

Bapuji Educational Association (Regd.)

# Bapuji Institute of Engineering and Technology

Affiliated to Visvesvaraya Technological University, Belgaavi, Accredited by NBA and NAAC with 'A' grade,

Recognized by UGC under 2(F) and 12(B)

## Department of Computer Science and Engineering

### LABORATORY MANUAL

2025-26

### OPERATING SYSTEM LAB MANUAL (BCSPCC304)

#### III SEMESTER

### Vision

*To be a centre-of-excellence by imbibing state-of-the-art technology in the field of Computer Science and Engineering, thereby enabling students to excel professionally and be ethical.*

### Mission

<b>M1</b>	<i>Adapting best teaching and learning techniques that cultivates Questioning and Reasoning culture among the students.</i>
<b>M2</b>	<i>Creating collaborative learning environment that ignites the critical thinking in students and leading to the innovation.</i>
<b>M3</b>	<i>Establishing Industry Institute relationship to bridge the skill gap and make them industry ready and relevant.</i>
<b>M4</b>	<i>Mentoring students to be socially responsible by inculcating ethical and moral values.</i>

### Program Educational Objectives:

<b>PEO1</b>	<i>To apply skills acquired in the discipline of Computer Science and Engineering for solving societal and industrial problems with apt technology intervention.</i>
<b>PEO2</b>	<i>To continue their career in industry/academia or to pursue higher studies and research.</i>
<b>PEO3</b>	<i>To become successful entrepreneurs, innovators to design and develop software products and services that meets the societal, technical and business challenges.</i>
<b>PEO4</b>	<i>To work in the diversified environment by acquiring leadership qualities with effective communication skills accompanied by professional and ethical values.</i>

### Program Specific Outcomes (PSOs):

<b>PSO1</b>	<i>Analyze and develop solutions for problems that are complex in nature by applying the knowledge acquired from the core subjects of this program.</i>
<b>PSO2</b>	<i>To develop Secure, Scalable, Resilient and distributed applications for industry and societal requirements.</i>
<b>PSO3</b>	<i>To learn and apply the concepts and construct of emerging technologies like Artificial Intelligence, Machine learning, Deep learning, Big Data Analytics, IoT, Cloud Computing, etc for any real time problems.</i>

### Programme outcome (PO's):

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering

problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Teaching Hours/Week (L: T: P: S) 3:0:2:0 SEE Marks 50

Total Hours of Pedagogy 40 hours Theory + 20 hours practicals Total Marks 100

Credits 04

**PRACTICAL COMPONENT OF IPCC (May cover all / major modules)**

**Experiments:**

<b>Practical Component</b>	
<b>Sl. No.</b>	<b>Programs for Conduction</b>
<b>1.</b>	Develop a C program to implement the Process system calls such as: fork (), exec(), wait(), create process, terminate process
<b>2.</b>	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF
<b>3.</b>	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) Round Robin b) Priority.
<b>4.</b>	Develop a C program to simulate producer-consumer problem using semaphores.
<b>5.</b>	Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo(), open(), read(), write() and close() APIs.
<b>6.</b>	Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance.
<b>7.</b>	Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit
<b>8.</b>	Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU c)Optimal
<b>9.</b>	Develop a C program to simulate SCAN disk scheduling algorithm
<b>10.</b>	Simulate following File Organization Techniques a) Single level directory b) Two level directory

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

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

int main() {
    pid_t child_pid;

    // Fork a child process
    child_pid = fork();

    if (child_pid < 0) {
        fprintf(stderr, "Fork failed\n");
        return 1;
    }

    if (child_pid == 0) {
        // Child process

        printf("Child Process: PID = %d\n", getpid());
        printf("\nWaiting for user to press Enter\n");
        printf("\nMeanwhile before pressing Enter\nOpen another terminal and run ps -a\n");
        getchar();

        printf("Executing 'ls' command...\n\n");

        // Execute 'ls' command in the child process
        execl("/bin/ls", "ls", NULL);

        // This line is reached only if execl fails
        perror("execl");
        exit(1);
    } else {
        // Parent process

        printf("Parent Process: PID = %d\n", getpid());

        // Wait for the child process to finish
        int status;
        wait(&status);

        if (WIFEXITED(status)) {
            printf("\nChild process exited with status %d\n", WEXITSTATUS(status));
        } else {
            fprintf(stderr, "Child process did not terminate normally\n");
        }
    }
}
```

```

        printf("Parent process exits\n");
    }
    return 0;
}

```

## 2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF

### a) FCFS

```

#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);

    printf("Enter process id of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&pid[i]);
    }

    printf("Enter burst time of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&bt[i]);
    }

    int i, wt[n];
    wt[0]=0;

    //for calculating waiting time of each process
    for(i=1; i<n; i++)
    {
        wt[i]= bt[i-1]+ wt[i-1];
    }

    printf("Process ID    Burst Time    Waiting Time    TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++)
    {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);
    }
}

```

```

        //calculating and printing turnaround time of each process
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");

        //for calculating total waiting time
        twt += wt[i];

        //for calculating total turnaround time
        tat += (wt[i]+bt[i]);
    }
    float att,awt;

    //for calculating average waiting time
    awt = twt/n;

    //for calculating average turnaround time
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}

```

## b) SJF

```

#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 process:");
    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;
    }

    //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

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

```

```

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

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

wt[0]=0;

//finding the waiting time of all the processes
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        //individual WT by adding BT of all previous completed processes
        wt[i]+=bt[j];

    //total waiting time
    total+=wt[i];
}

//average waiting time
avg_wt=(float)total/n;

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    //turnaround time of individual processes
    tat[i]=bt[i]+wt[i];

    //total turnaround time
    totalT+=tat[i];
    printf("\np%d\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
}

//average turnaround time
avg_tat=(float)totalT/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\n\nAverage Turnaround Time=%f",avg_tat);
}

```

### 3. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) Round Robin b) Priority.

#### a) Round Robin

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

```
int main()
{
```

```

//Input no of processed
int n;
printf("Enter Total Number of Processes:");
scanf("%d", &n);
int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
int x = n;

//Input details of processes
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];
}

//Input time slot
int time_slot;
printf("Enter Time Slot:");
scanf("%d", &time_slot);

//Total indicates total time
//counter indicates which process is executed
int total = 0, counter = 0;
printf("Process ID    Burst Time    Turnaround Time    Waiting Time\n");
for(total=0, i = 0; x!=0; )
{
    // define the conditions
    if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
    {
        total = total + temp_burst_time[i];
        temp_burst_time[i] = 0;
        counter=1;
    }
    else if(temp_burst_time[i] > 0)
    {
        temp_burst_time[i] = temp_burst_time[i] - time_slot;
        total += time_slot;
    }
    if(temp_burst_time[i]==0 && counter==1)
    {
        x--; //decrement the process no.
        printf("\nProcess No %d \t\t %d\t\t\t %d\t\t\t %d", i+1, burst_time[i],
            total-arr_time[i], total-arr_time[i]-burst_time[i]);
        wait_time = 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 average_wait_time = wait_time * 1.0 / n;
float average_turnaround_time = ta_time * 1.0 / n;
printf("\nAverage Waiting Time:%f", average_wait_time);
printf("\nAvg Turnaround Time:%f", average_turnaround_time);
return 0;
}

```

## **b. Priority**

```

#include <stdio.h>
//Function to swap two variables
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);

    // b is array for burst time, p for priority and index for process id
    int b[n],p[n],index[n];
    for(int i=0;i<n;i++)
    {
        printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
        scanf("%d %d",&b[i],&p[i]);
        index[i]=i+1;
    }
    for(int i=0;i<n;i++)
    {
        int a=p[i],m=i;

        //Finding out highest priority element and placing it at its desired position
        for(int j=i;j<n;j++)

```

```

    {
        if(p[j] > a)
        {
            a=p[j];
            m=j;
        }
    }

    //Swapping processes
    swap(&p[i], &p[m]);
    swap(&b[i], &b[m]);
    swap(&index[i],&index[m]);
}

// T stores the starting time of process
int t=0;

//Printing scheduled process
printf("Order of process Execution is\n");
for(int i=0;i<n;i++)
{
    printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
    t+=b[i];
}
printf("\n");
printf("Process Id   Burst Time   Wait Time   TurnAround Time\n");
int wait_time=0;
for(int i=0;i<n;i++)
{
    printf("P%d       %d       %d       %d\n",index[i],b[i],wait_time,wait_time + b[i]);
    wait_time += b[i];
}
return 0;
}

```

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

// C program for the above approach

```

#include <stdio.h>
#include <stdlib.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()
{
    // Decrease mutex value by 1
    --mutex;

    // Increase the number of full
    // slots by 1
    ++full;

    // Decrease the number of empty
    // slots by 1
    --empty;

    // Item produced
    x++;
    printf("\nProducer produces item %d",x);

    // Increase mutex value by 1
    ++mutex;
}

// Function to consume an item and
// remove it from buffer
void consumer()
{
    // Decrease mutex value by 1
    --mutex;

    // Decrease the number of full
    // slots by 1
    --full;

    // Increase the number of empty
    // slots by 1
    ++empty;
    printf("\nConsumer consumes item %d",x);
    x--;

    // Increase mutex value by 1
    ++mutex;
}

// Driver Code
int main()

```

```

{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");

    // Using '#pragma omp parallel for'
    // can give wrong value due to
    // synchronization issues.
    // 'critical' specifies that code is
    // executed by only one thread at a
    // time i.e., only one thread enters
    // the critical section at a given time
    #pragma omp critical

        for (i = 1; i > 0; i++) {

            printf("\nEnter your choice:");
            scanf("%d", &n);

            // Switch Cases
            switch (n) {
            case 1:

                // If mutex is 1 and empty
                // is non-zero, then it is
                // possible to produce
                if ((mutex == 1)
                    && (empty != 0)) {
                    producer();
                }

                // Otherwise, print buffer
                // is full
                else {
                    printf("Buffer is full!");
                }
                break;

            case 2:

                // If mutex is 1 and full
                // is non-zero, then it is
                // possible to consume
                if ((mutex == 1)
                    && (full != 0)) {
                    consumer();
                }

                // Otherwise, print Buffer

```

```

        // is empty
        else {
            printf("Buffer is empty!");
        }
        break;

    // Exit Condition
    case 3:
        exit(0);
        break;
    }
}
}

```

**5. Write a C/C++ program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs.**

```

/*Writer Process*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int fd;
    char buf[1024];
    /* create the FIFO (named pipe) */
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    printf("Run Reader process to read the FIFO File\n");
    fd = open(myfifo, O_WRONLY);
    write(fd, "Hi", sizeof("Hi"));
    /* write "Hi" to the FIFO */
    close(fd);
    unlink(myfifo); /* remove the FIFO */
    return 0;
}

```

```

/* Reader Process */
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#define MAX_BUF 1024
int main()

```

```

{
int fd;
/* A temp FIFO file is not created in reader */
char *myfifo = "/tmp/myfifo";
char buf[MAX_BUF];
/* open, read, and display the message from the FIFO */
fd = open(myfifo, O_RDONLY);
read(fd, buf, MAX_BUF);
printf("Writer: %s\n", buf);
close(fd);
return 0;
}

```

## 6. Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

```

#include<stdio.h>
int main()
{
/* array will store at most 5 process with 3 resoures if your process or
resources is greater than 5 and 3 then increase the size of array */
int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5],
terminate = 0;
printf("Enter the number of process and resources");
scanf("%d %d", & p, & c);
// p is process and c is diffrent resources
printf("enter allocation of resource of all process %dx%d matrix", p, c);
for (i = 0; i < p; i++) {
for (j = 0; j < c; j++) {
scanf("%d", & alc[i][j]);
}
}
printf("enter the max resource process required %dx%d matrix", p, c);
for (i = 0; i < p; i++) {
for (j = 0; j < c; j++) {
scanf("%d", & max[i][j]);
}
}
printf("enter the available resource");
for (i = 0; i < c; i++)
scanf("%d", & available[i]);

printf("\n need resources matrix are\n");
for (i = 0; i < p; i++) {
for (j = 0; j < c; j++) {
need[i][j] = max[i][j] - alc[i][j];
printf("%d\t", need[i][j]);
}
}
printf("\n");
}

```

```

/* once process execute variable done will stop them for again execution */
for (i = 0; i < p; i++) {
    done[i] = 0;
}
while (count < p) {
    for (i = 0; i < p; i++) {
        if (done[i] == 0) {
            for (j = 0; j < c; j++) {
                if (need[i][j] > available[j])
                    break;
            }
            //when need matrix is not greater then available matrix then if j==c will true
            if (j == c) {
                safe[count] = i;
                done[i] = 1;
                /* now process get execute release the resources and add them in available resources */
                for (j = 0; j < c; j++) {
                    available[j] += alc[i][j];
                }
                count++;
                terminate = 0;
            } else {
                terminate++;
            }
        }
    }
}
if (terminate == (p - 1)) {
    printf("safe sequence does not exist");
    break;
}

}
if (terminate != (p - 1)) {
    printf("\n available resource after completion\n");
    for (i = 0; i < c; i++) {
        printf("%d\t", available[i]);
    }
    printf("\n safe sequence are\n");
    for (i = 0; i < p; i++) {
        printf("p%d\t", safe[i]);
    }
}
return 0;
}

```

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

### a) Worst fit

```
#include <stdio.h>
void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)
{
    // This will store the block id of the allocated block to a process
    int allocation[processes];
    int occupied[blocks];

    // initially assigning -1 to all allocation indexes
    // means nothing is allocated currently
    for(int i = 0; i < processes; i++){
        allocation[i] = -1;
    }

    for(int i = 0; i < blocks; i++){
        occupied[i] = 0;
    }

    // pick each process and find suitable blocks
    // according to its size and assign to it
    for (int i=0; i < processes; i++)
    {
        int indexPlaced = -1;
        for(int j = 0; j < blocks; j++)
        {
            // if not occupied and block size is large enough
            if(blockSize[j] >= processSize[i] && !occupied[j])
            {
                // place it at the first block fit to accomodate process
                if (indexPlaced == -1)
                    indexPlaced = j;

                // if any future block is larger than the current block where
                // process is placed, change the block and thus indexPlaced
                else if (blockSize[indexPlaced] < blockSize[j])
                    indexPlaced = j;
            }
        }

        // If we were successfully able to find block for the process
        if (indexPlaced != -1)
        {
            // allocate this block j to process p[i]
            allocation[i] = indexPlaced;

            // make the status of the block as occupied
        }
    }
}
```

```

        occupied[indexPlaced] = 1;

        // Reduce available memory for the block
        blockSize[indexPlaced] -= processSize[i];
    }
}

printf("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < processes; i++)
{
    printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
    if (allocation[i] != -1)
        printf("%d\n", allocation[i] + 1);
    else
        printf("Not Allocated\n");
}
}

// Driver code
int main()
{
    int blockSize[] = {100, 50, 30, 120, 35};
    int processSize[] = {40, 10, 30, 60};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);

    implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0;
}

```

## **b) Best fit**

```

#include <stdio.h>
void implimentBestFit(int blockSize[], int blocks, int processSize[], int processes)
{
    // This will store the block id of the allocated block to a process
    int allocation[processes];
    int occupied[blocks];

    // initially assigning -1 to all allocation indexes
    // means nothing is allocated currently
    for(int i = 0; i < processes; i++){
        allocation[i] = -1;
    }

    for(int i = 0; i < blocks; i++){
        occupied[i] = 0;
    }
}

```

```

// pick each process and find suitable blocks
// according to its size and assign to it
for (int i = 0; i < processes; i++)
{
    int indexPlaced = -1;
    for (int j = 0; j < blocks; j++) {
        if (blockSize[j] >= processSize[i] && !occupied[j])
        {
            // place it at the first block fit to accommodate process
            if (indexPlaced == -1)
                indexPlaced = j;

            // if any future block is smaller than the current block where
            // process is placed, change the block and thus indexPlaced
            // this reduces the wastage thus best fit
            else if (blockSize[j] < blockSize[indexPlaced])
                indexPlaced = j;
        }
    }

    // If we were successfully able to find block for the process
    if (indexPlaced != -1)
    {
        // allocate this block j to process p[i]
        allocation[i] = indexPlaced;

        // make the status of the block as occupied
        occupied[indexPlaced] = 1;
    }
}

printf("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < processes; i++)
{
    printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
    if (allocation[i] != -1)
        printf("%d\n", allocation[i] + 1);
    else
        printf("Not Allocated\n");
}
}

// Driver code
int main()
{
    int blockSize[] = {100, 50, 30, 120, 35};
    int processSize[] = {40, 10, 30, 60};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);

```

```

implimentBestFit(blockSize, blocks, processSize, processes);

return 0 ;
}

```

### c) First fit.

```

#include <stdio.h>
void implimentFirstFit(int blockSize[], int blocks, int processSize[], int processes)
{
    // This will store the block id of the allocated block to a process
    int allocate[processes];
    int occupied[blocks];

    // initially assigning -1 to all allocation indexes
    // means nothing is allocated currently
    for(int i = 0; i < processes; i++)
    {
        allocate[i] = -1;
    }

    for(int i = 0; i < blocks; i++){
        occupied[i] = 0;
    }

    // take each process one by one and find
    // first block that can accomodate it
    for (int i = 0; i < processes; i++)
    {
        for (int j = 0; j < blocks; j++)
        {
            if (!occupied[j] && blockSize[j] >= processSize[i])
            {
                // allocate block j to p[i] process
                allocate[i] = j;
                occupied[j] = 1;

                break;
            }
        }
    }
}

printf("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < processes; i++)
{
    printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
    if (allocate[i] != -1)
        printf("%d\n", allocate[i] + 1);
}

```

```

        else
            printf("Not Allocated\n");
    }
}

void main()
{
    int blockSize[] = {30, 5, 10};
    int processSize[] = {10, 6, 9};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    implimentFirstFit(blockSize, m, processSize, n);
}

```

## 8. Develop a C program to simulate page replacement algorithms:

a) FIFO b) LRU c) Optimal

### FIFO:

```

#include<stdio.h>
int main()
{
    //int incomingStream[] = {4,1,2,4,5};
    int incomingStream[] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;
    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
    printf(" Incoming \t Frame 1 \t\t Frame 2 \t\t Frame 3 ");
    int temp[ frames ];
    for(m = 0; m < frames; m++)
    {
        temp[m] = -1;
    }
    for(m = 0; m < pages; m++)
    {
        s = 0;
        for(n = 0; n < frames; n++)
        {
            if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;
        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
    }
}

```

```

    }
    else if(s == 0)
    {
        temp[(pageFaults - 1) % frames] = incomingStream[m];
    }
    printf("\n");
    printf("%d\t\t\t",incomingStream[m]);
    for(n = 0; n < frames; n++)
    {
        if(temp[n] != -1)
            printf(" %d\t\t\t", temp[n]);
        else
            printf(" - \t\t\t");
    }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}

```

## LRU:

```

#include<stdio.h>
#include<limits.h>

```

```

int checkHit(int incomingPage, int queue[], int occupied){

```

```

    for(int i = 0; i < occupied; i++){
        if(incomingPage == queue[i])
            return 1;
    }

```

```

    return 0;
}

```

```

void printFrame(int queue[], int occupied)

```

```

{
    for(int i = 0; i < occupied; i++)
        printf("%d\t\t\t",queue[i]);
}

```

```

int main()

```

```

{
    int incomingStream[] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
    // int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1};
    // int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};
    // int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

```

```

    int n = sizeof(incomingStream)/sizeof(incomingStream[0]);
    int frames = 3;

```

```

int queue[n];
int distance[n];
int occupied = 0;
int pagefault = 0;

printf("Page\t Frame1 \t\t Frame2 \t\t Frame3\n");

for(int i = 0; i < n; i++)
{
    printf("%d: \t\t", incomingStream[i]);
    // what if currently in frame 7
    // next item that appears also 7
    // didnt write condition for HIT

    if(checkHit(incomingStream[i], queue, occupied)){
        printFrame(queue, occupied);
    }

    // filling when frame(s) is/are empty
    else if(occupied < frames){
        queue[occupied] = incomingStream[i];
        pagefault++;
        occupied++;

        printFrame(queue, occupied);
    }
    else{

        int max = INT_MIN;
        int index;
        // get LRU distance for each item in frame
        for (int j = 0; j < frames; j++)
        {
            distance[j] = 0;
            // traverse in reverse direction to find
            // at what distance frame item occurred last
            for(int k = i - 1; k >= 0; k--)
            {
                ++distance[j];

                if(queue[j] == incomingStream[k])
                    break;
            }

            // find frame item with max distance for LRU
            // also notes the index of frame item in queue
            // which appears furthest(max distance)
            if(distance[j] > max){
                max = distance[j];
                index = j;
            }
        }
    }
}

```

```

    }
    }
    queue[index] = incomingStream[i];
    printFrame(queue, occupied);
    pagefault++;
}

printf("\n");
}

printf("Page Fault: %d",pagefault);

return 0;
}

```

### Optimal :

```

#include <stdio.h>
#include <limits.h>

#define FRAMES 3 // Number of frames

// Function to check if a page is in the frame
int checkHit(int page, int frame[], int occupied) {
    for (int i = 0; i < occupied; i++) {
        if (frame[i] == page) {
            return 1; // Hit
        }
    }
    return 0; // Miss
}

// Function to find the index of the page to be replaced (the one that is used farthest in the future)
int findOptimalPageReplacement(int frame[], int pages[], int currentIndex, int n, int occupied) {
    int farthestIndex = -1;
    int pageToReplace = -1;

    for (int i = 0; i < occupied; i++) {
        int nextUse = -1;

        // Look ahead to find the next time each page will be used
        for (int j = currentIndex + 1; j < n; j++) {
            if (frame[i] == pages[j]) {
                nextUse = j;
                break;
            }
        }
    }
}

```

```

    // If the page will never be used again, replace it
    if (nextUse == -1) {
        return i;
    }

    // Find the page with the farthest next use
    if (nextUse > farthestIndex) {
        farthestIndex = nextUse;
        pageToReplace = i;
    }
}

return pageToReplace;
}

// Function to print the current state of frames
void printFrames(int frame[], int occupied) {
    for (int i = 0; i < occupied; i++) {
        printf("%d\t\t", frame[i]);
    }
}

int main() {
    // Incoming page reference stream
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};
    int n = sizeof(pages) / sizeof(pages[0]);

    int frame[FRAMES]; // Array to store the pages in memory
    int pageFaults = 0;
    int occupied = 0;

    printf("Page\tFrame 1\tFrame 2\tFrame 3\n");

    for (int i = 0; i < n; i++) {
        int currentPage = pages[i];

        // Check if the page is already in the frame
        if (checkHit(currentPage, frame, occupied)) {
            // If it's a hit, just print the current frame state
            printf("%d\t\t", currentPage);
            printFrames(frame, occupied);
            printf("\n");
        } else {
            // If it's a page fault, replace a page using OPT algorithm
            pageFaults++;

            if (occupied < FRAMES) {
                // There is space in the frame, just add the new page
                frame[occupied] = currentPage;
                occupied++;
            }
        }
    }
}

```

```

    } else {
        // Find the page to replace using OPT
        int replaceIndex = findOptimalPageReplacement(frame, pages, i, n, occupied);
        frame[replaceIndex] = currentPage;
    }

    // Print the current frame state
    printf("%d\t\t", currentPage);
    printFrames(frame, occupied);
    printf("\n");
}
}

printf("Total Page Faults: %d\n", pageFaults);
return 0;
}

```

## 9. Develop a C program to simulate SCAN disk scheduling algorithm

```

#include<stdio.h>
#include<math.h>
int main()
{
    int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],temp1=0,temp2=0;
    float avg;
    printf("Enter the max range of disk\n");
    scanf("%d",&max);
    printf("Enter the initial head position\n");
    scanf("%d",&head);
    printf("Enter the size of queue request\n");
    scanf("%d",&n);
    printf("Enter the queue of disk positions to be read\n");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&temp);
        if(temp>=head)
        {
            queue1[temp1]=temp;
            temp1++;
        }
        else
        {
            queue2[temp2]=temp;
            temp2++;
        }
    }
    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;
        }
    }
}
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;
        }
    }
}
for(i=1,j=0;j<temp1;i++,j++)
queue[i]=queue1[j];
queue[i]=max;
for(i=temp1+2,j=0;j<temp2;i++,j++)
queue[i]=queue2[j];
queue[i]=0;
queue[0]=head;
for(j=0;j<=n+1;j++)
{
    diff=abs(queue[j+1]-queue[j]);
    seek+=diff;
    printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}

```

## 10. Simulate following File Organization Techniques

### a) Single level directory

```

#include<stdio.h>
#include<stdlib.h>
#include<string.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\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter
        your choice -- ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: printf("\n Enter the name of the file -- ");
                    scanf("%s",dir.fname[dir.fcnt]);
                    dir.fcnt++;
                    break;
            case 2: printf("\n Enter 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("\n Enter 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("\n Directory Empty");

```

```

else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
getch();
}

```

### a) Two level directory

```

#include<stdio.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\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter 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");
break;
}
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++)
{
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();
}
```