Operating System

Manual

Course Title:	**Operating Systems**	
Course Code:	**BCS303**	**CIE Marks 50**
Course Type (Theory/Practical /Integrated)	**Integrated**	**SEE Marks 50**
		Total Marks100
Teaching Hours/Week (L:T:P: S)	**3:0:2**	**Exam Hours 3+2**
Total Hours of Pedagogy	**40 hours**	**Credits 04**

	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.
	Teaching-Learning Process (General Instructions) Teachers can use the following strategies to accelerate the attainment of the various course outcomes. 1. Lecturer methods (L) need not to be only traditional lecture method, but alternative effective teaching methods could be adopted to attain the outcomes. 2. Use of Video/Animation to explain functioning of various concepts. 3. Encourage collaborative (Group Learning) Learning in the class. 4. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather than simply recall it. 5. Role play for process scheduling. 6. Demonstrate the installation of any one Linux OS on VMware/Virtual Box
	Module-1 (8 Hours of Pedagogy)
	Introduction to operating systems, System structures: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management; Protection and Security; Distributed system; Special-purpose systems; Computing environments. **Operating System Services:** User - Operating System interface; System calls; Types of system calls; System programs; Operating system design and implementation; Operating System structure; Virtual machines; Operating System debugging, Operating System generation; System boot. **Textbook 1: Chapter – 1 (1.1-1.12), 2 (2.2-2.11)**
Teaching-Learning Process	Chalk and talk method/Power Point Presentation/ Web Content: 1. https://youtu.be/mXw9ruZaxzQ 2. https://youtu.be/vBURTt97EkA
	Module-2 (8 Hours of Pedagogy)

Process Management: Process concept; Process scheduling; Operations on processes; Inter process communication **Multi-threaded Programming:** Overview; Multithreading models; Thread Libraries; Threading issues. Process Scheduling: Basic concepts; Scheduling Criteria; Scheduling Algorithms; Thread scheduling; Multiple-processor scheduling, **Textbook 1: Chapter – 3 (3.1-3.4), 4 (4.1-4.4), 5 (5.1 -5.5)**	
Teaching-Learning Process	Chalk and talk method/ Power Point Presentation

	Module-3 (8 Hours of Pedagogy)
Process Synchronization: Synchronization: The critical section problem; Peterson's solution; Synchronization hardware; Semaphores; Classical problems of synchronization; **Deadlocks:** System model; Deadlock characterization; Methods for handling deadlocks; Deadlock prevention; Deadlock avoidance; Deadlock detection and recovery from deadlock. **Textbook 1: Chapter – 6 (6.1-6.6), 7 (7.1 -7.7)**	
Teaching-Learning Process	Chalk and talk method/ Power Point Presentation
	Module-4 (8 Hours of Pedagogy)
Memory Management: Memory management strategies: Background; Swapping; Contiguous memory allocation; Paging; Structure of page table; Segmentation. **Virtual Memory Management:** Background; Demand paging; Copy-on-write; Page replacement; Allocation of frames; Thrashing. **Textbook 1: Chapter -8 (8.1-8.6), 9 (9.1-9.6)**	
Teaching-Learning Process	Chalk and talk method/Power Point Presentation
	Module-5 (8 Hours of Pedagogy)
File System, Implementation of File System: File system: File concept; Access methods; Directory and Disk structure; File system mounting; File sharing; **Implementing File system:** File system structure; File system implementation; Directory implementation; Allocation methods; Free space management. **Secondary Storage Structure, Protection:** Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management; **Protection**: Goals of protection, Principles of protection, Domain of protection, Access matrix. **Textbook 1: Chapter – 10 (10.1-10.5) ,11 (11.1-11.5),12 (12.1-12.5), 14 (14.1-14.4)**	
Teaching-Learning Process	Chalk and talk method/Power Point Presentation

Course Outcomes (Course Skill Set)

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

CO1. Explain the structure and functionality of operating systems.
CO2. Apply appropriate CPU scheduling algorithms for the given problem.
CO3. Analyse the various techniques for process synchronization and deadlock handling.
CO4. Apply the various techniques for memory management.
CO5. Explain file and secondary storage management strategies.
CO6. Describe the need for information protection mechanisms.

Programming Assignments

1. Develop a C program to implement the Process System calls (fork(), exec(), wait(), create process, terminate process.
2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time:
   a. FCFS b. SJF c. Round Robin d. Priority
3. Develop a C program to simulate producer-consumer problem using semaphores.
4. 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. Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
6. Develop a C program to simulate the following contiguous memory allocation Techniques:
   a. Worst fit b. Best fit c. First fit.
7. Develop a C program to simulate page replacement algorithms:
   a. FIFO b. LRU
8. Simulate following File Organization Techniques
   a. Single level directory b. Two level directory
9. Develop a C program to simulate the Linked file allocation strategies.
10. Develop a C program to simulate SCAN disk scheduling algorithm.

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) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). 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 Tests (Two Tests, 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 students 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 experiments' 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 evaluations 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**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored by the student shall be proportionally scaled down to 50 Marks

The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component.

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

CO1	Explain the structure and functionality of operating system
CO2	Apply appropriate CPU scheduling algorithms for the given problem.
CO3	Analyse the various techniques for process synchronization and deadlock handling.
CO4	Apply the various techniques for memory management
CO5	Explain file and secondary storage management strategies.
CO6	Describe the need for information protection mechanisms

LAB ASSESSMENT RUBRICS FOR CIE

1. **Maximum CIE: 25 Marks**
   a. **Continuous Evaluation of Experiments:15 Marks**
   b. **Internal Test: 10 Marks**
2. **Rubrics for Continuous Evaluation of Experiments**

Continuous Evaluation for 15 marks		Marks Allotted
A	Observation Write up and Attendance	4
B	Conduction of Experiments and output	3
C	Viva Voce	4
D	Record Write up	4
Total		15

3. **Rubrics for Test Evaluation**: The Laboratory shall be conducted for 50 marks and scaled

down to 10 marks

Test Evaluation for 15 marks		Marks Allotted
A	Program Write up	15
B	Conduction of Experiments and output	20
C	Viva Voce	15
Total		50

Introduction to Principles of Programming Using C

Problem Solving Techniques: The process of working through details of a problem to reach a solution.

There are three approaches to problem solving:

- Algorithm
- Flowchart
- Pseudo Code

Algorithm: The algorithm is a step-by-step procedure to be followed in solving a problem. It provides a scheme to solve a particular problem in finite number of unambiguous steps. It helps in implementing the solution of a problem using any of the programming languages.
In order to qualify as an algorithm, a sequence of instructions must possess the following characteristics:

Input: It may accept a zero or more inputs.

Output: It should produce at least one output (result).

Definiteness: Each instruction must be clear, well defined and precise. There should not be any ambiguity.

Finiteness: It should be a sequence of finite instructions. That is, it should end after a fixed time. It should not enter into an infinite loop.

Effectiveness: This means that operations must be simple and are carried out in a finite time at one or more levels of complexity. It should be effective whenever tracedmanually for the results.

Key features of an algorithm: Any algorithm has a finite number of steps and some steps may involve decision making, repetition. Broadly speaking, an algorithm exhibits three key features that can be given as:

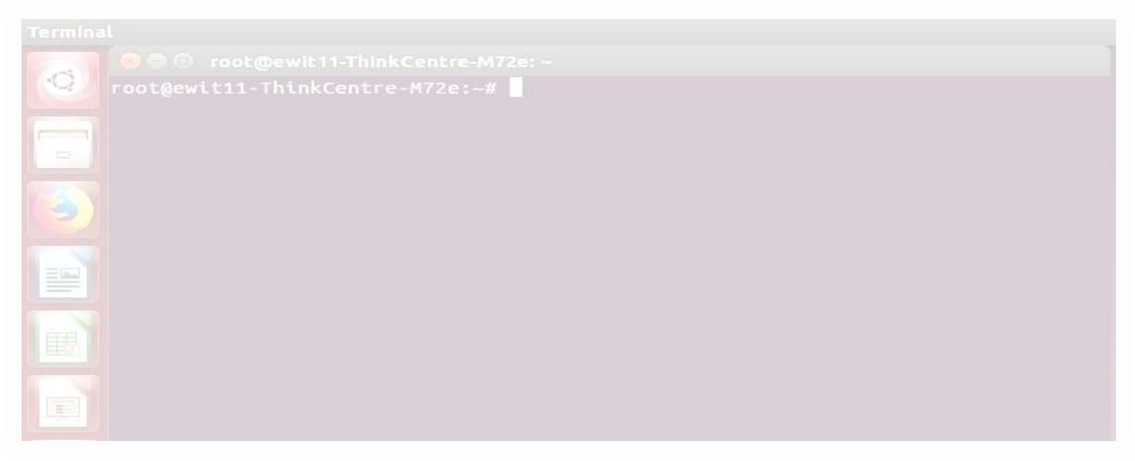
Sequence: Sequence means that each step of the algorithm is executed in the specified order.

Decision: Decision statements are used when the outcome of the process depends on some condition.

Repetition: Repetition which involves executing one or more steps for a number of times can be implemented using constructs like the while, do-while and for loops. These loops executed one or more steps until some condition is true.

C PROGRAMMING WITH Ubuntu

Step1: Open the terminal window:



```
root@ewit11-ThinkCentre-M72e:~#
```

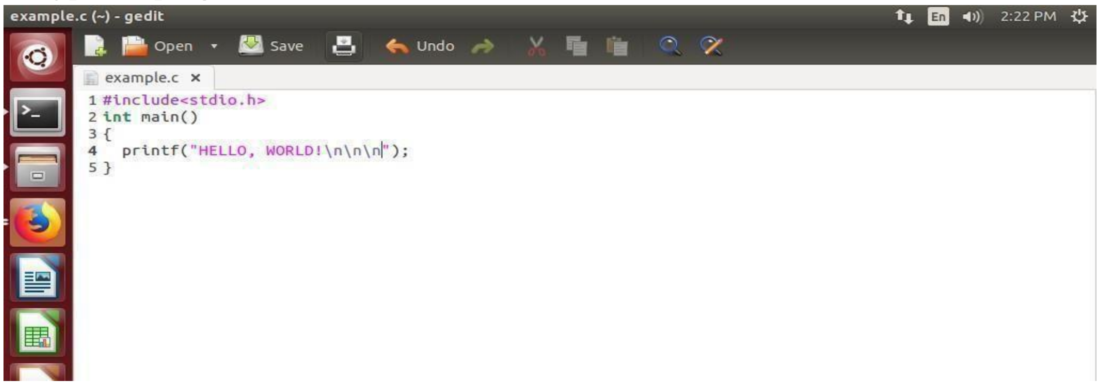
Step 2: Create a directory:
To create a directory, type the following command and press enter.
Syntax: mkdir<directoryname>

Step 3: Use the created directory:
To use the created directory use the change directory command. **Syntax : cd <directoryname>**

Step 4: Use the text editors:
There are several text editors already installed in Ubuntu OS like Gedit, vim,etc. To use Gedit text editor type the following command **Syntax : gedit<filename.c>**

As we are programming with C, the extension will be ".c". This is how a Gedit text editor of file name 'example.c' would look.

Type the program in the Gedit text editor save it and then close the text editor.



```
#include<stdio.h>
int main()
{
  printf("HELLO, WORLD!\n\n\n");
}
```

Step 5: Compiling the saved program:
To compile the saved program, go to the terminal window and type the above command. Syntax: gcc filename

Step 6: Get the Output
To get the output use the following command in the terminal Window.Syntax:
./a. out

NOTE: When using the 'math.h' header file in the C program, use the following command to compile. **Syntax: gcc -lm filename.**

PROGRAM-1

Develop a C program to implement the Process System calls (fork(), exec(), wait(), create process and terminate process)

AIM:

To write a program for implementing process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

ALGORITHM:

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID
4. execvp() system call used to execute that command given on that command line argument
5. execlp() system call used to execute specified command.
6. Open the directory at specified in command line input.
7. Display the directory contents.
8. Stop the program.

PROGRAM

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

void main()
{
        pid_t pid = 0;
        int status;
        pid = fork();

      if (pid == 0)
      {
              printf("I am the child.");
              execl("/bin/ls", "ls", "-l", "/home/ubuntu/", (char *) 0);
              perror("In exec(): ");
      }

      if (pid > 0)
      {
              printf("I am the parent %d, and the child is %d.\n", getppid(), pid);
             pid = wait(&status);
            printf("End of process %d: ", pid);

            if (WIFEXITED(status))
            {
                printf("The process ended with exit(%d).\n", WEXITSTATUS(status));
        }
```

```
            if (WIFSIGNALED(status))
           {
              printf("The process ended with kill -%d.\n", WTERMSIG(status));
             }
      }

      if (pid < 0)
      {
         perror("In fork():");
        }
   exit(0);

   }
```

VIVA QUESTIONS:

1. What is the purpose of system calls?
2. What system calls have to be executed by a command interpreter or shell in order to start a new process?
3. When a process creates a new process using the fork() operation, which of the following state is shared between the parent process and the child process?
4. What is the use of exec system call?
5. What system call is used for closing a file?
6. What is the value return by close system call?
7. What is the system call is used for writing to a file.

8, what are system calls used for creating and removing directories?

OUTPUT

```
I am the parent 223, and the child is 228.
ls: cannot access '/home/ubuntu/': No such file or directory
End of process 228: The process ended with exit(2).
```

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-2

Simulate the following Scheduling algorithms to find Turnaround Time and Waiting Time

AIM

To write a C program to implement First Come First Serve scheduling algorithm.

DESRIPTION:

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.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on order in which it requests CPU.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time.
Step 6: Print the details about all the processes.
Step 7: Stop the program.

PROGRAM

FCFS CPU Scheduling

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int bt[20], wt[20], tat[20], i, n;
    float wtavg, tatavg;
    //clrscr();
    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("\n \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 -- 3
Enter Burst Time for Process 0 -- 5 1 7

Enter Burst Time for Process 1 --
Enter Burst Time for Process 2 --
      PROCESS      BURST TIME      WAITING TIME    TURNAROUND TIME

      P0           5           0           5
      P1           1           5           6
      P2           7           6           13
Average Waiting Time -- 3.666667
Average Turnaround Time -- 8.000000
```

SJF CPU Scheduling

AIM
To write a C program to implement shortest job first (non-pre-emptive) scheduling algorithm.

DESCRIPTION:
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.

ALGORITHM:
Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on burst time.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time.
Step 6: Print the details about all the processes.
Step 7: Stop the program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg,
	tatavg;
	//clrscr();
	printf("\nEnter the number of processes -- ");
	scanf("%d", &n);
	for(i=0;i<n;i++)
	{
		p[i]=i;
		printf("Enter Burst Time for Process %d -- ", i);
		scanf("%d", &bt[i]);
	}
	for(i=0;i<n;i++)
	for(k=i+1;k<n;k++)
	if(bt[i]>bt[k])
	{
		temp=bt[i];
		bt[i]=bt[k];
		bt[k]=temp;
		temp=p[i];
```

```
                p[i]=p[k];
                p[k]=temp;
        }
        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("\n\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", p[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 -- 6
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 7
Enter Burst Time for Process 3 -- 3

      PROCESS     BURST TIME     WAITING TIME   TURNAROUND TIME

            P3        3               0                3
            P0        6               3                9
            P2        7               9                16
            P1        8               16               24
Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000
```

Round Robin CPU Scheduling

AIM:
To write a C program to implement Round Robin scheduling algorithm.

DESCRIPTION:
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.

ALGORITHM:
Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on priority.
Step 4: Compute the waiting time and turnaround time for each process. Step 5: Calculate the average waiting time and average turnaround time. Step 6: Print the details about all the processes.
Step 7: Stop the program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        //clrscr();
        printf("Enter the no of processes -- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter Burst Time for process %d -- ", i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
        printf("\nEnter the size of time slice -- ");
        scanf("%d",&t);
        max=bu[0];
        for(i=1;i<n;i++)
        if(max<bu[i])
                max=bu[i];

        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0)
                                if(bu[i]<=t)
                                {
                                        tat[i]=temp+bu[i];
                                        temp=temp+bu[i];
                                        bu[i]=0;
                        }
                        else
                        {
                                bu[i]=bu[i]-t;
                                temp=temp+t;
                        }
        for(i=0;i<n;i++)
        {
                wa[i]=tat[i]-
                ct[i]; att+=tat[i];
                awt+=wa[i];
        }
        printf("\nThe Average Turnaround time is -- %f",att/n);
        printf("\nThe Average Waiting time is -- %f ",awt/n);
        printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
        for(i=0;i<n;i++)
                printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

        getch();
}
```

OUTPUT:

```
Enter the no of processes -- 3
Enter Burst Time for process 1 -- 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 -- 3
Enter the size of time slice -- 3

The Average Turnaround time is -- 15.000000
The Average Waiting time is -- 5.000000
    PROCESS  BURST TIME      WAITING TIME   TURNAROUND TIME
         1        24                 6          30
         2        3                  3          6
         3        3                  6          9
```

RUN2:

```
Enter the no of processes -- 3
Enter Burst Time for process 1 -- 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 -- 3
Enter the size of time slice -- 2

The Average Turnaround time is -- 16.333334
The Average Waiting time is -- 6.333333

 PROCESS  BURST TIME      WAITING TIME   TURNAROUND TIME
    1              24         6                 30
    2              3          6                 9
    3              3          7                 10
```

Priority CPU Scheduling

AIM

To write a C program to implement Priority Scheduling algorithm.

DESCRIPTION:

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**.**

ALGORITHM:

Step 1: Start the program.
Step 2: Get the input process and their burst time.
Step 3: Sort the processes based on priority.
Step 4: Compute the waiting time and turnaround time for each process.
Step 5: Calculate the average waiting time and average turnaround time.
Step 6: Print the details about all the processes.
Step 7: Stop the program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg,
        tatavg;
        //clrscr();
        printf("Enter the number of processes --- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
            p[i] = i;
            printf("Enter the Burst Time & Priority of Process %d --- ",i);
            scanf("%d%d",&bt[i], &pri[i]);
        }
        for(i=0;i<n;i++)
        for(k=i+1;k<n;k++)
        if(pri[i] > pri[k])
        {
            temp=p[i];
            p[i]=p[k];
            p[k]=temp;
```

```
                        temp=bt[i];
                        bt[i]=bt[k];
                        bt[k]=temp;
                        temp=pri[i];
                        pri[i]=pri[k];
                        pri[k]=temp;
                }
                wtavg = wt[0] = 0;
                tatavg = tat[0] = 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("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND
        TIME");
                for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
                printf("\nAverage Waiting Time is --- %f",wtavg/n);
                printf("\nAverage Turnaround Time is --- %f",tatavg/n);
                getch();
}
```

OUTPUT:

```
Enter the number of processes --- 5
Enter the Burst Time & Priority of Process 0 --- 10 3
Enter the Burst Time & Priority of Process 1 --- 1 1
Enter the Burst Time & Priority of Process 2 --- 2 4
Enter the Burst Time & Priority of Process 3 --- 1 5
Enter the Burst Time & Priority of Process 4 --- 5 2

PROCESS      PRIORITY      BURST TIME      WAITING TIME    TURNAROUND TIME
   1             1             1                 0           1
   4             2             5                 1           6
   0             3             10                6           16
   2             4             2                 16           18
   3             5             1                 18           19
Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000
```

VIVA QUESTIONS:
1. Define operating system?
2. What are the different types of operating systems?
3. Define a process?
4. What is CPU Scheduling?
5. Define arrival time, burst time, waiting time, turnaround time?
6. What is the advantage of round robin CPU scheduling algorithm?
7. Which CPU scheduling algorithm is for real-time operating system?
8. In general, which CPU scheduling algorithm works with highest waiting time?
9. Is it possible to use optimal CPU scheduling algorithm in practice?
10. What is the real difficulty with the SJF CPU scheduling algorithm?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-3

Develop a C program to simulate Producer – Consumer problem using semaphores.

AIM:
To write a C program to implement the Producer & consumer Problem (Semaphore)

DESCRIPTION:
Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

ALGORITHM:

```
Step 1: The Semaphore mutex, full & empty are initialized.
Step 2: In the case of producer process
        i) Produce an item in to temporary variable.
        ii) If there is empty space in the buffer check the mutex value for enter into the critical section.
        iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.
Step 3: In the case of consumer process
        i) It should wait if the buffer is empty
        ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
        iii) Signal the mutex value and reduce the empty value by 1.
        iv) Consume the item.
Step 4: Print the result
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int buffer[10], bufsize, in, out, produce, consume,choice=0;
        in = 0;
        out = 0;
        bufsize = 10;
        while(choice !=3)
        {
                printf(" \n1. Produce \t 2. Consume \t3. Exit");
                printf(" \nEnter your choice: ");
                scanf("%d", &choice);
                switch(choice)
              {
                case 1:
                        if((in+1)%bufsize==out)
                                printf("\nBuffer is Full");
                        else
                        {
                                printf("\nEnter the value: ");
                                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 consumed value is %d", consume);
                                    out = (out+1)%bufsize;
                           }
                           break;
                    }
            }
}
```

OUTPUT:

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

Enter the value: 100

1. Produce     2. Consume    3. Exit
Enter your choice: 2

The consumed value is 100
1. Produce     2. Consume    3. Exit
Enter your choice: 1

Enter the value: 100

1. Produce     2. Consume    3. Exit
Enter your choice: 1

Enter the value: 100
```

```
1. Produce      2. Consume     3. Exit
Enter your choice: 2

The consumed value is 100
1. Produce      2. Consume     3. Exit
Enter your choice: 2

The consumed value is 100
1. Produce      2. Consume     3. Exit
Enter your choice: 2

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

VIVA QUESTIONS
1. Define Semaphore?
2. What is use of wait and signal functions?
3. What is mutual exclusion?
4. Define producer consumer problem?
5. What is the need for process synchronization?
6. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem?
7. Can producer and consumer processes access the shared memory concurrently?
If not which technique provides such a benefit?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-4

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

AIM:
To write a C program to implement demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs

DESCRIPTION:
In reader writer process the named pipes provide a mechanism for interprocess communication between a reader and a writer process. The writer opens the pipe for writing, and the reader opens it for reading. Data is written to the named pipe by the writer and can be read by the reader, facilitating communication between the two processes. Once communication is complete, the named pipe can be unlinked to clean up system resources.

Algorithm:

1. **Create a Named Pipe (FIFO)**:
   - Use the `mkfifo` function to create a named pipe. This should be done before running the reader and writer processes.
2. **Writer Process**:
   - Open the named pipe for writing using the `open` function:
   - Check for errors in opening the pipe.
   - Prepare the data to be written and store it in a buffer.
   - Use the `write` function to send data to the named pipe:
   - Close the file descriptor to release the named pipe:
3. **Reader Process**:
   - Open the named pipe for reading using the `open` function:
   - Check for errors in opening the pipe.
   - Create a buffer to receive the data.
   - Use the `read` function to read data from the named pipe:
   - Close the file descriptor when you're done reading data:
4. **Error Handling**:
   - Ensure proper error handling for `mkfifo`, `open`, `read`, and `write` functions to handle cases like failed pipe creation, opening, and reading/writing.
5. **Cleanup**:
   - After communication is complete, both the writer and reader processes should unlink the named pipe using the `unlink` function to remove the pipe:
6. **End of Processes**:
   - Terminate the reader and writer processes when they have completed their tasks.

Writer Process

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/type.h> // for ‘mkfifo()’ and permission modes.
#include <unistd.h>
int main()
 {
 int fd; // declare file descriptor
 char buf[1024];
 /* create the FIFO (named pipe) */
char * myfifo = "/tmp/myfifo"; //defining the path of FIFO
 mkfifo(myfifo, 0666) //Create named pipe with read and write permissions.
; printf("Run Reader process to read the FIFO File\n");
 fd = open(myfifo, O_WRONLY); //opens FIFO for write operation
write(fd,"Hi", sizeof("Hi")); // write the string “Hi” to the FIFO
close(fd); //close the file descriptor
unlink(myfifo);  // remove the FIFO
return 0;
 }
```

Reader 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];
    char * myfifo = "/tmp/myfifo";
    // Open the FIFO for reading
    fd = open(myfifo, O_RDONLY);

    // Read from the FIFO
    read(fd, buf, sizeof(buf));

    // Print the read message
    printf("Received: %s\n", buf);
buf[sizeof(buf) – 1] = ‘\0’; // initialize buf to zero to avoid unexpected result as buf contain previously
stored data.

    close(fd);

    return 0;
}
```

OUTPUT

To test the communication between the writer and reader programs:

Compile both programs:

```
gcc writer.c -o writer

gcc reader.c -o reader
```

Run the writer in one terminal:

```
./writer
```

This will create the FIFO, and write "Hi" into it.

Run the reader in another terminal:

```
./reader
```

This will read the message from the FIFO and display "Received: Hi" on the terminal.

VIVA QUESTIONS

1. What is IPC?
2. What is the use shared memory?
3. List commands used for shared memory communication?
4. What is the function of shmget function?
5. What is the use of shmctl fuction?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM 5

Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance

AIM:
To Simulate Algorithm for Deadlock avoidance

DESCRIPTION:
In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type

ALGORITHM:
Step 1: Start the Program
Step 2: Get the values of resources and processes.
Step 3: Get the avail value.
Step 4: After allocation find the need value.
Step 5: Check whether its possible to allocate. If possible it is safe state
Step 6: If the new request comes then check that the system is in safety or not if we allow the request.
Step 7: Stop the execution.
Step 8: Stop the program

PROGRAM:

```
#include<string.h>
#include<stdlib.h>
void main()
{
        int alloc[10][10],max[10][10];
        int avail[10],work[10],total[10],done[10];
        int i,j,k,n,need[10][10];
        int m,z=0;
        int count=0,c=0;
        char finish[10];
        //clrscr();
        printf("Enter the no. of processes and resources:");
        scanf("%d%d",&n,&m);
        for(i=0;i<=n;i++)
```

```
            finish[i]='n';

    printf("Enter the claim matrix:\n");
    for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                    scanf("%d",&max[i][j]);

    printf("Enter the allocation matrix:\n");
    for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                    scanf("%d",&alloc[i][j]);

    printf("Resource vector:");
    for(i=0;i<m;i++)
            scanf("%d",&total[i]);

    for(i=0;i<m;i++)
            avail[i]=0;

    for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                    avail[j]+=alloc[i][j];

    for(i=0;i<m;i++)
            work[i]=avail[i];

    for(j=0;j<m;j++)
            work[j]=total[j]-work[j];

    for(i=0;i<n;i++)
            for(j=0;j<m;j++)
                    need[i][j]=max[i][j]-alloc[i][j];

    A:
    for(i=0;i<n;i++)
    {
            c=0;
            for(j=0;j<m;j++)
                    if((need[i][j]<=work[j])&&(finish[i]=='n'))
                            c++;
                    if(c==m)
                    {
                            printf("All the resources can be allocated to Process %d", i+1);
                            printf("\n\nAvailable resources are:");
                            for(k=0;k<m;k++)
                            {
                                    work[k]+=alloc[i][k];
                                    printf("%4d",work[k]);
                            }
```

```
                        printf("\n");
                        finish[i]='y';
                        done[z++]=i+1;
                        printf("%d\t", finish[i]);
                        printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
                        count++;
                        }
        }

        if(count!=n)
                goto A;
        else
        printf("\n System is in safe mode");

        printf("\n The given state is safe state");
        printf("\n");
        for(z=0;z<n;z++)
                printf("\t%d\t", done[z]);

        getch();
}
```

OUTPUT:

```
Enter the no. of processes and resources:4 3
Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2

Available resources are:6       2       3
Process 2 executed?:y
All the resources can be allocated to Process 3

Available resources are:8       3       4
Process 3 executed?:y
All the resources can be allocated to Process 4

Available resources are:8       3       6
Process 4 executed?:y
All the resources can be allocated to Process 1
```

```
Available resources are:9       3       6
Process 1 executed?:y

 System is in safe mode
 The given state is safe state
2       3       4       1
```

VIVA QUESTIONS:

1. How to recover once the Deadlock has been detected?
2. List the steps to illustrate the Deadlock Detection Algorithm.
3. What are the advantages to check each resource request?
4. .How to fill the Allocation matrix?
5. How to identify whether the process exist in deadlock or not?
6. Define Deadlock Prevention.
7. List the difference Between Starvation and Deadlock.
8. Give the advantages of Deadlock.
9. List the disadvantages of Deadlock method.
10. Define resource. Give examples.
11. What are the conditions to be satisfied for the deadlock to occur?
12. How can be the resource allocation graph used to identify a deadlock situation?
13. How is Banker's algorithm useful over resource allocation graph technique?
14. Differentiate between deadlock avoidance and deadlock prevention?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM 6

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

AIM:
To write a C program to implement Memory Management concept using the technique best fit, worst fit and first fit algorithms.

ALGORITHM:
1. Get the number of process.
2. Get the number of blocks and size of process.
3. Get the choices from the user and call the corresponding switch cases.
4. First fit -allocate the process to the available free block match with the size of the process
5. Worst fit –allocate the process to the largest block size available in the list
6. Best fit-allocate the process to the optimum size block available in the list
7. Display the result with allocations

PROGRAM:

Worst Fit

```
#include<stdio.h>
#include<conio.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];
        //clrscr();
        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]);

	getch();
}
```

OUTPUT:

```
Memory Management Scheme - Worst 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             3               7              6
    2            4             1               5              1
```

Best fit

```
#include<stdio.h>
#include<conio.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];
	//clrscr();
	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\tBlockSize\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]);

	getch();
}
```

OUTPUT:

```
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     BlockSize    Fragment
     1          1            2             2            1
     2          4            1             5            1
```

First Fit

```
#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];
	//clrscr();
	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: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
```

VIVA QUESTIONS

1. What is Memory Management?
2. Why Use Memory Management?
3. List the memory allocation techniques
4. Define Best fit and its advantage?
5. What is the use of First fit and worst fir methods?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-7

Develop a C program to simulate page replacement algorithm:

a. FIFO b. LRU

AIM: To write a C program to implement FIFO page replacement algorithm.

DESCRIPTION :

The FIFO Page Replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen . There is not strictly necessary to record the time when a page is brought in. By creating a FIFO queue to hold all pages in memory and by replacing the page at the head of the queue. When a page is brought into memory, insert it at the tail of the queue.

ALGORITHM:

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

PROGRAM:

FIFO

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
	void display();
	int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
	int flag1=0,flag2=0,pf=0,frsize=3,top=0;
//clrscr();
	for(i=0;i<3;i++)
	{
		fr[i]=-1;
	}
	for(j=0;j<12;j++)
	{
		flag1=0; flag2=0;
		for(i=0;i<12;i++)
		{
			if(fr[i]==page[j])
			{
```

```
                                        flag1=1; flag2=1; break;
                                }
                        }
                        if(flag1==0)
                        {
                                for(i=0;i<frsize;i++)
                                {
                                        if(fr[i]==-1)
                                        {
                                                fr[i]=page[j]; flag2=1; break;
                                        }
                                }
                        }
                        if(flag2==0)
                        {
                                fr[top]=page[j];
                                top++;
                                pf++;
                                if(top>=frsize)
                                        top=0;
                        }
                display();
                }

                printf("Number of page faults : %d ",pf+frsize);

                getch();
}
void display()
{
int i; printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2     -1     -1
2      3     -1
2      3     -1
2      3      1
5      3      1
5      2      1
5      2      4
5      2      4
3      2      4
3      2      4
3      5      4
3      5      2      Number of page faults : 9
```

LRU

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

DESCRIPTION:
The Least Recently Used replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes the recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.

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

PROGRAM:
```
void main()
{
        void display();
        int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
        int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
        //clrscr();
        for(i=0;i<3;i++)
        {
                fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
                flag1=0,flag2=0;
                for(i=0;i<3;i++)
                {
                        if(fr[i]==p[j])
                        {
                                flag1=1;
                                flag2=1; break;
                        }
                }
                if(flag1==0)
                {
                        for(i=0;i<3;i++)
                        {
                                if(fr[i]==-1)
                                {
                                        fr[i]=p[j]; flag2=1;
                                        break;
                                }
                        }
                }

                if(flag2==0)
                {
                        for(i=0;i<3;i++)
                        fs[i]=0;

                        for(k=j-1,l=1;l<=frsize-1;l++,k--)
                        {
                                for(i=0;i<3;i++)
                                {
                                        if(fr[i]==p[k])
                                                fs[i]=1;
                                }
                        }
                        for(i=0;i<3;i++)
                        {
                                if(fs[i]==0)
                                index=i;
                        }
                        fr[index]=p[j];
                        pf++;
                }

                display();
        }
        printf("\n no of page faults :%d",pf+frsize);

        getch();
}

void display()
{
        int i;
        printf("\n");
        for(i=0;i<3;i++)
                printf("\t%d",fr[i]);
}
```

OUTPUT:

```
    2    -1    -1
    2    3     -1
    2    3     -1
    2    3     1
```

```
        2       5       1
        2       5       1
        2       5       4
        2       5       4
        3       5       4
        3       5       2
        3       5       2
        3       5       2
 no of page faults :7
```

VIVA QUESTIONS

1. What is the use Memory Management?
2. Define Memory Management Techniques
3. What is Swapping?
4. What is paging?
5. What are the advantages of non-contiguous memory allocation schemes?
6. What is the process of mapping a logical address to physical address with respect to the paging memory management technique?
7. Define the terms – base address, offset?
8. Differentiate between paging and segmentation memory allocation techniques?
9. What is the purpose of page table?
10. Whether the paging memory management technique suffers with internal or external fragmentation problem. Why?
11.What is the purpose of page replacement?
12. Define page fault?
13. Which replacement algorithms suffers from Belady's anomaly?
14. Reference bit is used in which page replacement algorithm?
15. What is LRU page replacement?
16. Define optimal page replacement.
17. Define the concept of thrashing? What is the scenario that leads to the situation of thrashing?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-8

Simulate following File Organization Techniques:

a. Single level directory b. Two level directory

Single level directory

```
#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];
	//clrscr();
	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\n
		Enter your choice -- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1: printf("\nEnter the name of the file -- ");
				scanf("%s",dir.fname[dir.fcnt]);
				dir.fcnt++;
				break;
			case 2: printf("\nEnter the name of the file -- ");
				scanf("%s",f);
				for(i=0;i<dir.fcnt;i++)
				{
					if(strcmp(f, dir.fname[i])==0)
					{
						printf("File %s is deleted ",f);
						strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
						break;
					}
				}
				if(i==dir.fcnt)
					printf("File %s not found",f);
				else
```

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

        getch();
}
```

OUTPUT:

```
Enter name of directory -- AIML

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

Enter the name of the file -- a
File a not found

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

Enter the name of the file -- A

1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 1
Enter the name of the file -- B

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

Enter the name of the file -- C

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

The Files are --        A       B       C

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

Enter the name of the file -- D
File D not found

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

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

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

The Files are --        A       C

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

Enter the name of the file -- B
File B not found

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

Enter the name of the file -- C
File C is found
1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice – 5
```

Program:

TWO LEVEL DIRECTORY

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];

void main()
{
        int i,ch,dcnt,k;
        char f[30], d[30];
        //clrscr();
        dcnt=0;
        while(1)
        {
                printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
                printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter name of directory -- ");
                                scanf("%s", dir[dcnt].dname);
                                dir[dcnt].fcnt=0;
                                dcnt++;
                                printf("Directory created"); break;
                        case 2: printf("\nEnter name of the directory -- ");
                                scanf("%s",d);
                                for(i=0;i<dcnt;i++)
                                if(strcmp(d,dir[i].dname)==0)
                                {
                                        printf("Enter name of the file -- ");
                                        scanf("%s",dir[i].fname[dir[i].fcnt]);
                                        dir[i].fcnt++;
                                        printf("File created");
                                }
                                if(i==dcnt)
                                        printf("Directory %s not found",d);
                                break;
                        case 3: printf("\nEnter name of the directory -- ");
                                scanf("%s",d);
                                for(i=0;i<dcnt;i++)
                                        for(i=0;i<dcnt;i++)
                                        {
                                                if(strcmp(d,dir[i].dname)==0)
                                        {
                                                printf("Enter name of the file -- ");
                                                scanf("%s",f);
                                                for(k=0;k<dir[i].fcnt;k++)
                                                {
                                                        if(strcmp(f, dir[i].fname[k])==0)
                                                        {
                                                                printf("File %s is deleted ",f);
                                                                dir[i].fcnt--;
                                                                strcpy(dir[i].fname[k],dir[i].fname[
                                                                dir[i].fcnt]);
                                                                goto jmp;
                                                        }
                                                }
                                                printf("File %s not found",f); goto jmp;
                                        }
                                }
                        printf("Directory %s not found",d);
                        jmp : break;
        case 4: printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                {
                        if(strcmp(d,dir[i].dname)==0)
                        {
                                printf("Enter the name of the file -- ");
                                scanf("%s",f);
                                for(k=0;k<dir[i].fcnt;k++)
                                {
                                        if(strcmp(f, dir[i].fname[k])==0)
                                        {
                                                printf("File %s is found ",f); goto jmp1;
                                        }
                                }
                        printf("File %s not found",f); goto jmp1;
                        }
                }

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

        getch();
}
```

OUTPUT:

```
1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --1

Enter name of directory -- DIR1
Directory created

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --1

Enter name of directory -- DIR2
Directory created

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --2

Enter name of the directory -- DIR2
Enter name of the file -- B1
File createdDirectory DIR2 not found

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --3

Enter name of the directory -- DIR1
Enter name of the file -- B1
File B1 not found

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --5

Directory       Files
DIR1
DIR2                    B1

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --4

Enter name of the directory -- DIR1
Enter the name of the file -- B2
File B2 not found
```

```
1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --4

Enter name of the directory -- DIR2
Enter the name of the file -- B1
File B1 is found

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --4

Enter name of the directory -- DIR1
Enter the name of the file -- B1
File B1 not found

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --3

Enter name of the directory -- DIR2
Enter name of the file -- B1
File B1 is deleted

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --5

Directory       Files
DIR1
DIR2

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit  Enter your choice --6
```

Rubrics for Continuous Evaluation-2022 Scheme (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-9

Develop a C Program to simulate the Linked File Allocation Strategies:

AIM:
To write a C program to implement File Allocation concept using the technique Linked List Technique.

ALGORITHM:

```
Step 1: Start the Program
Step 2: Get the number of files.
Step 3: Allocate the required locations by selecting a location randomly
Step 4: Check whether the selected location is free.
Step 5: If the location is free allocate and set flag =1 to the allocated locations.
Step 6: Print the results file no, length, blocks allocated.
Step 7: Stop the execution
```

PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
        int f[50],p,i,j,k,a,st,len,n,c;
        //clrscr();
        for(i=0;i<50;i++)
                 f[i]=0;
        printf("Enter how many blocks that are already allocated");
        scanf("%d",&p);
        printf("\nEnter the blocks no.s that are already allocated");
        for(i=0;i<p;i++)
        {
                scanf("%d",&a);
                f[a]=1;
        }
X:
        printf("Enter the starting index block & length");
        scanf("%d%d",&st,&len);
        k=len;
        for(j=st;j<(k+st);j++)
        {
                if(f[j]==0)
                {
                        f[j]=1;
                        printf("\n%d->%d",j,f[j]);
                }
                else
                {
                        printf("\n %d->file is already allocated",j);
                        k++;
                }
        }
        printf("\n If u want to enter one more file? (yes-1/no-0)");
        scanf("%d",&c);
        if(c==1)
                goto X;
        else
                exit(0);

        getch();
}
```

OUTPUT:

```
Enter how many blocks that are already allocated3

Enter the blocks no.s that are already allocated4 7 5
Enter the starting index block & length3 1

3->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length4 1

 4->file is already allocated
 5->file is already allocated
6->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length7 1

 7->file is already allocated
8->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length9 1

9->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length10 1

10->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length11 1

11->1
 If u want to enter one more file? (yes-1/no-0)1
Enter the starting index block & length12 1

12->1
 If u want to enter one more file? (yes-1/no-0)0
```

VIVA QUESTIONS:
1. Define file?
2. What are the different kinds of files?
3. What is the purpose of file allocation strategies?
4. Identify ideal scenarios where sequential, indexed and linked file allocation strategies are most appropriate?
5. What are the disadvantages of sequential file allocation strategy?
6. What is an index block?
7. What is the file allocation strategy used in UNIX?

RubricsforContinuousEvaluation-2022Scheme (15Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty

PROGRAM-10

Develop a C Program to simulate SCAN disk scheduling algorithm

AIM:
Write a C program to simulate disk scheduling algorithms SCAN.

DESCRIPTION
One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards 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. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

PROGRAM:

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

#define LOW 0
#define HIGH 1

// Function to sort the array
void sort(int arr[], int n)
  {
   for (int i = 0; i < n-1; i++)
  {
       for (int j = 0; j < n-i-1; j++)
    {
           if (arr[j] > arr[j+1])
        {
               int temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
           }
       }
    }
```

```
}

int main() {
    int n, initial, size, direction;
    int totalHeadMovement = 0;

    printf("Enter the number of requests: ");
    scanf("%d", &n);

    int requestQueue[n];
    printf("Enter the request sequence:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requestQueue[i]);
    }

    printf("Enter initial head position: ");
    scanf("%d", &initial);

    printf("Enter total disk size: ");
    scanf("%d", &size);

    printf("Enter the head movement direction (0 for LOW, 1 for HIGH): ");
    scanf("%d", &direction);
    sort(requestQueue, n);
    int currentPosition = initial;
    // Move in the chosen direction until the end or beginning is reached
    if (direction == HIGH) {
        for (int i = 0; i < n; i++) {
            if (requestQueue[i] >= currentPosition) {
                totalHeadMovement += abs(requestQueue[i] - currentPosition);
                currentPosition = requestQueue[i];
            }
        }

       if (currentPosition != size - 1) {
            totalHeadMovement += abs((size - 1) - currentPosition);
            currentPosition = size - 1;
        }

        // Move in the opposite direction
        for (int i = n - 1; i >= 0; i--) {
            if (requestQueue[i] < initial) {
                totalHeadMovement += abs(requestQueue[i] - currentPosition);
                currentPosition = requestQueue[i];
            }
        }
    }
                else {  // direction == LOW
        for (int i = n - 1; i >= 0; i--) {
            if (requestQueue[i] <= currentPosition) {
                totalHeadMovement += abs(requestQueue[i] - currentPosition);
                currentPosition = requestQueue[i];
            }
        }
       if (currentPosition != 0) {
            totalHeadMovement += currentPosition;
            currentPosition = 0;
        }
      // Move in the opposite direction
        for (int i = 0; i < n; i++) {
            if (requestQueue[i] > initial) {
                totalHeadMovement += abs(requestQueue[i] - currentPosition);
                currentPosition = requestQueue[i];
            }
        }
    }
        printf("Total head movement: %d\n", totalHeadMovement);
        return 0;
}
```

OUTPUT:

```
Enter the number of requests: 9
Enter the request sequence:
55
58
60
70
18
90
150
160
184

Enter initial head position: 53

Enter total disk size: 200

Enter the head movement direction (0 for LOW, 1 for HIGH): 0

Total head movement: 237
```

VIVA QUESTIONS:

1. What is disk scheduling?
2. List the different disk scheduling algorithms?
3. Define the terms – disk seek time, disk access time and rotational latency?
4. Define sequential file allocation?
5. What is the use of indexed file allocation?
6. What is the advantages if linked allocation?
7. What is the advantage of C-SCAN algorithm over SCAN algorithm?
8. Which disk scheduling algorithm has highest rotational latency? Why?

Rubrics for Continuous Evaluation (15 Marks)

Continuous Evaluation for15marks		**Marks Allotted**	**Marks Obtained**
A	Observation Writeup and Attendance	4	
B	Conduction of Experiment and Output	3	
C	Viva Voice	4	
D	Record Writeup	4	
Total		15	

Signature of Lab Faculty In-charge/Co-Faculty