Task States, Triggers and Timeline of New Multi-Level Feedback Queue Scheduler

M.V. Panduranga Rao¹, K.C. Shet²
¹Research Scholar, NITK, Surathkal, Mangalore, India
²Professor, NITK, Surathkal, Mangalore, India
¹raomvp@yahoo.com; ²kcshet@rediffmail.com

Abstract — We are articulating the task states of New Multi Level Feedback Queue [NMLFQ] Scheduler in this research paper. The contingent of task transitions with triggers which leads to a change of state is depicted. In real time scenario, the literal time line of real time process, with time instants and intervals are elucidated.

Keywords — NMLFQ, scheduler, states, triggers, process, time line, queue, deadline, preemption, priority, Multi Level, transition diagram and execution time

I. THE STATE TRANSITION DIAGRAM OF PROCESS

All implementations in NMLFQ are achieved using tasks or threads. In the transition each task can have one of the seven different states: new, ready, running, wait, blocked, zombie and succeeded state [2][5][10][14]. When the tasks are created, its state is said to be in new state. Tasks which are obtainable for running are said to be in ready state. Existing job is started to execute in running state [1][32][43][49]. Waiting tasks are the one waiting for a resource to be accessible. If a task is sleeping for certain predefined amount of time then it is said to be in blocked state. The zombie process is one, which is unable to fulfill its job, as it is killed due to uncertain event. The fairly completed task reaches the succeeded state of process [7][13][21][54].

The state transition diagram of process is described with the possible states in fig. 1. The transition of process must be accounted for all the states of a real time application [27][33][44][51].

The literal pseudo code of NMLFQ task state is provided as below.

METHOD BEGIN void processing()
    MONITOR BEGIN
        int i=0;
        WHILE BEGIN (pQueue.isEmpty())
            ExecutingProcess(pQueue.get(i++),"wait");
        END
    END
    CATCH BEGIN(Exception ex)
```java
ex.printStackTrace();
CATCH END
METHOD END

METHOD BEGIN void ExecutingProcess(Process process, String state)
// Actual Execution happens
IF BEGIN (state.equalsIgnoreCase("wait") || state.equalsIgnoreCase("Ready") ||
state.equalsIgnoreCase("DuringEvent"))
System.out.println("Execution started...");
IF END
ELSE BEGIN
System.out.println("Cannot execute process as its not in proper state to execute");
ELSE END
METHOD END

Fig. 1 The state transition diagram of process, representing the execution of an entire task, to be invoked through several states at dissimilar stages.
```
The object oriented programming concept is adopted in the developmental approach of New Multi Level Feedback Queue Scheduler [28][31][37][45][52]. The object oriented way of pseudo-code, which is almost similar to actual code, is explained in brief as follows.

```java
public class TaskStates {
    Vector<Process> readyQueue = new Vector();
    Hashtable<Integer, Process> pQueue = new Hashtable<Integer, Process>();
    public Process p = null;
    public Process removedProcess = null;
    public TaskStates(){
        // initially loading Queue with few number of processes.
        try{
            for(int i=1;i<=pQueue.size();i++){
                // Creating process
                p=Runtime.getRuntime().exec("ps -ef");
                insertIntoPQueue(i,p);
            }
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    public TaskStates(int i){
        try{
            // removing the requested number of processes from queue.
            if(i<10){
                for(int j=0;j<i;j++){
                    removedProcess = removeAtEnd(j);
                    if(removedProcess != null){
                        System.out.println("process is removed from queue");
                    }else{
                        System.out.println("process removed from the queue is null");
                    }
                }
            }else{
                addAtFront(i, Runtime.getRuntime().exec("ps -ef"));
            }
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    /* The following method is used to add the process to the ready-Queue based on Task-States.
    * It will check the Task-States and moves the other processes in ready-Queue accordingly. */
    public void addAtFront(int priority, Process temp){
        Enumeration enumitr = pQueue.keys();
        while(enumitr.hasMoreElements(){
            if(!(priority <(Integer)enumitr.nextElement())){
                pQueue.put(priority, temp);
                readyQueue.add(temp);
            }
        }
    }

    /* The following method, will insert some processes initially into queue and also for the ready-Queue */
    public void insertIntoPQueue(int priority, Process process){
        // adding process to the Queue and also into ready-Queue
    }
```
if(process != null){
pQueue.put(priority, process);
    readyQueue.addAll(pQueue.values());
    System.out.println("process is added into Ready Queue");
} else{
    System.out.println("process is null");
}

/* The following method, removes the process from the queue. */
public Process removeAtEnd(int index){
    // removing the process from the index specified from both ready-Queue and pQueue
    readyQueue.remove(index);
    return(pQueue.remove(index));
}

/* The following method, will allocate the CPU based on the priority of the processes. */
public void processing(){
    try{
        int i=0;
        while(pQueue.isEmpty()){
            ExecutingProcess(pQueue.get(i++),"wait");
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
}

/* Now the actual execution will start. */
public void ExecutingProcess(Process process,String state){
    //Actual Execution happens
    if(state.equalsIgnoreCase("wait") || state.equalsIgnoreCase("Ready") || state.equalsIgnoreCase("DuringEvent"))
        System.out.println("Execution started...");
    else
        System.out.println("Cannot execute process as it’s not in proper state to execute");
}

/* Here, we are creating pQueue as a placeholder for processes which are ready to execute.
However, as it is Priority-Queue based on task-states, we have to use HashTable for storing
both priority and processes. Processes will be added based on the priority to the Queue and will removed from
the end or based on priority. Correspondingly the task states are restored. */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    // creation of Queue
    new TaskStates();
    //Removing processes from Queue
    new TaskStates(2);
    new TaskStates(110);
    try {
        new TaskStates().addAtFront(2,Runtime.getRuntime().exec("ps -ef");
        new TaskStates().processing();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        }
II. TRANSITIONS WITH TRIGGERS WHICH LEADS TO A CHANGE OF STATE

![State diagram of process transitions](image)

Fig. 2 Task comprising of total period and execution time of a process. The state transitions along with the triggers for each of the state are represented.

The total period of process corresponds to, utilization time of CPU within a fixed amount of time. Sometimes it gets succeeded within deadline, consuming less execution time [6][11][18][23][40]. The earlier completion of task guarantees, to yield the output precisely, at the expected response time. Fig. 2 shows, the task consisting of total period comprising of execution and deadline time. Process arrives at t=0, this is also called the release time of process. The process starts waiting in the ready queue [36][39][53]. At t=1 millisecond, process gets admitted. The process elected from ready pool of processes. The process with its unique process identification number, [pid] starts to run at t=1 msec. On the occurrence of any event, such as Input/Output, [I/O] process is suspended to wait or blocked in anticipation of free resource. It sometimes sleeps to extend a delay time. The process resumes at t=3 msecs and succeeds at t=4 msecs, before the fixed deadline of process at t=5 msecs. The illustrations of start, suspend, wait, resume and succeeded are designated in fig. 2. The states of process for the execution of whole process, exploring through every state is shown in fig. 1.
Fig. 3 Preemption of a low priority process by the high priority process occurred at each trigger in a real time kernel, depending on the decision of scheduler.

Preemption of a low priority process by the high priority process occurred at each trigger in a real time kernel, depending on the decision of scheduler is shown in fig. 3. Initially process P1 is started [5][8][26][30][35]. It is suspended by kernel at $t=1$ msec in order to run medium priority process. Now the suspended process P1 waits till $t=5$ msecs. The medium priority process continues execution till $t=2$ msecs. The high priority process P3 preempts the medium priority process now to lock a processor peripheral [3][29][34][38][54]. It finds the peripheral is already locked by another process. Henceforth it suspends itself to give access for medium priority process to run at $t=3$ msecs. The process P2 runs and unlocks the peripheral at $t=4$ msecs. The process P3 is started again by kernel and completes the task at $t=5$ msecs as the peripheral is available.

III. TIME LINE OF REAL TIME PROCESS WITH TIME INSTANTS AND INTERVALS

Fig. 4 Representation of time line of real time process.
Fig. 4, represents the timeline of real time process depicting the deadline, response and execution time of process [15][19][25][47]. Several processes arrive and get CPU time, in accordance to the decision of scheduler. The top row shows the schedule and completion instance of processes. The processes P1, P2 and P3, arrives with increasing priority. The high priority process P3 pre-empts at t=1msec. The process P2 waits for a duration from t=1 to t=2 milliseconds. It completes immediately after starting at t=2 msecs. The arrow marks often shows several pre-emption points. A few deadline points for each process are also depicted in fig. 4.

IV. CONCLUSIONS

In this research paper, we have discussed the issues related to task states, triggers and timeline of new multi-level feedback queue scheduler. The process will be executed through dissimilar states. In the transition each task can have one of the seven different states: new, ready, running, wait, blocked, zombie and succeeded state. Pseudo code of NMLFQ with several task states is apprised. The object oriented way of pseudo-code is explained bearing respective methods. Comments are provided for the welfare of realization.

HashTable for storing both priority and processes are utilized, effectively for Priority-Queue depending on task-states. The state transitions along with the triggers for each of the state are represented. Preemption of a low priority process by the high priority process occurred at each trigger in a real time kernel, depending on the decision of scheduler is also articulated. Finally, the timeline of real time process depicting the deadline, response and execution time of process is conversed.

REFERENCES


[38] Panduranga Rao M.V., Shet K.C and K. Roopa, “Efficient and predictable process scheduling”, In proceedings of International Conference on Contemporary Computing IC3–MACMILLAN JOURNAL, Noida, New Delhi, India, University of FLORIDA (UFL & JIJTU) and Jaypee Institute of Information Technology University, page(s): 131 –140, August 7–9 2008.


AUTHOR BIOGRAPHIES

Prof. M.V. Panduranga Rao is a research scholar at National Institute of Technology Karnataka, Mangalore, India. He has completed Master of Technology in computer science and Bachelor of Engineering in electronics and communication.

His research interests are in the field of Real-Time and Embedded Systems on Linux platform. He has published various research papers in journal and conferences across India, Also in the IEEE international conference in Okinawa, Japan. He has authored two reference books on Linux Internals. He is the Life member of Indian Society for Technical Education and IAENG.

His webpage can be found via http://www.pandurangarao.i8.com/.

© 2014, IJCSMC All Rights Reserved 815
Dr. K.C. Shet obtained his PhD degree from Indian Institute of Technology, Bombay, Mumbai, India, in 1989. He has been working as a Professor in the Department of Computer Engineering, National Institute of Technology, Surathkal, Karnataka, India, since 1980.

He has published over 200 papers in the area of Electronics, Communication, & computers. He is a member of Computer Society of India, Mumbai, India, and Indian Society for Technical Education, New Delhi, India.

His webpage can be found via http://www.nitk.ac.in/~kcshet/index.html.