**Assignment 1b**

opt=1 while [ "$opt" -lt 6 ] do

echo -e "Choose one of the Following\n1. Create a New Address Book\n2. View

Records\n3. Insert new Record\n4. Delete a Record\n5. Modify a Record\n6. Exit" # echo -e, enables special features of echo to use \n \t \b etc.

read opt case $opt in

1) echo "Enter filename" read fileName if [ -e $fileName ] ; then # -e to check if file exists, if exits remove the file rm $fileName

fi

cont=1 echo

"NAME\tNUMBER\t\tADDRESS\n===============================\n" | cat >>

$fileName while [ "$cont" -gt 0 ] do echo "\nEnter Name" read name echo "Enter Phone Number of $name" read number echo "Enter Address of $name" read address echo "$name\t$number\t\t$address" | cat >> $fileName echo "Enter 0 to Stop, 1 to Enter next" read cont

done

;;

1. cat $fileName

;;

1. echo "\nEnter Name" read name echo "Enter Phone Number of $name" read number echo "Enter Address of $name"

read address echo "$name\t$number\t\t$address" | cat >> $fileName

;;

1. echo "Delete record\nEnter Name/Phone Number" read data temp="temp" grep -v $data $fileName | cat >> $temp rm $fileName cat $temp | cat >> $fileName rm $temp

;;

1. echo "Modify record\nEnter Name/Phone Number" read data temp="temp" grep -v $data $fileName | cat >> $temp rm $fileName cat $temp | cat >> $fileName rm $temp echo "Enter Name" read name echo "Enter Phone Number of $name" read number echo "Enter Address of $name" read address echo -e "$name\t$number\t$address" | cat >> $fileName

;;

esac

done

**Assignment 2a**

#include<unistd.h>

#include<sys/types.h> #include<stdio.h> void asc(int\*, int sz); void dsc(int\*, int sz); int main()

{

pid\_t pid; int \*a,sz,i; printf("enter the size of the array\n"); scanf("%d",&sz); a=(int\*)malloc(sz\*sizeof(int));

printf("enter element of %d size array\n",sz);

for(i=0;i<sz;i++)

{ printf("a[%d]:",i); scanf("%d",&a[i]);

} pid=fork(); switch(pid)

{

case -1:

printf("Error in fork\n"); exit(-1); case 0:

printf("child process\n"); asc(a,sz); exit(0); default: wait (NULL); printf("Parent process\n"); dsc(a,sz); exit(0);

} return 0;

}

void asc(int \*a, int sz)

{ int i,j,temp; for(i=0;i<sz;i++) {

for(j=i+1;j<sz;j++)

{ if(a[i]>a[j])

{

temp=a[i]; a[i]=a[j]; a[j]=temp;

}

}

}

printf("sorted array elements in ascending order are\n"); for(i=0;i<sz;i++) { printf("%d\t",a[i]);

} printf("\n");

}

void dsc(int \*a, int sz)

{ int i,j,temp; for(i=0;i<sz;i++) {

for(j=i+1;j<sz;j++)

{ if(a[i]<a[j])

{

temp=a[i]; a[i]=a[j]; a[j]=temp;

}

}

}

printf("sorted array elements in descending order are\n"); for(i=0;i<sz;i++) {

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

} printf("\n");

}

**Assignment 2b**

Save as Assignment2.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> #include <sys/wait.h>

int main(int argc, char \*argv[]) {

// Check if an array of integers is provided as a command-line argument if (argc < 2) { fprintf(stderr, "Usage: %s <integer\_array>\n", argv[0]); return 1;

}

int n = argc - 1; // Number of elements in the array int \*arr = (int \*)malloc(n \* sizeof(int));

// Parse the command-line arguments into an integer array for (int i = 0; i < n; i++) { arr[i] = atoi(argv[i + 1]);

}

// Fork a child process pid\_t pid = fork();

if (pid == -1) { perror("Fork failed"); return 1;

}

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

// Execute a new program to display the array in reverse order char \*args[2]; args[0] = "./reverse\_array"; // Assuming the reverse\_array program is in the same directory args[1] = NULL;

if (execve(args[0], args, NULL) == -1) { perror("Execve failed"); return 1;

}

} else { // Parent process

// Wait for the child process to finish wait(NULL);

// Sort the array in ascending order (you can use your own sorting algorithm) for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) {

int temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp;

}

}

}

// Print the sorted array printf("Sorted array: "); for (int i = 0; i < n; i++) { printf("%d ", arr[i]);

} printf("\n");

} free(arr);

return 0;

}

Save as rev.c

#include <stdio.h>

int main(int argc, char \*argv[]) {

if (argc < 2) { fprintf(stderr, "Usage: %s <integer\_array>\n", argv[0]); return 1;

}

printf("Reversed array: "); for (int i = argc - 1; i > 0; i--) { printf("%s ", argv[i]);

} printf("\n");

return 0;

}

**Assignment 4a**

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<semaphore.h> int maxsize=10;

pthread\_t prod,cons; int in=0,out=0; int buffer[10];

sem\_t sem\_s,sem\_n; pthread\_mutex\_t work\_mutex;

void\* producer(); void\* consumer(); int append(); int take(); void\* th\_res;

void main()

{

void\* th\_res;

int res;

res=pthread\_mutex\_init(&work\_mutex,0); res=sem\_init(&sem\_s,0,maxsize); res=sem\_init(&sem\_n,0,0); res=pthread\_create(&prod,NULL,producer,NULL); res=pthread\_create(&cons,NULL,consumer,NULL); res=pthread\_join(prod,&th\_res); res=pthread\_join(cons,&th\_res);

}

void\* producer()

{ int item; while(1)

{

sleep(1); sem\_wait(&sem\_s); pthread\_mutex\_lock(&work\_mutex); printf("Enter item to produce \n"); scanf("%d",&item); append(item); pthread\_mutex\_unlock(&work\_mutex); sem\_post(&sem\_n); if(item==999) break;

}

}

int append(int item)

{ buffer[in]=item; in=(in+1)%maxsize;

}

void\* consumer()

{

int item;

while(1)

{

sleep(1); sem\_wait(&sem\_n); pthread\_mutex\_lock(&work\_mutex); item= take(); printf("item to be consumed %d\n",item); pthread\_mutex\_unlock(&work\_mutex); sem\_post(&sem\_s); if(item==999) break;

}

}

int take()

{

int item=buffer[out]; out=(out+1)%maxsize;

return item;

}

**Assignment 4b**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h> #include<unistd.h> sem\_t mutex,writeblock; long int data=0,rcount=0; void \*reader(void \*arg)

{int f; f=((long int)arg); sem\_wait(&mutex); rcount=rcount+1; if(rcount==1) sem\_wait(&writeblock); sem\_post(&mutex); printf("data read by the reader %d id %ld\n",f,data); sleep(1); sem\_wait(&mutex); rcount=rcount-1; if(rcount==0) sem\_post(&writeblock); sem\_post(&mutex);

}

void \*writer(void \*arg)

{int f; f=((long int)arg); sem\_wait(&writeblock); data++; printf("data written by the writer %d id %ld\n",f,data); sleep(1); sem\_post(&writeblock);

}

int main()

{

long int i;

int b,pr; pthread\_t rtid[10],wtid[10]; sem\_init(&mutex,0,1); sem\_init(&writeblock,0,1); printf("enter no of process you want to create=="); scanf("%d",&pr); for(i=0;i<=pr;i++)

{

pthread\_create(&wtid[i],NULL,writer,(void\*)(long)i); pthread\_create(&rtid[i],NULL,reader,(void\*)(long)i);

}

for(i=0;i<=pr;i++)

{

pthread\_join(wtid[i],NULL); pthread\_join(rtid[i],NULL);

}}

**Assignment 3**

**//SJF**

#include <stdio.h> int main()

{

// Matrix for storing Process Id, Burst // Time, Average Waiting Time & Average // Turn Around Time. int A[100][4]; int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat; printf("Enter number of process: "); scanf("%d", &n); printf("Enter Burst Time:\n");

// User Input Burst Time and alloting Process Id.

for (i = 0; i < n; i++) {

printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1;

}

// Sorting process according to their Burst Time. for (i = 0; i < n; i++) {

index = i; for (j = i + 1; j < n; j++)

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1]; A[i][1] = A[index][1]; A[index][1] = temp;

temp = A[i][0]; A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

// Calculation of Waiting Times for (i = 1; i < n; i++) { A[i][2] = 0; for (j = 0; j < i; j++)

A[i][2] += A[j][1]; total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n");

// Calculation of Turn Around Time and printing the // data.

for (i = 0; i < n; i++) {

A[i][3] = A[i][1] + A[i][2]; total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],

A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**Assignment 3**

**//RR**

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

void main()

{

// initlialize the variable name int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10]; float avg\_wt, avg\_tat; printf(" Total number of process in the system: "); scanf("%d", &NOP); y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time for(i=0; i<NOP; i++)

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1); printf(" Arrival time is: \t"); // Accept arrival time scanf("%d", &at[i]);

printf(" \nBurst time is: \t"); // Accept the Burst time scanf("%d", &bt[i]); temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat printf("Enter the Time Quantum for the process: \t"); scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time "); for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define the conditions

{

sum = sum + temp[i]; temp[i] = 0; count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--; //decrement the process no.

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]); wt = wt+sum-at[i]-bt[i]; tat = tat+sum-at[i]; count =0;

}

if(i==NOP-1)

{ i=0;

}

else if(at[i+1]<=sum)

{ i++;

}

else

{ i=0;

}

}

// represents the average waiting time and Turn Around time

avg\_wt = wt \* 1.0/NOP; avg\_tat = tat \* 1.0/NOP; printf("\n Average Turn Around Time: \t%f", avg\_wt); printf("\n Average Waiting Time: \t%f", avg\_tat);

}

**Assignment 5**

// Banker's Algorithm #include <stdio.h>

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!!");

}

**Assignment 7a**

#include <stdio.h>

#include <stdlib.h> #include <errno.h>

#include <string.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#define FIFO\_NAME "mfifo1" #define FIFO\_NAME1 "mfifo2"

int main(void)

{

char s[300],s1[122]; int num, fd,i=0,character=0,word=0,line=0,fd1; mkfifo(FIFO\_NAME, 0666);

printf("THIS IS SECOND PROCESS\n"); fd = open(FIFO\_NAME, O\_RDONLY);

if ((num = read(fd, s, 300)) == -1) { printf("read error"); exit(0);

}

else {

s[num] = '\0'; printf("read sentences from First Process=====>\n \"%s\"\n", s);

}

FILE \*fp; fp = fopen("output.txt","w"); /\* open for writing \*/

while(s[i]!='\0')

{

if((s[i]>=65 && s[i]<=90) || ( s[i]>=97 && s[i]<=122))

{

character++;

} i++;

} i=0; while(s[i]!='\0')

{ if(s[i]!=' '&& s[i]!='\n')

{ i++;

}

else

{

word++; i++;

}

}

i=0; while(s[i]!='\0')

{ if(s[i]=='\n')

{

line++;

} i++;

}

// printf(fp,"Number of characters=%d \n Number of Words=%d \n Number of

Lines=%d\n",character,word,line); fprintf(fp,"Number of characters=%d \n Number of Words=%d \n Number of

Lines=%d",character,word,line); fclose(fp); /\* close the file before ending program \*/

mkfifo(FIFO\_NAME1, 0666);

//mknod(FIFO\_NAME1, S\_IFIFO | 0666, 0);

fd1 = open(FIFO\_NAME1, O\_WRONLY);

FILE \*f; i=0; char ch;

f=fopen("output.txt","r");

if(f==0)

{

printf("\n file not found"); exit(0);

}

while((ch=fgetc(f))!=EOF)

{

s1[i++]=ch;

}

fclose(f);

printf("\n Number of Chars, Words and Line are Counted, \n Output is stored in text file,

\n Output is sent to First Process using FIFO");

write(fd1,s1,strlen(s1));

return 0;

}

**Assignment 6**

#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\n6.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;

}

}

}

**Assignment 7b**

//client.c

/\*

\* shm-client - client program to demonstrate shared memory.

\*/

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

int shmid; key\_t key; char \*shm, \*s;

/\*

* We need to get the segment named \* "5678", created by the server.

\*/

key = 5678;

/\*

* Locate the segment.

\*/

if ((shmid = shmget(key, SHMSZ, 0666)) < 0) { perror("shmget"); exit(1);

}

/\*

* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat"); exit(1);

}

/\*

* Now read what the server put in the memory.

\*/

for (s = shm; \*s != NULL; s++) putchar(\*s);

putchar('\n');

/\*

* Finally, change the first character of the \* segment to '\*', indicating we have read \* the segment.

\*/

\*shm = '\*';

exit(0);

}

**Assignment 7b**

Sub

//Server.c

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

char c; int shmid; key\_t key; char \*shm, \*s;

/\*

* We'll name our shared memory segment \* "5678".

\*/

key = 5678;

/\*

* Create the segment.

\*/

if ((shmid = shmget(key, SHMSZ, IPC\_CREAT | 0666)) < 0) { perror("shmget"); exit(1);

}

/\*

* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) { perror("shmat"); exit(1);

}

/\*

* Now put some things into the memory for the \* other process to read.

\*/

s = shm;

for (c = 'a'; c <= 'z'; c++) \*s++ = c;

\*s = NULL;

/\*

* Finally, we wait until the other process
* changes the first character of our memory \* to '\*', indicating that it has read what \* we put there.

\*/

while (\*shm != '\*')

sleep(1);

exit(0);

}

**Assignment 8**

**//c look**

#include<stdio.h> #include<stdlib.h> int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move; 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); 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;

}

Output:-

**Assignment 8**

**//SCAN**

#include<stdio.h> #include<stdlib.h> int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move; 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); 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); return 0;

}

**Assignment 8**

**//SSTF**

#include<stdio.h> #include<stdlib.h> int main(){

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

// logic for sstf disk scheduling

/\* loop will execute until all process is completed\*/ 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); return 0;

}