KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

OS LAB -7

- Name :HITU RAJ
- Roll no. :2005025
- Branch :CSE

2005025_Hitu raj

MULTILEVEL QUEUE

```c
#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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\my codes\OSLAB\lab7_multi level queue> cd "d:\my codes\OSLAB\lab7_multi level queue\" ; if ($?) { gcc multilevel.c -o multilevel } ; if ($?) {
.\multilevel }
Enter the number of processes
5
Enter the priority of the process
5
Enter the burst time of the process
4
Enter the priority of the process
3
Enter the burst time of the process
3
Enter the priority of the process
1
Enter the burst time of the process
8
Enter the priority of the process
2
Enter the burst time of the process
5
Enter the priority of the process
6
Enter the burst time of the process
3
Queue 1 :       NULL
Queue 2 :       5 ->6 ->NULL
Queue 3 :       3 ->1 ->2 ->NULL
Queue 1 completed
Queue 3 in hand
Executing queue 3 and 1 process for a unit time. Process has priority of 3
Executing queue 3 and 1 process for a unit time. Process has priority of 3
Executing queue 3 and 1 process for a unit time. Process has priority of 3
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Broke from queue 3
Queue 1 completed
Queue 3 in hand
Executing queue 3 and 2 process for a unit time. Process has priority of 1
Executing queue 3 and 3 process for a unit time. Process has priority of 2
Executing queue 3 and 3 process for a unit time. Process has priority of 2
Executing queue 3 and 3 process for a unit time. Process has priority of 2
Executing queue 3 and 3 process for a unit time. Process has priority of 2
Executing queue 3 and 3 process for a unit time. Process has priority of 2
Broke from queue 3
Queue 1 completed
Queue 3 completed
Queue 2 in hand
Executing queue 2 and 1 process for a unit time. Process has priority of 6
Executing queue 2 and 1 process for a unit time. Process has priority of 6
Time taken for queue 2 to execute: 22
Process 1 of queue 2 took 18
Process 2 of queue 2 took 22

Time taken for queue 3 to execute: 15
Process 1 of queue 3 took 2
Process 2 of queue 3 took 10
Process 3 of queue 3 took 15


 Process | Turn Around Time | Waiting Time
 Queue 1
 Queue 2
 Process P 1      18              15
 Process P 2      22              18
 Queue 3
 Process P 1      2               -1
 Process P 2      10              2
 Process P 3      15              10

  The average turnaround time is : 13

  The average waiting time is : 8
PS D:\my codes\OSLAB\lab7_multi level queue>
```

2005205 Hitu Raj