1. Simulate the following non-preemptive CPU scheduling algorithms to find turnaround and waiting time: (a) FCFS (b) SJF (c) RR (preemptive) (d) Priority

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

void findWaitingTime(int processes[],int n,int bt[],int wt[],int
at[])
{
   int service_time[n];
   service_time[0]=0;
   wt[0]=0;
   for (int i = 1; i < n; i++)
   {
         service_time[i]=service_time[i-1]+bt[i-1];
         wt[i]=service_time[i]-at[i];
         if (wt[i]<0)
         {
         wt[i]=0;
         }
   }
}

void findTurnAroundTime(int processes[],int n,int bt[],int wt[],int
tat[])
{
   for(int i=0;i<n;i++)
         tat[i]=bt[i]+wt[i];
}

void findavgTime(int processes[],int n,int bt[],int at[])
{
   int wt[n],tat[n];
   findWaitingTime(processes,n,bt,wt,at);
   findTurnAroundTime(processes,n,bt,wt,tat);
   printf("\nProcesses  Burst Time  Arrival Time  Waiting Time  Turn-
Around Time  Completion Time");
   int total_wt=0,total_tat=0;
   for (int i = 0; i < n; i++)
   {
         total_wt=total_wt+wt[i];
         total_tat=total_tat+tat[i];
         int compl_time=tat[i]+at[i];
         printf("\n
%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",i+1,bt[i],at[i],wt[i],tat[i],compl_
time);
   }
   printf("\nAverage waiting time=%f",(float)total_wt/(float)n);
   printf("\nAverage turn around time=%f",(float)total_tat/(float)n);
}

void main()
{
   int processes[20],n,burst_time[20],arrival_time[20],i;
```

```
    printf("Enter the number of processes:");
    scanf("%d",&n);
    printf("\nEnter Burst Times(in ms) of Processes:");
    for (i = 0; i < n; i++)
    {
          printf("\nProcess %d:",i+1);
          scanf("%d",&burst_time[i]);
    }
    printf("\nEnter Arrival Times(in ms) of Processes:");
    for (i = 0; i < n; i++)
    {
          printf("\nProcess %d:",i+1);
          scanf("%d",&arrival_time[i]);
    }
    findavgTime(processes,n,burst_time,arrival_time);
}

// SJF
#include<stdio.h>

void main()
{
    int
bt[20],p[20],wt[20],tat[20],i,j,n,total_tat=0,total_wt=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter the number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Times(in ms) for Processes:");
    for(i=0;i<n;i++)
    {
          printf("\nProcess %d:",i+1);
          scanf("%d",&bt[i]);
          p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
          pos=i;
          for(j=i+1;j<n;j++)
          {
          if(bt[j]<bt[pos])
                pos=j;
          }

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

          temp=p[i];
          p[i]=p[pos];
          p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
```

```
        wt[i]=0;
        for(j=0;j<i;j++)
        wt[i]+=bt[j];

        total_wt+=wt[i];
    }
    avg_wt=(float)total_wt/n;
    printf("\nProcess  Burst Time  Waiting Time  Turn Around Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total_tat+=tat[i];
        printf("\n%d\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total_tat/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

// Round Robin (pre-emptive)

```
#include<stdio.h>

void main()
{
    int
count,j,n,time,remain,flag=0,time_quantum,wait_time=0,turnaround_time
=0,at[20],bt[20],rt[20];
    printf("Enter the number of Processes:");
    scanf("%d",&n);
    remain=n;
    printf("\nEnter Arrival Times(in ms) for Processes:");
    for (count = 0; count < n; count++)
    {
        printf("\nProcess %d:",count+1);
        scanf("%d",&at[count]);
    }
    printf("\nEnter Burst Times(in ms) for Processes:");
    for (count = 0; count< n; count++)
    {
        printf("\nProcess %d:",count+1);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("\nEnter the Time Quantum:");
    scanf("%d",&time_quantum);
    printf("\nProcesses  Turn Around Time  Waiting Time");
    for(time=0,count=0;remain!=0;)
    {
        if ((rt[count]<=time_quantum)&&(rt[count]>0))
        {
        time+=rt[count];
        rt[count]=0;
        flag=1;
        }
```

```
          else if (rt[count]>0)
          {
          rt[count]-=time_quantum;
          time+=time_quantum;
          }
          if ((rt[count]==0)&&(flag==1))
          {
          remain--;
          printf("\n%d\t\t%d\t\t%d",count+1,time-at[count],time-
at[count]-bt[count]);
          wait_time+=time-at[count]-bt[count];
          turnaround_time+=time-at[count];
          flag=0;
          }
          if (count==n-1)
          {
          count=0;
          }
          else if(at[count+1]<=time)
          {
          count++;
          }
          else
          {
          count=0;
          }
    }
    printf("\nAverage Waiting Time=%f",(float)(wait_time*1.0/n));
    printf("\nAverage Turn Around
Time:%f",(float)(turnaround_time*1.0/n));
}

// Priority
#include<stdio.h>

void main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total_wt=0,
total_tat=0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter the number of Processes:");
    scanf("%d", &n);
    printf("\nEnter the Burst Times(in ms) for Processes:");
    for(i=0;i<n;i++) {
          printf("\nProcess %d:",i+1);
          scanf("%d",&bt[i]);
          p[i]=i+1;
    }
    printf("\nEnter the Priority values for Processes:");
    for(i=0;i<n;i++) {
          printf("\nProcess %d:",i+1);
          scanf("%d",&pr[i]);
    }
    for(i=0;i<n;i++) {
```

```
            pos=i;
            for(j=i+1;j<n;j++)
                  if(pr[j]<pr[pos])
                        pos=j;
            temp=pr[i];
            pr[i]=pr[pos];
            pr[pos]=temp;

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

            temp=p[i];
            p[i]=p[pos];
            p[pos]=temp;
      }
      wt[0]=0;
      for(i=1;i<n;i++) {
            wt[i]=0;
            for(j=0;j<i;j++)
                  wt[i]+=bt[j];
            total_wt+=wt[i];
      }
      avg_wt=total_wt/n;
      printf("\nProcesses  Burst Time  Waiting Time  Turn Around Time");
      for(i=0;i<n;i++) {
            tat[i]=bt[i]+wt[i];
            total_tat+=tat[i];
            printf("\n%d\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tat[i]);
      }
      avg_tat=total_tat/n;
      printf("\nAverage Waiting Time=%f",avg_wt);
      printf("\nAverage Turn Around Time=%f",avg_tat);
}
```

Output:

(i) FCFS

```
Enter the number of processes:3

Enter Burst Times(in ms) of Processes:
Process 1:5

Process 2:9

Process 3:6

Enter Arrival Times(in ms) of Processes:
Process 1:0

Process 2:3

Process 3:6

Processes  Burst Time  Arrival Time  Waiting Time  Turn-Around Time  Completion Time
 1             5              0              0              5              5
 2             9              3              2              11             14
 3             6              6              8              14             20
Average waiting time=3.333333
Average turn around time=10.000000
```

(ii) SJF

```
Enter the number of process:4

Enter Burst Times(in ms) for Processes:
Process 1:6

Process 2:8

Process 3:7

Process 4:3

Process  Burst Time  Waiting Time  Turn Around Time
4               3               0               3
1               6               3               9
3               7               9               16
2               8               16              24

Average Waiting Time=7.000000
Average Turnaround Time=13.000000
```

(iii) Round-Robin (pre-emptive)

```
Enter the number of process:4

Enter Burst Times(in ms) for Processes:
Process 1:6

Process 2:8

Process 3:7

Process 4:3

Process  Burst Time  Waiting Time  Turn Around Time
4               3               0               3
1               6               3               9
3               7               9               16
2               8               16              24

Average Waiting Time=7.000000
Average Turnaround Time=13.000000
```

(iv) Priority

```
Enter the number of Processes:5

Enter the Burst Times(in ms) for Processes:
Process 1:11

Process 2:28

Process 3:2

Process 4:10

Process 5:16

Enter the Priority values for Processes:
Process 1:2

Process 2:0

Process 3:3

Process 4:1

Process 5:4

Processes  Burst Time  Waiting Time  Turn Around Time
2              28              0               28
4              10              28              38
1              11              38              49
3              2               49              51
5              16              51              67
Average Waiting Time=33.000000
Average Turn Around Time=46.000000
```

2. Simulate the following file organization techniques: (i) Single-level directory (ii) Two-level directory (iii) Hierarchical directory

```
// Single-level directory
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void main()
{
    int i, fCount=0, ch;
    char dName[10], fName[10][10], name[10];
    printf("Enter the directory name: ");
    scanf("%s", dName);

    while (1) {
        printf("\n1. Create file. \n2. Delete file. \n
3. Search file. \n4. Display files. \n5. Exit. \nENTER CHOICE: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1: // Create file
            if (fCount < 10) {
                printf("Enter filename: ");
                scanf("%s", name);

                for (i=0; i<fCount; i++) {
                    if (!strcmp(name, fName[i]))
                        break;
                }
                if (i==fCount) {
                    strcpy(fName[fCount++], name);
                    printf("File created\n");
                } else {
                    printf("File %s already exists!\n", name);
                }
            } else {
                printf("Directory full!\n");
            }
            break;
        case 2: // Delete file
            if (fCount) {
                printf("Enter the name of the file: ");
                scanf("%s", name);
                for (i = 0; i < fCount; i++) {
                    if (!strcmp(name, fName[i])) {
                        printf("Deleting file %s\n", name);
                        strcpy(fName[i], fName[--fCount]);
```

```
                        break;
                    }
                }
                if (i == fCount)
                    printf("File %s not found!\n", name);
            } else {
                printf("Directory empty!\n");
            }
            break;
        case 3: // Search file
            printf("Enter the name of the file: ");
            scanf("%s", name);
            for (i = 0; i < fCount; i++) {
                if (!strcmp(name, fName[i])) {
                    printf("File %s found!\n", name);
                    break;
                }
            }
            if (i == fCount)
                printf("File %s not found!\n", name);
            break;
        case 4: // Display files
            printf("\nFiles in directory %s: \n", dName);
            for (i = 0; i < fCount; i++)
                printf("%s\n", fName[i]);
            break;
        default:
            exit(0);
        }
    }
}


// Two-level directory.
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct {
    char dName[10], fName[10][10];
    int fCount; // no. of files
} directory;

void main()
{
    directory dir[10];
    int i, ch, dCount=0, k;
    char f[30], d[30];
```

```
    while (1)
    {
        printf("\n1. Create Directory. \n2. Create File. \n3. Delete
File. \n4. Search file. \n5. Display. \n6. Exit. \nENTER CHOICE: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1: // create directory
            printf("Enter the name of the directory: ");
            scanf("%s", dir[dCount].dName);
            dir[dCount].fCount = 0;
            dCount++;
            printf("Directory created\n");
            break;
        case 2:  // create file
            printf("Enter the name of the directory: ");
            scanf("%s", d);
            for (i=0; i<dCount; i++) {
                if (!strcmp(d, dir[i].dName)) {
                    printf("Enter the name of the file: ");
                    scanf("%s", f);
                    for (k=0; k<dir[i].fCount; k++) {
                        if (!strcmp(f, dir[i].fName[k]))
                            break;
                    }
                    if (k==dir[i].fCount) {
                        strcpy(dir[i].fName[dir[i].fCount++], f);
                        printf("File created\n");
                    } else {
                        printf("File %s already exists!\n\n", f);
                    }
                    break;
                }
            }
            if (i==dCount)
                printf("Directory %s not found!\n", d);
            break;
        case 3: // delete file
            printf("Enter the name of the directory: ");
            scanf("%s", d);
            for (i = 0; i < dCount; i++) {
                if (!strcmp(d, dir[i].dName)) {
                    if (dir[i].fCount) {
                        printf("Enter the name of the file: ");
                        scanf("%s", f);
                        for (k = 0; k < dir[i].fCount; k++) {
                            if (!strcmp(f, dir[i].fName[k])) {
                                printf("Deleted file: %s\n", f);
```

```
                                dir[i].fCount--;
                                strcpy(dir[i].fName[k], dir[i].fName[
dir[i].fCount]);
                                goto jmp;
                            }
                        }
                        printf("File %s not found!\n", f);
                        goto jmp;
                    } else {
                        printf("Directory empty!\n");
                        goto jmp;
                    }
                }
            }
            printf("Directory %s not found!\n", d);
            jmp: break;
        case 4: // search
            printf("Enter directory name: ");
            scanf("%s", d);
            for (i=0; i<dCount; i++) {
                if (!strcmp(d, dir[i].dName)) {
                    if (dir[i].fCount) {
                        printf("Enter name of the file: ");
                        scanf("%s", f);
                        for (k=0; k<dir[i].fCount; k++) {
                            if (!strcmp(f, dir[i].fName[k])) {
                                printf("File %s found in directory: %
s\n", f, dir[i].dName);
                                goto jmps;
                            }
                        }
                        printf("File %s not found!\n", f);
                        goto jmps;
                    } else {
                        printf("Directory empty!");
                        goto jmps;
                    }
                }
            }
            printf("Directory %s not found!\n", d);
            jmps: break;
        case 5: // display
            if (!dCount)
                printf("No directories!\n");
            else {
                for (i=0; i<dCount; i++) {
                    printf("DIRECTORY: %s\n", dir[i].dName);
                    if (dir[i].fCount) {
```

```
                        for (k=0; k<dir[i].fCount; k++)
                            printf("%s\n", dir[i].fName[k]);
                        printf("\n");
                    } else {
                        printf("Empty!\n\n");
                    }
                }
            }
            break;
        default:
            exit(0);
        }
    }
}

// Heirarchical Directory
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

struct node {
    char name[128];
    bool isDir;
    struct node *p; // parent
    struct node *c[100]; // children
    int i; // no of children
} * head, *curr;

void ls() {
    int i;
    if (!curr->i) {
        printf("Directory Empty!\n");
        return;
    }
    for (i = 0; i < curr->i; i++) {
        if (curr->c[i]->isDir)
            printf("*%s*  ", curr->c[i]->name);
        else
            printf("%s  ", curr->c[i]->name);
    }
    printf("\n");
}

void touch(bool d) {
    char *type = d ? "directory" : "file";
    printf("Enter %s name: ", type);
    char fname[128];
```

```
    scanf("%s", fname);
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    strcpy(temp->name, fname);
    temp->isDir = d;
    temp->p = curr;
    curr->c[curr->i] = temp;
    curr->i += 1; // increment the no. of children
}

void cd() // relative path - from current directory
{
    int i;
    printf("Enter directory name: ");
    char dname[128];
    scanf("%s", dname);
    for (i = 0; i < curr->i; i++) {
        if (!strcmp(curr->c[i]->name, dname) && curr->c[i]->isDir) {
            curr = curr->c[i];
            printf("Changed directory to: %s. \n", curr->name);
            return;
        }
    }
    printf("Directory not present.\n");
}

void cdup() {
    if (curr->p == NULL) {
        printf("You are at the root directory\n");
        return;
    }
    curr = curr->p;
    printf("Changed directory to: %s. \n", curr->name);
}

void rm(bool d) {
    char *type = d ? "directory" : "file";
    printf("Enter name of %s to delete: ", type);
    char name[128];
    scanf("%s", name);
    int i;
    for (i = 0; i < curr->i; i++) {
        if (!strcmp(curr->c[i]->name, name) && ((d && curr->c[i]-
>isDir) || (!d && curr->c[i]->isDir == false))) {
            int t = i;
            while (t < (curr->i) - 1) {
                curr->c[t] = curr->c[t + 1];
                t++;
            }
```

```
            curr->i -= 1;
            printf("Successfully deleted.\n");
            return;
        }
    }
    printf("Not found\n");
}

void main() {
    int in;
    head = (struct node *)malloc(sizeof(struct node));
    strcpy(head->name, "root");
    head->isDir = true;
    head->p = NULL;
    head->i = 0;
    curr = head;
    while (true) {
        printf("\n1. List directory. \n2. Change directory. \n3. Go t
o parent directory. \n4. Add new file. \n5. Delete file. \n6. Create
new directory. \n7. Delete directory. \n8. Print working directory. \
n9. Exit. \nENTER CHOICE: ", curr->name);
        scanf("%d", &in);
        switch (in) {
            case 1:
                ls();
                break;
            case 2:
                cd();
                break;
            case 3:
                cdup();
                break;
            case 4:
                touch(false);
                break;
            case 5:
                rm(false);
                break;
            case 6:
                touch(true);
                break;
            case 7:
                rm(true);
                break;
            case 8:
                printf("%s\n", curr->name);
                break;
            default:
```

```
                exit(0);
            }
        }
}
```

Output:

(i) Single-level directory

```
Enter the directory name: sk

1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 1
Enter filename: f1
File created

1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 1
Enter filename: f2
File created

1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 4

Files in directory sk:
f1
f2

1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 3
Enter the name of the file: f2
File f2 found!

1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 4

Files in directory sk:
f1
f2
```

```
1. Create file.
2. Delete file.
3. Search file.
4. Display files.
5. Exit.
ENTER CHOICE: 2
Enter the name of the file: f2
Deleting file f2
```

(ii) Two-level directory

```
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:1

Enter Name of Directory:d1

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:d2

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:d1

Enter Name of the File to Create:f1

File 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:d2

Enter Name of the File to Create:f2

File created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:5

Directory       Files
d1                      f1
d2                      f2
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:d1

Enter Name of the File to Delete:f1

File f1 Deleted
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:d2

Enter the Name of the File to Search:f2

File f2 Found
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:5

Directory       Files
d1
d2                      f2
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:6
```

(iii) Hierarchical directory

```
You are in root directory.
****************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:6

Enter Name:d1

You are in root directory.
****************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:2

Enter Directory Name:d1

You are in d1 directory.
****************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:4

Enter Name:f1

You are in d1 directory.
****************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:1

f1
You are in d1 directory.
****************************************
```

```
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:3

You are in root directory.
**************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:7

Enter Name of File or Directory to Delete:d1

Successfully Deleted.
You are in root directory.
**************************************
1.Show everything in this Directory
2.Change Directory
3.Go to Parent Directory
4.Create New File
5.Delete File
6.Create New Directory
7.Delete Directory
8.Exit
Enter your choice:8
```

3. Implement Banker's algorithm for deadlock avoidance.

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

void main() {
    int i, j, m, n, count=0, exec; // count is the no. of processes
that have completed
    bool safe = false;

    printf("Enter the no. of processes: ");
    scanf("%d", &n);
    printf("Enter the no. of resource types: ");
    scanf("%d", &m);

    int avlbl[m], max[n][m], alloc[n][m], need[n][m], work[m],
finish[n];

    for (i=0; i<n; i++)
        finish[i] = 0; // Initially, none of the processes have
finished

    printf("Enter Available Resources: \n");
    for (i=0; i<m; i++) {
        scanf("%d", &avlbl[i]);
        work[i] = avlbl[i]; // Initialize Work=Available
    }

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

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

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

    printf("\nNeed matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }


    while (count < n) {
        safe = false;
        for (i=0; i<n; i++) {
```

```
            if (!finish[i]) { // process hasn't terminated
                exec = 1;
                for (j=0; j<m; j++) {
                    if (need[i][j] > work[j]) {
                        exec = 0; // process can't execute
                        break;
                    }
                }
                if (exec) {
                    printf("\nP%d is executing\n", i);
                    finish[i] = 1;
                    count++;
                    safe = true;
                    for (j=0; j<m; j++)
                        work[j] += alloc[i][j]; // release the
resource after execution
                    break;
                }
            }
        }

        if (!safe) {
            printf("The processes are in unsafe state.\n");
            exit(0);
        }

        printf("Work: \n");
        for (i = 0; i < m; i++)
            printf("%d ", work[i]);
        printf("\n");
    }

    printf("\nThe processes are in safe state. \n");
}
```

Output:

```
Enter the no. of processes: 5
Enter the no. of resource types: 3
Enter Available Resources:
3 3 2
Enter Max. Resources:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Allocation:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Need matrix:
7 4 3
```

```
1 2 2
6 0 0
0 1 1
4 3 1

P1 is executing
Work:
5 3 2

P3 is executing
Work:
7 4 3

P0 is executing
Work:
7 5 3

P2 is executing
Work:
10 5 5

P4 is executing
Work:
10 5 7

The processes are in safe state.
```

4. Simulate the following disk scheduling algorithms: (i) FCFS (ii) SCAN (iii) C-SCAN

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

int head, n, max, req[25], p;

void fcfs() {
    int movts=0, i, cur=head;
    printf("Head Movements: \n");
    for (i=0; i<n; i++) {
          if (i != n-1) printf("%d --> ", req[i]);
          else printf("%d\n", req[i]);
          movts += abs(req[i]-cur);
          cur = req[i];
    }
    printf("No. of cylinder movements: %d\n", movts);
}

void sortRequests() {
    bool found = false;
    int i, j, temp, s;
    for (i=0; i<n-1; i++) {
          s=i;
          for (j=i+1; j<n; j++)
                if (req[j] < req[s])
                      s=j;
          temp = req[i];
          req[i] = req[s];
          req[s] = temp;
          if (!found && req[i] >= head) {
                found = true;
                p = i;
          }
    }
}

void scan() {
    int movts=0, i, cur=head;
    printf("Head Movements: \n");
    for (i=p; i<n; i++) {
          printf("%d --> ", req[i]);
          movts += abs(req[i]-cur);
          cur = req[i];
          if (i==n-1) {
                movts += abs(max-cur); // add the movt to the end of
the disk
                cur = max; // move to the end of the disk
          }
    }
    for (i=p-1; i>=0; i--) {
          if (i) printf("%d --> ", req[i]);
          else printf("%d\n", req[i]);
          movts += abs(req[i]-cur);
```

```
            cur = req[i];
        }
        printf("No. of cylinder movements: %d\n", movts);
}

void cScan() {
        int movts=0, i, cur=head, j=p;
        printf("Head Movements: \n");
        for (i=0; i<n; i++) {
                if (i != n-1) printf("%d --> ", req[j]);
                else printf("%d\n", req[j]);
                movts += abs(req[j]-cur);
                cur = req[j];
                if (j==n-1) {
                        movts += abs(max-cur) + max; // add the movt to the end
of the disk and from the end of the disk to the beginning
                        cur = 0; // move to the end of the disk
                }
                j=(j+1)%n;
        }
        printf("No. of cylinder movements: %d\n", movts);
}

void main()
{
        int i;
        char ch[2];
        do {
                printf("Enter the upper limit of cylinders: ");
                scanf("%d", &max);
                printf("Enter the disk head position: ");
                scanf("%d", &head);
                printf("Enter the no. of requests: ");
                scanf("%d", &n);

                printf("Enter the requests: \n");
                for (i=0; i<n; i++)
                        scanf("%d", &req[i]);

                printf("\nFCFS: \n");
                fcfs();

                sortRequests();

                printf("\nSCAN: \n");
                scan();

                printf("\nC-SCAN: \n");
                cScan();

                printf("\nDo you want to continue? Y/N: ");
                scanf("%s", ch);
        } while (ch[0]=='y' || ch[0]=='Y');
}
```

Output:

```
Enter the upper limit of cylinders: 199
Enter the disk head position: 100
Enter the no. of requests: 5
Enter the requests:
23 89 132 42 187

FCFS:
Head Movements:
23 --> 89 --> 132 --> 42 --> 187
No. of cylinder movements: 421

SCAN:
Head Movements:
132 --> 187 --> 89 --> 42 --> 23
No. of cylinder movements: 275

C-SCAN:
Head Movements:
132 --> 187 --> 23 --> 42 --> 89
No. of cylinder movements: 387

Do you want to continue? Y/N: n
```

5. Implement the Producer-Consumer problem using semaphores.

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

int mutex=1, n, full=0, empty, buffer[25], temp=0, f=-1, r=-1;

void wait(int *s) {
    (*s)--;
}
void signal(int *s) {
    (*s)++;
}

void producer() {
    int x;
    wait (&mutex);
    signal(&full);
    wait(&empty);
    // produce an item
    printf("Enter the item to be produced: ");
    scanf("%d", &x);

    // place the item in buffer
    if (f==-1) f++;
    r = (r+1)%n;
    buffer[r] = x;
    printf("Produced item: %d\n\n", x);

    signal(&mutex);
}

void consumer() {
    wait(&mutex);
    wait(&full);
    signal(&empty);

    //remove an item from buffer
    int x = buffer[f];
    if (f==r) f=r=-1;
    else f = (f+1)%n;

    signal(&mutex);

    // consume the item
    printf("Consumed item: %d\n\n", x);
}

void main()
{
    int ch;
    printf("Enter the size of the buffer: ");
    scanf("%d", &n);
    empty=n;
```

```
    while (true)
    {
          printf("1. Producer. \n2. Consumer. \n3. Exit. \nENTER
CHOICE: ");
          scanf("%d", &ch);
          switch(ch) {
                case 1:
                      if (empty)
                            producer();
                      else
                            printf("Buffer full!\n\n");
                      break;
                case 2:
                      if (full)
                            consumer();
                      else
                            printf("Buffer empty!\n\n");
                      break;
                default: exit(0);
          }
    }
}
```

Output:

```
Enter the size of the buffer: 3
1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 1
Enter the item to be produced: 5
Produced item: 5

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 1
Enter the item to be produced: 17
Produced item: 17

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 2
Consumed item: 5

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 1
Enter the item to be produced: 13
Produced item: 13

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 1
```

```
Enter the item to be produced: 12
Produced item: 12

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 1
Buffer full!

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 2
Consumed item: 17

1. Producer.
2. Consumer.
3. Exit.
ENTER CHOICE: 3
```

6. Simulate the working of Dining Philosophers' problem.

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N]; // semaphore for each philosopher

void test(int phnum) {
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        state[phnum] = EATING; // state of eating

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",
                      phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);

        // sem_post(&S[phnum]) has no effect during takefork
        // used to wake up hungry philosophers during putfork
        sem_post(&S[phnum]);
    }
}

void take_fork(int phnum) { // take up chopsticks
    sem_wait(&mutex);
    state[phnum] = HUNGRY; // state that hungry
    printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum); // eat if neighbours are not eating
    sem_post(&mutex);
    sem_wait(&S[phnum]); // if unable to eat wait to be signalled
    sleep(1);
}

void put_fork(int phnum) { // put down chopsticks
    sem_wait(&mutex);

    // state that thinking
    state[phnum] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);
```

```
    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

void* philospher(void* num)
{
    while (1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

int main()
{
   int i;
    pthread_t thread_id[N]; // create 5 threads

    sem_init(&mutex, 0, 1); // initialize the semaphores

    for (i = 0; i < N; i++)
        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++) { // create philosopher processes
        pthread_create(&thread_id[i], NULL, philospher, &phil[i]);
        printf("Philosopher %d is thinking\n", i + 1);
    }

    for (i = 0; i < N; i++)
        pthread_join(thread_id[i], NULL);
}
```

Output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
```

```
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
```