OS Group Mini Project

CSE 301: Operating Systems

Topic -: Develop a CPU scheduling algorithm for implementing the multi feedback queue.

Submitted by-: AP19110010499 - Harika
AP19110010503 - Sandeep Reddy
AP19110010506 - venkata Manas
AP19110010512 - Koushik
AP19110010513 - Avinash Reddy

Group Number:12

Date of Submission-: 15-06-2021

Submitted to-: Ravi Kant Kumar sir

Acknowledgment:

We, members of (group no:12) would like to express our gratitude to Ravi Kant Kumar sir for encouraging us throughout the semester because of that we are able to do this project.
Due to this project, we were able to do a lot of research and it also helped us to understand the concepts of “Develop a CPU scheduling algorithm for implementing the multi feedback queue” to a greater depth.

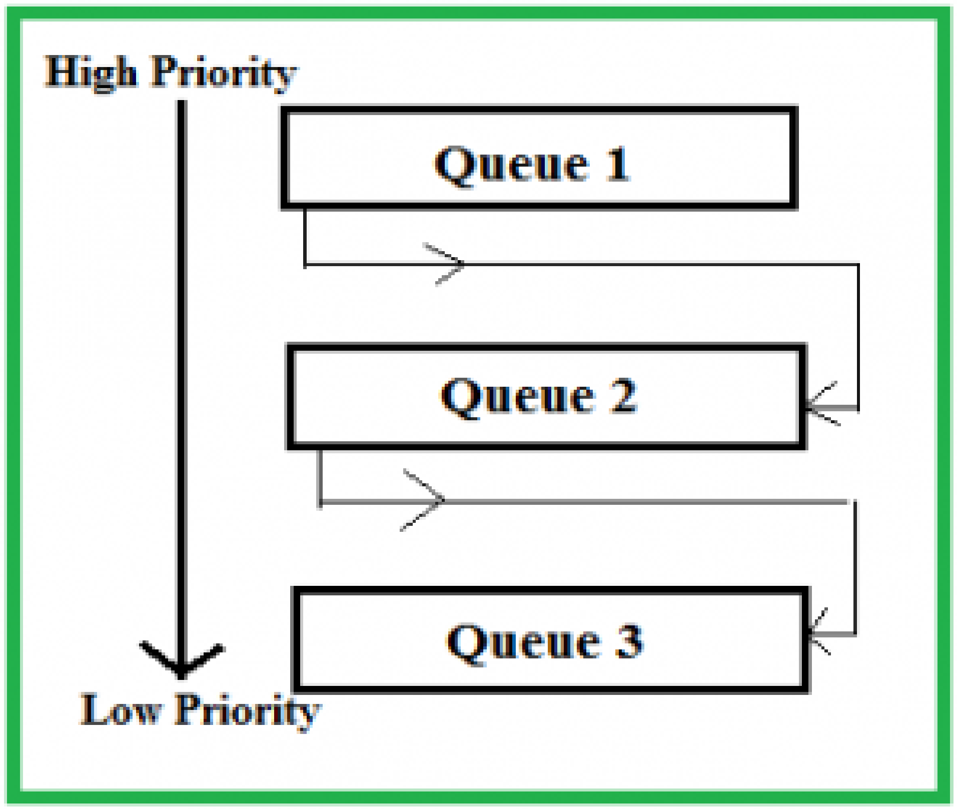
Index

Abstract:

In this project, we have to do coding for “Develop a CPU scheduling algorithm for implementing the multi feedback queue.”

We have developed this code using C language in this we need to do cpu Scheduling is like Multilevel Queue (MLQ) Scheduling but in this process can move between the queues. Multilevel Feedback Queue Scheduling keeps analyzing the behavior (time of execution) of processes and according to which it changes its priority.

Overview:



Process:

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

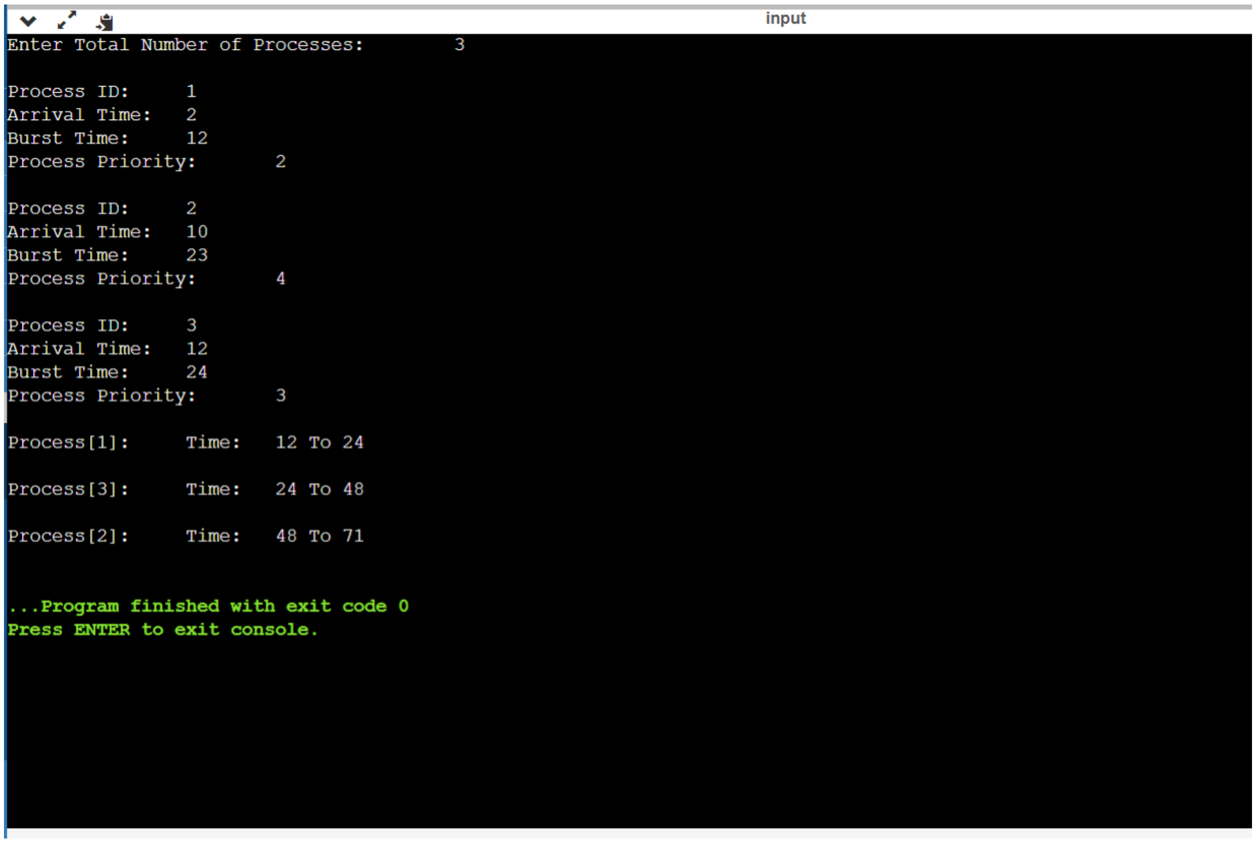
- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

CODE-:

```
#include<stdio.h>

#define N 10

typedef struct
{
    int process_id, arrival_time, burst_time, priority;
    int q, ready;
}process_structure;

int Queue(int t1)
{
    if(t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
    {
        return 1;
    }
    else
    {
        return 2;
    }
}

int main()
{
    int limit, count, temp_process, time, j, y;
    process_structure temp;
    printf("Enter Total Number of Processes:\t");
    scanf("%d", &limit);
    process_structure process[limit];
    for(count = 0; count < limit; count++)
    {
        printf("\nProcess ID:\t");
        scanf("%d", &process[count].process_id);
    printf("Arrival Time:\t");
    scanf("%d", &process[count].arrival_time);
    printf("Burst Time:\t");
    scanf("%d", &process[count].burst_time);
    printf("Process Priority:\t");
    scanf("%d", &process[count].priority);
    temp_process = process[count].priority;
    process[count].q = Queue(temp_process);
    process[count].ready = 0;
}
time = process[0].burst_time;
for(y = 0; y < limit; y++)
{
    for(count = y; count < limit; count++)
    {
        if(process[count].arrival_time < time)
        {
            process[count].ready = 1;
        }
    }
    for(count = y; count < limit - 1; count++)
    {
        for(j = count + 1; j < limit; j++)
        {
            if(process[count].ready == 1 && process[j].ready == 1)
            {
                if(process[count].q == 2 && process[j].q == 1)
                {
                    temp = process[count];
                    process[count] = process[j];
                    process[j] = temp;
                }
            }
        }
    }
```

```
        for(count = y; count < limit - 1; count++)
        {
            for(j = count + 1; j < limit; j++)
            {
                if(process[count].ready == 1 && process[j].ready == 1)
                {
                    if(process[count].q == 1 && process[j].q == 1)
                    {
                        if(process[count].burst_time > process[j].burst_time)
                        {
                            temp = process[count];
                            process[count] = process[j];
                            process[j] = temp;
                        }
                        else
                        {
                            break;
                        }
                    }
                }
            }
        }
        printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[y].process_id,
time, time + process[y].burst_time);
        time = time + process[y].burst_time;
        for(count = y; count < limit; count++)
        {
            if(process[count].ready == 1)
            {
                process[count].ready = 0;
            }
        }
    }
    return 0;
}
```

Output -:



```
Enter Total Number of Processes:        3

Process ID:     1
Arrival Time:   2
Burst Time:     12
Process Priority:       2

Process ID:     2
Arrival Time:   10
Burst Time:     23
Process Priority:       4

Process ID:     3
Arrival Time:   12
Burst Time:     24
Process Priority:       3

Process[1]:     Time:   12 To 24

Process[3]:     Time:   24 To 48

Process[2]:     Time:   48 To 71


...Program finished with exit code 0
Press ENTER to exit console.
```

<u>CONCLUSION-:</u>It is more adaptable than multilayer queueing. To improve turnaround times, algorithms such as SJF are required, which schedule operations based on their running time. However, the process's duration is unknown in advance. MFQS executes a process for a time quantum before changing the priority. As a result, it learns from the process's previous behaviour and then predicts its future behaviour. This optimizes turnaround time by attempting to run a shorter process first. The response time is also reduced using MFQS.

ADVANTAGES OF Multilevel Feedback Queue Scheduling (MLFQ):

It is more flexible. It allows different processes to move between different queues. It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.

References-:

1.Class slides
2.Research in Internet
3. Jeffrey D Ullman, 2nd Edition, Pearson Education, 2007.