write a c programs to simulate the following CPU scheduling algorithms

a) FCFS

```
#include <stdio.h>

Struct process
{
        int processNo, at, bt, ct, tat, wt;
};

Void readProcess (Struct process r[], int n)
{
        int i;
        for(i=0; i<n; i++)
        {
                printf ("Enter process Number : ");
                Scanf ("%d", &r[i].processNo);
                printf ("Enter Arrival time : ");
                Scanf ("%d", &r[i].at);
                printf ("Enter Burst time : ");
                Scanf ("%d", &r[i].bt);
        }
}
```

```
Void Time(struct process r[], int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        if (i==0 || r[i].at > r[i-1].ct)
            r[i].ct = r[i].at + r[i].bt;
        else
            r[i].ct = r[i-1].ct + r[i].bt;
        r[i].tat = r[i].ct - r[i].at;
        r[i].wt = r[i].tat - r[i].bt;
    }
}
Void SortProcess(struct process r[], int n)
{
    int i,j;
    struct process temp;
    for(i=0;i<n;i++)
        for(j=0;j<n-i;j++)
            if (r[j].at < r[j-1].at)
            {
                temp = r[j];
                r[j] = r[j-1];
                r[j-1] = temp;
            }
}
```

```
Void printProcess(struct process r[], int n)
{
    int i;
    printf("Process No. \t at \t bt \t ct \t tat \t wt \n");
    for(i=0; i<n; i++)
    {
        printf("%d \t\t %d \t %d \t %d \t %d \t %d \n",
            r[i].processNo, r[i].at, r[i].bt, r[i].ct,
            r[i].tat, r[i].wt);
    }
}

int main()
{
    struct process p[30];
    int n;
    printf("Enter number of processes : ");
    Scanf("%d", &n);
    readProcess(p,n);
    SortProcess(p,n);
    Time(p,n);
    printProcess(p,n);
}
```

INPUT:

Enter number of processes: 4

Enter process Number: 1

Enter Arrival time: 2

Enter Burst time: 1

Enter process Number: 2

Enter Arrival time: 0

Enter Burst time: 2

Enter process Number: 3

Enter Arrival time: 4

Enter Burst time: 2

Enter process Number: 4

Enter Arrival time: 5

Enter Burst time: 1

OUTPUT:

Process No.	at	bt	ct	tat	wt
2	0	2	2	2	0
1	2	1	3	1	0
3	4	2	6	2	0
4	5	1	7	2	1

b) SJF

```
#include <stdio.h>

struct process
{
        int processNo , at , bt , ct, tat, wt ;
};

void readProcess (struct process p[] , int n)
{
        int i ;
        for(i=0 ; i<n ; i++)
        {
                printf ("Enter process Number : ");
                scanf ("%d" , &p[i].processNo);
                printf ("Enter Arrival time : ");
                scanf ("%d" , &p[i].at) ;
                printf ("Enter Burst time :");
                scanf ("%d", &p[i]. bt);
        }
}

void Time (struct process p[] , int n)
{
        int i ;
        for(i=0 ; i<n ; i++)
        {
```

```
        if(i==0 || r[i].at > r[i-1].ct)
                r[i].ct = r[i].at + r[i].bt;
        else
                r[i].ct = r[i-1].ct + r[i].bt;
        r[i].tat = r[i].ct - r[i].at;
        r[i].wt = r[i].tat - r[i].bt;
    }
}

Void SortProcess(struct process r[], int n)
{
    int i, j, min=0, k=1, btime=0, count=1;
    Struct process temp;
    for(i=0; i<n; i++)
    {
        for(j=i; j<n-i; j++)
        {
            if(r[j].at < r[j-1].at)
            {
                temp = r[j];
                r[j] = r[j-1];
                r[j-1] = temp;
            }
            if(r[j].at == r[j-1].at)
            {
```

```
                if (r[j].bt < r[j-1].bt)
                {
                    temp = r[j];
                    r[j] = r[j-1];
                    r[j-1] = temp;
                }
            }
        }
    }
    for(i=0; i<n; i++)
    {
        if(i==0)
            btime = r[0].at + r[0].bt;
        else
            btime = btime + r[i].bt;
        min = r[k].bt;
        count = k;
        while(r[count].at <= btime && count < n)
        {
            count++;
        }
        for(j=k; j<count; j++)
        {
            if(btime >= r[j].at && r[j].bt < min)
            {
```

```
                temp = r[k];
                r[k] = r[j];
                r[j] = temp;
                min = r[k].bt;
            }
        }
        k++;
    }
}

void printProcess(struct process r[], int n)
{
    int i;
    printf("Process No.\t at \t bt \t ct \t wt \t tat");
    for(i=0;i<n;i++)
    {
        printf("%d \t\t %d \t %d \t %d \t
            %d \t %d \n", r[i].processNo,
            r[i].at, r[i].bt, r[i].ct,
            r[i].tat, r[i].wt;
    }
}
```

```
int main( )
{
        struct process p[30];
        int n;
        printf ("Enter Number of processes : ");
        scanf ("%d", &n);
        readProcess (p,n);
        sortProcess (p,n);
        Time (p,n);
        printProcess(p,n);
}
```

Input:

Enter number of processes : 5

Enter process number : 1

Enter Arrival time : 1

Enter Burst time : 5

Enter process number : 2

Enter Arrival time : 1

Enter Burst time : 5

Enter process number: 3

Enter Arrival time: 6

Enter Burst time: 12

Enter process Number: 4

Enter Arrival time: 8

Enter Burst time: 2

Enter process number: 5
Enter Arrival time: 10
Enter Burst time: 3

output:

process NO.	at	bt	ct	tat	wt
1	1	5	6	5	0
2	1	5	11	10	5
5	10	3	14	4	1
4	8	2	16	8	6
3	6	12	28	22	10

c) priority

```
#include <stdio.h>
struct process
{
    int processNo, at, bt, ct, tat, wt, pty;
};
void readProcess(struct process r[], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        printf("Enter process number : ");
        Scanf("%d", &r[i].processNo);
        printf("Enter priority : ");
        Scanf("%d", &r[i].pty);
        printf("Enter Arrival time : ");
        Scanf("%d", &r[i].at);
        printf("Enter Burst time : ");
        Scanf("%d", &r[i].bt);
    }
}
void Time(struct process r[], int n)
{
    int i;
    for(i=0; i<n; i++)
```

```
    {
        if (i==0 || r[i].at >r[i-1].ct)
            r[i].ct = r[i].at +r[i].bt;
        else
            r[i].ct =r[i-1].ct +r[i].bt;
        r[i].tat =r[i].ct -r[i].at;
        r[i].wt =r[i].tat -r[i].bt;
    }
}
Void SortProcess (struct process r[],int n)
{
    int i,j ,min=0,K=1,btime=0, count=1;
    struct process temp;
    for(i=0;i<n;i++)
    {
        for(j=1;j<n-i;j++)
        {
            if (r[j].at <r[j-1].at)
            {
                temp =r[j];
                r[j] =r[j-1];
                r[j-1] =temp;
            }
            if (r[j].at ==r[j-1].at)
                if (r[j].pty <r[j-1].pty)
                {
                    temp=r[j];
                    r[j]=r[j-1];
                    r[j-1]=temp;
                }
```

```
    }
}
for(i=0 ;i<n ; i++)
{
    if (i==0)
        btime = r[0].at + r[0].bt;
    else
        btime = btime + r[j].bt ;
    min = r[k].pty;
    count = k;
    while(r[count].at <= btime && count<n)
    {
        count++;
    }
    for(j=k ;j<count ; j++)
    {
        if(btime >= r[j].at && r[j].pty
                                <min)
        {
            temp = r[k];
            r[k] = r[j];
            r[j] = temp;
            min = r[k].pty;
        }
    }
    k++;
}
}
```

```
void printProcess(struct process r[], int n)
{
        int i ;
        printf ("Process No. \t priority \t at \t bt \t
                ct \t tat \t wt \n");
        for (i=0; i<n; i++)
        {
                printf ("%d \t\t %d \t\t %d \t %d
                \t %d \t %d \t %d \n",
                r[i].processNo , r[i].pty,
                r[i].at , r[i].bt , r[i].ct,
                r[i].tat , r[i].wt)
        }
}
int main( )
{
        struct process p[30];
        int n;
        printf ("Enter number of processes :");
        scanf ("%d", &n);
        readprocess (p,n);
        sortProcess(p,n);
```

```
        Time(p,n);
        printProcess(p,n);
}
```

Input:

```
Enter number of processes: 5
Enter process number: 1
Enter priority: 2
Enter Arrival time: 0
Enter Burst time: 2

Enter process number: 2
Enter priority: 1
Enter Arrival time: 1
Enter Burst time: 4

Enter process number: 3
Enter priority: 6
Enter Arrival time: 2
Enter Burst time: 8

Enter process number: 4
Enter priority: 4
Enter Arrival time: 3
Enter Burst time: 2

Enter process number: 5
Enter priority: 3
```

Enter Arrival time: 4
Enter Burst time: 3

OUTPUT:

Process No.	priority	at	bt	ct	tat	wt
1	2	0	2	2	2	0
2	1	1	4	6	5	1
5	3	4	3	9	5	2
4	4	3	2	11	8	6
3	6	2	8	19	17	9

30/1/2020

d) Round Robin

```
#include <stdio.h>
struct procs
{
    int no, at, bt, ct, tat, wt, tbt;
} p[10], m2[30], tmp;
int i, n, j, k=1, tq, cnt, pt, m=1;
void readProcs()
{
    printf("Enter time quantum: ");
    scanf("%d", &tq);
    for (i=0; i<n; i++)
    {
        p[i].no = i+1;
        printf("P%d - Arrival time: ", i+1);
        scanf("%d", &p[i].at);
        printf("P%d - Burst time: ", i+1);
        scanf("%d", &p[i].bt);
        p[i].tbt = p[i].bt;
    }
    cnt = n;
}
```

```
void sort()
{
    for(i=0;i<n-1;i++)
    {
        for(j=1;j<n-i;j++)
        {
            if(p[j].at<p[j-1].at)
            {
                tmp=p[j];
                p[j]=p[j-1];
                p[j-1]=tmp;
            }
            if(p[j].at==p[j-1].at)
            {
                if(p[j].bt<p[j-1].bt)
                {
                    tmp=p[j];
                    p[j]=p[j-1];
                    p[j-1]=tmp;
                }
            }
        }
    }
    mq[0]=p[0];
}
```

```
Void calcTime( )
{
    i=0, j=0;
    while (cnt >0)
    {
        if (mq[i] .tbt < tq)
        {
            mq[i].tbt -= tq;
            if (i==0)
                pt = mq[0].at + mq[i].bt;
            else
                pt = pt + mq[i].tbt + tq;
        }
        else
        {
            mq[i].tbt -= tq;
            if (i==0)
                pt = mq[0].at + tq;
            else
                pt += tq;
        }
        while (k<n && p[k].at <= pt)
        {
            mq[m] = p[k];
            m++;
```

```
        if (mq[i].tbt > 0)
        {
            mq[m] = mq[i];
            m++;
        }
        else
        {
            cnt--;
            mq[i].ct = pt;
            p[j] = mq[i];
            p[j].tat = p[j].ct - p[j].at;
            p[j].wt = p[j].tat - p[j].bt;
            j++;
        }
        i++;
    }
}

void printProcs()
{
    printf("pno  at  bt  ct  tat  wt \n");
    for(i=0;i<n;i++)
        printf("p%d  %3d  %3d  %3d  %4d  %3d \n",
            p[i].no, p[i].at, p[i].bt, p[i].ct,
            p[i].tat, p[i].wt);
}
```

```
int main()
{
    printf("Enter number of processes : ");
    scanf("%d",&n);
    readProcs();
    sort();
    calcTime();
    printProcs();
}
```

INPUT:

Enter number of processes : 6

Enter time quantum : 3

P1 - Arrival time : 5

P1 - Burst time : 5

P2 - Arrival time : 4

P2 - Burst time : 6

P3 - Arrival time : 3

P3 - Burst time : 7

P4 - Arrival time : 1

P4 - Burst time : 9

P5 - Arrival time : 2

P5 - Burst time : 2

P6 - Arrival time : 6

P6 - Burst time : 3

OUTPUT:

P.No	at	bt	ct	tat	wt
P5	2	2	6	4	2
P6	6	3	21	15	12
P2	4	6	27	23	17
P4	1	9	30	29	20
P1	5	5	32	27	22
P3	3	7	33	30	23

write a c program to simulate Bankers Algorithm for deadlock avoidance and prevention.

```
#include <stdio.h>
struct process
{
    int processNo, alloc[10], max[10], need[10];
};
int order[30];

void readProcess( struct process p[], int n, int m)
{
    int i, j;
    printf ("Enter allocation matrix :\n");
    for(i=0; i<n; i++)
    {
        printf("Enter process %d allocations :\n",
                                    i+1);

        for(j=0; j<m; j++)
        {
            printf("Enter alloc[%d] : ", j);
            scanf("%d", &p[i].alloc[j]);
        }
    }
    printf("Enter Max Matrix :\n");
```

```
    for(i=0 ; i<n ; i++)
    {
        printf("Enter process %d max
                resources :\n", i+1);

        for(j=0 ; j<m; j++)
        {
            printf("Enter max[%d] :", j);
            scanf("%d", &r[i].max[j]);
        }
    }
    for(i=0 ; i<n ; i++)
    {
        for(j=0 ; j<m ; j++)
            r[i].need[j] = r[i].max[j]
                - r[i].alloc[j];
    }
}
int resourceReq(struct process r[], int n, int m,
                int avail[])
{
    int i, x;
    printf("Enter process number for request :");
    scanf("%d", &x);
    for(i=0; i<m; i++)
    {
```

```
        if (req[i]<=r[x].need[i] &&
                    req[i]<=avail[i])
        {
            avail[i] -= req[i];
            r[x].alloc[i] += req[i];
            r[x].need[i] -= req[i];
        }
        else
            return 0;
    }
    return 1;
}

int safeState(struct process r[], int n, int m,
              int avail[])
{
    int i, j, finish[n], work[m], count=0, s=0,
        b=1, flag;
    for(i=0; i<m; i++)
        work[i] = avail[i];
    finish[0] = 0;
    while(b==1 && count<n)
    {
        flag=
        b=0;
```

```
for(i=0;i<n;i++)
{
    flag=0;
    for(j=0;j<m;j++)
    {
        if(r[i].need[j]<=
            work[j] &&
            finish[i]==0)
        {
            flag++;
        }
        else
            break;
    }
    if(flag==m)
    {
        for(int k=0;k<m;k++)
            work[k] +
            =r[i].alloc[k];
        order[s]=i;
        s++;
        finish[i]=1;
        count++;
        b=1;
    }
}
}
```

```
        if (count == n)
            return 1;
        else
            return 0;
}

int main()
{
    int n, m, avail[10], req[10], i, r, q;
    struct process p[30];
    printf("Enter number of processes : ");
    scanf("%d", &n);
    printf("Enter number of resource type : ");
    scanf("%d", &m);

    for(i=0; i<m; i++)
    {
        printf("Enter number of resources
                of type %d : ", i+1);
        scanf("%d", &avail[i]);
    }

    readProcess(p, n, m);
    printf("Enter Request Matrix : \n");
    for(i=0; i<m; i++)
    {
        printf("Enter request[%d] : ", i);
        scanf("%d", &req[i]);
    }
```

```
r=resourceReq(p,req,n,m,avail);
if(r==1)
{
    printf("Resource can be allocated");
    q=SafeState(p,n,m,avail);
    if(q==1)
    {
        printf("The system is in safe
                state\n");
        printf("The order of processes:\n");
        for(int a=0;a<n;a++)
            printf("%d ",order[a]);
    }
    else
    {
        printf("System is in deadlock
                state");
    }
}
else
    printf("Resource cannot be allocated\n");
```

INPUT:

Enter number of processes : 5

Enter number of resource types : 3

Enter number of resources of type 1 : 3

Enter number of resources of type 2 : 3

Enter number of resources of type 3 : 2

Enter Allocation Matrix :

Enter process 1 allocations :

Enter alloc [0] : 0

Enter alloc [1] : 1

Enter alloc [2] : 0

Enter process 2 allocations :

Enter alloc [0] : 2

Enter alloc [1] : 0

Enter alloc [2] : 0

Enter process 3 allocations :

Enter alloc [0] : 3

Enter alloc [1] : 0

Enter alloc [2] : 2

Enter process 4 allocations :

Enter alloc [0] : 2

Enter alloc [1] : 1

Enter alloc [2] : 1

```
Enter process 5 allocations:
Enter alloc[0]: 0
Enter alloc[1]: 0
Enter alloc[2]: 2

Enter Max Matrix:
Enter process 1 max resources:
Enter max[0]: 7
Enter max[1]: 5
Enter max[2]: 3

Enter process 2 max resources:
Enter max[0]: 3
Enter max[1]: 2
Enter max[2]: 2

Enter process 3 max resources:
Enter max[0]: 9
Enter max[1]: 0
Enter max[2]: 2

Enter process 4 max resources:
Enter max[0]: 2
Enter max[1]: 2
```

Enter max[2]: 2

Enter process 5 max resources:

Enter max[0]: 4

Enter max[1]: 3

Enter max[2]: 3

Enter Request Matrix:

Enter request[0]: 1

Enter request[1]: 0

Enter request[2]: 2

Enter process number for request: 1

OUTPUT:

Resource can be allocated.

The system is in safe state

The order of processes:

1 3 4 0 2

write a C program to implement producer consumer problem using semaphores using UNIX/LINUX system calls.

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

sem_t s,full,empty;
int i;
char buf[200];

void * producer (void * arg)
{
    for(i=0;i<8;i++)
    {
        sem_wait(&empty);
        sem_wait(&s);
        buf[i]=i;
        printf("Produced item is %d ",buf[i]);
        sem_post(&s);
        sem_post(&full);
    }
    pthread_exit(" ");
}
```

```
void * consumer (void *arg)
{
        for(i=0;i<2;i++)
        {
                sem_wait(&full);
                sem_wait(&s);
                printf("consumed item is %d ", buf[i]);
                sem_post(&s);
                sem_post(&empty);
        }
        pthread_exit(" ");
}
int main()
{
        pthread_t t1,t2;
        sem_init(&empty,0,1);
        sem_init(&full,0,0);
        sem_init(&s,1,1);
        pthread_create(&t1, NULL,producer,NULL);
        pthread_create(&t2, NULL, consumer, NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
}
```

OUTPUT:

produced item is 0
produced item is 1
consumed item is 0
consumed item is 1

write c programs to illustrate following IPC mechanisms

a) pipes

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

int main()
{
    int cid, l, fd[2];
    char s[20], s1[20];
    if (pipe(fd) == -1)
    {
        printf("pipe not created \n");
        exit(1);
    }
    printf("Enter message: ");
    gets(s);
    cid = fork();
    l = strlen(s);
    if (cid == -1)
    {
        printf("process not created \n");
        exit(0);
    }
```

```
        if (cid == 0)
        {
                close (fd[1]);
                read (fd[0], s1, 1);
                printf ("Message is : \n");
                puts(s1);
        }
        else
        {
                close (fd[0]);
                write (fd[1], s, 2);
                printf ("Message sent into pipe \n");
        }
}
```

INPUT:

Enter message : Anusoop

OUTPUT:

Message sent into pipe

Message is:

Anusoop.

b) FIFOs

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

int main()
{
        int id;
        char * myff = "/home/fifo";
        char s[20];
        mkfifo(myff, 0666);
        id = open(myff, O_WRONLY);
        printf("Enter msg: ");
        get(s);
        write(id, s, strlen(s)+1);
        close(id);
}
```

INPUT:

Enter msg: Rinku

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

int main()
{
        int id;
        char * myff = "/home/fifo";
        char s[20];
        mkfifo(myff, 0666);
        id = open (myff, O_RDONLY);
        read (id, s, 30);
        close(id);
        puts(s);
}
```

OUTPUT:

Rinku.

c) Message Queues

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/msg.h>

struct message
{
        long mesg_type;
        char mesg_text[100];
}msg;

int main()
{
        key_t  key = 6;
        int msgid;
        msgid = msgget(key, 0666 | IPC_CREAT);
        msg.mesg_type = 1;
        printf("write Data : ");
        gets(msg.mesg_text);
        msgsnd(msgid, &msg, sizeof(msg), 0);

}
```

INPUT:

write Data : Rinku

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/msg.h>

struct message
{
    long mesg_type;
    char mesg_text[100];
}msg;

int main()
{
    key_t key = 6;
    int msgid;
    msgid = msgget (key, 0666 | IPC_CREAT);
    msgrcv (msgid, &msg, sizeof(msg), 1, 0);
    printf("\n%s", msg.mesg_text);
    msgctl (msgid, IPC_RMID, Null);
}
```

OUTPUT:

Rinku

d) Shared Memory

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>

int main()
{
    key_t key = 6;
    int shmid = shmget(key, 1024, 0666|IPC_CREAT);
    char *str = (char*) shmat(shmid, (void*)0, 0);
    printf("Enter Data : ");
    gets(str);
}
```

INPUT:

Enter Data : Rinku

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main()
{
        key_t key=6;
        int shmid=shmget(key,1024,0666|IPC_CREAT);
        char *str=(char*)shmat(shmid,(void*)0,0);
        printf("%s\n",str);
        shmdt(str);
        shmctl(shmid,IPC_RMID,NULL);
}
```

OUTPUT:

Rinku

write c program to simulate following memory management techniques

a) paging

```
#include <stdio.h>
int main()
{
    int i, ladd, psize, ptable[10];
    int pageno, offset, phyadd, n;
    printf("Enter number of pages : ");
    scanf("%d", &n);
    printf("Enter logical address : ");
    scanf("%d", &ladd);
    printf("Enter page size : ");
    scanf("%d", &psize);
    printf("Enter frame numbers : \n");
    for(i=0; i<n; i++)
        scanf("%d", &ptable[i]);
    pageno = ladd/psize;
    offset = ladd % psize;
    phyadd = ptable[pageno] * psize + offset;
    printf("The physical address is : %d",
                phyadd);
}
```

~~OUTPUT 1~~ INPUT:

Enter number of pages: 4

Enter logical address: 11

Enter page size: 4

Enter frame numbers:

2

1

3

0

OUTPUT:

The physical address is: 15

b) Segmentation

```
#include <stdio.h>
struct segment
{
    int segno;
    int baseadd;
    int limit;
    int offset;
};
int main()
{
    int i,n,seg_no,phyadd;
    struct segment arr[10];
    printf("Enter number of segments: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter %d segno: ",i);
        scanf("%d",&arr[i].segno);
        printf("Enter %d baseaddress: ");
        scanf("%d",&arr[i].baseadd);
        printf("Enter %d limit:",i);
        scanf("%d",&arr[i].limit);
```

```
            printf("Enter offset of %d
                    segment : ", i);
            scanf("%d", &arr[i].offset);
    }
    printf("Enter segment_no : ");
    scanf("%d", &seg_no);
    if(arr[seg_no].limit >=arr[seg_no].offset)
    {
            phyadd =arr[seg_no].baseadd +
                    arr[seg_no].offset;
            printf("The physical address is
                    %d", phyadd);
    }
    else
    {
            printf("Not valid offset for this
                    segment");
}
```

INPUT:

Enter number of segments : 1

Enter 0 segno : 0

Enter 0 base address : 2514

Enter 0 limit : 514

Enter offset of 0 segment : 254

Enter segment_no : 0

OUTPUT: The physical address is : 2768.

write programs using I/O system calls of UNIX/ LINUX operating system

(open, read, write, close, fcntl, seek, stat)

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

int main()
{
        int n, fd, fd1, i, fd2;
        char buff[50], buff1[50];
        struct stat s;
        fd = open("Sample.txt", O_CREAT | O_RDWR, 0666);
        n = read(fd, buff, 18);
        printf("Numbers of characters : %d", n);
        fd1 = open("aaa.txt", O_CREAT | O_RDWR, 0666);
        write(fd1, buff, n);

        fd2 = open("abc.c", O_RDWR, 0666);
        lseek(fd2, 9, SEEK_SET);
        read(fd2, buff, 9);
        for(i=0; i<9; i++)
                printf("%c", buff1[i]);
```

```
    stat ("Sample.txt", &s);
    printf ("\nSize is %d", s.st_size);
    close (fd);
    close (fd1);
    close (fd2);
}
```

OUTPUT:

Number of characters : 18

<stdio.h>

Size is 19

(opendir, readdir)

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

int main()
{
        struct dirent *d;
        DIR *dr = opendir("/home/systemno-13/18SSIA0516");
        while((d=readdir(dr)) != NULL)
                printf("%s\n", d->d_name);
        closedir(dir);
}
```

OUTPUT:

.SysCall.c

.dir.c

.roundRobin.c

.SJF.C