CHANNABASAVESHWARA INSTITUTE OF TECHNOLOGY
Affiliated to VTU,Belagavi & Approved by AICTE,New-Delhi
(NAAC Accredited & ISO-9001:2015 Certified Institution)
NH 206,(B.H ROAD) GUBBI,TUMKUR – 572 216 Karnataka

Department of Information Science And Technology

(Year 2025-26)

Operating Systems laboratory

(BCSL303)

LAB MANUAL

Faculty Incharge **:**

Dr.Thara D K

(Dept. of ISE)

Ms.VARSHA N

(Dept.of ISE)

SYLLABUS

Course Title: Operating Systems laboratory

Course Code: BCSL303

Programming Exercises:

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.

TABLE OF CONTENTS

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

fork() system call

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

int main()
{
    int id, childid;
    id = getpid();

    if ((childid = fork()) > 0)
    {
        printf("\n I am in the parent process %d", id);
        printf("\n I am in the parent process %d", getpid());
        printf("\n I am in the parent process %d\n", getppid());
    }
    else
    {
        printf("\n I am in child process %d", id);
        printf("\n I am in the child process %d", getpid());
        printf("\n I am in the child process %d", getppid());
    }

    return 0;
}
```

OUTPUT:

```
I am in child process 3765
I am in the child process 3766
I am in the child process 3765
I am in the parent process 3765
I am in the parent process 3765
I am in the parent process 3680
```

wait() system call

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>  // Needed for exit()

int main()
{
    int i, pid;
    pid = fork();

    if (pid == -1)
    {
        printf("fork failed");
        exit(0);
    }
    else if (pid == 0)
    {
        printf("\n Child process starts");
        for (i = 0; i < 5; i++)
        {
            printf("\n Child process %d is called", i);
        }
        printf("\n Child process ends");
    }
```

```
    else
    {
        wait(0);
        printf("\n Parent process ends");
    }

    exit(0);
}
```

OUTPUT:

```
Child process 0 is called
Child process 1 is called
Child process 2 is called
Child process 3 is called
Child process 4 is called
Child process ends
Parent process ends
```

exec() system call

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

int main() {
    char *args[] = {"/bin/ls", "-l", NULL};
    execv("/bin/ls", args);
    printf("This line will not be executed\n");
    return 0;
}
```

OUTPUT :

```
total 40384
drwxrwxrwx  2 rextester_user407  rextester_user407  4096  Dec  30 18:42 1000030336
drwxr-xr-x  2 rextester_user289  rextester_user289  4096  Jun  22 2023  1000418078
drwxr-xr-x  2 root               root               4096  Jan  1  2022  1000524395
drwxr-xr-x  2 root               root               4096  Jan  1  2022  100068448
drwxr-xr-x  2 root               root               4096  Jan  1  2022  1000693863
```

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

//First Come First Serve (FCFS) Scheduling Algorithm

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

int main()
{
    char pn[10][10];
    int arr[10], bur[10], star[10], finish[10], tat[10], wt[10], i, n;
    float totwt = 0, tottat = 0;

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

    for (i = 0; i < n; i++)
    {
        printf("Enter the Process Name, Arrival Time & Burst Time: ");
        scanf("%s %d %d", &pn[i], &arr[i], &bur[i]);
    }

    for (i = 0; i < n; i++)
    {
        if (i == 0)
        {
            star[i] = arr[i];
        }
        else
        {
            star[i] = finish[i - 1];
        }

        if (star[i] < arr[i])  // Ensure CPU waits if process hasn't arrived
            star[i] = arr[i];

        finish[i] = star[i] + bur[i];
        tat[i] = finish[i] - arr[i];
        wt[i] = tat[i] - bur[i];
    }

    printf("\nPName\tArrTime\tBurTime\tStart\tTAT\tFinish\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%7d\t%7d\t%5d\t%4d\t%6d\t%3d",
            pn[i], arr[i], bur[i], star[i], tat[i], finish[i], wt[i]);

        totwt += wt[i];
        tottat += tat[i];
    }

    totwt = totwt / n;
    tottat = tottat / n;

    printf("\nAverage Waiting Time: %.2f", totwt);
    printf("\nAverage Turn Around Time: %.2f", tottat);

    getch();
    return 0;
}
```

OUTPUT :

```
Enter the number of processes:3
Enter the Process Name, Arrival Time & Burst Time:1 2 3
Enter the Process Name, Arrival Time & Burst Time:2 5 6
Enter the Process Name, Arrival Time & Burst Time:3 6 7
PName Arrtime     Burtime     Start    TAT    CompleteTime  WT
1          2          3          2          3          5          0
2          5          6          5          6          11         0
3          6          7          11         12         18         5
Average Waiting time:1.666667
Average Turn Around Time:7.000000
```

// Shortest Job First (SJF) Scheduling Algorithm

```c
#include <stdio.h>

int main()
{
    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, totalT = 0, pos, temp;
    float avg_wt, avg_tat;

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

    printf("\nEnter Burst Time:\n");
    for (i = 0; i < n; i++)
    {
        printf("p%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1; // process number
    }

    // Sorting burst times using selection sort (SJF)
    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
        {
            if (bt[j] < bt[pos])
                pos = j;
        }

        // Swap burst times
        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;

        // Swap process numbers
        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0; // first process has no wait

    // Calculate waiting time
    for (i = 1; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        total += wt[i];
    }

    // Calculate average waiting time
```

```
    avg_wt = (float)total / n;

    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i]; // TAT = BT + WT
        totalT += tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = (float)totalT / n;

    printf("\n\nAverage Waiting Time = %.2f", avg_wt);
    printf("\nAverage Turnaround Time = %.2f\n", avg_tat);

    return 0;
}
```

OUTPUT :

```
Enter number of process:4
Enter Burst Time:
p1:5
p2:4
p3:12
p4:7
Process  Burst Time      Waiting Time    Turnaround Time
p2               4               0                       4
p1               5               4                       9
p4               7               9                       16
p3               12              16                      28

Average Waiting Time=7.250000
Average Turnaround Time=14.250000
```

// Round Robin Scheduling algorithm

```
#include <stdio.h>

int main()
{
    int n;
    printf("Enter Total Number of Processes: ");
    scanf("%d", &n);

    int wait_time = 0, ta_time = 0;
    int arr_time[n], burst_time[n], temp_burst_time[n];
    int x = n;  // number of processes left

    // Input process details
    for (int i = 0; i < n; i++)
    {
        printf("Enter Details of Process %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &arr_time[i]);
        printf("Burst Time: ");
        scanf("%d", &burst_time[i]);
        temp_burst_time[i] = burst_time[i];  // store original burst time
    }

    int time_slot;
    printf("Enter Time Slot: ");
```

```
    scanf("%d", &time_slot);

    int total = 0, counter = 0, i;
    printf("\nProcess ID   Burst Time   Turnaround Time   Waiting Time\n");

    for (total = 0, i = 0; x != 0;)
    {
        if (temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
        {
            total += temp_burst_time[i];
            temp_burst_time[i] = 0;
            counter = 1;
        }
        else if (temp_burst_time[i] > 0)
        {
            temp_burst_time[i] -= time_slot;
            total += time_slot;
        }

        if (temp_burst_time[i] == 0 && counter == 1)
        {
            x--;
            printf("Process %2d   %10d   %16d   %13d\n", i + 1, burst_time[i],
                   total - arr_time[i], total - arr_time[i] - burst_time[i]);
            wait_time += total - arr_time[i] - burst_time[i];
            ta_time += total - arr_time[i];
            counter = 0;
        }

        if (i == n - 1)
            i = 0;
        else if (arr_time[i + 1] <= total)
            i++;
        else
            i = 0;
    }

    float avg_wait = (float)wait_time / n;
    float avg_tat = (float)ta_time / n;

    printf("\nAverage Waiting Time: %.2f", avg_wait);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

OUTPUT :

```
Enter Total Number of Processes: 3
Enter Details of Process 1
Arrival Time:  0
Burst Time:   10
Enter Details of Process 2
Arrival Time:  1
Burst Time:   8
Enter Details of Process 3
Arrival Time:  2
Burst Time:   7
Enter Time Slot:5
Process ID        Burst Time      Turnaround Time     Waiting Time

Process No 1              10              20                  10
Process No 2              8               22                  14
Process No 3              7               23                  16
 Average Waiting Time:13.333333
Avg Turnaround Time:21.666666
```

// Priority scheduling algorithm

```
#include <stdio.h>
#define max 5

int main()
{
    int i, j, n, t, p[max], bt[max], pr[max], wt[max], tat[max];
    int Total_wt = 0, Total_tat = 0;
    float awt = 0, atat = 0;

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

    // Input process details
    for (i = 0; i < n; i++)
    {
        printf("Enter the process number: ");
        scanf("%d", &p[i]);

        printf("Enter the burst time of the process: ");
        scanf("%d", &bt[i]);

        printf("Enter the priority of the process: ");
        scanf("%d", &pr[i]);
    }

    // Sort processes based on priority using bubble sort
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (pr[j] > pr[j + 1])
            {
                // Swap priorities
                t = pr[j];
                pr[j] = pr[j + 1];
                pr[j + 1] = t;

                // Swap burst times
                t = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = t;

                // Swap process IDs
                t = p[j];
                p[j] = p[j + 1];
                p[j + 1] = t;
            }
        }
    }

    // Calculate Waiting Time and Turnaround Time
    printf("\nProcessID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];

        tat[i] = wt[i] + bt[i];

        Total_wt += wt[i];
        Total_tat += tat[i];

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);
    }

    // Calculate and display averages
```

```
    awt = (float)Total_wt / n;
    atat = (float)Total_tat / n;

    printf("\nAverage Waiting Time = %.2f\n", awt);
    printf("Average Turnaround Time = %.2f\n", atat);

    return 0;
}
```

OUTPUT :

```
Enter the number of processes
2
Enter the process number
1
Enter the burst time of the process
5
Enter the priority of the process
3
Enter the process number
2
Enter the burst time of the process
4
Enter the priority of the process
2
Processid       Burst Time      Priority        Waiting Time    Turn Around Time
2               4               2               0               4
1               5               3               4               9
The average waiting time =  2.000000
The average turn aroud time = 6.500000
```

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

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

// Initialize a mutex to 1
int mutex = 1;

// Number of full slots as 0
int full = 0;

// Number of empty slots as size of buffer
int empty = 10, x = 0;

// Function to produce an item and add it to the buffer
void producer()
{
    --mutex;
    ++full;
    --empty;
    x++;
    printf("\nProducer produces item %d", x);
    ++mutex;
}

// Function to consume an item and remove it from buffer
void consumer()
{
    --mutex;
    --full;
    ++empty;
    printf("\nConsumer consumes item %d", x);
    x--;
    ++mutex;
}

int main()
{
    int n, i;
    printf("1. Press 1 for Producer");
    printf("\n2. Press 2 for Consumer");
    printf("\n3. Press 3 for Exit");

    for (i = 1; i > 0; i++) {
        printf("\n\nEnter your choice: ");
        scanf("%d", &n);

        switch (n) {
            case 1:
                if ((mutex == 1) && (empty != 0)) {
                    producer();
                } else {
                    printf("\nBuffer is full!");
                }
                break;

            case 2:
                if ((mutex == 1) && (full != 0)) {
                    consumer();
                } else {
                    printf("\nBuffer is empty!");
                }
                break;

            case 3:
                exit(0);
```

```
            default:
                printf("\nInvalid choice!");
        }
    }

    return 0;
}
```

OUTPUT :

```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1
Producer producesitem 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:2
Buffer is empty!
Enter your choice:1
Producer producesitem 1
Enter your choice:1
Producer producesitem 2
Enter your choice:1
1Producer producesitem 3
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!
```

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.

/*Writer Process*/

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char *myfifo = "/tmp/myfifo";

    // Create the FIFO (named pipe) with read-write permissions for all users
    mkfifo(myfifo, 0666);

    printf("Run Reader process to read the FIFO file\n");

    // Open the FIFO for writing only
    fd = open(myfifo, O_WRONLY);

    // Write "Hi" to the FIFO
    write(fd, "Hi", sizeof("Hi"));

    // Close the FIFO descriptor
    close(fd);

    // Remove the FIFO file
    unlink(myfifo);

    return 0;
}
```

/*Reader Process*/

```c
#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;
    char *myfifo = "/tmp/myfifo";
    char buf[MAX_BUF];

    // Open FIFO for reading only
    fd = open(myfifo, O_RDONLY);

    // Read data from FIFO into buffer
    read(fd, buf, MAX_BUF);

    // Display the message read from FIFO
    printf("Writer: %s\n", buf);

    // Close FIFO descriptor
    close(fd);

    return 0;
}
```

Terminal 1 (Reader):

```
$ ./reader
```

Terminal 2 (Writer):

```
$ ./writer
Run Reader process to read the FIFO file
Producer produces item 1
```

Output on **Reader Terminal**:

```
Writer: Hi
```

Output on **Writer Terminal**:

```
Run Reader process to read the FIFO file
```

5) Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance

```c
#include <stdio.h>

int main()
{
    int n, m, i, j, k, ind = 0, y = 0, flag = 0;
    int max[10][10], avail[10], alloc[10][10], need[10][10], f[10], ans[10];

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

    printf("Enter the number of resources: ");
    scanf("%d", &m);

    printf("Enter the Max matrix for each process:\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        for (j = 0; j < m; j++)
            scanf("%d", &max[i][j]);
    }

    printf("Enter the allocation for each process:\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    }

    printf("Enter the available resources: ");
    for (i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    // Initialize finish array to 0 (false)
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }

    // Calculate need matrix = max - allocation
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Try to find a safe sequence
    for (k = 0; k < n; k++) {       // Repeat n times to check all processes
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {       // If process not finished
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1; // Not enough resources available
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;   // Add to safe sequence
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];  // Release resources
                    f[i] = 1;  // Mark as finished
                }
            }
        }
    }

    // Check if all processes could finish
```

```
    flag = 1;
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {
            flag = 0;
            printf("The system is NOT SAFE.\n");
            break;
        }
    }

    if (flag == 1) {
        printf("Following is the SAFE sequence:\n");
        for (i = 0; i < n - 1; i++)
            printf("P%d -> ", ans[i] + 1);
        printf("P%d\n", ans[n - 1] + 1);
    }

    return 0;
}
```

OUTPUT:

```
Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :
For process 1 : 7 5 3
For process 2 : 3 2 2
For process 3 : 9 0 2
For process 4 : 2 2 2
For process 5 : 4 3 3

Enter the allocation for each process :
For process 1 : 0 1 0
For process 2 : 2 0 0
For process 3 : 3 0 2
For process 4 : 2 1 1
For process 5 : 0 0 2

Enter the Available Resources : 3 3 2
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

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

a) Worst Fit

```c
#include <stdio.h>

int main()
{
    int n, m, i, j, k, ind = 0, y = 0, flag = 0;
    int max[10][10], avail[10], alloc[10][10], need[10][10], f[10], ans[10];

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

    printf("Enter the number of resources: ");
    scanf("%d", &m);

    printf("Enter the Max matrix for each process:\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        for (j = 0; j < m; j++)
            scanf("%d", &max[i][j]);
    }

    printf("Enter the allocation for each process:\n");
    for (i = 0; i < n; i++) {
        printf("For process %d: ", i + 1);
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    }

    printf("Enter the available resources: ");
    for (i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    // Initialize finish array to 0 (false)
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }

    // Calculate need matrix = max - allocation
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Try to find a safe sequence
    for (k = 0; k < n; k++) {       // Repeat n times to check all processes
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {       // If process not finished
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1; // Not enough resources available
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;   // Add to safe sequence
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];  // Release resources
                    f[i] = 1;  // Mark as finished
                }
            }
        }
```

```
    }

    // Check if all processes could finish
    flag = 1;
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {
            flag = 0;
            printf("The system is NOT SAFE.\n");
            break;
        }
    }

    if (flag == 1) {
        printf("Following is the SAFE sequence:\n");
        for (i = 0; i < n - 1; i++)
            printf("P%d -> ", ans[i] + 1);
        printf("P%d\n", ans[n - 1] + 1);
    }

    return 0;
}
```

INPUT :

```
        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
```

OUTPUT :

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	

b) Best-fit

```
#include <stdio.h>
#define max 25

int main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest;
    static int bf[max], ff[max];

    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]);
        bf[i] = 0; // initialize block as free
    }

    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++)
    {
        lowest = 100000; // reset lowest for each file
        ff[i] = -1;      // initialize to no block allocated

        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1) // block is free
            {
                temp = b[j] - f[i];
                if (temp >= 0 && temp < lowest)
                {
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }

        if (ff[i] != -1)
        {
            frag[i] = lowest;
            bf[ff[i]] = 1; // mark block as allocated
        }
        else
        {
            frag[i] = -1; // no block allocated
        }
    }

    printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n");
    for (i = 1; i <= nf; i++)
    {
        if (ff[i] != -1)
        {
            printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, f[i], ff[i], b[ff[i]],
frag[i]);
        }
        else
        {
            printf("%d\t\t%d\t\tNot Allocated\n", i, f[i]);
        }
    }

    return 0;
}
```

INPUT :

```
        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
```

OUTPUT :

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

c) First Fit

```c
#include <stdio.h>
#define max 25

int 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\n");

    printf("Enter 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]);
        bf[i] = 0; // Initialize block as free
    }

    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++)
    {
        temp = -1; // reset temp for each file
        for(j = 1; j <= nb; j++)
        {
            if(bf[j] != 1) // block not allocated
```

```
            {
                int diff = b[j] - f[i];
                if(diff >= 0)
                {
                    ff[i] = j;
                    temp = diff;
                    break; // first fit found, break
                }
            }
        }
        frag[i] = temp;
        if (temp != -1) {
            bf[ff[i]] = 1; // mark block as allocated
        } else {
            ff[i] = -1; // no suitable block found
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for(i = 1; i <= nf; i++)
    {
        if (ff[i] != -1) {
            printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, f[i], ff[i], b[ff[i]],
frag[i]);
        } else {
            printf("%d\t\t%d\t\tNot Allocated\n", i, f[i]);
        }
    }

    return 0;
}
```

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:    Fragment:
1               1              1              5              4
2               4              3              7              3
```

7) Develop a C program to simulate page replacement algorithms:

a) FIFO b) LRU

a) FIFO

```c
#include <stdio.h>

int main()
{
    int a[5], b[20], n, p = 0, q = 0, m = 0, h, k, i, q1 = 1;
    char f = 'F';

    printf("Enter the Number of Pages: ");
    scanf("%d", &n);

    printf("Enter %d Page Numbers: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

    for (i = 0; i < n; i++)
    {
        if (p == 0)
        {
            if (q >= 3)
                q = 0;

            a[q] = b[i];
            q++;

            if (q1 < 3)
                q1 = q;
        }

        printf("\n%d\t", b[i]);

        for (h = 0; h < q1; h++)
            printf("%d", a[h]);

        if ((p == 0) && (q <= 3))
        {
            printf(" --> %c", f);
            m++;
        }

        p = 0;
        for (k = 0; k < q1; k++)
        {
            if (b[i + 1] == a[k])
                p = 1;
        }
    }

    printf("\nNumber of faults: %d\n", m);

    return 0;
}
```

OUTPUT:

```
Enter the Number of Pages: 12
Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2
2 2-->F
3 23-->F
2 23
```

```
1 231-->F
5 531-->F
2 521-->F
4 524-->F
5 524
3 324-->F
2 324
5 354-->F
2 352-->F
```

b) LRU

```c
#include <stdio.h>

int main()
{
    int a[5], b[20], p = 0, q = 0, m = 0, h, k, i, q1 = 1, j, u, n;
    char f = 'F';

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

    printf("Enter %d Page Numbers: ", n);
    for(i = 0; i < n; i++)
        scanf("%d", &b[i]);

    for(i = 0; i < n; i++)
    {
        if(p == 0)
        {
            if(q >= 3)
                q = 0;
            a[q] = b[i];
            q++;
            if(q1 < 3)
                q1 = q;
        }

        printf("\n%d\t", b[i]);
        for(h = 0; h < q1; h++)
            printf("%d", a[h]);

        if((p == 0) && (q <= 3))
        {
            printf(" --> %c", f);
            m++;
        }

        p = 0;

        if(q1 == 3)
        {
            // Check if next page is already in frame
            for(k = 0; k < q1; k++)
            {
                if(b[i+1] == a[k])
                    p = 1;
            }

            // Optimal Replacement Logic
            for(j = 0; j < q1; j++)
            {
                u = 0;
                k = i;
```

```
                while(k >= (i - 1) && k >= 0)
                {
                    if(b[k] == a[j])
                        u++;
                    k--;
                }
                if(u == 0)
                    q = j;
            }
        }
        else
        {
            for(k = 0; k < q; k++)
            {
                if(b[i+1] == a[k])
                    p = 1;
            }
        }
    }

    printf("\nNumber of faults: %d\n", m);
    return 0;
}
```

OUTPUT:

```
Enter the number of pages: 12
Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2
2 2-->F
3 23-->F
2 23
1 231-->F
5 251-->F
2 251
4 254-->F
5 254
3 354-->F
2 352-->F
5 352
2 352     No of faults: 7
```

8) Simulate following File Organization Techniques :

a) Single level directory b) Two level directory

a) Single level directory

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

int main() {
    int i, j = 0, ch;
    char mdname[10], fname[10][10], name[10];

    printf("Enter the directory name: ");
    scanf("%s", mdname);

    do {
        printf("Enter file name to be created: ");
        scanf("%s", name);

        // Check for duplicate
        for(i = 0; i < j; i++) {
            if(strcmp(name, fname[i]) == 0)
                break;
        }

        if(i == j) {
            strcpy(fname[j], name);
            j++;
        } else {
            printf("There is already a file named %s\n", name);
        }

        printf("Do you want to enter another file (yes - 1 / no - 0): ");
        scanf("%d", &ch);

    } while(ch == 1);

    // Output the directory contents
    printf("\nDirectory name is: %s\n", mdname);
    printf("File names are:\n");
    for(i = 0; i < j; i++) {
        printf("%s\n", fname[i]);
    }

    return 0;
}
```

OUTPUT:

```
Enter the directory name: ise
Enter the number of files: 3
Enter file name to be created: sample1
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created: sample2
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created: sample3
Do you want to enter another file(yes - 1 or no - 0):0
Directory name is: ise
Files names are:
sample1
sample2
sample3
```

b)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 name of directory %d: ", i + 1);
        scanf("%s", dir[i].dname);

        printf("Enter number of subdirectories in %s: ", dir[i].dname);
        scanf("%d", &dir[i].ds);

        for(j = 0; j < dir[i].ds; j++) {
            printf("Enter subdirectory name %d and number of files: ", j + 1);
            scanf("%s", dir[i].sdname[j]);
            scanf("%d", &dir[i].sds[j]);

            for(k = 0; k < dir[i].sds[j]; k++) {
                printf("Enter name of file %d in subdirectory %s: ", k + 1,
dir[i].sdname[j]);
                scanf("%s", dir[i].fname[j][k]);
            }
        }
    }

    printf("\nDirectory Structure:\n");
    printf("DirName\tSubDir\t\tFiles\n");
    printf("-------------------------------------------------\n");

    for(i = 0; i < n; i++) {
        printf("%s\n", dir[i].dname);
        for(j = 0; j < dir[i].ds; j++) {
            printf("\t%s\t\t", dir[i].sdname[j]);
            for(k = 0; k < dir[i].sds[j]; k++) {
                printf("%s ", dir[i].fname[j][k]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
enter number of directories:2
enter directory 1 names:Dir1
enter size of directories:2
enter subdirectory name and size:dir3
enter file name:bbb
enter file name:ccc
enter subdirectory name and size:dir4
enter file name:fff
enter file name:ggg
enter directory 2 names:Dir2
enter size of directories:1
enter subdirectory name and size:dir5 1
enter file name:hhh

dirname        size    subdirname      size    files
*********************************************************
Dir1             2      dir3              2        bbb     ccc
                        dir4              2        fff     ggg

Dir2             1      dir5              1        hhh
```

9) Develop a C program to simulate the Linked file allocation strategies.

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

int main() {
    int f[50], p, i, st, len, j, c, k, a;

    for (i = 0; i < 50; i++)
        f[i] = 0; // Initialize all blocks to free

    printf("Enter how many blocks are already allocated: ");
    scanf("%d", &p);

    printf("Enter the block numbers already allocated: ");
    for (i = 0; i < p; i++) {
        scanf("%d", &a);
        if (a >= 0 && a < 50)
            f[a] = 1;
    }

    do {
        printf("Enter index starting block and length of the file: ");
        scanf("%d%d", &st, &len);
        k = len;

        if (st < 0 || st >= 50) {
            printf("Invalid starting block!\n");
            continue;
        }

        if (f[st] == 0) {
            for (j = st; j < (st + k) && j < 50; j++) {
                if (f[j] == 0) {
                    f[j] = 1;
                    printf("%d --------> Allocated\n", j);
                } else {
                    printf("%d Block is already allocated.\n", j);
                    k++;  // Increase range to compensate
                }
            }
        } else {
            printf("%d Starting block is already allocated.\n", st);
        }

        printf("Do you want to enter more files? (Yes - 1 / No - 0): ");
        scanf("%d", &c);
    } while (c == 1);

    return 0;
}
```

OUTPUT:

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5
Enter index starting block and length: 2 2
2-------->1
3 Block is already allocated
4-------->1
Do you want to enter more file(Yes - 1/No - 0)1
Enter index starting block and length: 2 3
2 starting block is already allocated
Do you want to enter more file(Yes - 1/No -
```

10) Develop a C program to simulate SCAN disk scheduling algorithm.

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

int main() {
    int queue[100], queue1[20], queue2[20];
    int n, head, i, j, k;
    int seek = 0, max, diff, temp;
    int temp1 = 0, temp2 = 0;
    float avg;

    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the size of queue request: ");
    scanf("%d", &n);
    printf("Enter the queue of disk positions to be read:\n");

    for (i = 0; i < n; i++) {
        scanf("%d", &temp);
        if (temp >= head)
            queue1[temp1++] = temp; // Higher than head
        else
            queue2[temp2++] = temp; // Lower than head
    }

    // Sort queue1 in ascending
    for (i = 0; i < temp1 - 1; i++) {
        for (j = i + 1; j < temp1; j++) {
            if (queue1[i] > queue1[j]) {
                temp = queue1[i];
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }

    // Sort queue2 in descending
    for (i = 0; i < temp2 - 1; i++) {
        for (j = i + 1; j < temp2; j++) {
            if (queue2[i] < queue2[j]) {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }

    // Construct SCAN path
    int index = 0;
    queue[index++] = head;
    for (i = 0; i < temp1; i++) // Head to max
        queue[index++] = queue1[i];
    queue[index++] = max; // Reach disk end
    for (i = 0; i < temp2; i++) // Then reverse
        queue[index++] = queue2[i];
    queue[index++] = 0; // Reach disk start

    // Calculate seek
    for (i = 0; i < index - 1; i++) {
        diff = abs(queue[i + 1] - queue[i]);
        seek += diff;
        printf("Disk head moves from %d to %d with seek %d\n", queue[i], queue[i +
1], diff);
    }
```

```
    printf("\nTotal seek time: %d\n", seek);
    avg = (float)seek / n;
    printf("Average seek time: %.2f\n", avg);

    return 0;
}
```

OUTPUT:

```
Enter the max range of disk
200
Enter the initial head position
50
Enter the size of queue request
8
Enter the queue of disk positions to be read
90   120   35   122   38   128   65   68
Disk head moves from 50 to 65 with seek  15
Disk head moves from 65 to 68 with seek  3
Disk head moves from 68 to 90 with seek  22
Disk head moves from 90 to 120 with seek  30
Disk head moves from 120 to 122 with seek  2
Disk head moves from 122 to 128 with seek  6
Disk head moves from 128 to 200 with seek  72
Disk head moves from 200 to 38 with seek  162
Disk head moves from 38 to 35 with seek  3
Disk head moves from 35 to 0 with seek  35
Total seek time is 350
Average seek time is 43.750000
```

VIVA VOCE

1) Which system call is used to create a new process?
a) open()
b) **fork()**
c) exec()
d) wait()

2) What does `exec()` do in a process?
a) Ends the process
b) **Replaces the current process image**
c) Waits for a child process
d) Creates a pipe

3) Which system call waits for the child to finish execution?
a) fork()
b) exit()
c) **wait()**
d) exec()

4) Which algorithm executes processes in the order they arrive?
a) SJF
b) **FCFS**
c) Priority
d) Round Robin

5) Which scheduling algorithm may lead to starvation of long processes?
a) FCFS
b) Round Robin
c) **SJF**
d) Priority

6)What is the time quantum in Round Robin scheduling?
a) Execution time
b) Arrival time
c) **Fixed time slice for each process**
d) Turnaround time

7) What is the initial value of a binary semaphore?
a) 2
b) **1**
c) -1
d) 0

8) What problem does the Producer-Consumer model demonstrate?
a) Deadlock
b) **Synchronization**
c) Paging
d) Fragmentation

9) Which function creates a named FIFO in Linux?
a) pipe()
b) open()
c) **mkfifo()**
d) creat()

10) Which API is used to write to a FIFO file?
a) scanf()
b) **write()**
c) fopen()
d) exec()

11) Banker's Algorithm is used for:
a) Memory allocation
b) CPU Scheduling
c) **Deadlock Avoidance**
d) Synchronization

12) A system is in a safe state if:
a) It will deadlock soon
b) **It can allocate resources without deadlock**
c) Resources are full
d) Requests are pending

13) Which allocation strategy always finds the smallest suitable block?
a) Worst Fit
b) First Fit
c) **Best Fit**
d) Compact Fit

14) Which strategy may lead to most external fragmentation?
a) **First Fit**
b) Best Fit
c) Worst Fit
d) Next Fit

15) Which algorithm replaces the oldest page in memory?
a) LRU
b) **FIFO**
c) Optimal
d) Random

16) LRU stands for:
a) Last Ready Unit
b) Least Requested Unit
c) **Least Recently Used**
d) Load Reduced Usage

17) Which file system maintains a separate subdirectory for each user?
a) Single Level
b) **Two Level**
c) Hierarchical
d) Indexed

18) The major drawback of Single-Level Directory is:
a) Too many files
b) **Name conflicts**
c) Hard to create
d) Security issues

19) Linked File Allocation stores files as:
a) **Chain of blocks**
b) Indexed block
c) Contiguous space
d) Clusters

20) SCAN scheduling is also called:
a) First Fit
b) **Elevator Algorithm**
c) Round Robin
d) LRU