAM :5

Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance.

Description

Banker's Algorithm is used to avoid deadlocks in a system with multiple resources.
Requires that the system has some additional *a priori* information available
Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.
When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
System is in safe state if there exists a safe sequence of all processes.
Sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with j<i.
If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types

- *Available:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type R_j available.
- *Max: n x m* matrix. If *Max* [*i,j*] = *k*, then process P_i may request at most *k* instances of resource type R_j.
- *Allocation:* *n* x *m* matrix. If Allocation[*i,j*] = *k* then P_i is currently allocated *k* instances of R_j.
- *Need:* *n* x *m* matrix. If *Need*[*i,j*] = *k*, then P_i may need *k* more instances of R_j to complete its task.

Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*].

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:

Work = *Available*
Finish [*i*] = *false* for *i* - 1,3, …, *n.*
2. Find and *i* such that both:
(a) *Finish* [*i*] = *false*
(b) $Need_i \leq Work$
If no such *i* exists, go to step 4.
3. $Work = Work + Allocation_i$
$Finish[i] = true$
go to step 2.
4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants *k* instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:
   Available = Available = Request$_i$;
   Allocation$_i$ = Allocation$_i$ + Request$_i$;
   Need$_i$ = Need$_i$ – Request$_i$;
   If safe ⇒ the resources are allocated to Pi.
   If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

```
#include<stdio.h>
struct process
{
int all[6], max[6], need[6], finished, request[6];
}p[10];

int avail[6],sseq[10],ss=0,check1=0,check2=0,n,pid,nor,nori,work[6];
int main()
{
        int safeseq(void);
        int ch,k,i=0,j=0,pid,ch1;
        int violationcheck=0,waitcheck=0;
        do
        {
        printf("\n1.Input\n2.New Request\n3.Safe State or Not\n4.Print\n5.Exit\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\nEnter the number of processes:");
                scanf("%d",&n);
                printf("\nEnter the number of resources:");
                scanf("%d",&nor);
                printf("\nEnter the available resources:");
                for(j=0;j<n;j++)
                {
                        for(k=0;k<nor;k++)
                        {
                                if(j==0)
                        {
                                printf("\nFor Resource Type %d:",k);
                                scanf("%d",&avail[k]);
                        }
                        p[j].max[k]=0;
                        p[j].all[k]=0;
                        p[j].need[k]=0;
                        p[j].finished=0;
                        p[j].request[k]=0;
                        }
                }
                for(i=0;i<n;i++)
                {
                        printf("\nEnter Max and Allocated Resources for P %d :",i);
                        for(j=0;j<nor;j++)
                        {
                        printf("\nEnter the Max of Resources %d:",j);
                        scanf("%d",&p[i].max[j]);
```

```
		printf("\nAllocation of Resources %d:",j);
		scanf("%d",&p[i].all[j]);
		if(p[i].all[j]>p[i].max[j])
		{
			printf("\nAllocation should be less than or equal to Max\n");
			j--;
		}
		else
			p[i].need[j]=p[i].max[j]-p[i].all[j];
		}
}
break;

		case 2:
violationcheck=0;
waitcheck=0;
printf("\nRequesting Process ID:\n");
scanf("%d",&pid);
for(j=0;j<nor;j++)
{
		printf("\nNumber of Request for Resource %d:",j);
		scanf("%d",&p[pid].request[j]);
		if(p[pid].request[j]>p[pid].need[j])
		violationcheck=1;
		if(p[pid].request[j]>avail[j])
		waitcheck=1;
}
		if(violationcheck==1)
		printf("\nThe Process Exceeds its Max needs: Terminated\n");
else if(waitcheck==1)
		printf("\nLack of Resources: Process State - Wait\n");
else
{
		for(j=0;j<nor;j++)
		{
		avail[j]=avail[j]-p[pid].request[j];
		p[pid].all[j]=p[pid].all[j]+p[pid].request[j];
		p[pid].need[j]=p[pid].need[j]-p[pid].request[j];
		}
		ch1=safeseq();
		if(ch1==0)
		{
		for(j=0;j<nor;j++)
		{
			avail[j]=avail[j]+p[pid].request[j];
			p[pid].all[j]=p[pid].all[j]-p[pid].request[j];
			p[pid].need[j]=p[pid].need[j]+p[pid].request[j];
		}
		}
		else if(ch1==1)
		printf("\nRequest committed.\n");
}
break;

		case 3:
if(safeseq()==1)
		printf("\nThe System is in Safe State\n");
else
```

```
                printf("\nThe System is not in Safe State\n");
        break;
                case 4:
        printf("\nNumber of Process:%d\n",n);
        printf("\nNumber of Resources:%d\n",nor);
        printf("\nPid\tMax\tAllocated\tNeed\n");
    for(i=0;i<n;i++)
        {
                printf(" P%d :",i);
                for(j=0;j<nor;j++)
                printf(" %d ",p[i].max[j]);
                printf("\t");
                for(j=0;j<nor;j++)
                        printf(" %d ",p[i].all[j]);
                printf("\t");
                for(j=0;j<nor;j++)
                printf(" %d ",p[i].need[j]);
                printf("\n");
                }
    printf("\nAvailable:\n");
    for(i=0;i<nor;i++)
                printf(" %d ",avail[i]);
    break;
                case 5:  break;
    }
    }while(ch!=5);
    return 0;
}

int safeseq()
{
    int tj,tk,i,j,k;
    ss=0;
    for(j=0;j<nor;j++)
    work[j]=avail[j];
    for(j=0;j<n;j++)
    p[j].finished=0;
    for(tk=0;tk<nor;tk++)
    {
    for(j=0;j<n;j++)
    {
        if(p[j].finished==0)
        {
        check1=0;
        for(k=0;k<nor;k++)
        if(p[j].need[k]<=work[k])
        check1++;
        if(check1==nor)
        {
                for(k=0;k<nor;k++)
                {
                work[k]=work[k]+p[j].all[k];
                p[j].finished=1;
                }
                sseq[ss]=j;
                ss++;
        }
        }
```

```
        }
        }
        check2=0;
        for(i=0;i<n;i++)
if(p[i].finished==1)
        check2++;
        printf("\n");
        if(check2>=n)
        {
        for(tj=0;tj<n;tj++)
                printf("p%d",sseq[tj]);
        return 1;
        }
        else
        printf("\nThe System is not in Safe State\n");
        return 0;
}
```

/*__OUTPUT__

```
1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit
Enter your choice:1

Enter the number of processes:5

Enter the number of resources:3

Enter the available resources:
For Resource Type 0:3
For Resource Type 1:3
For Resource Type 2:2

Enter Max and Allocated Resources for P 0 :
Enter the Max of Resources 0:7
Allocation of Resources 0:0

Enter the Max of Resources 1:5
Allocation of Resources 1:1

Enter the Max of Resources 2:3
Allocation of Resources 2:0

Enter Max and Allocated Resources for P 1 :
Enter the Max of Resources 0:3
Allocation of Resources 0:2

Enter the Max of Resources 1:2
Allocation of Resources 1:0

Enter the Max of Resources 2:2
Allocation of Resources 2:0

Enter Max and Allocated Resources for P 2 :
Enter the Max of Resources 0:9
```

```
Allocation of Resources 0:3

Enter the Max of Resources 1:0
Allocation of Resources 1:0

Enter the Max of Resources 2:2
Allocation of Resources 2:2

Enter Max and Allocated Resources for P 3 :
Enter the Max of Resources 0:2
Allocation of Resources 0:2

Enter the Max of Resources 1:2
Allocation of Resources 1:1

Enter the Max of Resources 2:2
Allocation of Resources 2:1

Enter Max and Allocated Resources for P 4 :
Enter the Max of Resources 0:4
Allocation of Resources 0:0

Enter the Max of Resources 1:3
Allocation of Resources 1:0

Enter the Max of Resources 2:3
Allocation of Resources 2:2

1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit
Enter your choice:3

p1p3p4p0p2
The System is in Safe State

1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit

Enter your choice:4
Number of Process:5
Number of Resources:3

Pid      Max    Allocated       Need
P0 : 7  5  3      0  1  0          7  4  3
P1 : 3  2  2      2  0  0          1  2  2
P2 : 9  0  2      3  0  2          6  0  0
P3 : 2  2  2      2  1  1          0  1  1
P4 : 4  3  3      0  0  2          4  3  1

Available:
3  3  2
1.Input
```

```
        2.New Request
        3.Safe State or Not
        4.Print
        5.Exit
Enter your choice:2

Requesting Process ID:
1
Number of Request for Resource 0:1
Number of Request for Resource 1:0
Number of Request for Resource 2:2

p1p3p4p0p2
Request committed.
1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit

Enter your choice:4
Number of Process:5
Number of Resources:3

Pid      Max    Allocated      Need
P0 : 7  5  3      0  1  0        7  4  3
P1 : 3  2  2      3  0  2        0  2  0
P2 : 9  0  2      3  0  2        6  0  0
P3 : 2  2  2      2  1  1        0  1  1
P4 : 4  3  3      0  0  2        4  3  1

Available:
2  3  0
1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit

Enter your choice:3
p1p3p4p0p2

The System is in Safe State

1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit
Enter your choice:2

Requesting Process ID:
4

Number of Request for Resource 0:3
Number of Request for Resource 1:3
Number of Request for Resource 2:0
```

Lack of Resources: Process State - Wait

1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit
Enter your choice:2
Requesting Process ID:
0

Number of Request for Resource 0:0
Number of Request for Resource 1:2
Number of Request for Resource 2:0

The System is not in Safe State

1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit

Enter your choice:4
Number of Process:5
Number of Resources:3

Pid	Max	Allocated	Need
P0 :	7 5 3	0 1 0	7 0 3
P1 :	3 2 2	3 0 2	0 2 0
P2 :	9 0 2	3 0 2	6 0 0
P3 :	2 2 2	2 1 1	0 1 1
P4 :	4 3 3	0 0 2	4 3 1

Available:
2 3 0
1.Input
2.New Request
3.Safe State or Not
4.Print
5.Exit
Enter your choice:5

PROGRAM :6

Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.

DESCRIPTION

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. E ach partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-

fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

PROGRAM: a) WORST FIT

```
#include<stdio.h>
#define max 25
void main ()
{
int frag[max], b[max], f[max],i,j,nb,nf,temp,highest=0;
static int bf[max], ff[max];
printf("\n\tMemory 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]);
}
```

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	1	3	7	6
2	4	1	5	1

PROGRAM: b) BEST FIT

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

TEST CASES:

Memory Management Scheme - Best Fit

Enter the number of blocks:3

Enter the number of files:2

PROGRAM: c) FIRST FIT

```
#include<stdio.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]);
}
```

TEST CASES:

TEST CASE 1:

```
Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2
Enter the size of the blocks: -
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
```

File 1:1
File 2:4

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	1	1	5	4
2	4	3	7	3

TEST CASE 2:

Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2
Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:4
File 2:1

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	4	1	5	1
2	1	2	2	1

PROGRAM :7

7.Develop a C program to simulate page replacement algorithms:
a) FIFO b) LRU

```
#include <stdio.h>
#define MAX_FRAMES 3
#define MAX_PAGES 10

void fifo(int referenceString[MAX_PAGES])
{
   int frames [MAX_FRAMES] = {-1, -1, -1};
   int frameIndex = 0;
   int pageFaults = 0;
            printf("Page Replacement Using FIFO Algorithm\n");
   printf("Page Reference String: ");
   for (int i = 0; i < MAX_PAGES; i++) {
      printf("%d ", referenceString[i]);
   }
   printf("\n");

   for (int i = 0; i < MAX_PAGES; i++)
{
      int currentPage = referenceString[i];
      int pageFound = 0;
      for (int j = 0; j < MAX_FRAMES; j++)
{
         if (frames[j] == currentPage)
{
            pageFound = 1;
            break;
         }
      }
      if (!pageFound)
{
         printf("Page %d is not in the frames. Page Fault!\n", currentPage);
         pageFaults++;
         frames[frameIndex] = currentPage;
```

```
            frameIndex = (frameIndex + 1) % MAX_FRAMES;
        }
        printf("Frames: ");
        for (int j = 0; j < MAX_FRAMES; j++) {
            printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("Total Page Faults: %d\n", pageFaults);
}

int main () {
    int referenceString[MAX_PAGES] = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3};
    fifo(referenceString);
    return 0;
}
```

FIFO OUTPUT

```
Page Replacement Using FIFO Algorithm
Page Reference String: 1 2 3 4 1 2 5 1 2 3
Page 1 is not in the frames. Page Fault!
Frames: 1 -1 -1
Page 2 is not in the frames. Page Fault!
Frames: 1 2 -1
Page 3 is not in the frames. Page Fault!
Frames: 1 2 3
Page 4 is not in the frames. Page Fault!
Frames: 4 2 3
Page 1 is not in the frames. Page Fault!
Frames: 4 1 3
Page 2 is not in the frames. Page Fault!
Frames: 4 1 2
Page 5 is not in the frames. Page Fault!
Frames: 5 1 2
Frames: 5 1 2
Frames: 5 1 2
Page 3 is not in the frames. Page Fault!
Frames: 5 3 2
```
Total Page Faults: 8

2.LRU

```
#include <stdio.h>
#define MAX_FRAMES 3
#define MAX_PAGES 10

void lru(int referenceString[MAX_PAGES])
{
    int frames[MAX_FRAMES] = {-1, -1, -1};
    int counters[MAX_FRAMES] = {0};
    printf("Page Replacement Using LRU Algorithm\n");
    printf("Page Reference String: ");
    for (int i = 0; i < MAX_PAGES; i++)
{
        printf("%d ", referenceString[i]);
    }
    printf("\n");
```

```
    int pageFaults = 0;
    for (int i = 0; i < MAX_PAGES; i++)
{
        int currentPage = referenceString[i];

        int pageFound = 0;
        for (int j = 0; j < MAX_FRAMES; j++)
{
            if (frames[j] == currentPage)
{
                pageFound = 1;
                counters[j] = 0;
                break;
            }
        }
        if (!pageFound)
{
            printf("Page %d is not in the frames. Page Fault!\n", currentPage);
            pageFaults++;
            int lruIndex = 0;
            for (int j = 1; j < MAX_FRAMES; j++) {
                if (counters[j] > counters[lruIndex]) {
                    lruIndex = j;
                }
            }
            frames[lruIndex] = currentPage;
            counters[lruIndex] = 0;
        }
        for (int j = 0; j < MAX_FRAMES; j++) {
            counters[j]++;
        }
        printf("Frames: ");
        for (int j = 0; j < MAX_FRAMES; j++) {
            printf("%d(%d) ", frames[j], counters[j]);
        }
        printf("\n");
    }
    printf("Total Page Faults: %d\n", pageFaults);
}
int main() {
    int referenceString[MAX_PAGES] = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3};
    lru(referenceString);
    return 0;
}
```

LRU OUTPUT:

```
Page Replacement Using LRU Algorithm
Page Reference String: 1 2 3 4 1 2 5 1 2 3
Page 1 is not in the frames. Page Fault!
Frames: 1(1) -1(1) -1(1)
Page 2 is not in the frames. Page Fault!
Frames: 2(1) -1(2) -1(2)
Page 3 is not in the frames. Page Fault!
Frames: 2(2) 3(1) -1(3)
Page 4 is not in the frames. Page Fault!
Frames: 2(3) 3(2) 4(1)
Page 1 is not in the frames. Page Fault!
Frames: 1(1) 3(3) 4(2)
```

Page 2 is not in the frames. Page Fault!
Frames: 1(2) 2(1) 4(3)
Page 5 is not in the frames. Page Fault!
Frames: 1(3) 2(2) 5(1)
Frames: 1(1) 2(3) 5(2)
Frames: 1(2) 2(1) 5(3)
Page 3 is not in the frames. Page Fault!
Frames: 1(3) 2(2) 3(1**)**

Total Page Faults: 8

PROGRAM :8

Simulate following File Organization Techniques a) Single level directory b) Two level directory

Program: a) Single level directory

```
#include<stdio.h>
int main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("**************************************************\n"); for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}
```

Sample Output:

```
enter number of directorios:3
enter names of directories:
a
b
c
enter size of directories:2
4
3
enter the file names :d
e
f
g
```

```
h
i
j
k
l
directory size filenames
**************************************************
a 2 d
e
b 4 f
g
h
i
c 3 j
k
l
```

Program: a) Two level directory

```
#include<stdio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
int main()
{
int i,j,k,n;
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n**********************************************************\n"); for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
```

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

Sample Output:

```
enter number of directories:2
enter directory 1 names:a
enter size of directories:2
enter subdirectory name and size:b
2
enter file name:g
enter file name:h
enter subdirectory name and size:2
r
enter directory 2 names:enter size of directories:t
dirname size subdirname size files
************************************************************
a 2 b 2 g h
2 0
r 0
```

PROGRAM :9

Write a C Program to implement Linked File Allocation method.

Linked File Allocation is a Non-contiguous memory allocation method where the file is stored in random memory blocks and each block contains the pointer (or address) of the next memory block as in a linked list. The starting memory block of each file is maintained in a directory and the rest of the file can be traced from that starting block.

Program:

```
#include<stdio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
int main()
{
 int i,j,n;
 //clrscr();
 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);
 f[i].block[0]=f[i].start;
 printf("Enter no.of blocks:");
 scanf("%d",&f[i].size);
 printf("Enter block numbers:");
 for(j=1;j<=f[i].size;j++)
 {
 scanf("%d",&f[i].block[j]);
 }
 }
 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=1;j<=f[i].size-1;j++) printf("%d--->",f[i].block[j]); printf("%d",f[i].block[j]);
printf("\n");
//getch();
}
}
```

Sample Output:

```
Enter no. of files:2
Enter file name:a.c
Enter starting block:23
Enter no.of blocks:3
Enter block numbers:12
45
2
Enter file name:b.c
Enter starting block:34
Enter no.of blocks:4
Enter block numbers:6
90
25
5
File start size block
a.c 23 3 12--->45--->2 b.c 34 4 6--->90--->25--->5
```

PROGRAM :10

Develop a C program to simulate SCAN disk scheduling algorithm.

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

The SCAN algorithm is sometimes called the elevator algorithm, since the disk arms behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

```
#include <stdio.h>
 int main()
{
int i,j,sum=0,n;
int d[20];
int disk; //loc of head
int temp,max;
int dloc; //loc of disk in array
//clrscr();
printf("enter number of location\t");
scanf("%d",&n);
printf("enter position of head\t");
scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&d[i]);
}
d[n]=disk;
n=n+1;
for(i=0;i<n;i++) //sorting="" disk="" locations<="" span="">
{
for(j=i;j<n;j++)
```

```
{
if(d[i]>d[j])
{
temp=d[i];
d[i]=d[j];
d[j]=temp;
}
}

}
max=d[n];
for(i=0;i<n;i++) //to="" find="" loc="" of="" disc="" in="" array<="" span="">
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)//< span="">
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);
//getch();
return 0;
}
```

OUTPUT

```
enter number of location        8
enter position of head  53
enter elements of disk queue
98
183
37
122
14
124
65
57
53 -->37 -->14 -->0 -->57-->65-->98-->122-->124-->183-->
movement of total cylinders 53
```