**SJF (EXP 3)**

#include<iostream>

#include<iomanip>

using namespace std;

float average(int \*matrix,int n){

float avg = 0.0;

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

avg += matrix[i];

return avg/n;

}

void calc(int burst[], int arr[],int n){

//calculation

int seq[20], k=0, current\_time=0, totaltime=0, min;

bool complete[20];

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

totaltime += burst[i];

complete[i] = false;

}

do {

int min\_time=999;

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

if (burst[j]<min\_time && complete[j]==false){

if (arr[j]<=current\_time){

min\_time=burst[j];

min=j;

}

}

}

complete[min]=true;

seq[k]=min;

k++;

current\_time += burst[min];

} while (totaltime>current\_time);

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

cout<<seq[i]+1<<" ";

}

//gantt chart

int time=0;

int WT[200], TAT[20];

cout<<"Gantt chart:"<<endl<<endl;

cout<<"| ";

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

cout<<"P"<<seq[i]+1<<" | ";

cout<<endl;

cout<<arr[seq[0]];

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

WT[seq[i]] = time - arr[seq[i]];

TAT[seq[i]] = burst[seq[i]] + WT[seq[i]];

time += burst[seq[i]];

cout<<" "<<time<<" ";

}

//display columns and average

cout<<endl;

cout<<"P\tBT\tAT\tWT\tTAT"<<endl;

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

cout<<"P"<<i+1<<"\t"<<burst[i]<<"\t"<<arr[i]<<"\t"<<WT[i]<<"\t"<<TAT[i];

cout<<endl;

}

cout<<"AWT = "<<average(WT, n)<<endl;

cout<<"ATAT = "<<average(TAT, n);

}

int main(){

int n;

cout<<"Enter number of processes: ";

cin>>n;

int burst[n],arr[n];

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

cout<<"Process "<<i+1<<endl;

cout<<"Enter Burst Time: ";

cin>>burst[i];

cout<<"Enter Arrival Time: ";

cin>>arr[i];

}

cout<<endl;

calc(burst,arr,n);

return 0;

}

**NON PREEMPTIVE PRIORITY (exp 3)**

#include<iostream>

#include<iomanip>

using namespace std;

float average(int \*matrix,int n){

float avg = 0.0;

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

avg += matrix[i];

return avg/n;

}

void calc(int burst[], int arr[], int pr[], int n){

//calculation

int seq[20], k=0, current\_time=0, totaltime=0, min;

bool complete[20];

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

totaltime += burst[i];

complete[i] = false;

}

do {

int min\_priority=999;

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

if (pr[j]<min\_priority && complete[j]==false){

if (arr[j]<=current\_time){

min\_priority=pr[j];

min=j;

}

}

}

complete[min]=true;

seq[k]=min;

k++;

current\_time += burst[min];

} while (totaltime>current\_time);

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

cout<<seq[i]+1<<" ";

}

//gantt chart

int time=0;

int WT[200], TAT[20];

cout<<"Gantt chart:"<<endl<<endl;

cout<<"| ";

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

cout<<"P"<<seq[i]+1<<" | ";

cout<<endl;

cout<<arr[seq[0]];

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

WT[seq[i]] = time - arr[seq[i]];

TAT[seq[i]] = burst[seq[i]] + WT[seq[i]];

time += burst[seq[i]];

cout<<" "<<time<<" ";

}

//display columns and average

cout<<endl;

cout<<"P\tBT\tAT\tPr\tWT\tTAT"<<endl;

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

cout<<"P"<<i+1<<"\t"<<burst[i]<<"\t"<<arr[i]<<"\t"<<pr[i]<<"\t"<<WT[i]<<"\t"<<TAT[i];

cout<<endl;

}

cout<<"AWT = "<<average(WT, n)<<endl;

cout<<"ATAT = "<<average(TAT, n);

}

int main(){

int n;

cout<<"Enter number of processes: ";

cin>>n;

int burst[n],arr[n], pr[n];

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

cout<<"Process "<<i+1<<endl;

cout<<"Enter Burst Time: ";

cin>>burst[i];

cout<<"Enter Arrival Time: ";

cin>>arr[i];

cout<<"Enter Priority: ";

cin>>pr[i];

}

cout<<endl;

calc(burst, arr, pr, n);

return 0;

}

**DISK SCHEDULING – FCFS (exp 7)**

#include<iostream>

#include<cstdlib>

using namespace std;

void fcfs(int requests[], int n, int head){

int totalseek=0, current=head, seek;

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

cout<<"Seek "<<i+1<<": "<<requests[i];

cout<<endl;

seek = abs(current-requests[i]);

totalseek += seek;

current = requests[i];

}

cout<<endl<<endl<<"Total seek time: "<<totalseek;

}

int main()

{

int n, requests[20], head;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

fcfs(requests, n, head);

return 0;

}

**DISK SCHEDULING – SSTF (exp 7)**

#include<bits/stdc++.h>

using namespace std;

void sstf(int requests[], int n, int head)

{

int totalseek=0, current=head, head\_index;

sort(requests, requests+n);

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

if (requests[i] > head){

head\_index=i;

break;

}

}

int left=head\_index-1, right=head\_index;

int k=1;

while (left>=0 && right<n){

if (abs(current - requests[left]) < abs(current - requests[right])){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

}

else {

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

}

k++;

}

if (left == -1){

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

}

if (right == n){

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

}

cout<<endl<<"Total seek time: "<<totalseek;

}

int main()

{

int n, requests[20], head;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

sstf(requests, n, head);

return 0;

}

**DISK SCHEDULING – SCAN (exp 7)**

#include<bits/stdc++.h>

using namespace std;

void scan(int requests[], int n, int head, int dir, int len){

int totalseek=0, current=head, head\_index;

sort(requests, requests+n);

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

if (requests[i] > head){

head\_index=i;

break;

}

}

int left=head\_index-1, right=head\_index;

int k=1;

//moves towards left

if (dir == -1){

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

totalseek += current; //to include 0

totalseek += requests[right];

cout<<"Seek "<<k++<<": 0"<<endl;

current=requests[right];

cout<<"Seek "<<k++<<": "<<current<<endl; //to include first right element

right++;

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

}

//moves towards right

if (dir == 1){

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

totalseek += abs(current-len-1); //to include 199

totalseek += abs(requests[left]-len-1);

cout<<"Seek "<<k++<<": "<<len-1<<endl;

current=requests[left];

cout<<"Seek "<<k++<<": "<<current<<endl; //to include first left element

left--;

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

}

cout<<endl<<totalseek;

}

int main()

{

int n, requests[20], head, dir, len;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

cout<<"Enter length: ";

cin>>len;

cout<<"Enter direction in which you want to move (-1 for left, 1 for right): ";

cin>>dir;

scan(requests, n, head, dir, len);

return 0;

}

**DISK SCHEDULING – CSCAN (exp 7)**

#include<bits/stdc++.h>

using namespace std;

void cscan(int requests[], int n, int head, int dir, int len){

int totalseek=0, current=head, head\_index;

int end=n-1, begin=0; //to count backward while changing direction

sort(requests, requests+n);

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

if (requests[i] > head){

head\_index=i;

break;

}

}

int left=head\_index-1, right=head\_index;

int k=1;

//moves towards left

if (dir == -1){

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

totalseek += current; //to include 0

cout<<"Seek "<<k++<<": 0"<<endl;

totalseek += len-1; //to go from 0-199

cout<<"Seek "<<k++<<": "<<len-1<<endl;

current = len-1;

while (end > right){ //backward iterate until you reach 'right'

totalseek += abs(current - requests[end]);

current = requests[end];

cout<<"Seek "<<k++<<": "<<current<<endl;

end--;

}

}

//moves towards right

if (dir == 1){

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

totalseek += abs(current-len-1); //to include 199

cout<<"Seek "<<k++<<": "<<len-1<<endl;

totalseek += len-1; // to go from 199-0

cout<<"Seek "<<k++<<": 0"<<endl;

current=0;

while (begin <= left){ //forward iterate until you reach 'left'

totalseek += abs(current - requests[begin]);

current = requests[begin];

cout<<"Seek "<<k++<<": "<<current<<endl;

begin++;

}

}

cout<<endl<<totalseek;

}

int main()

{

int n, requests[20], head, dir, len;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

cout<<"Enter length: ";

cin>>len;

cout<<"Enter direction in which you want to move (-1 for left, 1 for right): ";

cin>>dir;

cscan(requests, n, head, dir, len);

return 0;

}

**DISK SCHEDULING – LOOK (exp 7)**

#include<bits/stdc++.h>

using namespace std;

void look(int requests[], int n, int head, int dir, int len){

int totalseek=0, current=head, head\_index;

sort(requests, requests+n);

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

if (requests[i] > head){

head\_index=i;

break;

}

}

int left=head\_index-1, right=head\_index;

int k=1;

//moves towards left

if (dir == -1){

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

}

//moves towards right

if (dir == 1){

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

}

cout<<endl<<totalseek;

}

int main()

{

int n, requests[20], head, dir, len;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

cout<<"Enter length: ";

cin>>len;

cout<<"Enter direction in which you want to move (-1 for left, 1 for right): ";

cin>>dir;

look(requests, n, head, dir, len);

return 0;

}

**DISK SCHEDULING – CLOOK (exp 7)**

#include<bits/stdc++.h>

using namespace std;

void clook(int requests[], int n, int head, int dir, int len){

int totalseek=0, current=head, head\_index;

int end=n-1, begin=0; //to count backward while changing direction

sort(requests, requests+n);

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

if (requests[i] > head){

head\_index=i;

break;

}

}

int left=head\_index-1, right=head\_index;

int k=1;

//moves towards left

if (dir == -1){

while (left >= 0){

totalseek += abs(current - requests[left]);

current = requests[left];

cout<<"Seek "<<k<<": "<<requests[left]<<endl;

left--;

k++;

}

totalseek += abs(current - requests[end]); // to go from first element to last element

current = requests[end];

cout<<"Seek "<<k++<<": "<<current<<endl;

end--;

while (end > right){ //backward iterate until you reach 'right'

totalseek += abs(current - requests[end]);

current = requests[end];

cout<<"Seek "<<k++<<": "<<current<<endl;

end--;

}

}

//moves towards right

if (dir == 1){

while (right < n){

totalseek += abs(current - requests[right]);

current = requests[right];

cout<<"Seek "<<k<<": "<<requests[right]<<endl;

right++;

k++;

}

totalseek += abs(current - requests[begin]); //to go from last element to first element

current = requests[begin];

cout<<"Seek "<<k++<<": "<<current<<endl;

begin++;

while (begin <= left){ //forward iterate until you reach 'left'

totalseek += abs(current - requests[begin]);

current = requests[begin];

cout<<"Seek "<<k++<<": "<<current<<endl;

begin++;

}

}

cout<<endl<<totalseek;

}

int main()

{

int n, requests[20], head, dir, len;

cout<<"Enter no of requests: ";

cin>>n;

cout<<"Enter requests sequence: ";

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

cin>>requests[i];

}

cout<<"Enter head position: ";

cin>>head;

cout<<"Enter length: ";

cin>>len;

cout<<"Enter direction in which you want to move (-1 for left, 1 for right): ";

cin>>dir;

clook(requests, n, head, dir, len);

return 0;

}

**PAGE REPLACEMENT – FIFO (exp 6)**

#include<iostream>

#include<queue>

#include<unordered\_set>

using namespace std;

void display(queue<int> temp)

{

while (!temp.empty()){

cout<<temp.front()<<" ";

temp.pop();

}

cout<<endl;

}

void fifo(int pages[], int n, int m)

{

int faults=0, oldestPage;

queue<int> pageQueue;

unordered\_set<int> pageSet;

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

if (pageSet.size() < m){

if (pageSet.find(pages[i]) == pageSet.end()){

pageSet.insert(pages[i]);

pageQueue.push(pages[i]);

cout<<"Page added (page faults): "<<pages[i]<<endl;

display(pageQueue);

cout<<endl;

faults++;

}

else {

cout<<"Page already in memory: "<<pages[i]<<endl;

display(pageQueue);

cout<<endl;

}

}

else{

if (pageSet.find(pages[i]) == pageSet.end()){

oldestPage = pageQueue.front();

pageQueue.pop();

pageSet.erase(oldestPage);

pageSet.insert(pages[i]);

pageQueue.push(pages[i]);

cout<<"Page added (page fault): "<<pages[i]<<endl;

display(pageQueue);

cout<<endl;

faults++;

}

else {

cout<<"Page already in memory: "<<pages[i]<<endl;

display(pageQueue);

cout<<endl;

}

}

}

cout<<endl<<"No of page faults: "<<faults;

}

int main()

{

int n, m, pages[20];

cout<<"Enter no of pages: ";

cin>>n;

cout<<"Enter no of frames: ";

cin>>m;

cout<<"Enter page string: ";

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

cin>>pages[i];

}

fifo(pages, n, m);

return 0;

}

**PAGE REPLACEMENT – LRU(exp 6)**

#include<iostream>

#include<list>

#include<unordered\_map>

using namespace std;

void display(list<int> temp)

{

while (!temp.empty()){

cout<<temp.front()<<" ";

temp.pop\_front();

}

cout<<endl;

}

void lru (int pages[], int n, int m)

{

int faults=0;

list<int> pageList;

unordered\_map<int, list<int>::iterator> pageMap;

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

if (pageMap.find(pages[i]) == pageMap.end()){ //if page not found

if (pageMap.size() == m){ //if size of list is full

int leastRecentlyUsed = pageList.back();

pageList.pop\_back();

pageMap.erase(leastRecentlyUsed);

}

faults++; //increment faults

cout<<"Page added: "<<pages[i]<<endl;

}

else { //if page found

cout<<"Page already in memory: "<<pages[i]<<endl;

pageList.erase(pageMap[pages[i]]);

}

pageList.push\_front(pages[i]); //push page to front of list

pageMap[pages[i]] = pageList.begin(); //map the added page to front of list

}

cout<<endl<<faults;

}

int main()

{

int n, m, pages[20];

cout<<"Enter no of pages: ";

cin>>n;

cout<<"Enter no of frames: ";

cin>>m;

cout<<"Enter page string: ";

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

cin>>pages[i];

}

lru(pages, n, m);

return 0;

}

**PAGE REPLACEMENT – Optimal(exp 6)**

#include<iostream>

using namespace std;

#define MAX 20

int pages[MAX], frames[MAX];

int n, m;

int flag1, flag2;

int faults = 0;

int optimal (int n, int m, int current){

int flag, k=0, temp[MAX];

for (int j=0; j<m; j++){ //for checking next occurence of a page in page sequence

flag = 0;

for (int i=current+1; i<n; i++){

if (frames[j] == pages[i]){ //next immediate occurence is found

temp[k++] = i; //occurence index assigned to temp

flag = 1;

break; //since next occurence is found, terminate loop for that page

}

}

if (flag == 0){ //if that page has no next occurence in sequence then return the current frame index

return j;

}

}

int max = temp[0], pos; //if all pages in the frames have next occurences, find the farthest (maximum index) to replace

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

if (temp[j] > max){

max = temp[j];

pos = j;

}

}

return pos;

}

int main()

{

cout<<"Enter no of pages: ";

cin>>n;

cout<<"Enter number of frames: ";

cin>>m;

cout<<"Enter page sequence: ";

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

cin>>pages[i];

}

for (int i=0; i<m; i++){ //initialize all frames to -1

frames[i] = -1;

}

for (int i=0; i<n; i++){ //iterate over all pages

flag1 = 0;

flag2 = 0;

for (int j=0; j<m; j++){ //to check if page in memory

if (frames[j] == pages[i]){

cout<<"Page already in memory"<<endl;

flag1 = 1;

flag2 = 1;

break;

}

}

if (flag1 == 0){ //page is not in memory => checking for empty frame

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

if (frames[j] == -1){

frames[j] = pages[i];

faults++;

cout<<"Page added"<<endl;

flag2 = 1;

break;

}

}

}

if (flag2 == 0){ //page not in mempry && no empty frame => finding frame to replace

int pos = optimal(n, m, i);

frames[pos] = pages[i];

cout<<"Page added"<<endl;

faults++;

}

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

cout<<frames[j]<<" ";

}

cout<<endl<<endl;

}

cout<<endl<<faults;

return 0;

}

**PRODUCER-CONSUMER PROBLEM**

#include<iostream>

using namespace std;

#define MAX 20

int buffer\_size, mutex=1, empty, full=0;

int in=0, out=0, counter=0;

int buffer[MAX];

void wait (int &S)

{

if (S<=0){

//cout<<"Cannot add/remove item"<<endl;

return;

}

S--;

}

void signal (int &S)

{

S++;

}

void produce()

{

wait(empty);

wait(mutex);

if (counter == buffer\_size){

cout<<"Storage full"<<endl;

}

else {

int next\_produced;

cout<<"Enter item to produce: ";

cin>>next\_produced;

buffer[in] = next\_produced;

cout<<"Item added: "<<next\_produced<<endl;

in = (in+1)%buffer\_size;

counter++;

signal(mutex);

signal(full);

}

}

void consume()

{

wait(full);

wait(mutex);

if (counter == 0){

cout<<"Storage empty"<<endl;

}

else {

int next\_consumed;

next\_consumed = buffer[out];

cout<<"Item consumed: "<<next\_consumed<<endl;

out = (out+1)%buffer\_size;

counter--;

signal(mutex);

signal(empty);

}

}

int main()

{

cout<<"Enter size of buffer: ";

cin>>buffer\_size;

empty = buffer\_size;

do {

int ch;

cout<<"\nSelection operation to be performed:\n1. Produce\n2. Consume\n3. Exit\n";

cout<<"Enter your choice: ";

cin>>ch;

cout<<endl;

switch(ch) {

case 1:

produce();

break;

case 2:

consume();

break;

case 3:

exit(1);

deafult:

cout<<"Enter valid choice";

break;

}

} while(true);

return 0;

}