Volume 4, Issue 7, July 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Development Approach of New Multi Level Feedback Queue Scheduler

**M.V. Panduranga Rao**
Research Scholar, NITK,
Surathkal , Mangalore, India
raomvp@yahoo.com

**K.C. Shet**
Professor, NITK,
Surathkal, Mangalore, India
kcshet@rediffmail.com

***Abstract— The scheduler is a vital part of the operating system software required to manage the tasks to assign processor time or allocate an critical resource. In this research paper, the distinguishing problems with existing Multi Level Feedback Queue (MLFQ) scheduling algorithm are discussed. This is taken as a milestone to develop a New Multi Level Feedback Queue (MLFQ). The relationships of job sets, record of tasks and queues are illustrated along with synchronization through semaphore. The algorithm of NMLFQ scheduler involving respective steps is debated along with pseudo-code. The approximate sort of object oriented code is also described with comments to justify the algorithm.***

***Keywords— NMLFQ, scheduler, process, queue, deadline, preemption, priority, Multi Level, semaphore and object oriented code.***

## I. INTRODUCTION

A scheduler is a portion of an operating system component. The objective of the scheduler is to optimize system performance corresponding to the principles set by the experts. It pertains to a set of policies and mechanisms, put together into the operating system, which decides the order in which work to be performed by computer system. In a multitasking system, several processes are kept and managed in main memory. A running program in a memory is known as process. In other words, every process along with its linked data is defined by the shared portion of memory or block of memory (Sinnen O and Sousa L. [27]). The contents of general purpose registers are used by the arithmetic logic unit (ALU) to execute. The control unit (CU) is used to update results. In this way Central Processing Unit (CPU) plays an important role to invoke a process. Several input / output (I/O) devices may be linked with the execution of every single process[1][33][54].

The CPU will normally be executing one process at a time. It will switch to a new process only when the high priority process interrupts. The previous process will be blocked or waits for an event to happen from scheduler in order to start later. Henceforth the processer is kept always busy executing several processes in time shared mode. This discussion concisely introduces the fundamental concept of scheduling through multiprogramming (Zhi Quan and Jong-Moon Chang [28]).

The scheduler is a vital part of the operating system software required to manage the tasks to assign processor time or allocate resource (Kruk L. et al. [29]). It usually attempts for minimizing response time, maximizing system throughput and ensuring fairness amongst the running processes in the system (Jean-François Hermant and Gérard Le Lann. [30]).

## II. REVIEW OF MULTI LEVEL FEEDBACK QUEUE SCHEDULER

It preserves several queues, using lower numbered queues staying at top position, consuming maximum priority (Tanenbaum [14]). The scheduler transfers the process to alternative next lower level queue if a process already has utilized a certain quanta in its queue. Indeed it uses Round Robin policy inside every queue to choose a process of the maximum priority. Interactive processes have high priority over longer ones, because long processes are transferred to lower priority queues. Normally it provides additional quantum time to lower priority queues.

In Multi Level Feedback Queue Scheduler, it is not clear about allocation of primary static priorities, the number of priority levels, fine-tuning or scaling of priorities at random time dynamically and decision to specify time quantum for every queue. MLFQ ignores the I/O bound processes, identification of foreground processes along with background processes, isolation of processes and ascertaining appropriate scheduling algorithm for every queue.

Allocation of primary static priorities, scaling or changing of priorities is not properly managed in MLFQ. Fixing of quantum time or time slice fine tuning to boost high priority job is not supported. Input output related urgent tasks are ignored in Multi Level Feedback Queue.

In MLFQ and other general operating system scheduler, it is not necessary to know the burst time of a task before-head. In our proposed NMLFQ, the scheduler is to be targeted for real time scenario. The concept of dynamic change of priority as well as change in quantum time must be employed. As the quantum time is raised for each next level of queue, the total burst time decreases and hence process completes early.

 

## III. DISTINGUISHING PROBLEMS OF MLFQ

The distinguishing problems with existing multi-level feedback queue (MLFQ) scheduling algorithm are,

- The quantity of priority levels of queues.
- Discovering appropriate scheduling algorithm for every queue.
- Allocating time quantum for every queue.
- Allocating primary static priorities.
- Fine-tuning or scaling of priorities at random time dynamically.
- Supporting I/O bound processes.
- Separation and identification of foreground processes and background processes.

The analysis of several scheduling algorithms is carried out by means of implementation, simulation and elementary algorithmic analysis approaches. A scheduling algorithm to be designed forms an important constituent of the kernel's process [2][7][13][36][48].

After studying policy mechanisms of different available schedulers, a New Multilevel Feedback Queue (NMLFQ) scheduling algorithm is proposed. NMLFQ includes all necessary modules to compete as a real time scheduler applied in embedded system domain [3][19][35].

The MLFQ principle of operation is used in NMLFQ scheduling algorithm in such a way that the response time is reduced and the functionality of the scheduling is improved [5][11][37][46]. The maximum number of queue and the quantum burst time for each queue are found using dynamic method. In NMLFQ scheduling, the operating system can modify the number of queues and the quantum of each queue according to the getable processes, deadline and as well as urgency consideration of the process (Garcia P. et al. [31]).

## IV. RELATIONSHIPS OF JOB SETS, RECORD OF TASKS AND QUEUES

The relationships of job sets, record of tasks and queues are illustrated in fig. 1. The job sets are declared for multiple processes. Several records are maintained. Task and task queue record are utilized for process information and allocation of resources including queues. The input output record is utilized for task specification and synchronization. The synchronization amongst them is achieved using semaphores. The critical region locking is maintained by mutual exclusion. Execution of processes and reordering of it is prepared in a number of queues. All queue related information is maintained using task queue record consisting of queue data, task sequencing, remaining quanta and process migrations [22][26][38].

A semaphore is a variable or abstract data type. This offers a simple generalization for controlling access for multiple processes to a common shared resource. The processes may compete for CPU time to lock the resource in a parallel programming environment [15][19][41]. A semaphore can be used as a record of count. This count exactly gives the number of times a particular resource to be utilized. The process if necessary made to wait until a unit of the resource becomes available. Thus, semaphores become a helpful tool in the avoidance of race conditions amongst processes competing for resource allocation. Nevertheless, the use of semaphore cannot guarantee the process scheduling is free from these problems of race condition [14][20][40]. Usually semaphores used for subjective resource count are known as counting semaphores. Semaphores utilized with values 0 and 1 are called binary semaphores.

The New Multi-Level Feedback Queue scheduler using the concept of multiple waiting queues have been designed where each of the ready processes wait for the CPU cycle [6][12][18][24]. The problem of providing precise allocation in scheduling of tasks mostly relies on some strict assumptions like full pre-emptibility of tasks, pre-emptive strategies, scheduling policies, etc (Abdelzaher T. F et al. [32]). Implementing an accurate timing mechanism in the active kernel enables such CPU scheduling strategies; hence improving the accuracy of scheduling tasks by making more realistic assumptions. Context switching is an anticipated characteristic of CPU scheduler; however too much context switching inserts undesirable dispatch latency to increase overheads of scheduler [4][16][27][39].

The **algorithm** of NMLFQ scheduler contains many issues. The salient steps basically involve the following steps.

- NMLFQ considers "Priority" as an important factor to be considered, while developing scheduling policies for real time systems. Here multiple waiting queue are considered, where each of the ready processes wait for the CPU cycle. The processes flow through each of the queues depending on their priority levels i.e. 0-50 high priority, 51-99 medium priority and 100-150 low priority (Lauzac S and Melhem R. [33]). The confirmation of completion of the quantum and burst time of process is illustrated in figure 3.1.2.

- With every process submitted to the system, CPU scheduling algorithm receives three major factors called Priority, Burst time and arrival time. Generally, when assigning the importance, priority fetches more importance followed by burst time and finally the arrival time (Brebner G and Diessel O. [34]).

- Processes are categorically linked with many components. These components are dynamic priority, burst or quantum time and arrival time. Significantly the same components will be used to decide the ordering of processes to be invoked, when they are available in ready queue. These components are used to sort the runnable processes to certain extent. This will indirectly help the scheduler to choose the prominent process to get execute. Primarily these three components used to constitute a decider component. The below equation governs this relation;
  DECIDER = dynamic priority **&&** burst or quantum time **&&** arrival time → (1)

 

Depending on the value of DECIDER, evaluated for each processes from above equation, the submitted processes are prearranged in the ready queue. This DECIDER influences, to decrease waiting time, turnaround time, response time and number of context switches. This DECIDER also influences, to rise CPU utilization time. The dynamic priority value changes depending upon load, urgency, associativity and dependency of processes. This priority may be high (0 to 50), low (100 to 150) or medium level (51 to 99). The two vital criteria's to elect for, early runnable processes are higher priority and smaller burst time. The burst time and initial arrival time are added to find a value. This value must be a minimum to elect the process to get executed next. Along with this value the priority of the process is checked. If it is high priority process, the minimum count process is recommended for execution. In this way the processes are arranged in ascending order for execution. When this value is same for two or more processes, then the order of FCFS is used to choose the process.

Example 1):
priority = 49 (high priority process, range 0 to 50)
burst time = 240 msec.
arrival time = 20 msec.
f = priority + burst time + arrival time
f = 49 + 240 + 20 = 309.
The value of form factor is 309; it is restored in lookup table along with the history of the process.

Example 2):
priority = 98 (medium priority process, range 51 to 99)
burst time = 190 msec.
arrival time = 40 msec.
f = priority + burst time + arrival time
f = 98 + 190 + 40 = 328.
The value of form factor is 328; it is also updated in lookup table.

Example 3):
priority = 145 (low priority process, range 100 to 150)
burst time = 320 msec.
arrival time = 10 msec.
f = priority + burst time + arrival time
f = 145 + 320 + 10 = 475.
The value of form factor is 475; these updated values are used along with deadline and urgency of the process to reorder the processes of execution.

- The initial values of form factor provide total weight of the count as 309, 328 and 475. The intention here is to check for the **completion time** of the process. The processes arrive at different times as 20, 40 or 10. Apart from these three factors an additional factor is quantum time, which is the time slice at which CPU executes each process. This quantum time is fixed as for example 10. This quantum time can be varied depending on the number processes arrives and the load on the CPU. The priority value can be set between 0 and 150, to distinguish as low, medium and high priority processes. The burst time decreases, as each of the process is completed with multiple queues with quantum value as 10.

The brief pseudo-code of algorithm of NMLFQ scheduler involves the following steps.

```
METHOD BEGIN public void dynamicQueueing(int numberOfProcesses)
        MONITOR
                int totalQueues = numberOfProcesses/10;
                ArrayList<Process> dynQueue = null;
                OUTER FOR BEGIN (int i=0;i<totalQueues;i++)
                        dynQueue = new ArrayList<Process>();
                        INNER FOR BEGIN (int j=0;j<totalQueues;j++)
                                dynQueue.add(Runtime.getRuntime().exec("ps -ef"));
                        INNER FOR END
                        pQueue.put(i, dynQueue);
                OUTER FOR END
        MONITOR END
        CATCH BEGIN (Exception ex)
                ex.printStackTrace();
        CATCH END
METHOD END
```

 

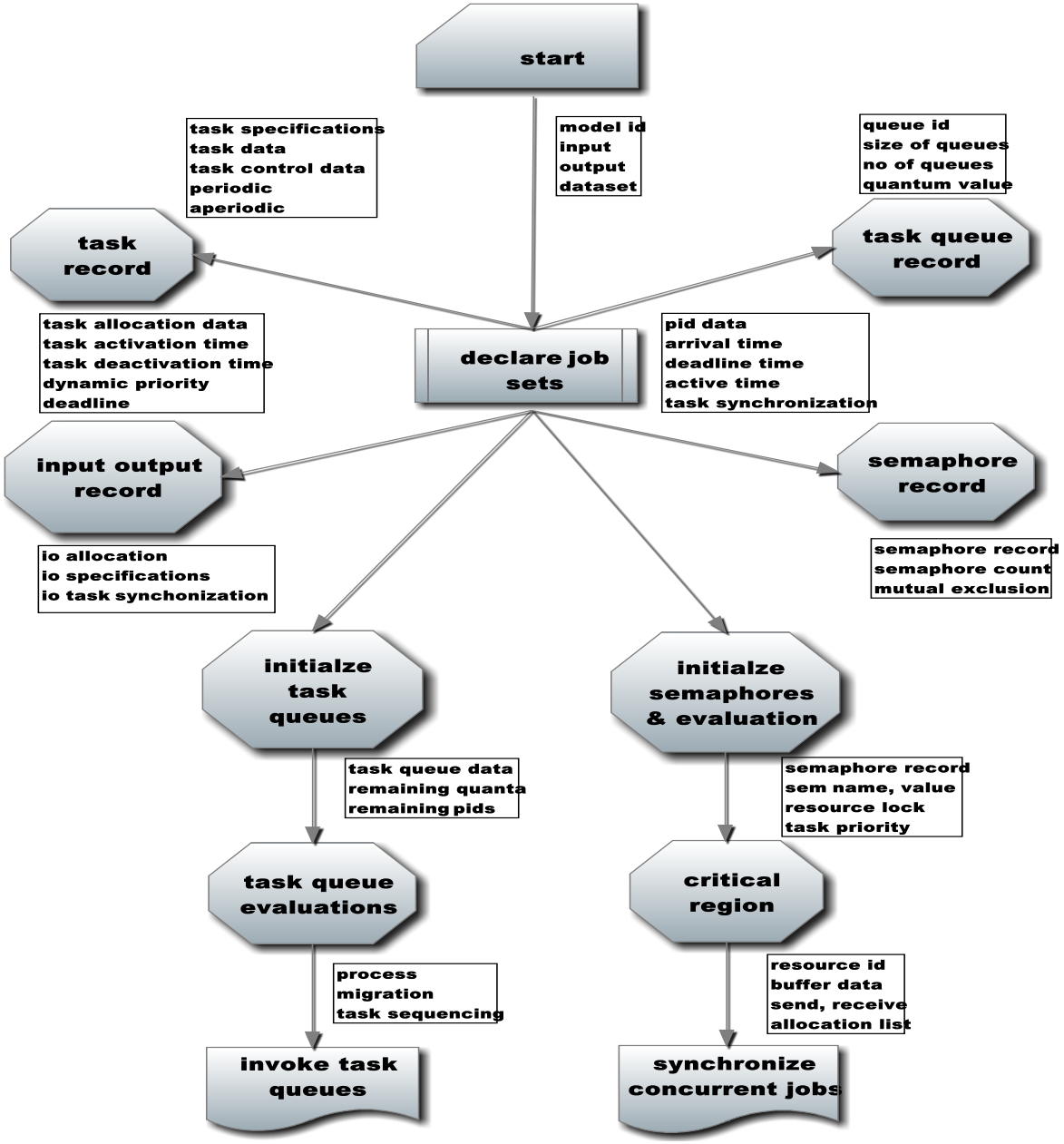


Fig. 1 Basic model involving job sets, task record, queue record and synchronization

```
/*  Creating dynamic queues based on number of processes. Here we are assigning the time quantum to each every
dynamic queues which are created [8][43].  */
    METHOD BEGIN public void dynamicQBasedOnPriority(int numberOfProcesses)
        MONITOR BEGIN
            int totalQueues = numberOfProcesses/10;
            ArrayList<Process> dynQueue = null;
            int dynamicQuantum=0;
            OUTER FOR BEGIN (int i=0;i<totalQueues;i++)
                dynQueue = new ArrayList<Process>();
                INNER FOR BEGIN (int j=0;j<totalQueues;j++)
                    dynQueue.add(Runtime.getRuntime().exec("ps -ef"));
                INNER FOR END
                dynamicQuantum =dynamicQuantum+10;
                pQueue.put(i*dynamicQuantum, dynQueue);
            OUTER FOR END
        MONITOR END
        CATCH BEGIN (Exception ex)
            ex.printStackTrace();
        CATCH END
    METHOD END
```

 

New Multi Level Feedback Queue Scheduler is developed initially using C, then Embedded C and later using object orientation approach of C++ and JAVA. The description of NMLFQ scheduling algorithm, in terms of false code [pseudo-code] is presented here. It is ambitious to represent all of the details. The brief version of it is explained in this research paper. The object oriented way of pseudo-code [9][42], which is almost similar to actual code, is enlightened in the following section.

## V. PSEUDO-CODE FORM OF NMLFQ OBJECT ORIENTED CODE

The object oriented way of pseudo-code with comments is provided for the ease of understanding of NMLFQ algorithm.

```
public class NMLFQ {
        ArrayList<Process> readyQueue =new ArrayList();
        Hashtable<Integer, ArrayList> pQueue = new Hashtable<Integer, ArrayList>();
        public Process p=null;
        public Process removedProcess=null;
        public NMLFQ(){
                // initially loading queue with quite a few number of processes.
                try{
                        // for simplicity, here we consider 3 queues.
                        for(int j=1;j<=3;j++){
                                for(int i=1;i<=5;i++){
                                        // Creation of process.
                                        p=Runtime.getRuntime().exec("ps -ef");
                                insertIntoMLQ(p);
                                }
                                pQueue.put(j,readyQueue);
                                readyQueue =null;
                        }
                }catch(Exception ex){
                        ex.printStackTrace();
                }
        }

public NMLFQ(int i){
                try{
                // removing the requested number of processes from queue.
                        removedProcess=removeAtFrontEnd(i);
                        if(removedProcess != null){
                             System.out.println("process is removed from queue");
                        }else{
                             System.out.println("process to be removed from the queue is null");
                        }
                }catch(Exception ex){
                        ex.printStackTrace();
                }
        }

/* The following method add the process to the ready-Queue based on priority of the queues. The addition operation will be done as in the case similar to First Come First Serve [44]. */

public void addAtFront(int priority, Process temp){
                ArrayList procQ = pQueue.get(priority);
                procQ.add(temp);
        }

/* The following method will insert some processes initially into queue along with ready-Queue.        */
public void insertIntoMLQ(Process process){
                //Adding process at the end of the Queue
                if(process != null){
                        readyQueue.add(process);
                        System.out.println("process is added into Queue");
                }else{
                        System.out.println("process is null");
```

 

```
		}
	}

	/* The following method removes the process from the queue. */
	public Process removeAtFrontEnd(int queueId){
		// using the specified index, removing the process from ready-Queue and pQueue.
		ArrayList tempArr = pQueue.get(queueId);
		return((Process)tempArr.remove(0));
	}

	/* Creating dynamic queues based on number of processes. Here we are dividing the number of processes by 10 to create the queues. */
	public void dynamicQueueing(int numberOfProcesses){
		try{
			int totalQueues = numberOfProcesses/10;
			ArrayList<Process> dynQueue = null;
			for(int i=0;i<totalQueues;i++){
				dynQueue = new ArrayList<Process>();
				for(int j=0;j<totalQueues;j++){
					dynQueue.add(Runtime.getRuntime().exec("ps -ef"));
				}
				pQueue.put(i, dynQueue);
			}
		}catch(Exception ex){
			ex.printStackTrace();
		}
	}

	/* Creation of dynamic queues depending upon number of processes. Here we are assigning the time quantum to each every dynamic queue which is created [10][16][52]. */
	public void dynamicQBasedOnPriority(int numberOfProcesses){
		try{
			int totalQueues = numberOfProcesses/10;
			ArrayList<Process> dynQueue = null;
			int dynamicQuantum=0;
			for(int i=0;i<totalQueues;i++){
				dynQueue = new ArrayList<Process>();
				for(int j=0;j<totalQueues;j++){
					dynQueue.add(Runtime.getRuntime().exec("ps -ef"));
				}
				dynamicQuantum =dynamicQuantum+10;
				pQueue.put(i*dynamicQuantum, dynQueue);
			}
		}catch(Exception ex){
			ex.printStackTrace();
		}
	}

	/* The processes inside the queue are sorted in a descending order of priority. Now when any I/O bound process arrives, increase the priority of the same and allow it for execution [17][21][23][45]. */
	public void processing(){
		try{
		int i=10;
		while(!pQueue.isEmpty()){
			if(true){
				// if process is I/O bound then allow it to get execute.
			}
			executingProcessQueue(pQueue.get(i--));
			if(true){
				// encountered beginning of the next process so it's coming out.
				return;
			}
		}
```

 

```
		}catch(Exception ex){
			ex.printStackTrace();
		}
	}

	/* Now actual execution starts. */
	public void executingProcessQueue(ArrayList process){
		// actual Execution happens.
		System.out.println("Execution started...");
	}
```

/* Here scheduler decides which process to execute and when exactly it should get executed. The priority of the process gets increased depending on the instances of dynamicity, aging or urgency using nice concept of prioritization [25][49][53]. */

```
	public void upgradeProcess(){
		// increase the priority.
		// selection of the queue.
		// allow the queue for execution.
	}

	public void demoteProcess(){
		//Select the queue
		//decrese priority

	}
```

/* Here, we are creating pQueue as a placeholder for processes, along with the priorities of processes, which are ready to execute. Though it is NMLFQ, here we are using HashTable for storing priority and processes queue. The addition of processes will be done as in the case similar to First Come First Serve. */

```
	public static void main(String[] args) {
		try{
		// TODO Auto-generated method stub.
		// creation of the queue.
		new NMLFQ();
		// removal of processes from queue.
		new NMLFQ(2);
		// adding processes to the queues depending upon their priority.
		new NMLFQ().addAtFront(2,Runtime.getRuntime().exec("ps -ef"));
		// creation of dynamic queue based on number of processes.
		new NMLFQ().dynamicQueueing(100);
		// creation of dynamic queue based on number of processes and quantum, which changes depending on
		 priority.
		new NMLFQ().dynamicQBasedOnPriority(100);
		// fetching the default queue, from the pool of queues based on their descending order of priority.
		new NMLFQ().upgradeProcess();
		new NMLFQ().demoteProcess();
		while(true){
			new NMLFQ().processing();
			Thread.currentThread().sleep(2000);
		}
		}catch(Exception ex){
			ex.printStackTrace();
		}
	}
}
```

In the NMLFQ scheduler, Queues will be created at runtime dynamically and thus number of queues varies depending on the priority and the number of processes that are available. Dynamic quantum will be different for different queues based on their priority. All the processes within a queue will have the same priority and thus get the same quantum of time for execution.

At the base level queue, processes are distributed in round robin fashion until they are complete and lastly leave the system. In NMLFQ before pushing a process down to a lower level queue, it is given only one chance to complete at a given queue level. Preempting the processes exceeding its quantum time and pushing back to one of its several priority

 

queues implies that kernel allocates the CPU for a specific quantum time only. A process before completion has to flow through several iterations through the feedback loop by executing at specific time quantum's [22][28][50].

If a job is interrupted for any reason it joins the end of the queue. The process must go to wait state and hence it has to give option for other processes to run. The interruption is because of preemption, occurring of event, scarcity of resource etc. Preemption of high priority job is usual. Occurring of event is for any exceptional cases or erroneous instances [9][47]. Scarcity of resource is due lack of resources and also due to need of particular resource, which is already locked by earlier process. The high priority queues acquires greater CPU cycles compared to lower priority ones [24][51]. In order to minimize the potentiality of a very high priority process being not executed for long time in the queue, the scheduler employs round robin scheduling scheme in the queues. This definitely helps in avoiding starvation.

## VI. CONCLUSIONS

In this research paper, we have discussed the issues related to problems of MLFQ. In Multi Level Feedback Queue Scheduler, It is not clear about allocation of primary static priorities, the number of priority levels, fine-tuning or scaling of priorities at random time dynamically and decision to specify time quantum for every queue. MLFQ ignores the I/O bound processes, identification of foreground processes along with background processes, isolation of processes and ascertaining appropriate scheduling algorithm for every queue.

Some of the features of NMLFQ are discussed. CPU Scheduling is a wide active subject of research. Here many developers and programmers make use to design effective scheduling algorithms for CPU processes, in order to obtain output in competent method.

A New MLFQ scheduler using the concept of multiple queuing mechanisms has been designed. Simpler design strategy would be to provide preference for shorter jobs and I/O bound processes. Rapidly capturing the nature of a process and scheduling the process consequently. This scheduler also considers the important parameters viz. preemption, priority, quantum time, deadline, CPU history and urgency of process. These factors are considered while designing scheduling policy for soft and hard real time systems.

## REFERENCES


[1] Abdelzaher T. F., Sharma V., and Lu C.,"A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling", in *IEEE Transactions on Computer*, Volume 53, Number 3, Page(s): 334-350, 2004.

[2] Albert Haque, "An Analysis and Comparison of Processor Scheduling Techniques", *University of Texas at Austin, ahaque@cs.utexas.edu,* December 7, 2012.

[3] Arezou Mohammadi and Selim G. Akl, "Scheduling Algorithms for Real-Time Systems", *Technical Report No. 2005-499, Natural Sciences and Engineering Research Council of Canada*, School of Computing, Queen's University, July 15, 2005.

[4] Ashiq Anjum, Richard McClatchey, Arshad Ali and Ian Willers, "Bulk Scheduling With Diana Scheduler", in *proceedings of IEEE Transactions on Nuclear Science, ISSN 0018-9499*, Volume 53, Issue 6, Page(s):: 3818-3829, 2006.

[5] Ayan Bhunia, "Enhancing the Performance of Feedback Scheduling", *International Journal of Computer Applications, ISSN 0975 – 8887*, Volume 18, Number.4, March 2011.

[6] Baskiyar S and Meghanathan N, "A Survey of Contemporary Real-time Operating Systems", *Informatica*, Volume 29, Auburn University, Auburn, USA, Page(s):: 233–240, 2005.

[7] Becchetti, L., Leonardi, S. and Marchetti S.A., "Average-Case and Smoothed Competitive Analysis of the Multilevel Feedback Algorithm", *Mathematics of Operation Research,* Volume 31, Number 1, February, Page(s):: 85–108, 2006.

[8] Behera H. S., Rakesh Mohanty, Sabyasachi Sahu and Sourav Kumar Bhoi, "Comparative Performance Analysis of Multi-Dynamic Time Quantum Round Robin (MDTQRR) Algorithm with Arrival Time", *Indian Journal of Computer Science and Engineering, IJCSE, ISSN: 0976-5166,* Volume 2, Number 2, Page(s):: 262-271, Apr-May 2011.

[9] Brebner G and Diessel O, "Chip-based Reconfigurable Task Management", in *Field-Programmable Logic and Applications (FPL'01),* Springer Verlag, Berlin, Germany, Page(s): 182–191, 2001.

[10] Burns A, Tindell K and Wellings A.J. "Fixed priority scheduling with deadline prior to completion", *in proceedings of Sixth Euromicro Workshop on Real-Time systems, Research Group Department of computer Science University of York,* UK. Page(s): 138 – 142, 1994.

[11] Cai Sufeng and Hugh Anderson, "Queueing Theory", *Operating System, Group Project Report, Group 36,* NUS, National University of Singapore, school of computing, 2007 - 2008.

[12] David B. Stewart, Donald E. Schmitz, and Pradeep K. Khosla, "The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications", *IEEE Transactions on Systems, Man, and Cybernetics*, Volume 22, Number 6, Page(s):: 1282-1295, Nov / Dec 1992.

[13] Dharamendra Chouhan, SM Dilip Kumar and Jerry Antony Ajay, "A MLFQ Scheduling Technique using M/M/c Queues for Grid Computing", *International Journal of Computer Science Issues, IJCSI, ISSN: 1694-0784,* Volume 10, Issue 2, Number 1, Page(s):: 357-364, March 2013.

[14] Dodd R.B, "Coloured Petri Net Modelling of a Generic Avionics Mission Computer" under "Avionics Enabling Research and Development", *Defense Science and Technology Organization, DSTO,* Australia, DSTO-TN-0692, April 2006.


 

[15] Dror G. Feitelson, "Notes on Operating Systems", *School of Computer Science and Engineering, The Hebrew University of Jerusalem,* Israel, 2011.

[16] Garcia P, Compton K, Schulte M, Blem E and Fu W., "An overview of reconfigurable hardware in embedded systems", *in EURASIP Journal on Embedded Systems*, Page(s): 1–19, 2006.

[17] Gary J. Nutt, "Implementing Processes, Threads, and Resources" and "scheduling" along with "Basic Synchronization Principles", *in text book Operating Systems, Third Edition, ISBN 0-201-77344-9,* Addison-Wesley, 2004.

[18] Hoganson, Kenneth, "Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging", *Consortium for Computing Sciences in Colleges, Southeastern Conference,* Georgia, 2009.

[19] Jean-François Hermant and Gérard Le Lann, "Fast asynchronous uniform consensus in real-time distributed systems", in *IEEE Transactions on Computers*, Volume 51, Number 8, Page(s): 931–944, 2002.

[20] krithi Ramamritham and John A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems", *Proceedings of the IEEE,* Volume 82, Issue 1, Page(s):: 55-67, Jan 1994.

[21] Kruk L, Lehoczky J. P, Shreve S. E and Yeung S.N, "Earliest deadline first service in heavy traffic acyclic networks", in *Annals of Applied Probability*, Volume 14, Number 3, Page(s): 1306-1352, 2004.

[22] Lauzac S and Melhem R, "An Improved Rate-Monotonic Admission Control and Its Applications", in *IEEE Transactions on Computers,* Volume 52, Number 3, Page(s): 337-350, 2003.

[23] Li Lo and Liang-Teh Lee, "An Interactive Oriented Fair Scheduler with Bounded Starvation for Desktop Time-Sharing Systems", *Thesis for Master of Science, Department of Computer Science and Engineering,* Tatung University, January 2008.

[24] Maricruz Valiente, Gonzalo Genova and Jesus Carretero, "UML 2.0 Notation for Modeling Real Time Task Scheduling", *proceedings in Journal of Object Technology,* Volume 5, Number 4, Page(s):: 91-105, May-June 2006.

[25] Michael B. Jones et al, "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities", *in Proceedings of the sixteenth ACM Symposium on Operating Systems Principles, ACM SOSP '97, ISBN:0-89791-916-5,* Page(s):: 198-211, 1997.

[26] Micro digital, "Simple Multitasking Executive", *in proceedings of online document SMX RTOS, smxinfo,* Embedded Software Outfitters, SMX , Symmetry Innovations, Australia, December 2004.

[27] Mohammad Reza Effat Parvar, Karim Faez, Mehdi EffatParvar, Mehdi Zarei, Saeed Safari, "An Intelligent MLFQ Scheduling Algorithm (IMLFQ) with Fault Tolerant Mechanism", *in proceedings of Sixth International Conference on Intelligent Systems Design and Applications, ISDA'06,* Volume 3, Page(s):: 80-85, Oct 2006.

[28] Mohammad Reza Effat Parvar, Mehdi Effat Parvar, Abolfazl Toroghi Haghighat, Reza Mahini, Mehdi Zarei, "An Intelligent MLFQ Scheduling Algorithm (IMLFQ)", *in proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA,* Las Vegas, Nevada, USA, Page(s):: 1033-1036, June 2006.

[29] Mohammad Reza Effat Parvar, Akbar Bemana, Mehdi EffatParvar, "IMLFQ Scheduling Algorithm with Combinational Fault Tolerant Method", *in proceedings of International Conference on Enformatika Systems Sciences and Engineering,* Prague, Czech Republic, Jan 2006.

[30] Mohammad Reza Effat Parvar, Mehdi Effat Parvar, Saeed Safari, "A Starvation Free IMLFQ Scheduling Algorithm Based on Neural Network", *International Journal of Computational Intelligence Research, ISSN 0973-1873,* Volume.4, Number.1, Page(s):: 27–36, 2008.

[31] Panduranga Rao M.V. and Shet K.C, "Study and Development of a New Multi Level Feedback Queue Scheduler for Embedded Processor", *International Journal of Computer Sciences and Engineering Systems. IJCSES, ISSN 0973-4406.* Volume 4, Number 3, Page(s): 203-213, July 2010.

[32] Panduranga Rao M.V. and Shet K.C, "A Simplified Study of Processor Scheduler for Real Time Operating System", *International Journal of* Computational Cognition. *IJCC, ISSN 1542-5908 (online); ISSN 1542-8060 (print).* Volume 8, Number 3, Page(s): 5-16, September 2010.

[33] Panduranga Rao M.V. and Shet K.C, "A Research in Real Time Scheduling Policy for Embedded System Domain", *CLEI Electronic Journal,* ISSN 0717- 5000. Volume 12, Number 2, Paper 4, August 2009.

[34] Panduranga Rao M.V. and Shet K.C, "Analysis and Refining of Scheduler for Real Time Operating System", ICFAI University Journal *of* Computer Sciences. ISSN: 0973-9904. Volume. 3, Number 4, Page(s): 7-22, October 2009.

[35] Panduranga Rao M.V. and Shet K.C, "New Approaches to Improve CPU Process Scheduler in the Embedded System Domain", *i-manager's Journal on Future Engineering and Technology, JFET,* ISSN: 0973- 2632. Volume 4, Number 1, Page(s): 72-84, July – September 2009.

[36] Panduranga Rao M.V. and Shet K.C, "A Simplistic Study of Scheduler for Real Time and Embedded System Domain", *International Journal of Computer Science and Applications, IJCSA, ISSN: 0974-1003*. Volume 2, Number 2, Page(s): 99-108, November 2009.

[37] Panduranga Rao M.V. and Shet K.C, R. Balakrishna and K. Roopa, "Development of Scheduler for Real Time and Embedded System Domain", *22nd IEEE International Conference on Advanced Information Networking and Applications - Workshops, WAINA '08, 2008.* FINA 2008, Fourth International Symposium on Frontiers in Networking with Applications, Gino-wan, Okinawa, JAPAN. IEEE Computer Society 2008. DOI 10.1109/WAINA.2008.33, Page(s):1 – 6, 25 to 28th March 2008.

 

[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 & JIITU) and Jaypee Institute of Information Technology University, page(s): 131–140, August 7–9 2008.

[39] Panduranga Rao M.V., Shet K.C, K. Roopa and K.J. Sri Prajna, "Implementation of a simple co-routine based scheduler", In proceedings of *Knowledge based computing systems & Frontier Technologies NCKBFT, MIT Manipal, Karnataka, INDIA*, Page(s): 161-164, 19 to 20th Feb 2007.

[40] Paolo Di Francesco, "Design and implementation of a MLFQ scheduler for the Bacula backup software", *Master thesis in Global Software Engineering*, Universita degli Studi dell'Aquila, Italy, 2011 / 2012.

[41] Peng Li and Binoy Ravindran, "Fast, Best-Effort Real-Time Scheduling Algorithms", *IEEE Transactions On Computers*, Volume 53, Issue 9, Page(s):: 1159-1175, Sept 2004.

[42] Raymond Keith Clark, "Scheduling Dependent Real-Time Activities", *PhD Thesis, School of Computer Science*, Carnegie Mellon University, CMU-CS-90-155, August 1990.

[43] Raymond Keith Clark, Jensen E. D and Rouquette N. F. "Software organization to facilitate dynamic processor scheduling", *in IEEE Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Page(s): 122b, 2004.

[44] Sami Khuri, Hsiu-Chin Hsu, "Visualizing the CPU Scheduler and Page Replacement Algorithms", *Proceedings of the 30th ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '99,* New Orleans, Louisiana, USA, ACM Press, New York, Page(s):: 227-231, March 1999.

[45] Shahrooz Feizabadi et al., "Utility Accrual Scheduling With Real-Time Java", *On The Move to Meaningful Internet Systems, OTM Workshops, ISBN 978-3-540-39962-9*, Page(s):: 550-563, 2003.

[46] Shahrooz S. Feizabadi, "Garbage Collection Scheduling for Utility Accrual Real-Time Systems", *PhD Thesis, Virginia Polytechnic Institute & State University*, Blacksburg, Virginia, December 7, 2006.

[47] Sifat Islam et al., "Concurrency Compliant Embedded System Modeling Methodology", *in 2nd Annual IEEE International Systems Conference, SysCon 2008, ISBN: 978-1-4244-2149-7*, Montreal, Canada, Page(s):: 1–8, April 2008 .

[48] Silberschatz, A., P.B. Galvin and G. Gagne, "Operating Systems Concepts", *7th Edn., John Wiley and Sons, USA., ISBN: 13: 978-0471694663*, Page(s):: 944, 2004.

[49] Sinnen O and Sousa L, "On Task Scheduling Accuracy: Evaluation Methodology and Results", in *The Journal of Supercomputing*, Volume 27, Number 2, Page(s): 177-194, 2004.

[50] Sukanya Suranauwarat, "A CPU Scheduling Algorithm Simulator", *Session F2H, 37th ASEE / IEEE Frontiers in Education Conference, Milwaukee, WI*, Page(s):: F2H-19 - F2H-24, October 10–13, 2007.

[51] Tanenbaum A.S., "Modern Operating Systems", *3rd Edn., Prentice Hall, ISBN: 13: 9780136006633*, Page(s):: 1104, 2008.

[52] Torrey, L.A., Coleman, J and Miller, B.P., "A Comparison of the Interactivity in the Linux 2.6 Scheduler and an MLFQ Scheduler", *Software Practice and Experience, John Wiley & Sons, Ltd*, Volume 37, Number 4, Page(s):: 347-364, 2007.

[53] Yaashuwanth C and Ramesh R, A New Scheduling Algorithms for Real Time Tasks, *International Journal of Computer Science and Information Security, IJCSIS, ISSN 1947-5500*, Volume 6, Number 2, Page(s):: 61-66, 2009.

[54] Zhi Quan and Jong-Moon Chang, "A Statistical Framework for EDF Scheduling", in *Journal on IEEE Communication Letters*, Volume 7, Number 10, Page(s): 493-495, 2003.

## 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/ .

**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 .

 