Development of Scheduler for Real Time and Embedded System Domain

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

K.C. Shet
NITK, Surathkal
Mangalore, India
kcshet@yahoo.co.uk

R.Balakrishna
SKU, Anantapur
Andhra Pradesh, India
rayankibala@yahoo.com

K. Roopa
Mphasis Limited
Bangalore, India.
roopa.sindhe@gmail.com

Abstract

We discuss scheduling techniques to be used for real-time, embedded systems. Though there are several scheduling policies, the preemptive scheduling policy holds promising results. In this research paper, the different approaches to design of a scheduler for real-time Linux kernel are discussed in detail. The comparison of different preemptive scheduling algorithms is performed. Hence, by extracting the positive characteristics of each of these preemptive scheduling policies, a new hierarchical scheduling policy is developed.

The proposed hierarchical scheduling for real time and embedded system will be implemented for a prototype system, using C or C++ language. It is expected that the new scheduling algorithm will give better performance with respect to satisfy the needs, such as time, capturing and usage of resources of different applications.

Key words: Linux, RTOS, round robin, fcfs, sjn, deadline, hrrn, rms, edf, preemption.

1. Introduction.

A real-time operating system (RTOS) is capable of handling multiple events simultaneously and within fixed-time frame. Computers running mission critical embedded system. A Real Time computer system is a computer system in which the correctness of the system behavior not only depends on the logical results of computation but also on the physical instant at which the results are produced [1].

Real-Time Operating Systems are systems in which certain processes or operations have guaranteed minimum and or maximum response times [10]. That is to say, the system ensures that it will complete operation x after time t but before time t′, whatever t and t′ are, without fail, even at the expense of other lower priority operations [7].

Speed, in and of itself, is not critical; the primary goal is predictability. A response time less than t may be just as bad as, or worse than, one greater than t'. One of the best-known applications need an operating system that responds quickly or within "real time" to requests.

Two essential features make an operating system "real time". The operating system must support multi-tasking with pre-emptive, priority-driven context switching with guaranteed interrupt handling [17]. What it means is that if the operating system receives an outside event, it should be able to switch between the running process and the event handler process immediately. The OS must also have a very efficient inter-process communication (IPC) subsystem. If a process wishes to talk to another, it should be able to do so immediately and without fail [5]].

Typical RTOS Task Model
Each task a triplet: (execution time, period, deadline)
Usually, deadline = period

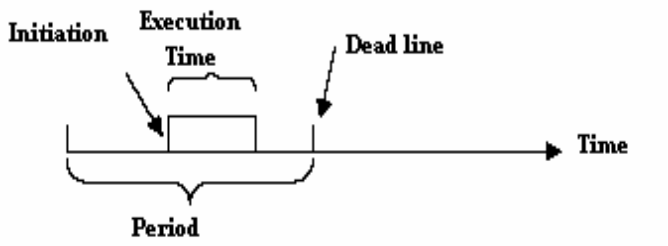

Fig. 1. Typical RTOS Task.

2. Definition Real-Time Systems

Any intelligent device that is embedded with in another system or device is called embedded Real Time OS for the x86 platform is QnX. Each system call of QnX is documented with a 'worst case completion time' [2].

3. Related Work by others

3.1 Different available Scheduling Algorithms and their characteristics

Known scheduling algorithms include Round Robin Scheduling, Priority-Based Scheduling, Earliest Deadline First Scheduling, Rate Monotonic Scheduling, Feedback Scheduling, ...here is a simple classification:

978-0-7695-3096-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WAINA.2008.33

- **#Real-Time Scheduling Algorithms**
  - **#Earliest Deadline First** (EDF)
  - #**Least Laxity First** (LLF)
  - **#Rate Monotonic Scheduling** (RMS)
- **#General Scheduling Algorithms**
  - **#First Come First Serve** (FCFS)
  - **#Round-Robin** (RR)
  - **#Priority-based Round-Robin** (PRR)
- **#Batch Scheduling Algorithms**
  - **#Shortest Process Next** (SPN)
  - **#Shortest Remaining Time** (SRT)
  - **#Highest Response Ratio Next** (HRRN)

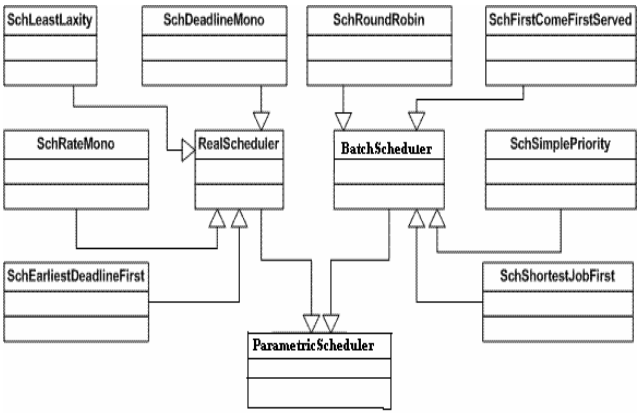

Fig. 2. Study of available Scheduling Algorithms to simulate a scheduler.

- Earliest Deadline First (EDF): Tasks can be periodic or not and are scheduled according to their deadline.
- Least Laxity First (LLF): Tasks can be periodic or not and are scheduled according to their laxity.

Laxity= deadline_time - current_time - CPU_time_needed (1)

- Rate Monotonic: Tasks have to be periodic, and deadline must be equal to period. Tasks are scheduled according to their period [13].
- Deadline Monotonic: Tasks have to be periodic and are scheduled according to their deadline. Rate Monotonic and Deadline Monotonic use the same scheduler engine except that priorities are automatically computed from task period or deadline [3].
- POSIX 1003.1b scheduler: Tasks can be periodic or not. Tasks are scheduled according to the priority and the policy of the tasks. POSIX 1003.1b scheduler supports SCHED_RR, SCHED_FIFO and SCHED_OTHERS queueing policies. SCHED_OTHERS is a time sharing policy. SCHED_RR and SCHED_FIFO tasks must have priorities ranging between 255 and 1. Priority level 0 is reserved for SCHED_OTHERS tasks. The highiest priority level is 255.
- A criticality level: The field idicates how the task is critical. Currently used by the MUF scheduler.
- User-defined scheduler can also be called as parametric scheduler [4].
- The **quantum** value associated with the scheduler. This information is useful if a scheduler has to manage several tasks with the same dynamic or static priority: in this case, the simulator has to choose how to share the processor between these tasks. The quantum is a bound on the delay a task can hold the processor (if the quantum is equal to zero, there is no bound on the processor holding time).

3.2 Research findings and gaps.

- Rate Monotonic Scheduling - a hard real-time scheduling algorithm - can guarantee time restraints only up to 70% CPU load. Beyond that it does not support dynamic systems very well.
- In addition to schedulable bounds that are less than 1.0, two problems exist for RM algorithms [14]: a) RM algorithms provide no support for dynamically changing task periods or priorities and b) tasks may experience priority inversion.
- The first problem can be resolved by considering fixed priority scheduling of periodic task with varying task execution priorities. Specifically tasks may have subtasks of various priorities [5].
- Priority inversion arises when a high priority job must wait for a lower priority job to execute, typically due to other resources being used by executing tasks. i.e tasks waiting on critical sections [18].
- This implies that applications have to state their run-time requirements beforehand - how often they must be called in a second, which maximum response time is acceptable etc. All this information must be provided by the application programmer.
- On the other hand, with earliest-deadline-first (EDF) and minimum-laxity-first (MLF) dynamic scheduling algorithms, a transient overload in the system may cause a critical task to fail, which is certainly undesirable.
- The maximum-urgency-first (MUF) combines the advantages of the RM, EDF, and MLF algorithms [3].
- Like EDF and MLF, MUF has a schedulable bound of 100% for the critical set. And like RM, a critical set can be defined that is guaranteed to meet all its deadlines.
- The MUF algorithm also allows the scheduler to detect forms of deadline failures and call failure handler routines for tasks, which fail to meet their deadlines.

4. Motivation, objectives and Goals

The **scheduler** is the part of the kernel responsible for deciding which task should be executing at any particular time. The kernel can suspend and later resume a task many times during the task lifetime [5].

The objectives and goals of this research are:

- The comparison of different preemptive scheduling algorithms.
- The technique of preemption, reordering of requests and variation of time slice to be used in preemptive scheduling policies.
- Justification for the of execution of user-defined code by the scheduling engine.
- Principle is to develop scheduling policy in a real time environment.

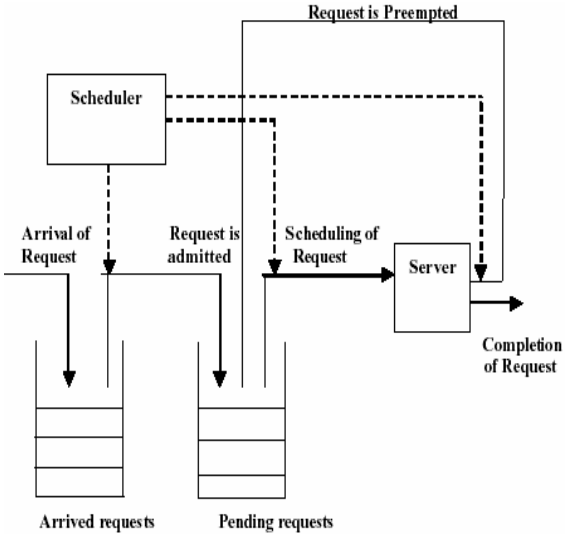

Fig. 3. A Schematic of scheduling

- A mix of CPU-bound and I/O-bound processes exists in the system.
- An I/O-bound process has a higher priority than a CPU-bound process.
- Process priorities are static, i.e., they do not change with time.
- Process scheduling is preemptive; a low priority *running* process is preempted if a higher priority process becomes *ready*. In effect, a low priority process cannot be *running* if a higher priority process exists in *ready* state.

The **scheduling policy** is the algorithm used by the scheduler to decide which task to execute at any point in time [1]. The policy of a (non real-time) multi user system will most likely allow each task a "fair" proportion of processor time [3]. In addition to being suspended involuntarily by the RTOS kernel a task can choose to suspend itself. It will do this if it either wants to delay (**sleep**) for a fixed period, or wait (**block**) for a resource to become available (Ex a serial port) or an event to occur (Ex a key press). A blocked or sleeping task is not able to execute and will not be allocated any processing time.

5. Research Plan

5.1 Problem statement

The main aim is to study the policy mechanisms of different real time schedulers in embedded domain, evaluation of performance of these mechanisms. In addition, to arrive at a common solution to simulate a parametric scheduling policy on real-time Linux kernel for embedded system domain.

5.2 Background significance

Real-time or embedded systems are designed to provide a timely response to real world events. Events occurring in the real world can have deadlines before which the real-time or embedded system must respond and the RTOS scheduling policy must ensure these deadlines are met.

To achieve this objective, first assign a priority to each task. The scheduling policy of the RTOS is then to simply ensure that the highest priority task that is able to execute is the task given processing time [6]. This may require sharing processing time "fairly" between tasks of equal priority if they are ready to run simultaneously.

6. Research Methodology

6.1 Mechanism and policy modules of process scheduler

In preemptive multitasking, a higher priority task will forcibly stop a lower priority task and take the CPU time. This will be very much needed in Real Time Systems so that the execution time of a task can be guaranteed [5]. The trick with any scheduling algorithms is that it must fulfill a number of criteria [12]:

- No task must be starved of resources - all tasks must get their chance at CPU time.
- If using priorities, a low-priority task must not hold up a high-priority task.
- The scheduler must scale well with a growing number of tasks, i.e. taking constant time no matter how many tasks are queued.

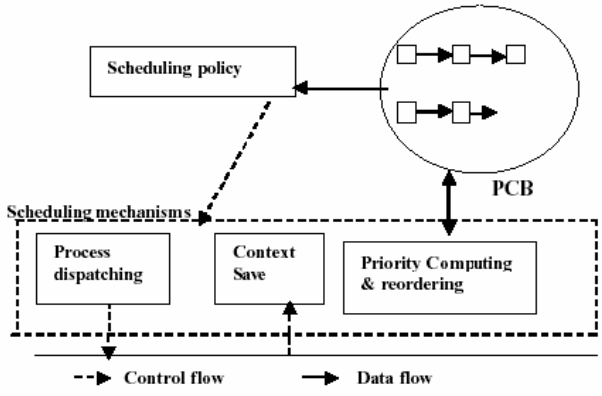

Fig. 4. Mechanism and policy modules of process scheduler.

The scheduler can maintain separate lists of *ready* and *blocked* processes and always select the highest priority process from the *ready* list. However, process priorities are static and scheduling is preemptive [4]. Hence a simpler arrangement can be designed as follows: The scheduler can maintain a single list of PCBs in which PCBs are arranged in the order of reducing priorities. It can scan this list and simply select the first *ready* process it finds. This is the highest priority *ready* process in the system.

6.2 Priority-based scheduling using CRP

In addition to the PCB list, the scheduler maintains a pointer called *currently running process pointer* (CRP pointer). This pointer points to the PCB of the process that is in the *running* state. When an interrupt occurs, the context save mechanism saves the PSW and CPU registers into appropriate fields of this PCB [16].

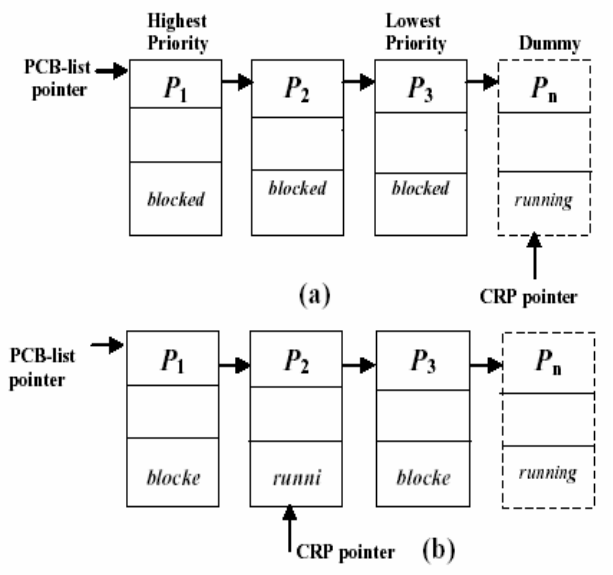

Fig. 5. Priority-based scheduling.

Example: Figure 5 illustrates the situation when all user processes are *blocked*. The only PCB showing a process in the *ready* state is the one for the dummy process [8]. The scheduler selects this process and dispatches it. Figure 5(b) shows the situation after process P_2 becomes *ready*. The PCB of P_2 is the first PCB in the PCB list that shows a process in the *ready* state, so P_2 is scheduled and the CRP pointer is set to point at it.

- A single list of PCBs is maintained in the system.
- PCBs in the list are organized in the order of decreasing priorities.
- The PCB of a newly created process is entered in the list in accordance with its priority.
- When a process terminates, its PCB is removed from the list.
- The scheduler scans the PCB list from the beginning and schedules the first *ready* process it finds.

If there is **no** *ready* processes exist in the system, then scheduler should simply 'freeze' the CPU so that the CPU does not execute any instructions, but remains in an interruptible state so that occurrence of an event can be processed by the event handler. If the architecture lacks a 'freeze' state for the CPU [6], the scheduler can achieve an equivalent effect quite simply by defining a dummy process that contains an infinite loop. This process is always in the *ready* state. It is assigned the lowest priority so that it gets scheduled only when no *ready* processes exist in the system. Once scheduled, this process executes until some higher priority process becomes *ready* [10].

6. 3 Task of the simulator to run user-defined code

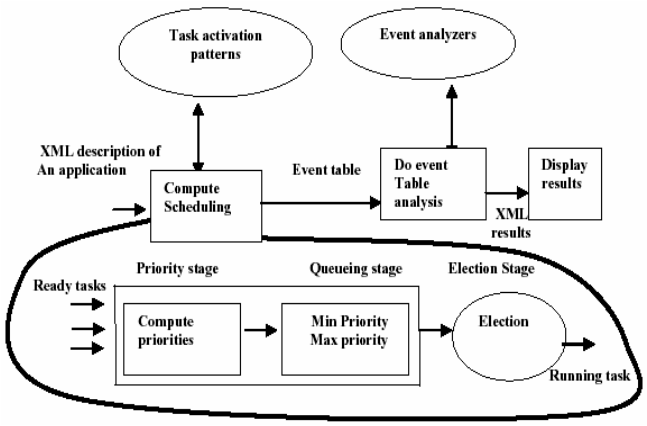

Fig. 6. Description of how a user-defined code is run by the scheduling engine.

Figure 6, gives an idea on the way the simulation engine is implemented in the framework with three-step process.

6.3.1 Computing the scheduling

The first step consists of computing the scheduling. It is required now to decide which events occur for each unit of time [9]. Events can be allocating / releasing shared resources, writing / reading buffers, sending / receiving messages and of course running a task at a given time. At the end of this step, a table is built which stores all the generated events. The event table is built according to the XML description file of the studied application and according to a set of task activation patterns and schedulers.

6.3.2 Analysis of the event table

In the second step, the analysis of the event table is performed. The table is scanned by "event analyzers" to find properties on the studied system. At this step, some standard

information can be extracted by predefined event analyzers, but users can also define their own event analyzers to look for ad-hoc properties. The results produced during this step are XML formatted and can be exported towards other programs.

6.3.3 Defining task activation patterns

Now, let's see how user-defined schedulers or task activation patterns can be added into the framework. All tasks are stored in a set of arrays [19]. Each array stores a given information for all tasks. The job of a scheduler is to find a task to run from a set of ready tasks. To achieve this job, our framework models a scheduler with a 3 stages pipe-line which is similar to the POSIX 1003.1b scheduler [5].
These 3 stages are :
The priority stage: For each ready task, a priority is computed.

The queuing stage: Ready tasks are inserted into different queues. There is one queue per priority level. Each queue contains all the ready tasks with the same priority value. Queues are managed like POSIX scheduling queues: if a quantum is associated with the scheduler, queues work like the SCHED_RR scheduling queuing policy. Otherwise, the SCHED_FIFO queuing policy is applied.

The election stage: The scheduler looks for the non-empty queue with the highest priority level and allocates the processor to the task at the head of this queue. The elected task keeps the processor during one unit of time if the designed scheduler is preemptive or during all its capacity if the scheduler is not preemptive.

6.3.4 User-defined schedulers are organized by means of several sections

The start section: In this section, we declare variables needed to schedule the tasks. Some of them are those defined at task/processor/buffer/message definition. This set of predefined variables can be extended with the "Edit/Add/Add a Task" submenu. The others are managed by the simulator engine and describe the state of tasks/processors/buffers/messages at simulation time.
The priority section: The section contains the code necessary to compute task priorities. The code given here is called each time a scheduling decision has to be made.
The election section: This section just decides which task should receive the processor for next units of time. This section should only contain one return statement [11].
The task activation section: This section describes how tasks could be activated during a simulation. In our framework, 3 kinds of tasks exists [20]: aperiodic tasks which are activated only one time and periodic or poissons process tasks which are activated several times [15]. In the case of periodic tasks, two successive task activations are delayed by an amount of fixed time called period. In the case of poisson process tasks, two successive task activations are delayed by an exponential random delay. The task activation section is used to define new kinds of task activation patterns.

7. Conclusion

- The primary goal of this research paper is to study the policy mechanisms of different real time schedulers in embedded domain.
- The main aim is to analyze and evaluate performance of these real time scheduling mechanisms.
- Implement a good, robust, fully preemptive real-time scheduler.
- Arrive at a common solution to simulate a parametric scheduler policy to real-time Linux kernel for embedded system domain.
- As we mentioned before, there are all sorts of variations on these basic algorithms. The thing to keep in mind is that the more complicated a scheduling algorithm gets, the more it lowers system throughput. It may be possible to write a fancy scheduling algorithm that calculates the precise amount of time each task will take and determines its need for user interaction and then schedules tasks based on mathematical equation.

8. Acknowledgment

We would like to thank the supervisor for guidance and support in the work on my research. Our gratitude goes to the people, who previously succeeded in implementation of modular kernel, scheduler mechanisms, process communication, event analysis, interrupt handling etc and making it available for further development.

References

[1] Chih-Lin Hu, "On-Demand Real-Time Information Dissemination: A General Approach with Fairness, Productivity and Urgency", *21st International Conference on Advanced Information Networking and Applications,* AINA '07, 2007. Page(s):362 – 369, 21-23 May 2007.

[2] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son and M. Marley, "Performance Specifications and Metrics for Adaptive Real-Time Systems," *IEEE Real-Time Systems Symposium,* Orlando, FL, Dec 2006.

[3] Jensen 03a, *A Timeliness Paradigm for Mesosynchronous Real-Time Systems,* E. Douglas Jensen, *9th Embedded and Real-Time Applications and Systems Symposium*, May 2005.

[4] Jensen et al. 02a, *Guest Editors,* " Introduction to Special Section on Asynchronous Real-Time Distributed System*",* E. Douglas Jensen and Binoy Ravindran, *IEEE Transactions on Computers*, August 2005.

[5] Clark et al. 04, "Software Organization to Facilitate Dynamic Processor Scheduling", Raymond K. Clark, E. Douglas Jensen, and Nicolas F. Rouquette, *Proc. of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, Jan 2007.

[6] L. Gauthier, S. Yoo and A. Jerraya, "Automatic generation and targeting of application-specific operating systems and embedded systems software," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(11), pp.1293-1301, November 2005.

[7] Lu, C., Stankovic, A., Tao, G. and Son, H.S. "Feedback Control Real-time Scheduling: Framework, Modeling and Algorithms", special *issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, Vol. 23, No. 1/2 July/September, pp. 85-126, 2002.

[8] A. Wierman, and M. Harchol-Balter. "Classifying scheduling policies with respect to unfairness in an M/GI/1". *In Proceedings of ACM Sigmetrics*, 2003.

[9] C. D. Gill, D. L. Levine and D. C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *Real-Time Syst., vol. 20*, pp. 117-154, 2001.

[10] W.T. Chan, T.W. Lam and K.S. Mak, "Online Deadline Scheduling with Bounded Energy Efficiency", *Proceedings of the 4th Annual Conference on Theory and Applications of Models of Computation (TAMC),* 416-427, 2007.

[11] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. "Implementation of SRPT scheduling in web servers". *ACM Transactions on Computer Systems* 21(2): 207-233, 2003.

[12] A. Bar-Noy, R. E. Ladner, and T. Tamir. "Windows scheduling as a restricted version of bin packing". In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 224-233, 2004.

[13] I. Rai, G. Urvoy-Keller, and E. Biersack. "Analysis of LAS scheduling for job size distributions with high variance". *In Proceedings of ACM Sigmetrics,* 2003.

[14] Lam, T., Ngan, T.J. and TO, K. "Performance Guarantee for EDF under Overload", *In proceedings of the Journal of Algorithms, vol. 52*, pp. 193-206, 2004.

[15] I. Rai, G. Urvoy-Keller, M. Vernon, and E. Biersack. "Performance modeling of LAS based scheduling in packet switched networks". In *Proceedings of ACM Sigmetrics-Performance*, 2004.

[16] H.L. Chan, W.T. Chan, T.W. Lam, L.K. Lee and K.S. Mak, "Energy Efficient Online Deadline Scheduling", *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* pp. 795—804, 2007.

[17] D. Dyachuk and R. Deters, "Scheduling of Composite Web Services" in DOA'06: *In proceedings of the OTM Workshops 2006, LNCS 4277*, pp. 19 – 20, 2006.

[18] S. Aalto, U. Ayesta, and E. Nyberg-Oksanen. "Two-level processor-sharing scheduling disciplines: Mean delay analysis", In *Proceedings.of ACM SIGMETRICS '04*, pages 97–105, 2004.

[19] A. Streit. "Evaluation of an Unfair Decider Mechanism for the Self-Tuning dynP Job Scheduler", In Proceedings of the *13th International Heterogeneous Computing Workshop (HCW) at IPDPS*, pages 108 (book of abstracts, paper only on CD). IEEE Computer Society Press, 2004.

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

Author Biographies

M.V. Panduranga Rao is a research scholar in department of computer engineering, National Institute of Technology Karnataka, Mangalore, India. His research interests are in the field of Real time and Embedded systems on Linux platform and Security.

He has published various research papers across India**.** He has also authored two reference books on Linux Internals for KSOU and Kuvempu University. 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 .

R.Balakrishna is a research scholar at Sri Krishna Devarya University, Anantapur, Andra pradesh, India. His research interests are in the field of Artificial Intelligence, Artificial Neural Networks, Data mining, Operating System and Security.

He has published various papers across India**.** He is the Life member of Indian Society for Technical Education and IAENG. His webpage can be found via http://www.balakrishnar.i8.com/

K.Roopa is a software engineer at Mphasis Limited. She is involved in the development of many research projects on C++ / Linux Plat form in Philips Software India Ltd and LOGICA CMG Bangalore.

She received the BE degree in computer science and engineering from the PES College of Engineering, Mandya, India, in 1997 and the MTech degree in computer science and engineering from the Visvesvaraya Technological University, in 2005.

Her research interests include hardware and software for embedded and real-time systems. She has also authored two reference books on Linux Internals for KSOU and Kuvempu University. She is the Life member of Indian Society for Technical Education