

OS LAB -7

• Name :HITU RAJ

• Roll no. :2005025

• Branch :CSE

MULTILEVEL QUEUE

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct process

{

    int priority;

    int burst\_time;

    int tt\_time;

    int total\_time;

};

struct queues

{

    int priority\_start;

    int priority\_end;

    int total\_time;

    int length;

    struct process \*p;

    int executed;

};

int notComplete(struct queues q[])

{

    int a = 0;

    int countInc = 0;

    for (int i = 0; i < 3; i++)

    {

        countInc = 0;

        for (int j = 0; j < q[i].length; j++)

        {

            if (q[i].p[j].burst\_time != 0)

            {

                a = 1;

            }

            else

            {

                countInc += 1;

            }

        }

        if (countInc == q[i].length)

        {

            q[i].executed = 1;

        }

    }

    return a;

}

void sort\_ps(struct queues q)

{ // Queue q has to be sorted according to priority of processes

    for (int i = 1; i < q.length; i++)

    {

        for (int j = 0; j < q.length - 1; j++)

        {

            if (q.p[j].priority < q.p[j + 1].priority)

            {

                struct process temp = q.p[j + 1];

                q.p[j + 1] = q.p[j];

                q.p[j] = temp;

            }

        }

    }

}

void checkCompleteTimer(struct queues q[])

{

    int a = notComplete(q);

    for (int i = 0; i < 3; i++)

    {

        if (q[i].executed == 0)

        {

            for (int j = 0; j < q[i].length; j++)

            {

                if (q[i].p[j].burst\_time != 0)

                {

                    q[i].p[j].total\_time += 1;

                }

            }

            q[i].total\_time += 1;

        }

    }

}

int main()

{ // Initializing 3 queues

    struct queues q[3];

    for (int i = 0; i < 3; i++)

    {

        q[i].total\_time = 0;

        q[i].length = 0;

    }

    q[0].priority\_start = 7;

    q[0].priority\_end = 9;

    q[1].priority\_start = 4;

    q[1].priority\_end = 6;

    q[2].priority\_start = 1;

    q[2].priority\_end = 3;

    q[0].executed = q[1].executed = q[2].executed = 0;

    int no\_of\_processes, priority\_of\_process, burst\_time\_of\_process; // Entering Processes and assigning it to respective queues.

    printf("Enter the number of processes\n");

    scanf("%d", &no\_of\_processes);

    struct process p1[no\_of\_processes];

    for (int i = 0; i < no\_of\_processes; i++)

    {

        p1[i].total\_time = 0;

        printf("Enter the priority of the process\n");

        scanf("%d", &priority\_of\_process);

        printf("Enter the burst time of the process\n");

        scanf("%d", &burst\_time\_of\_process);

        p1[i].priority = priority\_of\_process;

        p1[i].burst\_time = burst\_time\_of\_process;

        p1[i].tt\_time = burst\_time\_of\_process;

        for (int j = 0; j < 3; j++)

        {

            if (q[j].priority\_start <= priority\_of\_process && priority\_of\_process <= q[j].priority\_end)

            {

                q[j].length++;

            }

        }

    }

    for (int i = 0; i < 3; i++)

    {

        int len = q[i].length;

        q[i].p = malloc(len \* sizeof(struct process));

    }

    int a = 0;

    int b = 0;

    int c = 0;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < no\_of\_processes; j++)

        {

            if ((q[i].priority\_start <= p1[j].priority) && (p1[j].priority <= q[i].priority\_end))

            {

                if (i == 0)

                {

                    q[i].p[a++] = p1[j];

                }

                else if (i == 1)

                {

                    q[i].p[b++] = p1[j];

                }

                else

                {

                    q[i].p[c++] = p1[j];

                }

            }

        }

    }

    a--;

    b--;

    c--;

    for (int i = 0; i < 3; i++)

    {

        printf("Queue %d : \t", i + 1);

        for (int j = 0; j < q[i].length; j++)

        {

            printf("%d ->", q[i].p[j].priority);

        }

        printf("NULL\n");

    } // While RR on multiple queues is not complete, keep on repeating

    int timer = 0;

    int l = -1;

    int rr\_timer = 4;

    int counter = 0;

    int counterps = 0;

    int counterfcfs = 0;

    while (notComplete(q))

    {

        if (timer == 10)

        {

            timer = 0;

        }

        l += 1;

        if (l >= 3)

        {

            l = l % 3;

        } // Process lth queue if its already not executed // If its executed change the value of l

        if (q[l].executed == 1)

        {

            printf("Queue %d completed\n", l + 1);

            l += 1;

            if (l >= 3)

            {

                l = l % 3;

            }

            continue;

        } // Finally you now have a queue which is not completely executed // Process the incomplete processes over it

        if (l == 0)

        {

            printf("Queue %d in hand\n", l + 1); // Round Robin Algorithm for q=4

            if (rr\_timer == 0)

            {

                rr\_timer = 4;

            }

            for (int i = 0; i < q[l].length; i++)

            {

                if (q[l].p[i].burst\_time == 0)

                {

                    counter++;

                    continue;

                }

                if (counter == q[l].length)

                {

                    break;

                }

                while (rr\_timer > 0 && q[l].p[i].burst\_time != 0 && timer != 10)

                {

                    printf("Executing queue 1 and %d process for a unit time. Process has priority of %d\n", i + 1, q[l].p[i].priority);

                    q[l].p[i].burst\_time--;

                    checkCompleteTimer(q);

                    rr\_timer--;

                    timer++;

                }

                if (timer == 10)

                {

                    break;

                }

                if (q[l].p[i].burst\_time == 0 && rr\_timer == 0)

                {

                    rr\_timer = 4;

                    if (i == (q[i].length - 1))

                    {

                        i = -1;

                    }

                    continue;

                }

                if (q[l].p[i].burst\_time == 0 && rr\_timer > 0)

                {

                    if (i == (q[i].length - 1))

                    {

                        i = -1;

                    }

                    continue;

                }

                if (rr\_timer <= 0)

                {

                    rr\_timer = 4;

                    if (i == (q[i].length - 1))

                    {

                        i = -1;

                    }

                    continue;

                }

            }

        }

        else if (l == 1)

        {

            printf("Queue %d in hand\n", l + 1);

            sort\_ps(q[l]); // Priority Scheduling

            for (int i = 0; i < q[l].length; i++)

            {

                if (q[l].p[i].burst\_time == 0)

                {

                    counterps++;

                    continue;

                }

                if (counterps == q[l].length)

                {

                    break;

                }

                while (q[l].p[i].burst\_time != 0 && timer != 10)

                {

                    printf("Executing queue 2 and %d process for a unit time. Process has priority of %d\n", i + 1, q[l].p[i].priority);

                    q[l].p[i].burst\_time--;

                    checkCompleteTimer(q);

                    timer++;

                }

                if (timer == 10)

                {

                    break;

                }

                if (q[l].p[i].burst\_time == 0)

                {

                    continue;

                }

            }

        }

        else

        {

            printf("Queue %d in hand\n", l + 1); // FCFS

            for (int i = 0; i < q[l].length; i++)

            {

                if (q[l].p[i].burst\_time == 0)

                {

                    counterfcfs++;

                    continue;

                }

                if (counterfcfs == q[l].length)

                {

                    break;

                }

                while (q[l].p[i].burst\_time != 0 && timer != 10)

                {

                    printf("Executing queue 3 and %d process for a unit time. Process has priority of %d\n", i + 1, q[l].p[i].priority);

                    q[l].p[i].burst\_time--;

                    checkCompleteTimer(q);

                    timer++;

                }

                if (timer == 10)

                {

                    break;

                }

                if (q[l].p[i].burst\_time == 0)

                {

                    continue;

                }

            }

        }

        printf("Broke from queue %d\n", l + 1);

    }

    for (int i = 0; i < 3; i++)

    {

        printf("\nTime taken for queue %d to execute: %d\n", i + 1, q[i].total\_time);

        for (int j = 0; j < q[i].length; j++)

        {

            printf("Process %d of queue %d took %d\n", j + 1, i + 1, q[i].p[j].total\_time);

        }

    }

    int sum\_tt = 0;

    int sum\_wt = 0;

    printf("\n\nProcess | Turn Around Time | Waiting Time\n");

    for (int i = 0; i < 3; i++)

    {

        printf("Queue %d\n", i + 1);

        for (int j = 0; j < q[i].length; j++)

        {

            printf("Process P %d\t %d\t\t %d\n", j + 1, q[i].p[j].total\_time, q[i].p[j].total\_time - q[i].p[j].tt\_time);

            sum\_tt += q[i].p[j].total\_time;

            sum\_wt += q[i].p[j].total\_time - q[i].p[j].tt\_time;

        }

    }

    printf("\n The average turnaround time is : %d\n", sum\_tt / no\_of\_processes);

    printf("\n The average waiting time is : %d\n", sum\_wt / no\_of\_processes);

    return 0;

}

OUTPUT

