Disk Scheduling

#include <stdio.h>

#include <stdlib.h>

int n, head,max,min,range,i,fmax=0,fmin=9999;

char direction;

int sum = 0;

void fcfs(int rs[])

{

sum=abs(head-rs[0]);

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

sum=sum+abs(rs[j]-rs[j-1]);

}

printf("Total seek operation:%d",sum);

restore();

}

void sstf(int rs[]){

int visited[n];

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

visited[i] = 0;

}

int sum = 0;

int current = head;

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

int minDiff = 1e9;

int index = -1;

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

if (!visited[j]) {

int diff = abs(current - rs[j]);

if (diff < minDiff) {

minDiff = diff;

index = j;

}

}

}

visited[index] = 1;

sum += minDiff;

current = rs[index];

}

printf("Total Seek Time: %d\n", sum);

restore();

}

void scan(int rs[])

{

printf("Enter the direction(l or r)");

scanf(" %c", &direction);

if (direction == 'l')

{

max=rs[0];

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

{

if (rs[i] > max)

{

max=rs[i];

}

}

sum = head +max;

printf("Total Seek Time: %d\n", sum);

}

else if (direction == 'r')

{

printf("enter the range:");

scanf("%d",&range);

min=999;

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

{

if (rs[i]<min)

{

min=rs[i];

}

}

sum = (range-head)+(range-min);

printf("Total Seek Time: %d\n", sum);

}

else

{

printf("invalid input try l or r");

}

restore();

}

void look(int rs[]){

printf("Enter the direction(l or r)");

scanf(" %c", &direction);

max=rs[0];

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

{

if (rs[i] > max)

{

max=rs[i];

}

}

min=999;

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

{

if (rs[i]<min)

{

min=rs[i];

}

}

if (direction == 'l')

{

sum = (head-min)+(max-min);

printf("Total Seek Time: %d\n", sum);

}

else if (direction == 'r')

{

sum = (max-head)+(max-min);

printf("Total Seek Time: %d\n", sum);

}

else

{

printf("invalid input try l or r");

}restore();

}

void cscan(int rs[]){

printf("Enter the direction(l or r)");

scanf(" %c", &direction);

if (direction == 'r')

{

max=0;

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

{

if (rs[i] > max && rs[i]<head )

{

max=rs[i];

}

}

sum = (range-head)+max;

printf("Total Seek Time: %d\n", sum);

}

else if (direction == 'l')

{

min=range;

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

{

if (rs[i]<min && rs[i]> head)

{

min=rs[i];

}

}

sum = head+(range-min);

printf("Total Seek Time: %d\n", sum);

}

else

{

printf("invalid input try l or r");

}restore();

}

void clook(int rs[])

{

printf("Enter the direction(l or r)");

scanf(" %c", &direction);

fmax=0;

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

{

if (rs[i] > fmax)

{

fmax=rs[i];

}

}

fmin=range;

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

{

if (rs[i]<fmin)

{

fmin=rs[i];

}

}

if (direction == 'r')

{

max=0;

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

{

if (rs[i] > max && rs[i]<head )

{

max=rs[i];

}

}

sum = (fmax-head)+(max-fmin);

printf("Total Seek Time: %d \n", sum);

}

else if (direction == 'l')

{

min=range;

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

{

if (rs[i]<min && rs[i]> head)

{

min=rs[i];

}

}

sum = (head-fmin)+(fmax-min);

printf("Total Seek Time: %d \n", sum);

}

else

{

printf("invalid input try l or r");

}restore();

}

void restore()

{

sum=0;

max=0;

min=0;

}

void main()

{

printf("Enter the number of requests (n):");

scanf("%d", &n);

int rs[n];

printf("Enter request sequence one by one (in ascending)\n");

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

{

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

}

printf("Enter initial head position (p):");

scanf("%d", &head);

printf("enter the range:");

scanf("%d",&range);

printf("\n fcfs");

fcfs(rs);

printf("\n sstf");

sstf(rs);

printf("\n scan ");

scan(rs);

printf("\n look");

look(rs);

printf("\n cscan");

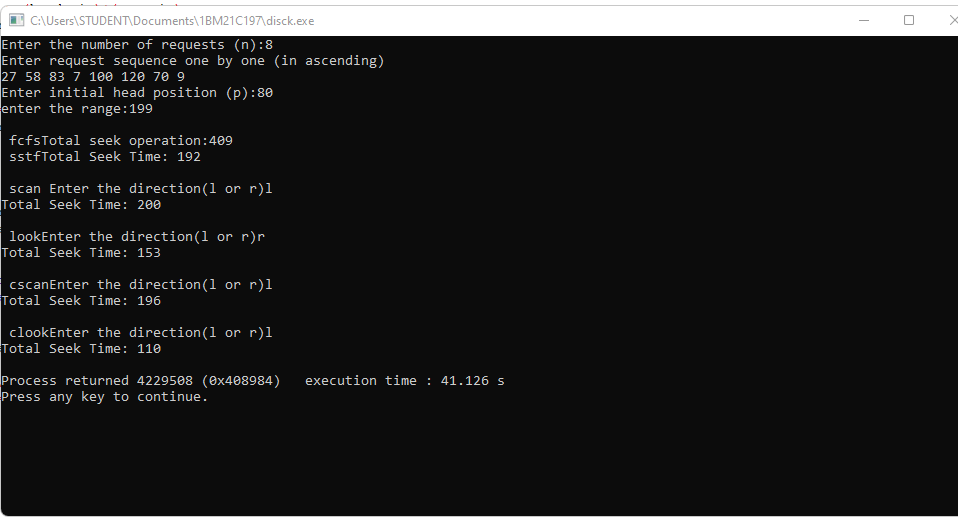
cscan(rs);

printf("\n clook");

clook(rs);

}

Output:



Page replacement

#include<stdio.h>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

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();

}

void main()

{

int choice=0;

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);

while(choice!=4)

{

printf("\nPage Replacement Algorithms\n1.FIFO\n2.Optimal\n3.LRU\n4.Exit\nEnter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

fifo();

break;

case 2:

optimal();

break;

case 3:

lru();

break;

case 4:

printf("program exiting...");

break;

default:

printf("Invalid input");

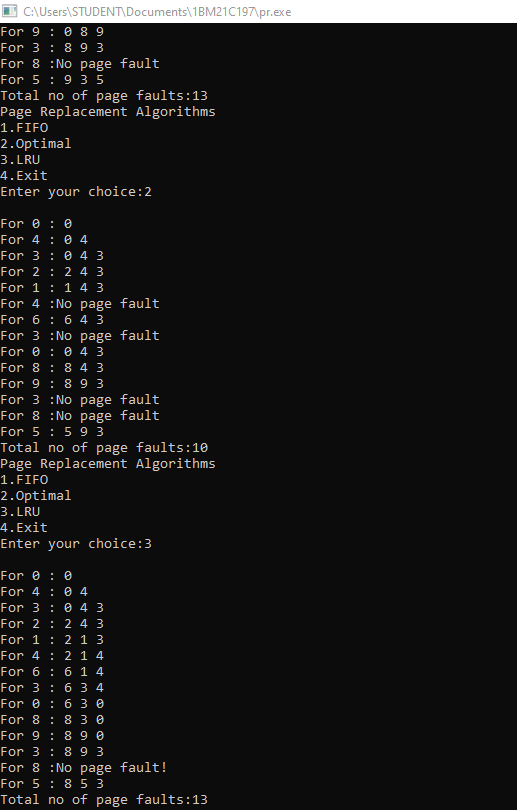
break;

}

}

}

Output:



Memory Management

#include<stdio.h>

int n,m;

struct mem{

int d;

int f;

};

struct mem p[100];

void best(int r[])

{

int i,j,k,a[m],min;

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

{

min=9999;

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

{

if(p[j].d>=r[i] && p[j].f==0)

{

if(p[j].d<=min)

min=p[j].d;

}

}

a[i]=min;

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

{

if(p[k].d==min && p[j].f==0 )

{

p[j].f=1;

j=n;

}

}

}

printf("the pits assigned are:\n");

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

{

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

}

}

void worst(int r[])

{

int i,j,k,a[m],max;

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

{

max=-1;

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

{

if(p[j].d>=r[i] && p[j].f==0)

{

if(p[j].d>=max)

max=p[j].d;

}

}

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

{

if(p[k].d==max && p[j].f==0 )

{

p[k].f=1;

k=n;

}

}

a[i]=max;

}

printf("the pits assigned are:\n");

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

{

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

}

}

void first(int r[])

{

int i,j,a[m];

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

{

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

{

if(p[j].d>=r[i] && p[j].f==0)

{

a[i]=p[j].d;

p[j].f=1;

j=n;

}

}

}

printf("the pits assigned are:\n");

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

{

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

}

}

void main()

{ int c,i,temp;

printf("enter the no of pits:");

scanf("%d",&n);

printf("enter the size of pits:");

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

{

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

p[i].f=0;

}

printf("enter the no of files to run:");

scanf("%d",&m);

int r[m];

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

{

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

}

printf("enter the 1.first fit 2.best fit 3.worst fit");

scanf("%d",&c);

switch (c)

{

case 1: first(r); /\* code \*/

break;

case 2: best(r);

break;

case 3: worst(r);

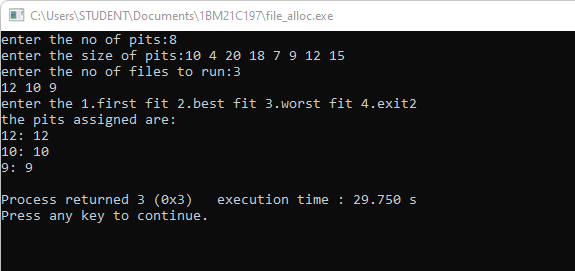
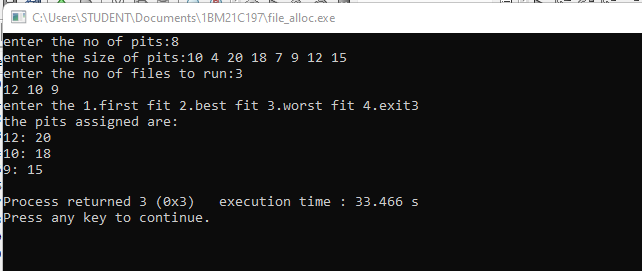
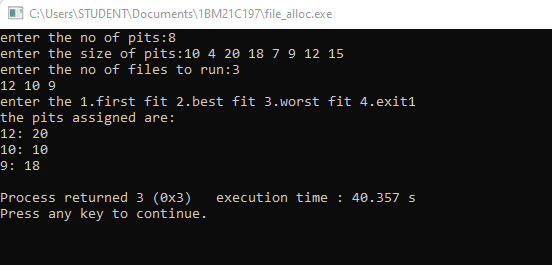
break;

default:printf("entered invalid no");

break;

}

Output:

 Bankers Algorithm

#include <stdio.h>

int main() {

int n, m, all[10][10], req[10][10], ava[10], need[10][10];

int i, j, k, flag[10], prev[10], c, count = 0,array[10],z=0;

printf("Enter number of processes and number of resources required \n");

scanf("%d %d", &n, &m);

printf("Enter total number of required resources %d for each process\n", n);

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

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

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

printf("Enter number of allocated resources %d for each process\n", n);

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

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

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

printf("Enter number of available resources \n");

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

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

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

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

need[i][j] = req[i][j] - all[i][j];

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

flag[i] = 1;

k = 1;

while (k) {

k = 0; // Reset the value of k for each iteration of the loop

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

if (flag[i]) {

c = 0;

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

if (need[i][j] <= ava[j]) {

c++;

}

}

if (c == m) {

array[z++]=i;

printf("Resouces can be allocated to Process:%d and available resources are: ", (i + 1));

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

printf("%d ", ava[j]);

}

printf("\n");

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

ava[j] += all[i][j];

all[i][j] = 0;

}

flag[i] = 0;

count++;

}

}

}

// Check if the current state is different from the previous state

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

if (flag[i] != prev[i]) {

k = 1;

break;

}

}

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

prev[i] = flag[i];

}

}

printf("\nNeed Matrix:\n");

for (i = 0; i < n; i++) //printing need matrix

{

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

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

printf("\n");

}

if (count == n) {

printf("\nSystem is in safe mode \n<");

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

printf("P%d ",(array[i]+1));

printf(">\n");

} else {

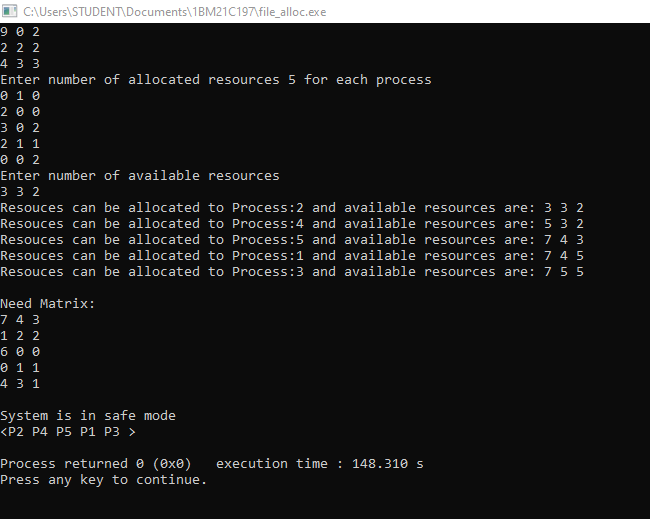
printf("\nSystem is not in safe mode deadlock occurred \n");

}

return 0;

}

Output:



Producer consumer

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

int mutex=1, full=0, empty=3, x=0;

int main() {

void producer();

int n;

void consumer();

int signal(int); printf("\n1.Producer\n2.Consumer");

int wait (int);

while(1)

{

printf("\nEnter your choice:"); scanf("%d", &n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0)) consumer ();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait (int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{mutex=wait(mutex);

full=signal(full); empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full); empty=signal(empty);

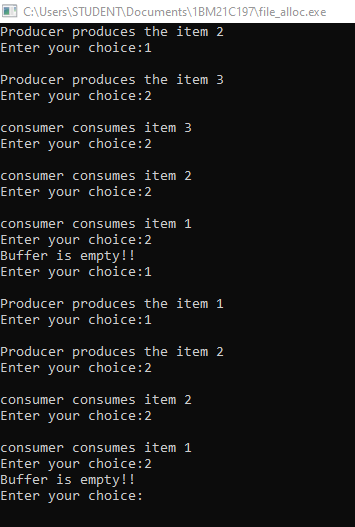
printf("\nconsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

Output:



Dining Philosopher

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

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

sem\_init(&chopstick[i],0,1);

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

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

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

pthread\_join(tid[i],NULL);

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

Output:

