```
 #include<unistd.h>
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
sem_t ch[5];
void *philosopher(void *arg)
{
        int i=(int)arg;
        int left,right;
        printf("\n%d philosopher is created\n",arg);
        sleep(5);
        left=i;
        right=(i+1)%5;
        printf("\n%d philosopher is hungry\n",arg);
        sem_wait(&ch[left]);
        sem_wait(&ch[right]);
        printf("\n%d philosopher is eating\n",arg);
        sleep(5);
        sem_post(&ch[right]);
        sem_post(&ch[left]);
        printf("\n%d philosopher is complete\n",arg);
}
int main()
{
        int i;
        pthread_t pid[5];
        for(i=0;i<5;i++)
                sem_init(&ch[i],0,1);
        for(i=0;i<5;i++)
        {
                pthread_create(&pid[i],NULL,philosopher,(void *)i);
                sleep(1);
        }
        for(i=0;i<5;i++)
                pthread_join(pid[i],NULL);
}
```

```
//PRODUCER - CONSUMER PROBLEM

#include<stdio.h>
#include<pthread.h>
int buff[10];
pthread_mutex_t m;              // Delcaration

void producer()
{
        int i=0,n;
        while(1)
                {
                        pthread_mutex_lock(&m);
                        n=random()%10;
                        buff[i++]=n;
                        printf("%d Element is added at %d Location", n, i-1);
                        if(i==10) i=0;
                        pthread_mutex_unlock(&m);
                        sleep(2);
                }
}

void consumer()
{
        int key, i=0;
        while(1)
        {
                pthread_mutex_lock(&m);
                key=buff[i++];
                printf("%d Element is extracted at %d Location", key, i-1);
                if(i==10) i=0;
                pthread_mutex_unlock(&m);
                sleep(2);

        }
}

void main()
{
        pthread_mutex_init(&m,NULL);            // Initialize
        pthread_t pt, ct;
        pthread_create(&pt, NULL, producer, NULL);
        pthread_create(&ct, NULL, consumer, NULL);

        pthread_join(pt, NULL);
        pthread_join(ct, NULL);

}
```

```
/*<--------------PREPROCESSING STATEMENTS--------------->*/
#include <stdio.h>
#include <conio.h>

/*<--------------MAIN FUNCTION--------------->*/

int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10],
safeSequence[10];
int p, r, i, j, process, count;
count = 0;

printf("Enter the no of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
        completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
                scanf("%d", &Max[i][j]);
}

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

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
                scanf("%d", &avail[i]);


        for(i = 0; i < p; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = Max[i][j] - alloc[i][j];

do
{
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < p; i++)
        {
                for( j = 0; j < r; j++)
                        printf("%d  ", Max[i][j]);
                printf("\t\t");
                for( j = 0; j < r; j++)
                        printf("%d  ", alloc[i][j]);
                printf("\n");
        }

        process = -1;

        for(i = 0; i < p; i++)
        {
                if(completed[i] == 0)//if not completed
                {
                        process = i ;
                        for(j = 0; j < r; j++)
                        {
                                if(avail[j] < need[i][j])
                                {
                                        process = -1;
                                        break;
                                }
                        }
                }
                if(process != -1)
                        break;
        }

        if(process != -1)
        {
                printf("\nProcess %d runs to completion!", process + 1);
                safeSequence[count] = process + 1;
                count++;
                for(j = 0; j < r; j++)
                {
                        avail[j] += alloc[process][j];
                        alloc[process][j] = 0;
                        Max[process][j] = 0;
                        completed[process] = 1;
                }
        }
}while(count != p && process != -1);

if(count == p)
{
        printf("\nThe system is in a safe state!!\n");
        printf("Safe Sequence : < ");
        for( i = 0; i < p; i++)
```

```
                        printf("%d  ", safeSequence[i]);
        printf(">\n");
}
else
        printf("\nThe system is in an unsafe state!!");
getch();
}
```

```c
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;

void getData()
{
    printf("\nEnter length of page reference sequence:");
    scanf("%d",&n);
    printf("\nEnter the page reference sequence:");
    for(i=0; i<n; i++)
        scanf("%d",&in[i]);
    printf("\nEnter no of frames:");
    scanf("%d",&nf);
}

void initialize()
{
    pgfaultcnt=0;
    for(i=0; i<nf; i++)
        p[i]=9999;
}

int isHit(int data)
{
    hit=0;
    for(j=0; j<nf; j++)
    {
        if(p[j]==data)
        {
            hit=1;
            break;
        }

    }

    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }

}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}

void fifo()
{
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];

            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}


void optimal()
{
    initialize();
    int near[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i; k<n; k++)
                {
                    if(pg==in[k])
                    {
                        near[j]=k;
                        found=1;
                        break;
                    }
                    else
                        found=0;
                }
                if(!found)
                    near[j]=9999;
            }
            int max=-9999;
            int repindex;
            for(j=0; j<nf; j++)
            {
                if(near[j]>max)
                {
                    max=near[j];
                    repindex=j;
                }
            }
            p[repindex]=in[i];
            pgfaultcnt++;

            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}

void lru()
{
    initialize();
    int least[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i-1; k>=0; k--)
                {
                    if(pg==in[k])
                    {
                        least[j]=k;
                        found=1;
                        break;
                    }
                    else
                        found=0;
                }
                if(!found)
                    least[j]=-9999;
            }
            int min=9999;
            int repindex;
            for(j=0; j<nf; j++)
            {
                if(least[j]<min)
                {
                    min=least[j];
                    repindex=j;
                }
            }
            p[repindex]=in[i];
            pgfaultcnt++;

            dispPages();
        }
        else
            printf("No page fault!");
    }
    dispPgFaultCnt();
}

int main()
{
    int choice;
    while(1)
    {
        printf("\nPage Replacement Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n\n5.Exit\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            getData();
            break;
        case 2:
            fifo();
            break;
        case 3:
            optimal();
            break;
        case 4:
            lru();
            break;
        default:
            return 0;
            break;
        }
    }
}
```

```c
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<unistd.h>
#define FIFO_PIPE "xyz"

int main()
{
        int fd, i=0;
        char str[200],ch;

        mkfifo(FIFO_PIPE, 0644);

        printf("\n Enter Your message (press @ to terminate) \n");
        while((ch=fgetc(stdin)) != '@')
        {
                str[i++] = ch;
        }

        str[i]='\0';

        fd=open(FIFO_PIPE,O_WRONLY);
        write(fd, str, strlen(str));

        close(fd);
        return 0;
}
```

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<unistd.h>
#define FIFO_PIPE "xyz"

int main()
{
int fd, i, c=0, w=0, l=0;
char str[100];

//mkfifo(FIFO_PIPE, 0777);              // mkfifo - create named pipe special file
with 777 permission

fd=open(FIFO_PIPE,O_RDONLY);    // Open a File in Read Only Mode
read(fd, str, sizeof(str));             // Read file and store into "str" buffer

printf("\nReceived Data : %s", str);    // Print "str"

for(i=0; str[i]!='\0'; i++)                             // Calculate no of lines,
words and characters
{
        if(str[i] == ' ' || str[i] == '\n')             w++;
        if(str[i] == '\n')
l++;

        c++;
}
printf("\n No of characters = %d", c);  // print no of lines, words and characters
printf("\n No of lines = %d", l+1);
printf("\n No of words = %d \n", w+1);

close(fd);
return 0;
}
```

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

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
        int running = 1;
        void *shared_memory = (void *)0;
        struct shared_m *shared_stuff;
        int shmid;
        srand((unsigned int)getpid());
        shmid = shmget((key_t)1234, sizeof(struct shared_m), 0666 | IPC_CREAT);
        if (shmid == -1)
        {
                fprintf(stderr,"shmget failed\n");
                exit(EXIT_FAILURE);
        }
        shared_memory = shmat(shmid, (void *)0, 0);
        if (shared_memory == (void *)-1)
        {
                fprintf(stderr, "shmat failed\n");
                exit(EXIT_FAILURE);
        }
        printf("Memory attached at %X\n", (int)shared_memory);
        shared_stuff = (struct shared_m *)shared_memory;
        shared_stuff->flag = 0;
        while(running)
        {
                if (shared_stuff->flag)
                {
                        printf("You wrote: %s", shared_stuff->text);
                        sleep( rand() % 4 ); /* make the other process wait for us
! */
                        shared_stuff->flag = 0;
                        if (strncmp(shared_stuff->text, "end", 3) == 0)
                        {
                                running = 0;
                        }
                }
        }


        if (shmdt(shared_memory) == -1)
        {
                fprintf(stderr, "shmdt failed\n");
                exit(EXIT_FAILURE);
        }

                exit(EXIT_SUCCESS);
}
```

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

int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;

void getData()
{
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
}
void sstf()
{
while(count!=n)
    {
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
           d=abs(RQ[i]-initial);
           if(min>d)
           {
               min=d;
               index=i;
           }

        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }

    printf("Total head movement is %d",TotalHeadMoment);
}

void scan()
{

printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for Scan disk scheduling
        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        //  last movement for max size
        TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
        initial = size-1;
        for(i=index-1;i>=0;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        initial =0;
        for(i=index;i<n;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
}

void clook()
{
printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for C-look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for( j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=n-1;i>=index;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
int main()
{
    int choice;
    while(1)
    {
        printf("\nDisk Scheduling Alogrithms\n1.Enter 
data\n2.SSTF\n3.SCAN\n4.C-LOOK\n\n5.Exit\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            getData();
            break;
        case 2:
            sstf();
            break;
        case 3:
            scan();
            break;
        case 4:
            clook();
            break;
        default:
            return 0;
            break;
        }
    }
}
```