CPU scheduling algorithms

## a) FCFS

code:

|  |
| --- |
| #include <iostream>  using namespace std;  // Function to find the waiting time for all  // processes  void findWaitingTime(int processes[], int n,                       int bt[], int wt[])  {      // waiting time for first process is 0      wt[0] = 0;      // calculating waiting time      for (int i = 1; i < n; i++)          wt[i] = bt[i - 1] + wt[i - 1];  }  // Function to calculate turn around time  void findTurnAroundTime(int processes[], int n,                          int bt[], int wt[], int tat[])  {      // calculating turnaround time by adding      // bt[i] + wt[i]      for (int i = 0; i < n; i++)          tat[i] = bt[i] + wt[i];  }  // Function to calculate average time  void findavgTime(int processes[], int n, int bt[])  {      int wt[n], tat[n], total\_wt = 0, total\_tat = 0;      // Function to find waiting time of all processes      findWaitingTime(processes, n, bt, wt);      // Function to find turn around time for all processes      findTurnAroundTime(processes, n, bt, wt, tat);      // Display processes along with all details      cout << "Processes  "           << " Burst time  "           << " Waiting time  "           << " Turn around time\n";      // Calculate total waiting time and total turn      // around time      for (int i = 0; i < n; i++)      {          total\_wt = total\_wt + wt[i];          total\_tat = total\_tat + tat[i];          cout << "   " << i + 1 << "\t\t" << bt[i] << "\t    "               << wt[i] << "\t\t  " << tat[i] << endl;      }      cout << "Average waiting time = "           << (float)total\_wt / (float)n;      cout << "\nAverage turn around time = "           << (float)total\_tat / (float)n;  }  // Driver code  int main()  {      // process id's      int processes[] = {1, 2, 3};      int n = sizeof processes / sizeof processes[0];      // Burst time of all processes      int burst\_time[] = {10, 5, 8};      findavgTime(processes, n, burst\_time);      return 0;  }  **Output**  Processes Burst time Waiting time Turn around time  1 10 0 10  2 5 10 15  3 8 15 23  Average waiting time = 8.33333  Average turn around time = 16 |

2-SJF

#include <stdio.h>

int main()

{

    int A[100][4]; // Matrix for storing Process Id, Burst

                   // Time, Average Waiting Time & Average

                   // Turn Around Time.

    int i, j, n, total = 0, index, temp;

    float avg\_wt, avg\_tat;

    printf("Enter number of process: ");

    scanf("%d", &n);

    printf("Enter Burst Time:\n");

    // User Input Burst Time and alloting Process Id.

    for (i = 0; i < n; i++) {

        printf("P%d: ", i + 1);

        scanf("%d", &A[i][1]);

        A[i][0] = i + 1;

    }

    // Sorting process according to their Burst Time.

    for (i = 0; i < n; i++) {

        index = i;

        for (j = i + 1; j < n; j++)

            if (A[j][1] < A[index][1])

                index = j;

        temp = A[i][1];

        A[i][1] = A[index][1];

        A[index][1] = temp;

        temp = A[i][0];

        A[i][0] = A[index][0];

        A[index][0] = temp;

    }

    A[0][2] = 0;

    // Calculation of Waiting Times

    for (i = 1; i < n; i++) {

        A[i][2] = 0;

        for (j = 0; j < i; j++)

            A[i][2] += A[j][1];

        total += A[i][2];

    }

    avg\_wt = (float)total / n;

    total = 0;

    printf("P     BT     WT     TAT\n");

    // Calculation of Turn Around Time and printing the

    // data.

    for (i = 0; i < n; i++) {

        A[i][3] = A[i][1] + A[i][2];

        total += A[i][3];

        printf("P%d     %d     %d      %d\n", A[i][0],

               A[i][1], A[i][2], A[i][3]);

    }

    avg\_tat = (float)total / n;

    printf("Average Waiting Time= %f", avg\_wt);

    printf("\nAverage Turnaround Time= %f", avg\_tat);

}

3-round robin

#include <iostream>

using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

                     int bt[], int wt[], int quantum)

{

    // Make a copy of burst times bt[] to store remaining

    // burst times.

    int rem\_bt[n];

    for (int i = 0; i < n; i++)

        rem\_bt[i] = bt[i];

    int t = 0; // Current time

    // Keep traversing processes in round robin manner

    // until all of them are not done.

    while (1)

    {

        bool done = true;

        // Traverse all processes one by one repeatedly

        for (int i = 0; i < n; i++)

        {

            // If burst time of a process is greater than 0

            // then only need to process further

            if (rem\_bt[i] > 0)

            {

                done = false; // There is a pending process

                if (rem\_bt[i] > quantum)

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t += quantum;

                    // Decrease the burst\_time of current process

                    // by quantum

                    rem\_bt[i] -= quantum;

                }

                // If burst time is smaller than or equal to

                // quantum. Last cycle for this process

                else

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t = t + rem\_bt[i];

                    // Waiting time is current time minus time

                    // used by this process

                    wt[i] = t - bt[i];

                    // As the process gets fully executed

                    // make its remaining burst time = 0

                    rem\_bt[i] = 0;

                }

            }

        }

        // If all processes are done

        if (done == true)

            break;

    }

}

// Function to calculate turn around time

void findTurnAroundTime(int processes[], int n,

                        int bt[], int wt[], int tat[])

{

    // calculating turnaround time by adding

    // bt[i] + wt[i]

    for (int i = 0; i < n; i++)

        tat[i] = bt[i] + wt[i];

}

// Function to calculate average time

void findavgTime(int processes[], int n, int bt[],

                 int quantum)

{

    int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

    // Function to find waiting time of all processes

    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes

    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details

    cout << "PN\t "

         << " \tBT "

         << "  WT "

         << " \tTAT\n";

    // Calculate total waiting time and total turn

    // around time

    for (int i = 0; i < n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

        cout << " " << i + 1 << "\t\t" << bt[i] << "\t "

             << wt[i] << "\t\t " << tat[i] << endl;

    }

    cout << "Average waiting time = "

         << (float)total\_wt / (float)n;

    cout << "\nAverage turn around time = "

         << (float)total\_tat / (float)n;

}

// Driver code

int main()

{

    // process id's

    int processes[] = {1, 2, 3};

    int n = sizeof processes / sizeof processes[0];

    // Burst time of all processes

    int burst\_time[] = {10, 5, 8};

    // Time quantum

    int quantum = 2;

    findavgTime(processes, n, burst\_time, quantum);

    return 0;

}

output

PN BT WT TAT

1 10 13 23

2 5 10 15

3 8 13 21

Average waiting time = 12

Average turn around time = 19.6667

4-priority

#include <bits/stdc++.h>

using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process

{

    int at, bt, pr, pno;

};

process proc[50];

/\*

Writing comparator function to sort according to priority if

arrival time is same

\*/

bool comp(process a, process b)

{

    if (a.at == b.at)

    {

        return a.pr < b.pr;

    }

    else

    {

        return a.at < b.at;

    }

}

// Using FCFS Algorithm to find Waiting time

void get\_wt\_time(int wt[])

{

    // declaring service array that stores cumulative burst time

    int service[50];

    // Initialising initial elements of the arrays

    service[0] = proc[0].at;

    wt[0] = 0;

    for (int i = 1; i < totalprocess; i++)

    {

        service[i] = proc[i - 1].bt + service[i - 1];

        wt[i] = service[i] - proc[i].at;

        // If waiting time is negative, change it into zero

        if (wt[i] < 0)

        {

            wt[i] = 0;

        }

    }

}

void get\_tat\_time(int tat[], int wt[])

{

    // Filling turnaroundtime array

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

    {

        tat[i] = proc[i].bt + wt[i];

    }

}

void findgc()

{

    // Declare waiting time and turnaround time array

    int wt[50], tat[50];

    double wavg = 0, tavg = 0;

    // Function call to find waiting time array

    get\_wt\_time(wt);

    // Function call to find turnaround time

    get\_tat\_time(tat, wt);

    int stime[50], ctime[50];

    stime[0] = proc[0].at;

    ctime[0] = stime[0] + tat[0];

    // calculating starting and ending time

    for (int i = 1; i < totalprocess; i++)

    {

        stime[i] = ctime[i - 1];

        ctime[i] = stime[i] + tat[i] - wt[i];

    }

    cout << "Process\_no\tStart\_time\tComplete\_time\tTurn\_Around\_Time\tWaiting\_Time" << endl;

    // display the process details

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

    {

        wavg += wt[i];

        tavg += tat[i];

        cout << proc[i].pno << "\t\t" << stime[i] << "\t\t" << ctime[i] << "\t\t" << tat[i] << "\t\t\t" << wt[i] << endl;

    }

    // display the average waiting time

    // and average turn around time

    cout << "Average waiting time is : ";

    cout << wavg / (float)totalprocess << endl;

    cout << "average turnaround time : ";

    cout << tavg / (float)totalprocess << endl;

}

int main()

{

    int arrivaltime[] = {1, 2, 3, 4, 5};

    int bursttime[] = {3, 5, 1, 7, 4};

    int priority[] = {3, 4, 1, 7, 8};

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

    {

        proc[i].at = arrivaltime[i];

        proc[i].bt = bursttime[i];

        proc[i].pr = priority[i];

        proc[i].pno = i + 1;

    }

    // Using inbuilt sort function

    sort(proc, proc + totalprocess, comp);

    // Calling function findgc for finding Gantt Chart

    findgc();

    return 0;

}

output

Process\_no Start\_time Complete\_time Turn\_Around\_Time Waiting\_Time

1 1 4 3 0

2 5 10 8 3

3 4 5 2 1

4 10 17 13 6

5 17 21 16 12

Average Waiting Time is : 4.4

Average Turn Around time is : 8.4