# OS LAB

# Assignments

# Created by

Kotha Dileep kumar

Roll no:108CS015

# Day-1

## Q1.Write a programme to take the input time in seconds and convert it into HH:MM:SS format ?

#!/bin/bash

echo "$(($1/3600))hr:$((($1%3600)/60))min:$((($1%3600)%60))sec"

## Q2.Write a Programme to take two numbers as input and display all the numbers in between them ?

#!/bin/bash

for ((i=$1;i<=$2;i++))

do

echo $i

done

## Q3.Write a Programme to display date,time,username,hostname and the number of users currently logged in ?

#!/bin/bash

clear

date +%D

date +%T

hostname

pwd

logname

echo "$((`who|wc -l`-1))"

## Q4.Write a Programme to create child process using fork system call and return it’s PID ?

#include<stdio.h>

#include<sys/types.h>

main()

{int pid;

printf("The process id of the parent process is %d\n",getpid());

pid=fork();

if(pid==0)

printf("The process id of the child process is %d\n",getpid());

else if(pid<0)

printf("Sorry, couldn't create the child process");

else

printf("The process id of parent process is %d\n",getpid());

}

# Day-2

## Q1.Write a C-programme that uses “pipe” system call to communicate using a string between a parent and a child process ?

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{int n,l;

char buf[100];

int fd[2];

printf("Starting the piping process\n");

pipe(fd);

l=fork();

if(l==0)

{close(fd[1]);

printf("Running the child process\n");

n=read(fd[0],buf,100);

printf("The value of n is:%d\n",n);

printf("The value received is\n%s\n",buf);

}

else if(l<0)

printf("Child cannot be created\n");

else

{close(fd[0]);

printf("Running the parent process\n");

write(fd[1],"I am a good boy\n",16);

printf("Wrote to the pipe\n");

}

getche();

return 0;

}

## Q2.Write a C-programme to create n-processes, the first process accepts an integer increments it by “1” and writes it to the pipe and the second process receives, increments it by “1” and so on. The last process reads, increments by “1” and finally displays it’s value ?

#include<stdio.h>

int main()

{int fd[2];

int n,i,l,a[1];

if(fork()==0)

{read(fd[0],a,1);

a[0]++;printf("%d\n",a[0]);

write(fd[1],a,1);

exit(0);

}

else

{

printf("Enter the number of processes you want:");

scanf("%d",&n);

printf("Enter the number to be incremented:");

scanf("%d",&a[0]);

read(fd[0],a,1);

write(fd[1],a,1);

exit(0);

}

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

{l=fork();

if(l==0)

{read(fd[0],a,1);

a[0]++;

write(fd[1],a,1);

exit(0);

}

}

if(i==n)

{read(fd[0],a,1);

printf("%d",a);

exit(0);

}

return 0;

}

## Q3.Write a C-programme to create a child process, the parent process will run, “ls –l” command and sends the input to child process, which runs “wc –l” command ?

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{int n,l;

FILE \*fp;

char command[400];//For storing the command

char buf[300];

int fd[2];

printf("Starting the piping process\n");

pipe(fd);

l=fork();

if(l==0)

{close(fd[1]);

printf("Running the child process\n");

n=read(fd[0],buf,300);

sprintf(command,"wc -l %s",buf);

system("command");

}

else if(l<0)

printf("Child cannot be created\n");

else

{close(fd[0]);

printf("Running the parent process\n");

fp=popen("ls -l","r");//For system calls reading

write(fd[1],fp,200);

printf("Wrote to the pipe\n");

}

getche();

return 0;

}

# Sheduling Algorithms

## Input text file (same for all the algorithms):

3

A 5 30

B 10 20

C 15 10

## 1.FCFS

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{FILE \*fp;

int i,n,TET,temp,j;

float AV\_WT,AV\_TA;

char pro\_name[20][10],tempch[10];

int e[20],w[20],ta[20],a[20],sw;

fp=fopen("Input.txt","r");

printf("Enter the number of processes:\n");

fscanf(fp,"%d",&n);

printf("Enter the name of process, it's arrival time and it's execution time, side by side :\n");

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

fscanf(fp,"%s%d%d",&pro\_name[i],&a[i],&e[i]);

fclose(fp);

//

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

{sw=0;

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

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

{temp=a[j];

a[j]=a[i];

a[i]=temp;

//

strcpy(tempch,pro\_name[j]);

strcpy(pro\_name[j],pro\_name[i]);

strcpy(pro\_name[i],tempch);

//

temp=e[j];

e[j]=e[i];

e[i]=temp;

sw=sw+1;}

}

if(sw==0)

break;

}

//

printf("The processes you entered after sorting are:\nName\tArr\_time\tExec\_time\n");

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

printf("%s\t%d\t%d\n",pro\_name[i],a[i],e[i]);

w[0]=0;

AV\_WT=0;

ta[0]=e[0];

AV\_TA=ta[0];

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

{

if(a[i]>(a[i-1]+e[i-1]+w[i-1]))

w[i]=0;

else

w[i]=a[i-1]+e[i-1]+w[i-1]-a[i];

AV\_WT=AV\_WT+w[i];

ta[i]=w[i]+e[i];

AV\_TA=AV\_TA+ta[i];

}

AV\_WT/=n;

AV\_TA/=n;

TET=w[i-1]+e[i-1]+a[i-1]-a[0];

printf("The processes you entered along with waiting times and turn around times are:\n");

printf("Process name Exectime Waitingtime Turnaroundtime\n");

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

printf("%s\t\t\t%d\t%d\t%d\n",pro\_name[i],e[i],w[i],ta[i]);

printf("The total execution time is %d\n\nThe avarage waiting time is %f\n\nThe avarage turn around time is %f\n",TET,AV\_WT,AV\_TA);

getche();

return 0;

}

## 2.Preemptive (SRTF)

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{FILE \*fp;

int i,n,fl,k,j,min,temp,MT=0,sw=0,flag,t;

float AV\_WT,AV\_TA;

char pro\_name[20][10],tempch[10];

int e[20],w[20],ta[20],a[20],l[20],ex[20],pr[20];

fp=fopen("Input.txt","r");

printf("Enter the number of processes:\n");

fscanf(fp,"%d",&n);

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

l[i]=0;

printf("Enter the name of process, it's arrival time and it's execution time, side by side as per the order they arrived:\n");

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

fscanf(fp,"%s%d%d",&pro\_name[i],&a[i],&e[i]);

fclose(fp);

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

{sw=0;

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

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

{temp=a[j];

a[j]=a[i];

a[i]=temp;

strcpy(tempch,pro\_name[j]);

strcpy(pro\_name[j],pro\_name[i]);

strcpy(pro\_name[i],tempch);

temp=e[j];

e[j]=e[i];

e[i]=temp;

sw=sw+1;}

}

if(sw==0)

break;

}

printf("The processes you entered after sorting are:\nName\tArr\_time\tExec\_time\n");

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

printf("%s\t%d\t%d\n",pro\_name[i],a[i],e[i]);

//

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

l[i]=0;

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

ex[i]=e[i];

AV\_WT=0;

AV\_TA=0;

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

if(a[i]<a[i+1])

break;

i=i+1;

fl=1;

while(fl)

{fl=0;

e[18]=10000;

min=18;

//To find the minimum element

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

if(l[j]!=1)

{if(e[j]<e[min])

min=j;}

flag=1;

while(i<=n)

{flag=0;

if(MT+e[min]>a[i])

{flag=1;i++;}

else

break;

if(flag==1)

{

if(a[min]<MT)

{

e[min]=e[min]-a[i-1]+MT;

MT=a[i-1];

}

else

{

e[min]=e[min]-a[i-1]+a[min];

pr[min]=a[i-1]-a[min];

MT=a[i-1];

}

printf("\nThe process executed just is %s",pro\_name[min]);

e[18]=10000;

min=18;

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

if(l[j]!=1)

{if(e[j]<e[min])

min=j;}

}}

if(flag==0)

{

if(a[min]<MT)

{MT=MT+e[min];

}

else

{MT=a[min]+e[min];

}

}

printf("\nThe process completed just is %s",pro\_name[min]);

l[min]=1;

w[min]=MT-a[min]-ex[min];

ta[min]=w[min]+ex[min];

AV\_WT=AV\_WT+w[min];

AV\_TA=AV\_TA+ta[min];

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

if(l[i]!=1)

if(a[i]>MT)

break;

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

if(l[j]==0)

{fl=1;break;}

}

AV\_WT/=n;

AV\_TA/=n;

printf("\nThe processes you entered along with waiting times and turn around times are:\n");

printf("Process name Exectime Waitingtime Turnaroundtime\n");

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

printf("%s\t\t\t%d\t%d\t%d\n",pro\_name[i],e[i],w[i],ta[i]);

printf("The Final time after execution is %d\n\nThe avarage waiting time is %f\n\nThe avarage turn around time is %f\n",MT,AV\_WT,AV\_TA);

getche();

return 0;}

## 3.Non-Preemptive ( SJNF)

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{FILE \*fp;

int i,n,fl,k,j,min,temp,MT=0,sw=0;

float AV\_WT,AV\_TA;

char pro\_name[20][10],tempch[10];

int e[20],w[20],ta[20],a[20],l[20];

fp=fopen("Input.txt","r");

printf("Enter the number of processes:\n");

fscanf(fp,"%d",&n);

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

l[i]=0;

printf("Enter the name of process, it's arrival time and it's execution time, side by side as per the order they arrived:\n");

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

fscanf(fp,"%s%d%d",&pro\_name[i],&a[i],&e[i]);

fclose(fp);

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

{sw=0;

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

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

{temp=a[j];

a[j]=a[i];

a[i]=temp;

strcpy(tempch,pro\_name[j]);

strcpy(pro\_name[j],pro\_name[i]);

strcpy(pro\_name[i],tempch);

temp=e[j];

e[j]=e[i];

e[i]=temp;

sw=sw+1;}

}

if(sw==0)

break;

}

printf("The processes you entered after sorting are:\nName\tArr\_time\tExec\_time\n");

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

printf("%s\t%d\t%d\n",pro\_name[i],a[i],e[i]);

AV\_WT=0;

AV\_TA=0;

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

if(a[i]<a[i+1])

break;

i=i+1;

fl=1;

while(fl)

{fl=0;

e[18]=10000;

min=18;

for(j=0;j<i;j++)//To find the minimum element

if(l[j]!=1)

{if(e[j]<e[min])

min=j;}

l[min]=1;

if(a[min]<MT)

MT=MT+e[min];

else

MT=a[min]+e[min];

printf("\nThe process executed just is %s",pro\_name[min]);

w[min]=MT-a[min]-e[min];

ta[min]=w[min]+e[min];

AV\_WT=AV\_WT+w[min];

AV\_TA=AV\_TA+ta[min];

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

if(l[i]!=1)

if(a[i]>MT)

break;

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

if(l[j]==0)

{fl=1;break;}

}

AV\_WT/=n;

AV\_TA/=n;

printf("\nThe processes you entered along with waiting times and turn around times are:\n");

printf("Process name Exectime Waitingtime Turnaroundtime\n");

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

printf("%s\t\t\t%d\t%d\t%d\n",pro\_name[i],e[i],w[i],ta[i]);

printf("The Final time after execution is %d\n\nThe avarage waiting time is %f\n\nThe avarage turn around time is %f\n",MT,AV\_WT,AV\_TA);

getche();

return 0;}

## 4.Round Robin Solution

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{FILE \*fp;

int i,n,fl,k,j,min,temp,MT=0,sw=0,flag;

float AV\_WT,AV\_TA;

char pro\_name[20][10],tempch[10];

int e[20],w[20],ta[20],a[20],l[20],ex[20],ts;

fp=fopen("Input.txt","r");

printf("Enter the number of processes:\n");

fscanf(fp,"%d",&n);

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

l[i]=0;

printf("Enter the name of process, it's arrival time and it's execution time, side by side as per the order they arrived:\n");

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

fscanf(fp,"%s%d%d",&pro\_name[i],&a[i],&e[i]);

fclose(fp);

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

{sw=0;

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

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

{temp=a[j];

a[j]=a[i];

a[i]=temp;

strcpy(tempch,pro\_name[j]);

strcpy(pro\_name[j],pro\_name[i]);

strcpy(pro\_name[i],tempch);

temp=e[j];

e[j]=e[i];

e[i]=temp;

sw=sw+1;}

}

if(sw==0)

break;

}

printf("The processes you entered after sorting are:\nName\tArr\_time\tExec\_time\n");

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

printf("%s\t%d\t%d\n",pro\_name[i],a[i],e[i]);

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

l[i]=0;

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

ex[i]=e[i];

AV\_WT=0;

AV\_TA=0;

ts=2;

fl=1;

MT=0;

while(fl)

{fl=0;

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

{if(l[i]!=1)

{if(MT>a[i])

{

if(e[i]>ts)

{

MT=MT+ts;

e[i]=e[i]-ts;

printf("\nThe process executed just is %s",pro\_name[i]);

}

else

{

MT=MT+e[i];

w[i]=MT-a[i]-ex[i];

ta[i]=w[i]+ex[i];

AV\_WT=AV\_WT+w[i];

AV\_TA=AV\_TA+ta[i];

l[i]=1;

printf("\nThe process executed just is %s",pro\_name[i]);

}}

else

{

if(e[i]>ts)

{

MT=a[i]+ts;

e[i]=e[i]-ts;

printf("\nThe process executed just is %s",pro\_name[i]);}

else

{

MT=a[i]+e[i];

w[i]=MT-a[i]-ex[i];

ta[i]=w[i]+ex[i];

AV\_WT=AV\_WT+w[i];

AV\_TA=AV\_TA+ta[i];

l[i]=1;

printf("\nThe process executed just is %s",pro\_name[i]);

}}}

}

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

if(l[j]==0)

fl=1;

}

//Round Robin end

AV\_WT/=n;

AV\_TA/=n;

printf("\nThe processes you entered along with waiting times and turn around times are:\n");

printf("Process name Exectime Waitingtime Turnaroundtime\n");

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

printf("%s\t\t\t%d\t%d\t%d\n",pro\_name[i],e[i],w[i],ta[i]);

printf("The Final time after execution is %d\n\nThe avarage waiting time is %f\n\nThe avarage turn around time is %f\n",MT,AV\_WT,AV\_TA);

getche();

return 0;}

## 5.Round Robin With Aging

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{FILE \*fp;

int i,n,fl,k,j,min,temp,MT=0,sw=0,flag,fl1;

float AV\_WT,AV\_TA;

char pro\_name[20][10],tempch[10];

int e[20],w[20],ta[20],a[20],l[20],ex[20],ts,tsa,pre,ag;

fp=fopen("Input.txt","r");

printf("Enter the number of processes:\n");

fscanf(fp,"%d",&n);

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

l[i]=0;

printf("Enter the name of process, it's arrival time and it's execution time, side by side as per the order they arrived:\n");

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

fscanf(fp,"%s%d%d",&pro\_name[i],&a[i],&e[i]);

fclose(fp);

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

{sw=0;

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

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

{temp=a[j];

a[j]=a[i];

a[i]=temp;

strcpy(tempch,pro\_name[j]);

strcpy(pro\_name[j],pro\_name[i]);

strcpy(pro\_name[i],tempch);

temp=e[j];

e[j]=e[i];

e[i]=temp;

sw=sw+1;}

}

if(sw==0)

break;

}

printf("The processes you entered after sorting are:\nName\tArr\_time\tExec\_time\n");

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

printf("%s\t%d\t%d\n",pro\_name[i],a[i],e[i]);

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

l[i]=0;

//To find maximum solution

ag=18;

e[ag]=1;

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

if(e[i]>e[ag])

ag=i;

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

ex[i]=e[i];

AV\_WT=0;

AV\_TA=0;

//Roundrobin start

ts=2;

fl=1;

MT=0;

tsa=ts;

pre=ts;

while(fl)

{fl=0;

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

{if(l[i]!=1)

{if(MT>a[i])

{fl1=0;

if(i==ag)

{if(e[i]>tsa)

{fl1=1;

MT=MT+tsa;

e[i]=e[i]-tsa;

printf("\nThe process executed just is %s",pro\_name[i]); }}

else

{

if(e[i]>ts)

{fl1=1;

MT=MT+ts;

e[i]=e[i]-ts;

printf("\nThe process executed just is %s",pro\_name[i]);}}

if(fl1!=1)

{

MT=MT+e[i];

w[i]=MT-a[i]-ex[i];

ta[i]=w[i]+ex[i];

AV\_WT=AV\_WT+w[i];

AV\_TA=AV\_TA+ta[i];

l[i]=1;

printf("\nThe process executed just is %s",pro\_name[i]);

}}

else

{fl1=0;

if(i==ag)

{if(e[i]>tsa)

{fl1=1;

MT=a[i]+tsa;

e[i]=e[i]-tsa;

printf("\nThe process executed just is %s",pro\_name[i]);

}}

else

{if(e[i]>ts)

{fl1=1;

MT=a[i]+ts;

e[i]=e[i]-ts;

printf("\nThe process executed just is %s",pro\_name[i]);}}

if(fl1!=1)

{

MT=a[i]+e[i];

w[i]=MT-a[i]-ex[i];

ta[i]=w[i]+ex[i];

AV\_WT=AV\_WT+w[i];

AV\_TA=AV\_TA+ta[i];

l[i]=1;

printf("\nThe process executed just is %s",pro\_name[i]);

}}}

}

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

if(l[j]==0)

fl=1;

tsa=2\*pre+tsa;

pre=2\*pre;

}

AV\_WT/=n;

AV\_TA/=n;

printf("\nThe processes you entered along with waiting times and turn around times are:\n");

printf("Process name Exectime Waitingtime Turnaroundtime\n");

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

printf("%s\t\t\t%d\t%d\t%d\n",pro\_name[i],e[i],w[i],ta[i]);

printf("The Final time after execution is %d\n\nThe avarage waiting time is %f\n\nThe avarage turn around time is %f\n",MT,AV\_WT,AV\_TA);

getche();

return 0;}

# Synchronisation Problems

## 1.Producer-Consumer Problem

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#define SIZE 5

typedef int buffer\_item;

sem\_t mutex;//semaphore mutex lock declaration

sem\_init(&mutex,0,1);//Getting default attributes to mutex

int empty=0;

int full=1;

int HEAD=-1;

int TAIL=-1;

buffer\_item Buffer[SIZE];

//Wait function

void wait(int flag)

{

if(flag==0)

{while((HEAD+1)%SIZE==TAIL)

sleep(2);}

else

{while(HEAD==-1 && TAIL==-1)

sleep(2);

}

}

//Signal function

void signal(int flag)

{

if(flag==0)

{

if(HEAD==TAIL)

{HEAD=-1;

TAIL=-1;

}

else

TAIL=(TAIL+1)%SIZE;

}

else

{

HEAD=(HEAD+1)%SIZE;

if(TAIL==-1)

TAIL=0;

}

}

//Inserting item function for Producer thread

int insert\_item(buffer\_item item)

{wait(empty);

sem\_wait(&mutex);

//Critical section start

Buffer[((HEAD+1)%SIZE)]=item;

//Critical section end

sem\_post(&mutex);

signal(full);

}

//Removing or consuming item function for consumer thread

int remove\_item()

{buffer\_item temp;

wait(full);

sem\_wait(&mutex);

//Critical section start

temp=Buffer[TAIL];

//Critical section end

sem\_post(&mutex);

signal(empty);

return temp;

}

//Thread for Producer process

void \*producer(void \*param)

{

buffer\_item item;

item=rand();

insert\_item(item);

printf("Item - %d successfully produced",item);

}

//Thread for consumer process

void \*consumer(void \*param)

{

buffer\_item temp;

temp=remove\_item();

printf("The item consumed successfully is %d",temp);

}

int main()

{int i,j;

pthread\_t tid1[5],tid2[5];//Thread id's

pthread\_attr\_t attr;//Thread attributes

pthread\_attr\_init(&attr);//Setting thread attributes to their default value

printf("Enter the number of producer and consumer processes you want:");

scanf("%d%d",&i,&j);

while(i)

{pthread\_create(&tid1[i],&attr,producer,i);

i--;

}

while(j)

{pthread\_create(&tid2[j],&attr,consumer,i);

j--;

}

sleep(60);

}

## 2.Reader-Writer Starvation Free Solution

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

int mutex[3],wrt,rd,flag;//semaphore mutex lock declaration

int readercount,writercount;

int read[6][2],write[6][2];

int p(int \*mutex)

{

while(\*mutex==0)

sleep(2);

\*mutex=0;

}

int v(int \*mutex)

{

\*mutex=1;

}

//Wait function

int readerwriterlockwait(char \*mode)

{

if(mode=="write")

{p(&flag);

p(&mutex[0]);//To make the below atomic and exclusive

writercount++;

if(writercount==1)//If first writer, then there is a need to check if any one is reading

p(&rd);

v(&mutex[0]);

p(&wrt);//Write lock, so that when one is modifying, the other cannot touch it

}

else

{ p(&flag);

p(&mutex[1]);//To make the below atomic and exclusive

p(&rd);//Read lock to show that some one has started reading

p(&mutex[2]);//To make the below atomic and exclusive and also to avoid signal of other read to modify readercount at the same time

readercount++;

if(readercount==1)//If he is a first reader, he must make sure that no one is modifying or writing

p(&wrt);

v(&mutex[2]);

v(&rd);//Lock is removed before the reading process has begun because two readers can read at the same time

v(&mutex[1]);

v(&flag);

}

}

//Signal function

int readerwriterlocksignal(char \*mode)

{

if(mode=="write")

{ v(&wrt);//Writing lock is removed

v(&flag);

p(&mutex[0]);//To make the below ones atomic and exclusive

writercount--;

if(writercount==0)

v(&rd);//Reading lock is removed, now any one can read since no writer is modifying

v(&mutex[0]);

}

else

{p(&mutex[2]);

readercount--;

if(readercount==0)

v(&wrt);

v(&mutex[2]);

}

}

//Reader thread

DWORD WINAPI Reader(LPVOID \*Param)

{DWORD i=\*(DWORD \*)Param;

char \*mode;

//Not arrived

sleep(read[i][0]\*1000);

//Arrived

mode="read";

readerwriterlockwait(mode);

//Reads the item

sleep(read[i][1]\*1000);

//Completed reading

readerwriterlocksignal(mode);

printf(" Reader-%d has completed reading\n",i+1);

}

//Writer thread

DWORD WINAPI Writer(LPVOID \*Param)

{char \*mode;

DWORD i=\*(DWORD \*)Param;

//Not arrived

sleep(write[i][0]\*1000);

//Arrived

mode="write";

readerwriterlockwait(mode);

//Modifies an item

sleep(write[i][1]\*1000);

//Completed modifying

readerwriterlocksignal(mode);

printf(" Writer-%d has completed modifying\n",i+1);

}

int main()

{readercount=0;

writercount=0;

flag=1;

mutex[0]=1;

mutex[1]=1;

mutex[2]=1;

wrt=1;

rd=1;

int l;

int Param[6];

DWORD ThreadId[6];

int i,j;

int readcount,writecount;

//Reader input start

printf("Enter the number of readers:");

scanf("%d",&readcount);

printf("Enter the arrival time and reading time of %d readers\n",readcount);

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

for(j=0;j<2;j++)

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

//Reader input end

//Writer input start

printf("Enter the number of writers:");

scanf("%d",&writecount);

printf("Enter the arrival time and writing time of %d writers\n",writecount);

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

for(j=0;j<2;j++)

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

//Writer input end

HANDLE ThreadHandleReader[6];

l=readcount-1;

while(l!=-1)

{Param[l]=l;

ThreadHandleReader[l]=CreateThread(NULL,0,Reader,&Param[l],0,&ThreadId[l]);

l--;

}

HANDLE ThreadHandleWriter[6];

l=writecount-1;

while(l!=-1)

{Param[l]=l;

ThreadHandleWriter[l]=CreateThread(NULL,0,Writer,&Param[l],0,&ThreadId[l]);

l--;

}

getche();

}

## 3.Dining Philosopher Problem

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<windows.h>

int Chopstick[5];//Each chopstick is a semaphore

//If chopstick=1, the chopstick is free, otherwise someone grabbed it

int l;

int TE[5][2];//For storing the eating and thinking time of philosophers

Wait(int i)

{

while(1)

{while(Chopstick[i]==0)//He goes on checking for left chopstick

sleep(2);

Chopstick[i]=0;//Immediately grabs the left chopstick

if(Chopstick[(i+1)%5]==0)//If right chopstick found he breaks

break;

Chopstick[i]=1;//Otherwise he will release the left chopstick and again goes back

sleep(2);

}

Chopstick[(i+1)%5]=0; //Right chopstick is also grabbed

}

Signal(int i)

{Chopstick[i]=1;

Chopstick[(i+1)%5]=1;

}

DWORD WINAPI Eat(LPVOID \*Param)

{DWORD i=\*Param;

sleep(TE[i][0]\*1000);//Thinking

Wait(i);

sleep(TE[i][1]\*1000);//Eating

Signal(i);

printf("Philosopher %d has completed eating\n",\*(DWORD\*)Param+1);

return 0;}

int main()

{int i,j;

l=4;

while(l!=-1)

{Chopstick[l]=1;

l--;}

printf("Enter the values of thinking and eating for five philosophers:\n");

for(i=0;i<5;i++)

for(j=0;j<2;j++)

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

HANDLE ThreadHandle[5];

DWORD ThreadId[5];

int Param[5];

l=4;

while(l!=-1)

{Param[l]=l;

ThreadHandle[l]=CreateThread(NULL,0,Eat,&Param[l],0,&ThreadId[l]);

l--;

}

getche();

return 0;}

# Banker’s algorithm

## Input text file

Total

10 5 7

Maximum Resources

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Allocated Resources

0 1 0

2 0 0

1 0 1

2 1 1

0 0 2

## Programme for Banker’s algorithm

#include<stdio.h>

int main()

{FILE \*fp;

int i,j;

char temp1[10];

fp=fopen("banker.txt","r");

int max[5][5],allo[5][5],need[5][5];//For the processes sake to give maximum, allocated and needed //resources

int avai[3];//Available resources

fscanf(fp,"%s",temp1);//Reading from file

printf("%s\n",temp1);

printf("\n");

for(i=0;i<3;i++)

{fscanf(fp,"%s",temp1);

avai[i]=atoi(temp1);

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

}

printf("\n");

printf("\n");

//For maximum resources start

fscanf(fp,"%s",temp1);//Reading from file

printf("%s\t",temp1);

fscanf(fp,"%s",temp1);//Reading from file

printf("%s\n",temp1);

printf("\n");

for(i=0;i<5;i++)

{for(j=0;j<3;j++)

{fscanf(fp,"%s",temp1);

max[i][j]=atoi(temp1);

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

}

printf("\n");}

//Maximum rexsources end

printf("\n\n");

//Allocated resouces start

fscanf(fp,"%s",temp1);//Reading from file

printf("%s\t",temp1);

fscanf(fp,"%s",temp1);//Reading from file

printf("%s\n",temp1);

printf("\n");

for(i=0;i<5;i++)

{for(j=0;j<3;j++)

{fscanf(fp,"%s",temp1);

allo[i][j]=atoi(temp1);

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

printf("\n");}

//Allocated resources end

//Needed resources start

printf("\n\n");

printf("Needed Resources\n\n");

for(i=0;i<5;i++)

{for(j=0;j<3;j++)

{

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

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

}

printf("\n");}

//Needed resources end

//Making the available resources apt

for(j=0;j<3;j++)

for(i=0;i<5;i++)

avai[j]=avai[j]-allo[i][j];

printf("\n");

printf("The available resources at present are:\n");

printf("\n");

for(j=0;j<3;j++)

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

printf("\n\n");

//End of making available resources apt

//Input start

int id,req[5][3];

printf("Enter the process id number:");

scanf("%d",&id);

printf("Enter the three resources");

scanf("%d%d%d",&req[id][0],&req[id][1],&req[id][2]);

//Input end

//Solution start

int Fin[5];

for(i=0;i<5;i++)

Fin[i]=0;

//Consider the resources are allocated

printf("\nAfter allocating the resources to p-%d the available resources are\n",id);

for(j=0;j<3;j++)

{avai[j]=avai[j]-req[id][j];

allo[id][j]=allo[id][j]+req[id][j];

need[id][j]=max[id][j]-allo[id][j];

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

}

printf("\n");

//End of consideration

//Checking for safe/Unsafe state

while(1)

{for(i=0;i<5;i++)

{if(Fin[i]==0&&need[i][0]<=avai[0]&&need[i][1]<=avai[1]&&need[i][2]<=avai[2])

break;}

if(i==5)

break;

for(j=0;j<3;j++)

avai[j]=avai[j]+allo[i][j];

Fin[i]=1;

printf("\nThe process finished is p-%d",i);}

//End of checking

//Checking for finish of all

for(i=0;i<5;i++)

if(Fin[i]==0)

break;

if(i==5)

printf("\nThe allocation is safe\n");

else

printf("\nThe allocation is unsafe\n");

//End of checking

//Solution end

getche();}

# Page Replacement Algorithms

## Input file (Input.txt – same for all algorithms)

20

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

## 1.FIFO

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21];//To take the input of reference string

int frame[SIZE];//This is for frames

int repptr=0;//The pointer for the page that needs to be replaced

int count=0;//For counting number of page faults

int display()//To display the elements of the frame

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Pagerep(int ele)

{

int temp;

temp=frame[repptr];

frame[repptr]=ele;

repptr++;//The pointer moves to the next frame since the current frame became the newest

if(repptr==SIZE)

repptr=0;

return temp; //Returns the victim page

}

int Pagefault(int ele)

{if(full!=SIZE)

frame[full++]=ele;//Untill all the frames fill, there is no call for page replacement

else

printf("The page replaced is %d",Pagerep(ele));//Displaying of victim page

}

int Search(int ele)//This would search and return the flag that tells whether the page is already //present in the frame or not

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;

break;

}}

return flag;

}

int main()

{int n,i;

FILE \*fp;//For taking input from a file

//Start

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

fclose(fp);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

//End

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

{

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();//For waiting

return 0;

}

## 2.Optimal Page Replacement Algorithm

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21],n;//To take the input

int frame[SIZE];

static int f=0;

int repptr;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Longestopt()

{int temp[SIZE]={0};/\*This is for checking the occurence of nearest possible future pages, considering no page is nearest in the beginning\*/

int c=0;//Counter to break the loop once we get two nearest future pages

int id,i,k,j=SIZE;

id=0;

for(i=f+1;i<n;i++)//Checking from the current time of use till the end of string for future references

{ for(k=0;k<j;k++) //Checking whether a page occurs in future or not

{ if(a[i]==frame[k])

{if(temp[k]!=1)

{temp[k]=1;

c++;

}

break;}}

if(c==2)

break;//Once we get two future pages then we may break

}

id=2;

while(id!=0)

{if(temp[id]==0)//Checking for the page which is not the nearest future reference

break;

id--;

}

repptr=id;

return repptr;//Returning the replacement pointer with the value of victim page

}

int Pagerep(int ele)

{

int temp;

repptr=Longestopt();

temp=frame[repptr];

frame[repptr]=ele;

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

frame[full++]=ele;

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;

break;

}}

return flag;

}

int main()

{int i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{f=i;

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 3.LRU using Stack

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21],n;//To take the input

int frame[SIZE];

int stk[SIZE];//Stack for storing the page numbers as per their reference criteria

int repptr;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

/\* A function for removing the referenced element from the stack and keeping it in the top, while moving all

the elements up or down corresepondingly\*/

int LRstackopt(int p)//The input to this function is the referenced page

{int temp;

int i;

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

if(stk[i]==p)

break;

temp=stk[i];//Storing the referenced value in temp

while(i!=SIZE-1)//Moving the other elements so that the TOP is empty

{stk[i]=stk[i+1];i++;

}

stk[i]=temp;//Storing the element in the TOP from temp

}

int Pagerep(int ele)

{

int temp;

repptr=stk[0];//Always the victim page is selected as the first element of stack

temp=frame[repptr];

frame[repptr]=ele;

LRstackopt(repptr);//Now the page that is brought is the latest referenced one, so it is moved to TOP

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{stk[full]=full;//First stack is populated till all the pages are full

frame[full++]=ele;}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;LRstackopt(i);//When ever reference is made, but pae fault doesn't occur, the referenced page is moved to TOP

break;

}}

return flag;

}

int main()

{int i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{

if(Search(a[i])!=1)

{Pagefault(a[i]);

count++;

}

display();

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 4.LRU using Counter

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21],n;//To take the input

int frame[SIZE];

int ctr[SIZE]={0};

static int f=0;//This is a counter that keeps track of current time

int repptr;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Longestopt()//Fucntion for discivering the least recently used page using their corresponding counter values

{int i,min;

min=0;

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

if(ctr[min]>ctr[i])

min=i;

repptr=min;

return repptr;

}

int Pagerep(int ele)

{

int temp;

repptr=Longestopt();//Gets the LRU page from a function

temp=frame[repptr];

frame[repptr]=ele;

ctr[repptr]=f;//When ever a page is brought it's counter is kept as per the current time of use

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ctr[full]=f;//Setting the counter as per current time of use

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;ctr[i]=f;//If page fault doesn't occur, but the element is referenced, it's counter is set to the current time of use

break;

}}

return flag;

}

int main()

{int i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{f++;//Time of use is incremented

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 5.LFU

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21],n;//To take the input

int frame[SIZE];

int ctr[SIZE]={0};//Counter to know the frequency, intially all have zero frequency of use

int repptr;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Longestopt()

{int i,min;

min=0;

for(i=0;i<SIZE;i++)//The page with least frequency is selected as victim

if(ctr[min]>ctr[i])

min=i;

repptr=min;

return repptr;

}

int Pagerep(int ele)

{

int temp;

repptr=Longestopt();//The victim page is selected with the help of a function

temp=frame[repptr];

frame[repptr]=ele;

ctr[repptr]=1;//A new page is brought, hence it's counter is set to 1

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ctr[full]++;//The counters will increase initially for all frames till they are full

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;ctr[i]++;//Whenever a reference is made the counter is incremented

break;

}}

return flag;

}

int main()

{int i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 6.MFU

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21],n;//To take the input

int frame[SIZE];

int ctr[SIZE]={0};

static int f;

int repptr;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Longestopt()

{int i,max;

max=0;//The increment of counter value here is same as that for of LFU

for(i=0;i<SIZE;i++)//The page with maximum frequency is selected

if(ctr[max]<ctr[i])

max=i;

repptr=max;

return repptr;

}

int Pagerep(int ele)

{

int temp;

repptr=Longestopt();

temp=frame[repptr];

frame[repptr]=ele;

ctr[repptr]=1;

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ctr[full]++;

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;ctr[i]++;

break;

}}

return flag;

}

int main()

{int i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{f=i;

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 7.Second Chance Algorithm

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21];//To take the input

int ref[SIZE];//This is for reference bits for each frame

int frame[SIZE];

int repptr=0;//Initialised to first frame

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

int Pagerep(int ele)

{

int temp;

/\*When ever a page needs to be replaced the repptr moves from page to page checking whether it's reference bit is 0 or not, if it is 0 it

coomes out of the while loop and if it is one, it gives a second chance setting the reference bit to 0\*/

while(ref[repptr]!=0)

{ ref[repptr++]=0;

if(repptr==SIZE)

repptr=0;

}

temp=frame[repptr];

frame[repptr]=ele;

ref[repptr]=1;//The latest page reference, hence it is set to 1

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ref[full]=1;//All the ref bits are set to 1 as each frame is being filled firstly

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;ref[i]=1;//When ever page reference occurs, it's rference bit is set to 1

break;

}}

return flag;

}

int main()

{int n,i;

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

## 8.Additional Reference Bits Algorithm

#include<stdio.h>

#define SIZE 3

int full=0;//To check whether all frames are filled

int a[21];//To take the input

int ref[SIZE];

char r=1;

int frame[SIZE];

unsigned char addref[SIZE]={0};

int repptr=0;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

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

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

}

//For showing bits

int bitrep(char f)//Program for showing the bits

{char p,fl;

int j;

char i=1;

for(j=7;j>=0;j--)

{p=i<<j;

fl=p&f;

if(fl==0)

printf("0");

else

printf("1");

}

printf("\n");

}

//For showing bit wise

int additionalreference()

{

int i,min;

min=0;

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

if(addref[min]>addref[i])//Finding the smallest decimal value from all the strings available

min=i;

repptr=min;

return repptr;

}

int Pagerep(int ele)

{

int temp;

repptr=additionalreference();

temp=frame[repptr];

frame[repptr]=ele;

ref[repptr]=1;

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ref[full]=1;

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

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

if(ele==frame[i])

{ flag=1;ref[i]=1;

break;

}}

return flag;

}

int main()

{int n,i,k;

r=r<<7;

bitrep(r);

FILE \*fp;

fp=fopen("Input.txt","r");

printf("The number of elements in the reference string are :");

fscanf(fp,"%d",&n);

printf("%d",n);

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

fscanf(fp,"%d",&a[i]);

printf("\nThe elements present in the string are\n");

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

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

printf("\n\n");

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

{

if(Search(a[i])!=1)

{Pagefault(a[i]);

display();

count++;

}

printf("\nThe values in shift registers related to three frames are\n");

//Shifing of registers

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

{

addref[k]=addref[k]>>1;//Shifting all the shift registers by one bit

if(ref[k]==1)

{ addref[k]=(r|addref[k]);//Bit OR operation for shifting

ref[k]=0;

}

bitrep(addref[k]);

}

//End of shifting

}

printf("\nThe number of page faults are %d\n",count);

getche();

return 0;

}

# Disk Sheduling Algorithms

## Input file(Input.txt – same for all programs)

8

98 183 37 122 14 124 65 67

53

## FCFS

#include<stdio.h>

int main()

{int n,a[20],i,rwp,rwc,j;

rwc=0;

int f[20]={0};

int flag=1;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

i=0;

printf("\n\nThe order of scanning is \n\n");

while(flag)

{

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

i++;

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}

## SSTF

#include<stdio.h>

int n,a[20],rwp,rwc,j;

rwc=0;

int f[20]={0};

int SSTF()

{int ss[20];

int min;

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

if(rwp>a[j])

ss[j]=rwp-a[j];

else

ss[j]=a[j]-rwp;

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

if(f[j]==0)

{min=j;

break;

}

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

if(ss[j]<ss[min]&&f[j]!=1)

min=j;

return min;

}

int main()

{int i;

int flag=1;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

printf("\n\nThe order of scanning is \n\n");

i=0;

while(flag)

{

i=SSTF();

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}

## SCAN

#include<stdio.h>

int n,a[20],rwp,rwc,j,leftflag=1;

int rwc=0;

int f[20]={0};

int SCAN()

{if(leftflag)

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

if(rwp<=a[j])

break;

j=j-1;

return j;}

else

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

if(rwp<=a[j]&&f[j]!=1)

break;

return j;

}

}

int main()

{

int i,flag=1,temp;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

i=0;

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

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

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

{temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

printf("\n\nThe order of scanning is \n\n");

while(flag)

{

i=SCAN();

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

if(i==0)

{ leftflag=0;

rwc=rwc+a[i];

rwp=0;

printf("%d\n\n",rwp);

}

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}

## C-SCAN

#include<stdio.h>

int n,a[20],rwp,rwc,j,leftflag=1;

int rwc=0;

int f[20]={0};

int SCAN()

{if(leftflag)

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

if(rwp<=a[j])

break;

j=j-1;

return j;}

else

{for(j=n-1;j>0;j--)

if(rwp>=a[j]&&f[j]!=1)

break;

return j;

}

}

int main()

{

int i,flag=1,temp;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

i=0;

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

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

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

{temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

printf("\n\nThe order of scanning is \n\n");

while(flag)

{

i=SCAN();

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

if(i==0)

{ leftflag=0;

rwc=rwc+a[i];

rwp=199;

printf("%d\n\n",rwp);

}

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}

## Look

#include<stdio.h>

int n,a[20],rwp,rwc,j,leftflag=1;

int rwc=0;

int f[20]={0};

int SCAN()

{if(leftflag)

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

if(rwp<=a[j])

break;

j=j-1;

return j;}

else

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

if(rwp<=a[j]&&f[j]!=1)

break;

return j;

}

}

int main()

{

int i,flag=1,temp;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

i=0;

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

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

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

{temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

printf("\n\nThe order of scanning is \n\n");

while(flag)

{

i=SCAN();

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

if(i==0)

{ leftflag=0;

}

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}

## C – Look

#include<stdio.h>

int n,a[20],rwp,rwc,j,leftflag=1;

int rwc=0;

int f[20]={0};

int SCAN()

{if(leftflag)

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

if(rwp<=a[j])

break;

j=j-1;

return j;}

else

{for(j=n-1;j>0;j--)

if(rwp>=a[j]&&f[j]!=1)

break;

return j;

}

}

int main()

{

int i,flag=1,temp;

FILE \*fp;

fp=fopen("Input.txt","r");

fscanf(fp,"%d",&n);

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

fscanf(fp,"%d",&a[i]);

fscanf(fp,"%d",&rwp);

fclose(fp);

printf("\n\nThe size of the disk sheduling queue is %d",n);

printf("\n\nThe elements in the disk sheduling queue are\n\n");

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

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

i=0;

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

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

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

{temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

printf("\n\nThe order of scanning is \n\n");

while(flag)

{

i=SCAN();

if(rwp>a[i])

rwc=rwc+rwp-a[i];

else

rwc=rwc+a[i]-rwp;

rwp=a[i];

f[i]=1;

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

if(i==0)

{ leftflag=0;

rwp=a[n-1];

}

flag=0;

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

if(f[j]==0)

{ flag=1;

break;

}

}

printf("\n\nThe number of movements of read write head are %d",rwc);

getche();

}