Efficient End-Host Resource Management with Kernel Optimizations for Multimedia Applications

S. Lakshminarayanan[1] and Karthikeyan Mahesh[2]

[1] Department of Computer Science and Eng.
Indian Institute of Technology-Madras
Chennai 600 036 India
lakme@meena.iitm.ernet.in

[2] Department of Computer Science and Eng.
Indian Institute of Technology-Madras
Chennai 600 036 India
karthik@cs.iitm.ernet.in

Abstract. Multimedia applications have timing requirements that cannot be satisfied using time-sharing scheduling algorithms of general operating systems. Our approach is to provide for a resource reservation mechanism to cater to the real-time resource requirement[5] of multimedia applications. We propose the design of a *Resource Manager* which allocates and manages the end-host resources among the processes[7]. We identify three important resources at the end-host namely, the processor, memory and system bus cycles. A process reserves these resources by negotiating with the Resource manager.

The goals that we seek to achieve are: (a) real-time resource management using kernel supported reservation mechanisms, (b) optimal utilization of the various resources of the end-system and (c) kernel optimizations for reducing end-host communication overheads in distributed multimedia applications. We use a two-pronged approach to accomplish our goals. First, we adopt a reservation strategy coupled with priority process scheduling[13,14] to achieve real-time resource management. The reservation mechanism includes a processor and memory reserve abstraction which controls the allocation of processor cycles and memory space to the processes. The reservation scheme can allow applications to dynamically adapt in real-time based on system load and application requirements. Device requirements of a certain multimedia application is abstracted out as a kernel process for system bus reservation and device activation. Secondly, we adopt kernel optimizations to minimize end-host communication overhead in real-time multimedia applications. To improve the end-host performance in distributed multimedia applications, we unveil a new connectionless protocol, Reliable-UDP, in the kernel. We also present aggressive caching mechanism as a scheme for improving end-host performance. The performance of the Resource Manger was tested out with the generation of processes at random times and the results match the expected theoretical results. The connectionless protocol was tested out in a local distributed system and the results are also presented.

H. Leopold and N. García (Eds.): ECMAST'99, LNCS 1629, pp. 46–57, 1999.

1 Introduction

Multimedia systems are computing systems that employ several types of media ranging from static ones, such as text and graphics, to dynamic ones such as audio and video. A multimedia system is often a distributed system in the sense that its components are located in different processing nodes of a local or wide area network. One of the important aspects of multimedia applications is real-time resource management. Many of the new multimedia applications like *Video-conferences* or *Distributed Video on Demand* share the resources of the distributed system. In order to provide efficient services to these applications, one must utilize these resources properly. Resource management policies dictate how resources are allocated, monitored optimized and released.

The resources of the distributed system are of two types: one being the resources at the end-host and the other being the network resources. The resources at the end-host include the processor, memory, the system bus and the peripherals and the network resources include the channels used for communication and the routers present in the network. In this paper we propose an efficient end-host resource management strategy. The resources that are in contention for such a system include CPU time, memory capacity and bus cycles. In many multimedia applications audio and video streams are transmitted and these streams have to be collected by the end-host and processed. A multimedia application, unlike normal applications has timing requirements that cannot be satisfied by normal operating systems. The schedulers in existing operating systems do no take care of the timing requirements of these applications into account. Due to this reason, the performance of the multimedia applications is often suboptimal.

In this paper we present a process scheduling strategy that takes into consideration the timing requirements of the multimedia application. We extend the idea of *processor capacity reserves*[4]—an abstraction of the Real Time Mach micro-kernel to our model to suit processor, memory and system bus requirements. The processor capacity reserves[3] allows application threads to specify their CPU requirements in terms of timing requirements. We extend the above concept to a model in which an application thread can request for a certain memory capacity to suit its applications. The capacity of memory allocated can be dynamically varied to suit the needs of the application. A multimedia application involving audio or video streams has to transfer data to the peripheral devices at periodic intervals of time. To support this mechanism, we provide bus reservation strategies as part of the *Resource manager* which controls the resource reservations of the various applications. The *Resource manager* is designed to allocate the minimum resource requirements to the multimedia application to achieve optimal performance.

In this paper, we have discussed the design of a connectionless and reliable protocol in the kernel called Reliable UDP. This is a high performance protocol over a local area network and the timeout algorithm of TCP has been modified to suit a LAN. We also discuss an aggressive caching mechanism where extra memory capacity reserve is allocated for bulk transfer in a distributed system. The performance of Reliable-UDP for small packets is substantially high when

compared to that of TCP. The variation of timeouts has also been studied in this paper.

In section 2, we present our Working model followed by a discussion of the *Resource Manager* in section 3. In section 4, a description of Reliable UDP is given and the results of our simulations are presented in section 5. In section 6, we conclude giving directions for future work.

1.1 Related Work

In [3],[4] there is a discussion of operating systems support for Multimedia Applications. The approach towards solving this problem has been fundamentally different from the normal solutions for problems in Real Time systems. But the work has manly concentrated on processor capacity reserves[3] and not on other resources of an end-host system.

2 Working Model

A Distributed system is one in which various nodes are interconnected by the underlying network. The resources at the end-host can be abstracted using three parameters namely processing speed, memory capacity and peripherals. Let P_i, M_i, R_i represent the processing capability, memory capacity and device availability of the i^{th} node in the multimedia system. A channel in the network offers a certain bandwidth B_j with a certain delay D_j where j is an index for the channel. Any distributed process running over the entire system uses both the network resources and the end-host resources. Any distributed process Q_k can be represented by its time varying resource requirements. Let $x_{ij}(t), y_{ij}(t), z_{ij}(t)$ represent the time-varying processor, memory and resource requirement of process i at node j. Let $b_{ij}(t)$ represent the bandwidth requirement of process i in the channel j at time t. Let K represent the total number of distributed processes. We have the following requirements from the system.

Node Requirements

$$\sum_{i=1}^{k} x_{ij}(t) \leq P_j$$

$$\sum_{i=1}^{k} y_{ij}(t) \leq M_j$$

$$\sum_{i=1}^{k} z_{ij}(t) \leq R_j$$

Channel Requirements

$$\sum_{i=1}^{k} b_{ij}(t) \leq B_j$$

Designing a distributed scheduling algorithm to service all the processes efficiently is the goal of the system. Many multimedia applications may have deadlines associated with it. The scheduling algorithm must include deadlines as an important parameter while scheduling the processes. A distributed multimedia application may involve various *QoS* requirements from the network[8]. In this paper, we concentrate on scheduling processes at the end-host based on their resource requirements. We assume that the network offers the required *QoS* for the various processes[9]. We obtain the resource abstraction of a distributed process at a single node and consider this abstraction to be the requirements of a local process at the node.

3 Resource Manager

Every multimedia application has certain minimum resource requirements to be satisfied for optimum performance. These requirements can be specified by the processor time, memory buffer and I/O bandwidth required in a given time interval. For example, an MPEG video player which reads a file from disk and plays 30 frames of video per second, with say 10 ms processing time per frame, will need to be scheduled for 10 ms out of every 33 ms (1/30 s), buffer space in physical memory to hold one frame of video and enough bus cycles to read an MPEG frame from disk and write the decoded frame to the video framebuffer in the 33 ms. The OS must provide a way for the application to specify these requirements. In this case, the parameters specified are the processor time required in a given time interval, the size of the memory buffer, and the amount of data to be transferred through the bus in a given interval. In the rest of this section, we describe methods to specify and implement resource reservations for these three resources—processor time, I/O bandwidth and memory buffer resources.

3.1 Processor Time

To guarantee a certain amount of processor time to a process, the scheduler must take its reservations into account while scheduling it. The scheduling algorithm is typically priority based—using either fixed or dynamic priorities.

Fixed Priority Scheduling In a fixed priority scheduling scheme one must be able to assign priorities to processes such that each one is processed at its required rate. This can be done using the *rate monotonic* (RM) algorithm of Liu and Layland[13]. Here, the highest priority is assigned to the highest frequency task and the lowest priority is assigned to the lowest frequency task.

Let n be the number of tasks and let C_i and T_i be the computation time and period of task i. Liu and Layland[13] showed that all tasks will successfully meet their deadlines if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

When n is large, $n(2^{1/n} - 1) = \ln 2 = 0.69$. This means that as long as the total processor reservation is less than 69%, all tasks will meet their deadlines. This is a pessimistic bound. Here, a lot of computation time (31%), cannot be reserved. Lehoczky et.al[14] gave an average case analysis showing that tasks can be scheduled with upto 88% utilization.

Dynamic Priority Scheduling In contrast with fixed priority scheduling, the *earliest deadline*(ED) scheduling policy uses a dynamic priority scheme. The deadline of a task is defined as the end of the period during which the computation is to be finished. The task with the earliest deadline is scheduled to run. It can be shown that all tasks will successfully meet their deadlines under ED scheduling if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

In ED scheduling the priority of a particular process is inversely proportional to the difference between the deadline and the present time. This however, does not require continuous updation of priority, but the scheduler needs to make updates at the end of every task.

We see that with ED scheduling, it is possible to reserve upto 100% of the CPU time and still meet all deadlines whereas with the rate monitoring method, deadlines are guaranteed only if the reservation is less than 69%.

3.2 Bus Resources

In section 3.1, we studied how the processing time of a CPU can be efficiently shared between processes. But processor time is not the only resource for which applications compete. Multimedia applications often need to transfer large amounts of data to a peripheral device in a given amount of time. In order to achieve this, applications must have guaranteed access to the bus.

The OS will provide system calls for bulk transfer of data between the memory and the device. Whenever a particular application requires the transfer of data from a resource to memory or vice versa before a particular deadline, then we adopt a policy of bus reservation. Note that direct control of bus resources is not an efficient methodology since it reduces the throughput considerably. to overcome this, we follow a policy of *preemption*, where the resource involved in the transfer requests for bus access in advance. With a small predictable delay, the application effectively gets access to the bus. Here again, priorities can be allocated to resources. Effectively the data transfer is accomplished within the required deadline with high probability.

3.3 Memory Buffer Resources

Multimedia applications typically use large memory buffers. For performance reasons, it is advantageous if the memory buffer is present in physical memory

before it is accessed. If a buffer is not in memory when it is required, a page fault occurs, and the process sleeps until the page is fetched. This can cause a delay of the order of several ms—too large for a multimedia application. Memory buffers that are meant for holding data that is read from or to be written to a device should also be contiguous in physical memory so that they can be directly used for DMA transfers without a copy into kernel space. Support for reservation of memory buffers implies that the OS should provide a service by which the application can specify the size of the memory buffer (*memory capacity* of the application), the interval during which it should be present in physical memory and whether it needs to be contiguous.

In our method, the memory is logically split into 3 parts- A,B,C. A is used as a memory capacity reserve for multimedia applications. B is used as a common memory for all applications and C is used as a temporary memory for Distributed multimedia applications. The number of local and distributed multimedia applications dictate the logical split of memory between A, B and C. Whenever a process P requests a memory capacity M, then it is assigned a high proportion of M say pM in A and the rest of the capacity is adjusted in the common memory slot B. Memory capacity reservation is implemented by scheduling a transfer to bring the required pages into physical memory at a time before the time when the process wants to use the buffer. Processes that involve data transfer to a reserved memory buffer of an application are given high priority by the scheduler. The logical split is made dynamic with the help of memory location transfers between A, B and C. A change occurs in the logical split when memory of a particular finished task is allocated to another task.

3.4 Implementation of Resource Reservation in a Kernel

In this section, we propose a unified approach for managing all three resources—processor time, bus resources and memory buffer resources. The OS runs a *Resource Manager* process which keeps track of all existing reservations and performs admission control. All processes register their reservations with the *Resource Manager*. A particular registration of a process is a three tuple (a, b, c) denoting the reserves of the three resources. The *Resource Manager* verifies whether this reservation can be guaranteed at the current load (for example, if RM scheduling is used, the total reservation of processor time cannot exceed 69%) before admitting it. The *Scheduler* and the *Resource Manager* together implement the resource reservation service.

Reservations for processor time are handled directly. A reservation for a bus transfer of x bytes is handled with the help of *preemption*. An OS call is made to inform the device that data transfer of x bytes is accomplished within a particular deadline. A memory buffer reservation is always tied to a processor or a bus transfer reservation. It is required that the buffer be in physical memory *before* the task or the transfer is scheduled. The *Resource Manager* ensures that the buffer is in memory when it is required by the following method. It checks if the required space is present in the allocated space in A and if present , the transfer is immediately accomplished. If the buffer is not present, the memory is

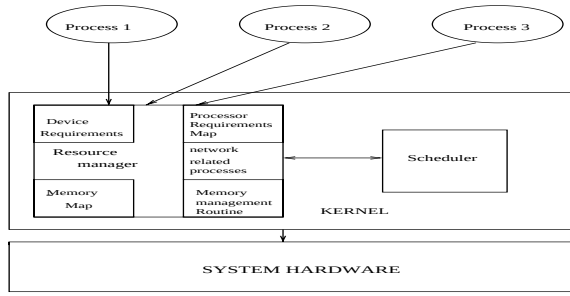

Fig. 1. Block Diagram of Implementation

allocated in B. This task is guaranteed enough time to evict an existing page and bring in the new page. Whenever the processor finds that the memory allocation in A for a particular processor is much below the expected value, a change in the logical split between A, B, C occurs.

The *Resource Manager* thus provides a unified interface to handle reservations for three classes of resources—CPU time, bus resources and memory buffer resources.

4 Kernel Optimizations for Distributed Multimedia Applications

Any Multimedia Application that runs over a distributed system communicates with the other systems present in the distributed domain. All these communications involve the use of the operating system at the end-hosts. Many multimedia applications like Videoconferencing or Video on Demand have real-time issues associated with them[11]. In order to effectively address these real time issues, one needs to minimize the overhead due to the data transfers between the end-host systems. In this section, we address issues relating to kernel optimizations at the end-hosts as part of providing efficient end-host support for communication.

4.1 A Connectionless and Reliable Transport Protocol — RUDP

We have implemented a transport protocol called Reliable UDP (RUDP) which is connectionless, reliable and provides timely delivery of packets in the kernel of our system to suit multimedia applications. This protocol supports efficient interprocess communication between two nodes in the distributed system without the overheads of connection oriented protocols like TCP.

Drawbacks of TCP TCP is poorly suited to frequent, short, request-response style traffic. Frequent connection setup and tear down costs burden servers with many connections in TIME_WAIT state. TCP's slow start algorithm for congestion avoidance interacts with short connections in a rather poor manner. The

overhead involved in connection management severely limits the scalability and performance of our system. These mismatches between the needs of multimedia applications and the services of TCP have caused increased latency in real-time applications.

Implementation We have developed RUDP using the unreliable and connectionless UDP as a base protocol and have provided reliable and in-sequence delivery over it. We use sequence numbers and acknowledgments in order to achieve this.

Every packet sent from a node is assigned a sequence number which depends on the IP address of the receiving node and the destination port. A packet is uniquely identified by its sequence number, destination IP address and port number. A sequence number list is associated with every port in the node.

Acknowledgments are handled by allocating a standard port in each node as an acknowledgment port. A node sends acknowledgments to the acknowledgment port of the sender. When a node receives an acknowledgment, it updates the sequence number list of the corresponding port.

We have modified the TCP's congestion control mechanism to provide a time-out adjustment suitable to our application. Our timeout mechanism is well suited over a LAN set-up. Whenever a packet loss occurs the timeout is multiplied by a constant factor γ typically around 2. We have based our time-out algorithm to explicitly suit the sudden variations in RTT. Our time-out algorithm aims at keeping the time-out to within a constant factor of the RTT. Our algorithm ensures that following a congestion the timeout drops back to its initial value rapidly when compared to TCP.

4.2 Aggressive Caching Mechanism

Memory capacities are allocated to all multimedia applications. We propose the use of aggressive caching in multimedia applications which need to obtain streaming multimedia data from a remote source. This mechanism is especially very useful to the Distributed Video on Demand applications where one can perform bulk transfer between the two end nodes. Through this mechanism, a process that needs to obtain data from a remote node can fill its memory capacity alloted in partition C, by performing bulk transport of the data from the remote node. The presence of a large cache at the end-host improves the performance of the above operation. The message is broken into various packets and the RPC transport at the other end reassembles the fragments. A single acknowledgment is used for all the fragments. On the other hand a collective set of all selectively rejected packets and sequence numbers of unreceived packets is sent as a collective negative acknowledgment to the packets. The sending RPC transport retransmits these fragments alone. This process is used to ensure reliable data delivery.

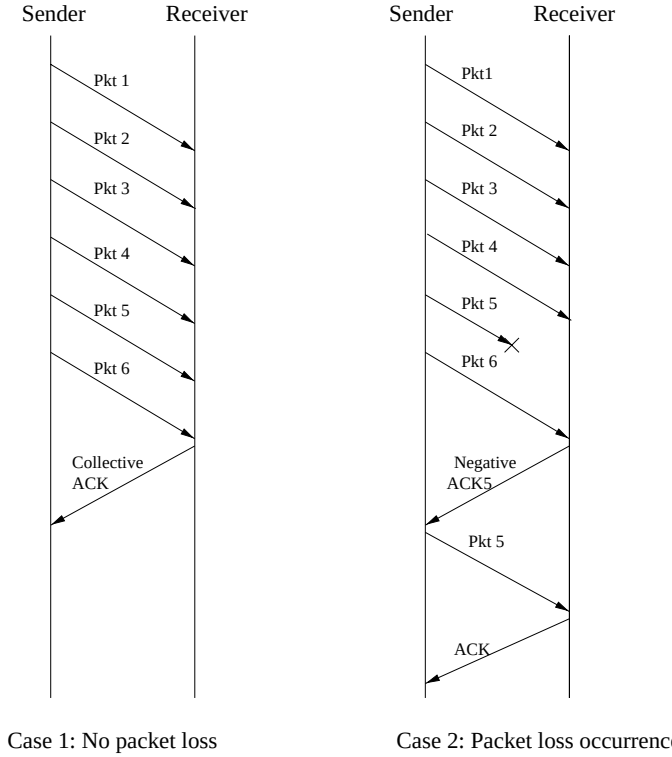

Fig. 2. Transfer mechanism used in Aggressive caching

5 Simulation Experiments

Experiments were performed to study the utility of operating system optimization on applications. Simulations were performed to analyze the effect of processor reservation support and communication optimizations on applications.

In the first experiment, we studied the impact of processor reservation strategies. We simulated a set of processes with different reservations and examined the long term behavior of the system. In our experimental setup, we assumed three processes A,B,C with reservations of 40%, 30% and 20%, with the rest of the processor time given to processes without reservations. The processor utilization of each process vs time is shown in Figure 3.

From the above graph, we see that the processor time available to each process stabilizes around its reservation. This means that each process is guaranteed its reserved processor time.

The performance of RUDP was tested over a distributed cluster comprising of 3 nodes connected by a 10Mbps Ethernet link. The protocol was implemented on the Linux OS. Message passing of varying sizes were accomplished and the

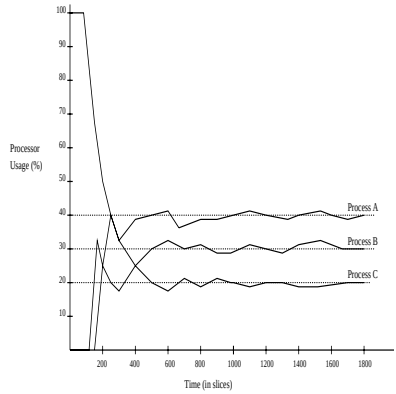

Fig. 3. Processor Time vs Time

performance of our protocol in comparison to TCP was studied. The timeout variation of our protocol have also been measured.

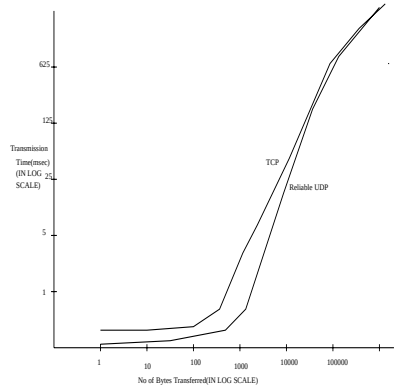

Fig. 4. Comparison of the Transmission time of our protocol with TCP

The results have clearly indicated that our protocol has a tremendous gain in performance for small and medium sized packets. The connection management overhead in our protocol is very less. The timeout mechanism of our protocol tries to model the fluctuation of the RTT of the system. When the network is relatively free the timeout is reduced to within a small factor of RTT to support multiple transmissions. Our algorithm supports fast recovery from congestion by ensuring that the timeout drops much rapidly to the steady value in comparison to TCP. This protocol is thereby well suited for real time applications running over a distributed system.

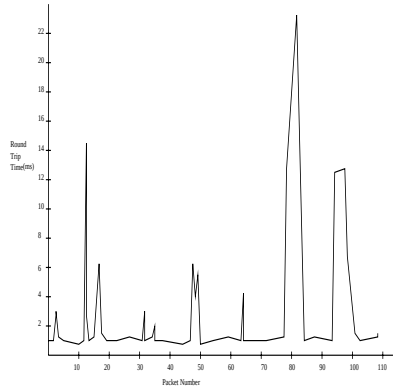

Fig. 5. Variation of RTT with time on a LAN

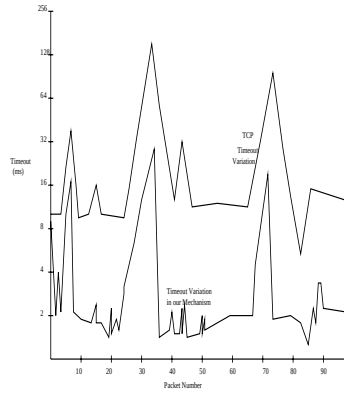

Fig. 6. Comparison of our Timeout mechanism with TCP

6 Conclusion and Future Work

In this paper, we have motivated the design of a reservation strategy of the resources of a system to support continuous media applications. The scheduling framework discussed, based on time -varying resource availability provides an effective way to specify real-time resource requirements. The Resource manager maintains the resources of the system. This reservation scheme can be directly extended to the distributed multimedia systems. In this reservation mechanism

each reserve contain reservations for various resources in the distributed systems. In such a model, one needs to even address network resource reservation issues. We have also proposed kernel optimizations to reduce the communication overhead between systems. The use of the aggressive caching mechanism reduces the number of data transfers between the two end-systems. We plan to use these optimizations effectively in the development of a Distributed Video on Demand application.

We are also working towards the development of a full-fledged operating system sensitive to multimedia applications. This operating system will include the scheduling features and the optimizations provided over here.

References

1. John.K.Ousterhout et.al, The Sprite Network Operating System, *IEEE Computer*, Vol. 21(2): 23-36, 1988.
2. Michael N.Nelson, BrentB.Welch, John.K.Ousterhout, Caching in the Sprite Network File System, *ACM Transactions on Computer Systems*, Vol.6(1): 134-154, 1988.
3. Clifford W.Mercer, Stefan Savage and Hideyuki Tokuda, Processor Capacity Reserves: Operating Systems Support for Multimedia Applications, *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.
4. Chen Lee, Ragunathan Rajkumar and Cliff Mercer, Experiences with Processor Reservation and Dynamic QoS in Real-Time Mach, *In Proceedings of Multimedia Japan*, March 1996.
5. Hideyuki Tokuda, Tatsuo Nakajima and Prithvi Rao, Real-Time Mach: Towards a Predictable Real-Time System, *Proceedings of USENIX Mach Workshop*, October 1990.
6. M.Ott, G.Michelitsch and D.Reininger and G.Welling, An Architecture for Adaptive QoS and its Application to Multimedia Systems Design, *Special Issue of Computer Communications on Building Quality of Service into Distributed Systems.*
7. T.Abdelzaher and Kang G.Shin, End-host Architecture for QoS Adaptive Communication, *IEEE Real-Time Technology and Applications Symposium*, June 3-5, 1998.
8. A.Campbell, C.Aurrecoechea, L.Hauw, A review of QoS Architectures, *ACM Multimedia Systems Journal*, 1996.
9. A.Campbell, G.Coulson and D.Hutchison, A Quality of Service Architecture, *ACM Computer Communications Review*, April 1994.
10. A.Campbell, R.Liao, Y.Shobatake, Delivering Scalable Flows using QoS Controlled Handoff, *ATM Forum*/97-0341.
11. A.Vogel et al., Distributed Multimedia and QoS: A Survey, *IEEE Multimedia*, Summer 1995, pp 10-18.
12. A.Hafid and G.V.Bochmann, An Approach to Quality of Service Management for Distributed Multimedia Systems, *International Conference on Open Distributed Processing (ICODP-95)*, Australia, Feb 1995, pp 319-340.
13. C.L.Liu and J.W.Layland, Scheduling Algorithms for Multiprogramming in Hard Real Time Environment, *JACM*, 20(1): 46-61, 1973.
14. J.P.Lehoczky, L.Sha and Y.Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour, *In Proceedings of the* 10^{th} *IEEE Real-Time Systems Symposium*, pp 166-171, Dec. 1989.