**OPERATING SYSTEMS**

**ASSIGNMENT SUBMISSION**

**CPU SCHEDULING:**

1. FCFS

#include <iostream>

using namespace std;

int main(){

int process;

cout << "Input the number of processes" << endl;

cin >> process;

int burst[process], arr[process], turn\_around[process], wait[process], service[process];

int total\_wait = 0, total\_turn\_around = 0;

cout << "Input the arrival time" << endl;

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

cin >> arr[i];

}

cout << "Input the burst time" << endl;

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

cin >> burst[i];

}

service[0] = arr[0];

wait[0] = 0;

//for waiting time

for(int i=1; i<process; i++){

service[i] = service[i-1] + burst[i-1];

wait[i] = service[i] - arr[i];

if(wait[i]<0){

wait[i]=0;

}

}

//for turnaround time

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

turn\_around[i] = wait[i] + burst[i];

}

//average waiting time

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

total\_wait+=wait[i];

}

float avg\_wait\_time = (float)total\_wait/(float)process;

//average turn\_around time

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

total\_turn\_around+=turn\_around[i];

}

float avg\_turn\_around\_time = (float)total\_turn\_around/(float)process;

cout << "Total wait time is " << total\_wait << endl;

cout << "Average wait time is " << avg\_wait\_time << endl;

cout << "Total turn around time is " << total\_turn\_around << endl;

cout << "Average turn around time is " << avg\_turn\_around\_time << endl;

}

Text

Description automatically generated

2. SJF (Non Pre-emptive)

#include <iostream>

using namespace std;

int mat[10][6];

void swap(int\* a, int\* b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void sortArr(int num, int mat[][6]){

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

for (int j = 0; j < num - i - 1; j++) {

if (mat[j][1] > mat[j + 1][1]) {

for (int k = 0; k < 5; k++) {

swap(mat[j][k], mat[j + 1][k]);

}

}

}

}

}

void completionTime(int num, int mat[][6]){

int temp, val;

mat[0][3] = mat[0][1] + mat[0][2];

mat[0][5] = mat[0][3] - mat[0][1];

mat[0][4] = mat[0][5] - mat[0][2];

for (int i = 1; i < num; i++) {

temp = mat[i - 1][3];

int low = mat[i][2];

for (int j = i; j < num; j++) {

if (temp >= mat[j][1] && low >= mat[j][2]) {

low = mat[j][2];

val = j;

}

}

mat[val][3] = temp + mat[val][2];

mat[val][5] = mat[val][3] - mat[val][1];

mat[val][4] = mat[val][5] - mat[val][2];

for (int k = 0; k < 6; k++) {

swap(mat[val][k], mat[i][k]);

}

}

}

int main(){

int process, temp;

cout << "Input the number of processes" << endl;

cin >> process;

int burst[process], arr[process], turn\_around[process], wait[process], service[process];

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

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

cout << "Enter Process Id: ";

cin >> mat[i][0];

cout << "Enter Arrival Time: ";

cin >> mat[i][1];

cout << "Enter Burst Time: ";

cin >> mat[i][2];

}

cout << "Before sorting processes based on arrival times" << endl;

cout << "Process ID\tArrival Time\tBurst Time\n";

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

cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"<< mat[i][2] << "\n";

}

sortArr(process, mat);

completionTime(process, mat);

cout << "Final Result...\n";

cout << "Process ID\tArrival Time\tBurst Time\tWaiting "

"Time\tTurnaround Time\n";

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

cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"

<< mat[i][2] << "\t\t" << mat[i][4] << "\t\t"

<< mat[i][5] << "\n";

}

}

Text

Description automatically generated

3. Round Robin

#include <iostream>

using namespace std;

void updateQueue(int queue[], int timer, int arrival[], int n, int maxProcessInd){

int zeroInd;

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

if(queue[i]==0){

zeroInd=i;

break;

}

}

queue[zeroInd] = maxProcessInd + 1;

}

void maintainQueue(int queue[], int n){

for(int i=0; (i<n-1)&&(queue[i+1]!=0); i++){

int temp=queue[i];

queue[i]=queue[i+1];

queue[i+1]=temp;

}

}

void checkNewArr(int timer, int arrival[], int n, int maxProcessInd, int queue[]){

if(timer <= arrival[n-1]){

bool newArr = false;

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

if(arrival[j]<=timer){

if(maxProcessInd<j){

maxProcessInd=j;

newArr=true;

}

}

}

if(newArr){

updateQueue(queue, timer, arrival, n, maxProcessInd);

}

}

}

int main(){

int n, timeQuanta;

int timer=0, maxProcessInd=0;

float avgWait=0, avgTurnAroundTime=0;

cout << "Num of processes" << endl;

cin >> n;

cout << "Time Quanta" << endl;

cin >> timeQuanta;

int arrTime[n], burstTime[n], waitTime[n], turnAroundTime[n], queue[n], tempBurst[n];

bool complete[n];

cout << "Arrival time of the processes" << endl;

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

cin >> arrTime[i];

}

cout << "Burst time of the processes" << endl;

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

cin >> burstTime[i];

tempBurst[i]=burstTime[i];

}

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

complete[i] = false;

queue[i] = 0;

}

while(timer<arrTime[0]){

timer++;

}

queue[0] = 1;

while(true){

bool flag=true;

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

if(tempBurst[i]!=0){

flag=false;

break;

}

}

if(flag){

break;

}

for(int i=0; (i<n)&&(queue[i]!=0); i++){

int ctr=0;

while((ctr<timeQuanta)&&(tempBurst[queue[0]-1]>0)){

tempBurst[queue[0]-1] -=1;

timer+=1;

ctr++;

checkNewArr(timer, arrTime, n, maxProcessInd, queue);

}

if((tempBurst[queue[0]-1]==0)&&(complete[queue[0]-1]==false)){

turnAroundTime[queue[0]-1] = timer;

complete[queue[0]-1]=true;

}

bool idle=true;

if(queue[n-1]==0){

for(int i=0; i<n && queue[i]!=0; i++){

if(complete[queue[i]-1]==false){

idle=false;

}

}

}

else{

idle=false;

}

if(idle){

timer++;

checkNewArr(timer, arrTime, n, maxProcessInd, queue);

}

maintainQueue(queue, n);

}

}

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

turnAroundTime[i] = turnAroundTime[i]-arrTime[i];

waitTime[i] = turnAroundTime[i] - burstTime[i];

}

cout << "\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"<< endl;

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

cout <<i+1<<"\t\t"<<arrTime[i] <<"\t\t"<<burstTime[i]<<"\t\t"<<waitTime[i]<<"\t\t"<<turnAroundTime[i]<<endl;

}

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

avgWait+=waitTime[i];

avgTurnAroundTime+=turnAroundTime[i];

}

cout << "Average wait time " << float(avgWait/n) << endl;

cout << "Average turn around time " << float(avgTurnAroundTime/n);

}

Text

Description automatically generated

4. Priority (Non Pre-emptive)

#include <iostream>

using namespace std;

class Process

{

public:

int pid;

int arrivalTime;

int burstTime;

int priority;

int remainingTime;

int waitingTime;

int turnAroundTime;

int completionTime;

Process(int pid = 0, int arrivalTime = 0, int burstTime = 0, int priority = 0)

{

this->pid = pid;

this->arrivalTime = arrivalTime;

this->burstTime = burstTime;

this->priority = priority;

this->remainingTime = burstTime;

this->waitingTime = 0;

this->turnAroundTime = 0;

this->completionTime = 0;

}

};

void input(Process \*processes, int n)

{

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

{

cout << "Enter pid, arrival time, burst time and priority of process " << i + 1 << ": ";

cin >> processes[i].pid >> processes[i].arrivalTime >> processes[i].burstTime >> processes[i].priority;

processes[i].remainingTime = processes[i].burstTime;

}

}

void display(Process \*processes, int n)

{

cout << "PID\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurn Around Time" << endl;

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

{

cout << processes[i].pid << "\t" << processes[i].arrivalTime << "\t\t" << processes[i].burstTime << "\t\t" << processes[i].priority << "\t\t" << processes[i].completionTime << "\t\t" << processes[i].waitingTime << "\t\t" << processes[i].turnAroundTime << endl;

}

}

void swap(Process \*a, Process \*b)

{

Process temp = \*a;

\*a = \*b;

\*b = temp;

}

void sortByArrivalTime(Process \*processes, int n)

{

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

{

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

{

if (processes[j].arrivalTime > processes[j + 1].arrivalTime)

{

swap(&processes[j], &processes[j + 1]);

}

else if (processes[j].arrivalTime == processes[j + 1].arrivalTime)

{

if (processes[j].priority < processes[j + 1].priority)

{

swap(&processes[j], &processes[j + 1]);

}

}

}

}

}

void sortByPid(Process \*processes, int n)

{

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

{

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

{

if (processes[j].pid > processes[j + 1].pid)

{

swap(&processes[j], &processes[j + 1]);

}

}

}

}

void calculateCompletionTime(Process \*processes, int n)

{

sortByArrivalTime(processes, n);

processes[0].completionTime = processes[0].arrivalTime + processes[0].burstTime;

int time = processes[0].completionTime;

int processIndex = 0, minPriority;

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

{

minPriority = 999;

// Find process with highest priority among the processes that have arrived

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

{

if (processes[j].arrivalTime <= time && processes[j].priority < minPriority && processes[j].remainingTime > 0)

{

processIndex = j;

minPriority = processes[j].priority;

}

}

if (processes[processIndex].remainingTime > 0)

{

processes[processIndex].completionTime = time + processes[processIndex].burstTime;

processes[processIndex].remainingTime = 0;

time = processes[processIndex].completionTime;

}

}

}

void calculateTurnAroundTime(Process \*processes, int n)

{

int totalTurnAroundTime = 0;

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

{

processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;

totalTurnAroundTime += processes[i].turnAroundTime;

}

}

void calculateWaitingTime(Process \*processes, int n)

{

int totalWaitingTime = 0;

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

{

processes[i].waitingTime = processes[i].turnAroundTime - processes[i].burstTime;

totalWaitingTime += processes[i].waitingTime;

}

cout << "Average waiting time: " << (float)totalWaitingTime / n << endl;

}

int main()

{

int n;

cout << "Enter number of processes: ";

cin >> n;

Process \*processes = new Process[n];

input(processes, n);

calculateCompletionTime(processes, n);

calculateTurnAroundTime(processes, n);

calculateWaitingTime(processes, n);

sortByPid(processes, n);

display(processes, n);

return 0;

}

Text

Description automatically generated

5. Bankers

#include <iostream>

using namespace std;

int main()

{

int a = 10, b = 5, c = 7;

int suma = 0, sumb = 0, sumc = 0;

int allocation[5][3], maxneed[5][3], available[3], remaining[5][3];

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

{

cout << "Enter the allocation for process " << i << endl;

cin >> allocation[i][0] >> allocation[i][1] >> allocation[i][2];

suma += allocation[i][0];

sumb += allocation[i][1];

sumc += allocation[i][2];

}

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

{

cout << "Enter the allocation for max " << i << endl;

cin >> maxneed[i][0] >> maxneed[i][1] >> maxneed[i][2];

}

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

{

remaining[i][0] = maxneed[i][0] - allocation[i][0];

remaining[i][1] = maxneed[i][1] - allocation[i][1];

remaining[i][2] = maxneed[i][2] - allocation[i][2];

}

available[0] = a - suma;

available[1] = b - sumb;

available[2] = c - sumc;

int counter = 0;

while (1)

{

if (available[0] == a && available[1] == b && available[2] == c)

{

break;

}

if (counter == 5)

{

cout << "Deadlock\n";

break;

}

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

{

if (available[0] >= remaining[i][0] && available[1] >= remaining[i][1] && available[2] >= remaining[i][2])

{

remaining[i][0] = INT\_MAX;

remaining[i][1] = INT\_MAX;

remaining[i][2] = INT\_MAX;

cout << "Process " << i << " ";

available[0] += allocation[i][0];

available[1] += allocation[i][1];

available[2] += allocation[i][2];

}

else

{

counter++;

}

}

}

}

Text

Description automatically generated

**DISK SCHEDULING:**

1. FCFS

#include<iostream>

#include<cstdlib>

using namespace std;

void fcfs(int a[],int n,int h)

{

int d,ad=0;

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

{

d=abs(h-a[i]);

ad+=d;

h=a[i];

}

cout<<"Absolute distance= "<<ad<<endl;

cout<<"Sequence: ";

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

cout<<a[i]<<" ";

}

int main()

{

int n,h;

cout<<"Size: ";

cin>>n;

cout<<"Head: ";

cin>>h;

int \*a=new int[n];

cout<<"Enter access points: \n";

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

cin>>a[i];

fcfs(a,n,h);

delete []a;

return 0;

}

Text

Description automatically generated

1. SSTF

#include<iostream>

#include<cstdlib>

#include<vector>

using namespace std;

void sstf(int a[],int n,int h)

{

vector<int> s;

int size=n;

int ad=0;

int pos,min;

while(n>0)

{

min=abs(h-a[0]);

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

{

if(min >= abs(h-a[i]))

{

min=abs(h-a[i]);

pos=i;

}

}

ad+=min;

s.push\_back(a[pos]);

h=a[pos];

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

a[i]=a[i+1];

n--;

}

cout<<"Absolute difference= "<<ad<<endl;

cout<<"Sequence: ";

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

cout<<s[i]<<" ";

}

int main()

{

int n,h;

cout<<"Size: ";

cin>>n;

cout<<"Head: ";

cin>>h;

cout<<"Sequence:\n";

int \*a=new int[n];

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

cin>>a[i];

sstf(a,n,h);

return 0;

}

Text

Description automatically generated

1. Scan

#include <iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

vector<int> requests;

requests.push\_back(82);

requests.push\_back(170);

requests.push\_back(43);

requests.push\_back(140);

requests.push\_back(24);

requests.push\_back(16);

requests.push\_back(190);

int head = 50;

int distance = 0;

int max\_left\_index, min\_right\_index = -1;

string direction = "right";

sort(requests.begin(), requests.end());

for (int i = 0; i < requests.size(); i++)

{

if (requests[i] > head)

{

max\_left\_index = i - 1;

min\_right\_index = i;

break;

}

}

if (direction == "left")

{

if (max\_left\_index == -1)

{

distance = requests[6] - head;

}else if(min\_right\_index==-1){

distance = head;

}

else

{

distance = head + requests[6];

}

}

else

{

if (min\_right\_index == -1)

{

distance = head - requests[0];

}else if(max\_left\_index==-1){

distance = 199-head;

}

else

{

distance = (199 - head) + (199 - requests[0]);

}

}

cout << "Distance is " << distance;

}

Graphical user interface, text, application

Description automatically generated

1. Look

#include <iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

vector<int> requests;

requests.push\_back(82);

requests.push\_back(170);

requests.push\_back(43);

requests.push\_back(140);

requests.push\_back(24);

requests.push\_back(16);

requests.push\_back(190);

int head = 50;

int distance = 0;

int max\_left\_index, min\_right\_index = -1;

string direction = "right";

sort(requests.begin(), requests.end());

for (int i = 0; i < requests.size(); i++)

{

if (requests[i] > head)

{

max\_left\_index = i - 1;

min\_right\_index = i;

break;

}

}

if (direction == "left")

{

if (max\_left\_index == -1)

{

distance = requests[6] - head;

}

else

{

distance = (head-requests[0]) + (requests[6]-requests[0]);

}

}

else

{

if (min\_right\_index == -1)

{

distance = head - requests[0];

}

else

{

distance = (requests[6] - head) + (requests[6] - requests[0]);

}

}

cout << "Distance is " << distance;

}

Graphical user interface, text

Description automatically generated

1. C-look

#include <iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

vector<int> requests;

requests.push\_back(82);

requests.push\_back(170);

requests.push\_back(43);

requests.push\_back(140);

requests.push\_back(24);

requests.push\_back(16);

requests.push\_back(190);

int head = 50;

int distance = 0;

int max\_left\_index, min\_right\_index = -1;

string direction = "right";

sort(requests.begin(), requests.end());

for (int i = 0; i < requests.size(); i++)

{

if (requests[i] > head)

{

max\_left\_index = i - 1;

min\_right\_index = i;

break;

}

}

if (direction == "left")

{

if (max\_left\_index == -1)

{

distance = requests[6] - head;

}

else

{

distance = (head-requests[0]) + (requests[6]-requests[0]) + (requests[6]-requests[min\_right\_index]);

}

}

else

{

if (min\_right\_index == -1)

{

distance = head - requests[0];

}

else

{

distance = (requests[6] - head) + (requests[6]-requests[0]) + (requests[max\_left\_index]-requests[0]);

}

}

cout << "Distance is " << distance;

}

Graphical user interface, text

Description automatically generated with medium confidence