**AddressBook – Shell Programming**

#!/bin/sh

create()

{

touch addresbook.txt

}

insert()

{

echo " enter a name "

read name

echo " enter a address "

read address

echo " enter phone no "

read phoneno

echo " enter email id "

read email

echo " $name $address $phoneno $email " >> addresbook.txt

}

veiw()

{

echo " show the contents of the address book "

echo " name address phoneno email "

cat addresbook.txt

}

modify()

{

echo " enter ur name which u want to modify "

read name

grep -v " $name " addresbook.txt > addresbook1.txt

cp addresbook1.txt addresbook.txt

rm addresbook1.txt

echo " enter a name "

read name

echo " enter a address "

read address

echo " enter phone no "

read phoneno

echo " enter email id "

read email

echo " $name $address $phoneno $email ">> addresbook.txt

}

delete()

{

echo " enter ur name which u want to delete "

read name

grep -v " $name " addresbook.txt > addresbook1.txt

cp addresbook1.txt addresbook.txt

rm addresbook1.txt

}

n=0

while [ $n -ne 6 ]

do

echo " enter ur choice "

echo " 1.create "

echo " 2.insert "

echo " 3.veiw "

echo " 4.modify "

echo " 5.delete "

echo " 6.exit"

read ch

case $ch in

"1") create;;

"2")insert;;

"3")veiw;;

"4")modify;;

"5")delete;;

"6")exit;;

esac

done

**Fork – Bubble Sorting**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

void bubble\_sort(int [],int);

void bubble\_sort\_2(int [],int);

void main()

{

int a,b;

int array[100], n, c, d, swap;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d numbers\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

pid\_t pid;

pid=fork();

if(pid==0)

{

printf("Hello,I am the Child process\n");

bubble\_sort(array, n);

printf("Sorted list in ascending order:\n");

for ( c = 0 ; c < n ; c++ )

printf("%d\n", array[c]);

}

else

{

printf("Hello,I am the Parent process\n");

bubble\_sort\_2(array, n);

printf("Sorted list in decending order:\n");

for ( c = 0 ; c < n ; c++ )

printf("%d\n", array[c]);

}

}

void bubble\_sort(int list[], int n)

{

int c, d, t;

for (c = 0 ; c < ( n - 1 ); c++)

{

for (d = 0 ; d < n - c - 1; d++)

{

if (list[d] > list[d+1])

{

/\* Swapping \*/

t = list[d];

list[d] = list[d+1];

list[d+1] = t;

}

}

}

}

void bubble\_sort\_2(int list[], int n)

{

int c, d, t;

for (c = 0 ; c < ( n - 1 ); c++)

{

for (d = 0 ; d < n - c - 1; d++)

{

if (list[d] < list[d+1])

{

/\* Swapping \*/

t = list[d];

list[d] = list[d+1];

list[d+1] = t;

}

}

}

}

**Fork – Sorting**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

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

{

int nEle=argc-2,elem,temp,pid,fd[2], nbytes;

char \*args[nEle+1],f1[nEle],tt[nEle];

int i,j,a[5];

j=2;

args[0]=argv[1]; //name of second program tobe exe

for(i=1;i<=nEle;i++) //copy all elements to from command line to args[]

{

args[i]=argv[j];

j++;

}

args[i]= "NULL";//args[] last element should be NULL

j=0;

for(i=1;i<=nEle;i++)//convert all elemetns of args[] into integer

{

elem=atoi(args[i]);

a[j]=elem;

j++;

}

for(i=0;i<nEle;i++)//simple sorting of elements

{

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

{

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

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("\n Sorted Array is \n");

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

{

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

}

j=1;

for(i=0;i<nEle;i++) // convert integer to string array

{

sprintf(f1,"%d",a[i]);

strcpy(args[j],f1);

printf("\n Converted to string args[%d]=%s\n",j,args[j]);

j++;

}

args[j]=(char\*)0;//last char of args[] should be NULL

pid=fork();//creating child process

if(pid==0)

{

wait(0);

}

else

{

printf("\n Now parent is passing the sorted array to %s executable program\n",args[0]);

execv(argv[1],args);

}

return 0;

}

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

int binary(int a[],int n,int m,int l,int u)

{

int mid,c=0;

if(l<=u)

{

mid=(l+u)/2;

if(m==a[mid])

{

c=1;

}

else

if(m<a[mid])

{

return binary(a,n,m,l,mid-1);

}

else

return binary(a,n,m,mid+1,u);

}

else

return c;

}

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

{

int nEle=argc-1,elem,temp,pid,l,u,c,x;

char line[100];

char \*args[nEle+1];

int i,j=2,a[nEle];

j=0;

for(i=1;i<=nEle;i++)

{

elem=atoi(argv[i]);

a[j]=elem;

printf("a[%d]=%d\n",j,a[j]);

j++;

}

printf("Please Enter the Element to be search using binary search\n");

scanf("%d",&x);

l=0,u=nEle-1;

c=binary(a,nEle,x,l,u);

if(c==0)

printf("Number is not found.\n");

else

printf("Number is found.\n");

return 0;

}

//Commands for execution

//gcc f1.c -o f1

//gcc s1.c -o s1

//./f1 s1 6 7 8 3

**Round Robin – Scheduling Algorithm**

#include<stdio.h>

void main()

{

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;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");

scanf("%d", &at[i]);

printf(" \nBurst time is: \t");

scanf("%d", &bt[i]);

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

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)

{

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--;

printf("\nP%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;

}

}

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_tat);

printf("\n Average Waiting Time: \t%f", avg\_wt);

}

**Shortest Job First – Scheduling Algorithm**

#include <stdio.h>

int main()

{

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");

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

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

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;

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");

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]);

}

**Producer Consumer problem using counting semaphores and mutex**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h> //for system calls

#include <sys/types.h>

#include <sys/syscall.h> //to print thread id

#include <stdlib.h>

#define SIZE 3

void \*producer(void \*argp); //thread function for producer

void \*consumer(void \*argc); //thread function for consumer

struct Shared //structure

{

int buff[SIZE];

sem\_t full, empty;

};

int front = -1, rear = -1;

pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER; //initialize mutex variable

struct Shared Sh;

int main()

{

int prod, cons, i, j, k, l;

pthread\_t ptid, ctid; //producer and consumer thread ids

sem\_init(&Sh.empty, 0, 1); //initialize semaphore variable empty with value 1..used in thread of single process

sem\_init(&Sh.full, 0, 0); //initialize semaphore variable full with value 0..used in thread of single process

printf("\nEnter the no. of producers :\n");

scanf("%d", &prod);

printf("\nEnter the no. of consumers :\n");

scanf("%d", &cons);

for (i = 0; i < prod; i++) //calling producer thread

{

pthread\_create(&ptid, NULL, producer, NULL);

}

for (j = 0; j < cons; j++) //calling consumer thread

{

pthread\_create(&ctid, NULL, consumer, NULL);

}

for (k = 0; k < prod; k++) //for joining producer thread

{

pthread\_join(ptid, NULL);

}

for (l = 0; l < cons; l++) //for joining consumer thread

{

pthread\_join(ctid, NULL);

}

return 0;

}

void \*producer(void \*argp) //producer function

{

int i, item;

while (1)

{

if (rear >= SIZE - 1) //if buffer is full

{

sleep(1);

printf("\nBuffer full\n");

exit(0);

}

else

{

if (front == -1) //for first element of bufffer

{

sem\_wait(&Sh.empty); //critical section begins here

pthread\_mutex\_lock(&mut);

sleep(3);

printf("\n\n");

printf("\nEnter the product to be produced:\n");

scanf("%d", &item);

Sh.buff[++rear] = item;

printf("\nProducer id of producer:");

printf("%ld\t", syscall(SYS\_gettid));

printf("\nProduced item by producer: %d\n", item);

front = rear;

pthread\_mutex\_unlock(&mut);

sem\_post(&Sh.full); //critical section ends here

}

else //for other elements

{

sem\_wait(&Sh.empty); //critical section begins here

pthread\_mutex\_lock(&mut);

sleep(3);

printf("\n\n");

printf("\nEnter the product to be produced:\n");

scanf("%d", &item);

Sh.buff[++rear] = item;

printf("\nProducer id of producer:");

printf("%ld\t", syscall(SYS\_gettid));

printf("\nProduced item by producer: %d\n", item);

pthread\_mutex\_unlock(&mut);

sem\_post(&Sh.full); //critical section ends here

}

}

}

return NULL;

pthread\_exit(0);

}

void \*consumer(void \*argc) //consumer function

{

int i, item;

while (1)

{

if (front == rear == -1) //if buffer is empty

{

printf("\nBuffer Empty..");

break;

}

else

{

sem\_wait(&Sh.full); //critical section begins here

pthread\_mutex\_lock(&mut);

item = Sh.buff[front++];

printf("\nConsumer id of consumer:");

printf("%ld\t", syscall(SYS\_gettid));

printf("\nConsumed item by consumer: %d\n", item);

sem\_post(&Sh.empty);

pthread\_mutex\_unlock(&mut); //critical section ends here

}

}

return NULL;

pthread\_exit(0);

}

**Reader Writer Problem**

#include <stdio.h>

#include <pthread.h>

#include <sys/syscall.h>

#include <unistd.h>

void \*reader(void \*);

void \*writer(void \*);

int getItemforBuff();

void readItemfromBuff(int buffer);

int buffer;

pthread\_mutex\_t mutex1 = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t wrt = PTHREAD\_MUTEX\_INITIALIZER;

int flag = 0;

int read\_count = 0;

int main()

{

pthread\_t rd1\_tid;

pthread\_t wr\_tid;

pthread\_t rd2\_tid;

pthread\_create(&wr\_tid, NULL, writer, NULL);

pthread\_create(&rd1\_tid, NULL, reader, NULL);

pthread\_create(&rd2\_tid, NULL, reader, NULL);

pthread\_join(wr\_tid, NULL);

pthread\_join(rd1\_tid, NULL);

pthread\_join(rd2\_tid, NULL);

return 0;

}

void \*reader(void \*argp)

{

while (1)

{

pthread\_mutex\_lock(&mutex1);

read\_count++;

if (read\_count == 1)

{

pthread\_mutex\_lock(&wrt);

}

pthread\_mutex\_unlock(&mutex1);

if (flag == 1)

{

readItemfromBuff(buffer);

sleep(1);

flag = 0;

}

pthread\_mutex\_lock(&mutex1);

read\_count--;

if (read\_count == 0)

{

pthread\_mutex\_unlock(&wrt);

}

pthread\_mutex\_unlock(&mutex1);

}

}

void \*writer(void \*argp)

{

while (1)

{

pthread\_mutex\_lock(&mutex1);

if (flag == 0)

{

buffer = getItemforBuff();

flag = 1;

}

pthread\_mutex\_unlock(&mutex1);

}

}

int getItemforBuff()

{

int item;

printf("writer:enter an item into buffer\n");

scanf("%d", &item);

return item;

}

void readItemfromBuff(int buffer)

{

printf("thread=%ld\n", syscall(SYS\_gettid));

printf("reader:read item from buffer=%d\n", buffer);

}

**FCFS – Page Replacement Algorithm**

#include<stdio.h>

int main()

{

int n,i,bt[10],wt[10],tat[10],awt,atat;

printf("How many Elements you want to enter:");

scanf("%d",&n);

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

{

printf("Enter Burst time for P%d:",i);

scanf("%d",&bt[i]);

}

printf("\nProcess Details are:\n");

printf("Process\t Burst Time\n");

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

{

printf(" P%d\t %d\n",i,bt[i]);

}

printf("\n");

//Waiting Time

wt[0]=0;

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

{

wt[i+1]=wt[i]+bt[i];

printf("Waiting Time of P%d is %d\n",i,wt[i]);

}

printf("\n");

//Turn Around Time

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

{

tat[i]=bt[i]+wt[i];

printf("Turn Around Time of P%d is %d\n",i,tat[i]);

}

printf("\nProcess\t Burst Time\t Waiting Time\t Turn-Around Time\n");

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

printf(" P%d\t %d\t\t %d\t \t\t%d\n",i,bt[i],wt[i],tat[i]); }

awt=0;

atat=0;

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

{

awt=awt+wt[i];

atat+=tat[i];

}

printf("\nAverage Waiting Time: %d",(awt/n));

printf("\nAverage Turn Around Time: %d\n",(atat/n));

}

**LRU, FIFO, OPT.**

#include <stdio.h>

void FIFO(char[], char[], int, int);

void lru(char[], char[], int, int);

void opt(char[], char[], int, int);

int main()

{

int ch, YN = 1, i, l, f;

char F[10], s[25];

system("clear");

printf("\n\n\tEnter the no of empty frames: ");

scanf("%d", &f);

printf("\n\n\tEnter the length of the string: ");

scanf("%d", &l);

printf("\n\n\tEnter the string: ");

scanf("%s", s);

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

F[i] = -1;

do

{

system("clear");

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*");

printf("\n\n\t1:FIFO\n\n\t2:LRU\n\n\t3:OPT\n\n\t4:EXIT");

printf("\n\n\tEnter your choice: ");

scanf("%d", &ch);

system("clear");

switch (ch)

{

case 1:

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

{

F[i] = -1;

}

FIFO(s, F, l, f);

break;

case 2:

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

{

F[i] = -1;

}

lru(s, F, l, f);

break;

case 3:

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

{

F[i] = -1;

}

opt(s, F, l, f);

break;

case 4:

exit(0);

}

printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\tIF NO PRESS 0 : ");

scanf("%d", &YN);

} while (YN == 1);

return (0);

}

//FIFO

void FIFO(char s[], char F[], int l, int f)

{

int i, j = 0, k, flag = 0, cnt = 0;

printf("\n\tPAGE\t FRAMES\t FAULTS");

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

{

for (k = 0; k < f; k++)

{

if (F[k] == s[i])

flag = 1;

}

if (flag == 0)

{

printf("\n\t%c\t", s[i]);

F[j] = s[i];

j++;

for (k = 0; k < f; k++)

{

printf(" %c", F[k]);

}

printf("\tPage-fault%d", cnt);

cnt++;

}

else

{

flag = 0;

printf("\n\t%c\t", s[i]);

for (k = 0; k < f; k++)

{

printf(" %c", F[k]);

}

printf("\tNo page-fault");

}

if (j == f)

j = 0;

}

}

//LRU

void lru(char s[], char F[], int l, int f)

{

int i, j = 0, k, m, flag = 0, cnt = 0, top = 0;

printf("\n\tPAGE\t FRAMES\t FAULTS");

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

{

for (k = 0; k < f; k++)

{

if (F[k] == s[i])

{

flag = 1;

break;

}

}

printf("\n\t%c\t", s[i]);

if (j != f && flag != 1)

{

F[top] = s[i];

j++;

if (j != f)

top++;

}

else

{

if (flag != 1)

{

for (k = 0; k < top; k++)

{

F[k] = F[k + 1];

}

F[top] = s[i];

}

if (flag == 1)

{

for (m = k; m < top; m++)

{

F[m] = F[m + 1];

}

F[top] = s[i];

}

}

for (k = 0; k < f; k++)

{

printf(" %c", F[k]);

}

if (flag == 0)

{

printf("\tPage-fault%d", cnt);

cnt++;

}

else

printf("\tNo page fault");

flag = 0;

}

}

//optimal

void opt(char s[], char F[], int l, int f)

{

int i, j = 0, k, m, flag = 0, cnt = 0, temp[10];

printf("\n\tPAGE\t FRAMES\t FAULTS");

for (i = 0; i < 10; i++)

temp[i] = 0;

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

F[i] = -1;

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

{

for (k = 0; k < f; k++)

{

if (F[k] == s[i])

flag = 1;

}

if (j != f && flag == 0)

{

F[j] = s[i];

j++;

}

else if (flag == 0)

{

for (m = 0; m < f; m++)

{

for (k = i + 1; k < l; k++)

{

if (F[m] != s[k])

{

temp[m] = temp[m] + 1;

}

else

break;

}

}

m = 0;

for (k = 0; k < f; k++)

{

if (temp[k] > temp[m])

{

m = k;

}

}

F[m] = s[i];

}

printf("\n\t%c\t", s[i]);

for (k = 0; k < f; k++)

{

printf(" %c", F[k]);

}

if (flag == 0)

{

printf("\tPage-fault %d", cnt);

cnt++;

}

else

printf("\tNo Page-fault");

flag = 0;

for (k = 0; k < 10; k++)

temp[k] = 0;

}

}

**Bankers**

#include <stdio.h>

void main()

{

int alloc[10][10], max[10][10], avail[10], tot[10], need[10][10], pflag[10] = {0}, safe[10], flag1, flag2, p, r, i, j, k = 0, m;

printf("Enter the no of processes: ");

scanf("%d", &p);

printf("\nEnter the no of resources: ");

scanf("%d", &r);

printf("\nEnter the total instances of resources: ");

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

{

scanf("%d", &tot[i]);

avail[i] = tot[i];

}

printf("\nEnter the allocated instances for each process ");

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

{

printf("\nProcess%d: ", i);

for (j = 0; j < r; j++)

scanf("%d", &alloc[i][j]);

}

printf("\nEnter the max instances required for each process");

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

{

printf("\nProcess %d: ", i);

for (j = 0; j < r; j++)

scanf("%d", &max[i][j]);

}

printf("\nThe available matrix is: ");

for (j = 0; j < r; j++)

{

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

avail[j] = avail[j] - alloc[i][j];

printf("\t%d", avail[j]);

}

printf("\n\nThe need matrix is: ");

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

{

printf("\nProcess %d:", i);

for (j = 0; j < r; j++)

{

need[i][j] = max[i][j] - alloc[i][j];

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

}

}

for (m = 0; m < p; m++)

{

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

{

if (pflag[i] == 0)

{

flag1 = 0;

printf("\n\nFor process %d:", i);

for (j = 0; j < r; j++)

{

if (need[i][j] > avail[j])

{

flag1 = 1;

break;

}

}

if (flag1 == 0)

{

for (j = 0; j < r; j++)

avail[j] = avail[j] + alloc[i][j];

pflag[i] = 1;

printf("\nProcess %d can be granted

resources..",i);

printf("\nNew Available resources are\n");

for (j = 0; j < r; j++)

printf("\t%d", avail[j]);

safe[k] = i;

k++;

}

if (flag1 == 1)

printf("\nProcess %d cannot be granted

resources....Going to next process", i);

} //outer if

} //outer for

} //outer for

flag2 = 0;

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

{

if (pflag[i] == 0)

{

printf("\n\nSystem is NOT in a safe state");

flag2 = 0;

break;

}

else

flag2 = 1;

}

if (flag2 == 1)

{

printf("\n\nSystem is in a SAFE STATE\nSAFE SEQUENCE is\n");

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

printf("Process%d ", safe[i]);

}

}

**FIFO – Inter process Communication**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#define MAX\_BUF 1024

int main()

{

int fd, c = 0;

char \*fifo1 = "fifo1";

char \*fifo2 = "fifo2";

int fd1;

int words = 1, lines = 1, chars = 0;

char buf1[MAX\_BUF];

mkfifo(fifo1, 0666);

fd = open(fifo1, O\_RDWR);

char str;

printf("\nEnter the String:");

while ((str = getchar()) != '#')

buf1[c++] = str;

buf1[c] = '\0';

write(fd, buf1, sizeof(buf1));

close(fd);

unlink(fifo1);

fd1 = open(fifo2, O\_RDWR);

read(fd1, buf1, sizeof(buf1));

printf("\nThe contents of file are %s", buf1);

int i = 0;

while (buf1[i] != '\0')

{

if (buf1[i] == ' ' || buf1[i] == '\n')

{

words++;

}

else

{

chars++;

}

if (buf1[i] == '\n')

{

lines++;

}

i++;

}

printf("\n No of Words: %d", words);

printf("\n No of Characters: %d", chars);

printf("\n No of Lines: %d\n", lines);

close(fd1);

return 0;

}

**Shared Memory - Inter process Communication**

#include <stdio.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <wait.h>

int main()

{

int x, y, ret, ret\_v;

long int add;

int shmid;

int \*shmptr;

key\_t key;

pid\_t pid;

printf("\nEnter a number:");

scanf("%d", &x);

key = ftok(".", 'M');

shmid = shmget(key, sizeof(x), IPC\_CREAT | 0666);

if (shmid < 0)

{

printf("\nShared memory creation error!");

\_exit(-1);

}

printf("\nShared Memory is Created.");

printf("\nShmid is:%d", shmid);

shmptr = (int \*)shmat(shmid, 0, 0);

add = (int)shmptr;

if (add != -1)

printf("\nShared Memory is attached at address:%u", shmptr);

else

{

printf("\nShared Memory not attached!");

\_exit(-1);

}

\*shmptr = x;

ret = shmdt((void \*)shmptr);

if (ret == 0)

printf("\nShared Memory detached successfully\n");

pid = fork();

if (pid == 0)

{

printf("\n-----------------------------------------------\nThis is Child Process\n-----------------------------------------------");

shmptr = (int \*)shmat(shmid, 0, 0);

add = (int)shmptr;

if (add != -1)

printf("\nShared Memory is attached at address:%u", shmptr);

else

{

printf("\nShared Memory not attached!");

\_exit(-1);

}

y = \*shmptr;

printf("\nThe data read is:%d", y);

ret = shmdt((void \*)shmptr);

if (ret == 0)

printf("\nShared Memory detached successfully\n");

ret\_v = shmctl(shmid, IPC\_RMID, 0);

if (ret\_v == 0)

printf("\nShared Memory removed successfully!\n\n");

printf("-----------------------------------------------\n");

}

else

{

wait(0);

}

return 0;

}

**SSTF Algorithm**

#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;

}

**SCAN Algorithm**

#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;

}

**C–Look Algorithm**

#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;

}