Lab Assignment 11

# **Round Robin Scheduling**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#include <stdlib.h>

struct process\_struct

{

    int pid;

    int at;

    int bt;

    int ct, wt, tat, rt, start\_time;

    int bt\_remaining;

} ps[100];

int pfline(int num)

{

    for (int i = 0; i < num; i++)

    {

        printf("-");

    }

    printf("\n");

}

int findmax(int a, int b)

{

    return a > b ? a : b;

}

int comparatorAT(const void \*a, const void \*b)

{

    int x = ((struct process\_struct \*)a)->at;

    int y = ((struct process\_struct \*)b)->at;

    if (x < y)

        return -1;

    else if (x >= y)

        return 1;

}

int comparatorPID(const void \*a, const void \*b)

{

    int x = ((struct process\_struct \*)a)->pid;

    int y = ((struct process\_struct \*)b)->pid;

    if (x < y)

        return -1;

    else if (x >= y)

        return 1;

}

int main()

{

    int n, index;

    int cpu\_utilization;

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

    int current\_time = 0, max\_completion\_time;

    int completed = 0, tq, total\_idle\_time = 0, length\_cycle;

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

    scanf("%d", &n);

    int queue[100], front = -1, rear = -1;

    float sum\_tat = 0, sum\_wt = 0, sum\_rt = 0;

    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);

        ps[i].bt\_remaining = ps[i].bt;

    }

    printf("\nEnter time quantum: ");

    scanf("%d", &tq);

    qsort((void \*)ps, n, sizeof(struct process\_struct), comparatorAT);

    front = rear = 0;

    queue[rear] = 0;

    visited[0] = true;

    while (completed != n)

    {

        index = queue[front];

        front++;

        if (ps[index].bt\_remaining == ps[index].bt)

        {

            ps[index].start\_time = findmax(current\_time, ps[index].at);

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

            current\_time = ps[index].start\_time;

            is\_first\_process = false;

        }

        if (ps[index].bt\_remaining - tq > 0)

        {

            ps[index].bt\_remaining -= tq;

            current\_time += tq;

        }

        else

        {

            current\_time += ps[index].bt\_remaining;

            ps[index].bt\_remaining = 0;

            completed++;

            ps[index].ct = current\_time;

            ps[index].tat = ps[index].ct - ps[index].at;

            ps[index].wt = ps[index].tat - ps[index].bt;

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

            sum\_tat += ps[index].tat;

            sum\_wt += ps[index].wt;

            sum\_rt += ps[index].rt;

        }

        for (int i = 1; i < n; i++)

        {

            if (ps[i].bt\_remaining > 0 && ps[i].at <= current\_time && visited[i] == false)

            {

                queue[++rear] = i;

                visited[i] = true;

            }

        }

        if (ps[index].bt\_remaining > 0)

            queue[++rear] = index;

        if (front > rear)

        {

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

            {

                if (ps[i].bt\_remaining > 0)

                {

                    queue[rear++] = i;

                    visited[i] = true;

                    break;

                }

            }

        }

    }

    max\_completion\_time = INT\_MIN;

    for (int i = 0; i < n; i++)

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

    length\_cycle = max\_completion\_time - ps[0].at;

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

    qsort((void \*)ps, n, sizeof(struct process\_struct), comparatorPID);

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

    pfline(90);

    for (int i = 0; i < n; i++)

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

    printf("\n");

    printf("\nAverage Turn Around time= %.2f", (float)sum\_tat / n);

    printf("\nAverage Waiting Time= %.2f", (float)sum\_wt / n);

    printf("\nAverage Response Time= %.2f", (float)sum\_rt / n);

    return 0;

}

Output

