Shortest Job First

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

struct process\_struct

{

    int pid;

    int at;

    int bt;

    int ct, wt, tat, rt, start\_time;

} ps[20];

int findmax(int a, int b)

{

    return a > b ? a : b;

}

int findmin(int a, int b)

{

    return a < b ? a : b;

}

int main()

{

    int n;

    bool is\_completed[100] = {false}, is\_first\_process = true;

    int current\_time = 0;

    int completed = 0;

    printf("Enter total number of processes: ");

    scanf("%d", &n);

    int sum\_tat = 0, sum\_wt = 0, sum\_rt = 0, total\_idle\_time = 0, prev = 0, length\_cycle;

    float cpu\_utilization;

    int max\_completion\_time, min\_arrival\_time;

    for (int i = 0; i < n; i++)

    {

        printf("\nEnter Process %d Arrival Time: ", i);

        scanf("%d", &ps[i].at);

        ps[i].pid = i;

    }

    for (int i = 0; i < n; i++)

    {

        printf("\nEnter Process %d Burst Time: ", i);

        scanf("%d", &ps[i].bt);

    }

    while (completed != n)

    {

        int min\_index = -1;

        int minimum = INT\_MAX;

        for (int i = 0; i < n; i++)

        {

            if (ps[i].at <= current\_time && is\_completed[i] == false)

            {

                if (ps[i].bt < minimum)

                {

                    minimum = ps[i].bt;

                    min\_index = i;

                }

                if (ps[i].bt == minimum)

                {

                    if (ps[i].at < ps[min\_index].at)

                    {

                        minimum = ps[i].bt;

                        min\_index = i;

                    }

                }

            }

        }

        if (min\_index == -1)

        {

            current\_time++;

        }

        else

        {

            ps[min\_index].start\_time = current\_time;

            ps[min\_index].ct = ps[min\_index].start\_time + ps[min\_index].bt;

            ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at;

            ps[min\_index].wt = ps[min\_index].tat - ps[min\_index].bt;

            ps[min\_index].rt = ps[min\_index].wt;

            sum\_tat += ps[min\_index].tat;

            sum\_wt += ps[min\_index].wt;

            sum\_rt += ps[min\_index].rt;

            total\_idle\_time += (is\_first\_process == true) ? 0 : (ps[min\_index].start\_time - prev);

            completed++;

            is\_completed[min\_index] = true;

            current\_time = ps[min\_index].ct;

            prev = current\_time;

            is\_first\_process = false;

        }

    }

*// Output*

    printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");

    for (int i = 0; i < n; i++)

        printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n", ps[i].pid, ps[i].at, ps[i].bt, ps[i].ct, ps[i].tat, ps[i].wt, ps[i].rt);

    printf("\n");

    printf("\nAverage Turn Around time= %f ", (float)sum\_tat / n);

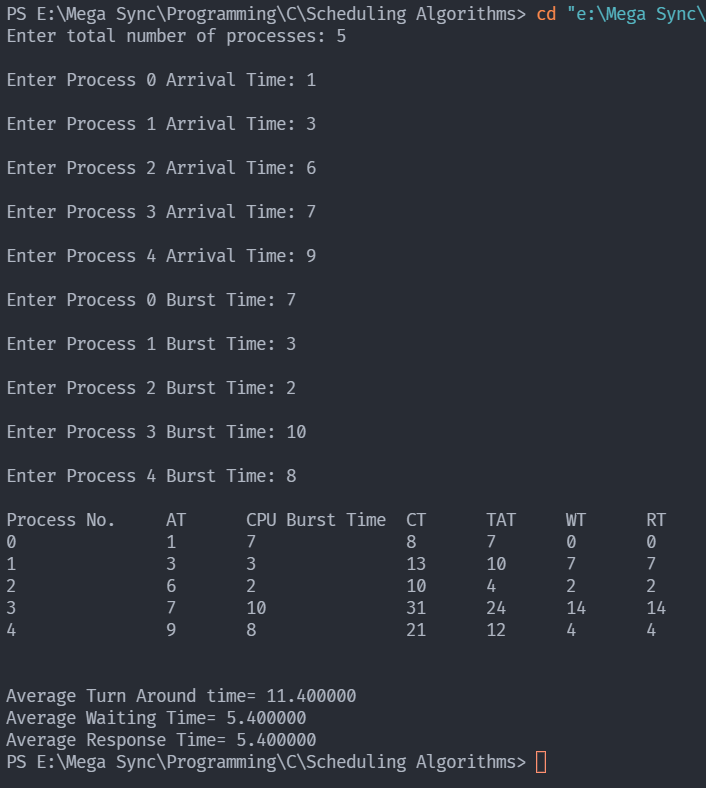
    printf("\nAverage Waiting Time= %f ", (float)sum\_wt / n);

    printf("\nAverage Response Time= %f ", (float)sum\_rt / n);

    return 0;

}

Output



Shortest Remaining Time First

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

struct process\_struct

{

    int pid;

    int at;

    int bt;

    int ct, wt, tat, rt, start\_time;

} ps[100];

int findmax(int a, int b)

{

    return a > b ? a : b;

}

int findmin(int a, int b)

{

    return a < b ? a : b;

}

int main()

{

    int n;

    float bt\_remaining[100];

    bool is\_completed[100] = {false}, is\_first\_process = true;

    int current\_time = 0;

    int completed = 0;

    ;

    float sum\_tat = 0, sum\_wt = 0, sum\_rt = 0, total\_idle\_time = 0, length\_cycle, prev = 0;

    float cpu\_utilization;

    int max\_completion\_time, min\_arrival\_time;

    printf("Enter total number of processes: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++)

    {

        printf("\nEnter Process %d Arrival Time: ", i);

        scanf("%d", &ps[i].at);

        ps[i].pid = i;

    }

    for (int i = 0; i < n; i++)

    {

        printf("\nEnter Process %d Burst Time: ", i);

        scanf("%d", &ps[i].bt);

        bt\_remaining[i] = ps[i].bt;

    }

    while (completed != n)

    {

        int min\_index = -1;

        int minimum = INT\_MAX;

        for (int i = 0; i < n; i++)

        {

            if (ps[i].at <= current\_time && is\_completed[i] == false)

            {

                if (bt\_remaining[i] < minimum)

                {

                    minimum = bt\_remaining[i];

                    min\_index = i;

                }

                if (bt\_remaining[i] == minimum)

                {

                    if (ps[i].at < ps[min\_index].at)

                    {

                        minimum = bt\_remaining[i];

                        min\_index = i;

                    }

                }

            }

        }

        if (min\_index == -1)

        {

            current\_time++;

        }

        else

        {

            if (bt\_remaining[min\_index] == ps[min\_index].bt)

            {

                ps[min\_index].start\_time = current\_time;

                total\_idle\_time += (is\_first\_process == true) ? 0 : (ps[min\_index].start\_time - prev);

                is\_first\_process = false;

            }

            bt\_remaining[min\_index] -= 1;

            current\_time++;

            prev = current\_time;

            if (bt\_remaining[min\_index] == 0)

            {

                ps[min\_index].ct = current\_time;

                ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at;

                ps[min\_index].wt = ps[min\_index].tat - ps[min\_index].bt;

                ps[min\_index].rt = ps[min\_index].start\_time - ps[min\_index].at;

                sum\_tat += ps[min\_index].tat;

                sum\_wt += ps[min\_index].wt;

                sum\_rt += ps[min\_index].rt;

                completed++;

                is\_completed[min\_index] = true;

            }

        }

    }

    max\_completion\_time = INT\_MIN;

    min\_arrival\_time = INT\_MAX;

    for (int i = 0; i < n; i++)

    {

        max\_completion\_time = findmax(max\_completion\_time, ps[i].ct);

        min\_arrival\_time = findmin(min\_arrival\_time, ps[i].at);

    }

    length\_cycle = max\_completion\_time - min\_arrival\_time;

    printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n");

    for (int i = 0; i < n; i++)

        printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n", ps[i].pid, ps[i].at, ps[i].bt, ps[i].ct, ps[i].tat, ps[i].wt, ps[i].rt);

    printf("\n");

    cpu\_utilization = (float)(length\_cycle - total\_idle\_time) / length\_cycle;

    printf("\nAverage Turn Around time= %f ", (float)sum\_tat / n);

    printf("\nAverage Waiting Time= %f ", (float)sum\_wt / n);

    printf("\nAverage Response Time= %f ", (float)sum\_rt / n);

    printf("\nThroughput= %f", n / (float)length\_cycle);

    printf("\nCPU Utilization(Percentage)= %f", cpu\_utilization \* 100);

    return 0;

}

Output

