**Assignment Five**

**Scheduling Process first come First out**

*#include<bits/stdc++.h>*

*using namespace std ;*

*void solve(){*

*int n ; // number of process*

*cin >> n ;*

*int time , burstTime ;*

*vector<pair<int ,int >> processArray; // pair < arrivingTime , processNumber >*

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

*cout << "Enter the process burstTime and arriving time : \n" ;*

*cin >> burstTime >> time ;*

*processArray.push\_back(make\_pair(time , burstTime)) ;*

*}*

*double averageTime = 0; // calculate average time*

*cout << endl;*

*// sorting based on arrival time*

*sort(processArray.begin() , processArray.end()) ;*

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

*cout <<  "Start Executing the Process Number " << i+1 << "  which have burst Time equal to "  ;*

*cout << processArray[i].second << endl;*

*averageTime += processArray[i].second ; // time adding into averageTime*

*}*

*cout << "All Process are Completely Exectued\n" ;*

*cout << "Average Response Time is " << double(averageTime/n) << endl;*

*}*

*int main(){*

*int t ; // how many times cpu will be run*

*cout << "Enter How Many Process Will Come : " ;*

*cin >> t ;*

*cout << endl;*

*while(t--)solve() ;*

*return 0;*

*}*

**Scheduling process Shortes job first**

*#include<bits/stdc++.h>*

*using namespace std ;*

*// i assume all process are come on same time ... (time == 0)*

*void solve(void) ; //declaration of function*

*int main(){*

*int Task ; // number of task*

*cin >> Task ;*

*while(Task--)solve() ;*

*return 0;*

*}*

*void solve(){*

*long n , BurstTime ; // number of process , BurstTime of process*

*cin >> n ;*

*vector< pair <long , long > > storing ;*

*// takin input*

*for(long  i=1 ; i<= n ;i++){ // i represent process number*

*cin >> BurstTime ;*

*storing.push\_back(make\_pair(BurstTime , i)) ;*

*}*

*sort(storing.begin() ,storing.end()) ;// based on burst time*

*double averageSum = 0;*

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

*averageSum += storing[i].first ;// adding value into the averageSum  '*

*cout << "Execute Process number : " <<  storing[i].second << " " << "Which have Burst time " << storing[i].first << endl;*

*}*

*cout << endl;*

*cout << "Average Execution Time : "  << double(averageSum/double(n)) << endl;*

*}*

**Scheduling process Round Robin scheduling**

#include<bits/stdc++.h>

#include<**windows.ui.h**> // include to use Sleep Function

using namespace std ;

#define ll long //macro defined

// i assume all process are come on same time ... (time == 0)

void solve(void) ; //declaration of function

int main(){

  ll  Task ; // number of task

  cin >> Task ;

  while(Task--)solve() ;

  return 0;

}

**ll n , BurstTime , Priority , TimeQuanta;** // number of process

vector<pair<ll , ll  > > ProcessQueue ;

queue<ll> Process ;

void solve(){

  cout << "Enter the process Number and Time Quanta : " ;

  cin >> n >> TimeQuanta;

  // arrival timimg is same

  for(ll i = 1 ; i <= n;i++){

    cout << "\nEnter The Process BurstTime and Priority of the Process : "  ;

    cin >> BurstTime >> Priority ;

    ProcessQueue.push\_back(make\_pair(Priority,BurstTime)) ;

  }

  sort(ProcessQueue.begin() , ProcessQueue.end()) ; // sorting on reverse order

  // inserting all Element Into the Queue

  for(ll i =0 ; i < n;i++){

    Process.push(ProcessQueue[i].second) ;

  }

  cout << "\nExecuting Starting : \n" ;

  while(!Process.empty()){ // Run a Loop on a Process queue

    ll time = Process.front() ; // accessing an element

    cout << "Time of Process : " << time << endl;

    Process.pop() ; // removing an element

    if(time - TimeQuanta > 0) {

      ll newTime = time-TimeQuanta ;

      Process.push(newTime) ;

    }

    // Process sleep for a Particular Time Quanta

    ll sleepTime = TimeQuanta\*1000 ; // conert in to mili second

    Sleep(sleepTime) ; // sleep Process

  }

  cout << "CompleteTheProcess" << endl ;

}