In Proceedings of Eighth IEEE Workshop on Real-Time Operatings Systems
and Software, in conjunction with 17th IFAC/IFIP Workshop on Real-Time
Programming, Atlanta, GA, pp. 144-150, May 1991.

Real-Time Scheduling of Sensor-Based Control Systems

David B. Stewart and Pradeep K. Khosla

Department of Electrical and Computer Engineering and
The Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213

Abstract: *Many sensor-based control systems are dynamically changing, and thus require a flexible scheduler. The rate monotonic (RM) real-time scheduling algorithm does not support such dynamic systems very well. 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. This paper proposes a new real-time scheduling algorithm, which we call maximum-urgency-first (MUF), which combines the advantages of the RM, EDF, and MLF algorithms. 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 three forms of deadline failures, and call failure handler routines for tasks which fail to meet their deadlines.The MUF scheduler has been implemented as the default scheduler of CHIMERA II, a real-time operating system being used to control sensor-based control systems both at Carnegie Mellon University and elsewhere. There are still many issues to be addressed with regards to the MUF algorithm. This paper also presents those issues, with possible approaches that should be investigated further.*

Keywords*: real-time scheduling, dynamic scheduling, sensor-based control, maximum-urgency-first, rate monotonic algorithm, earliest-deadline-first, minimum-laxity-first, CHIMERA II Real-Time Operating System.*

1 Introduction

Many sensor-based control systems are dynamically changing, and require a flexible scheduler. For example consider the case of a tactile sensor, on the end of a robotic manipulator, that is used to explore an object. Assume the tactile sensor has a resolution of 2^n by 2^m taxels, where n and m can vary dynamically between 1 and 5. When exploring uninteresting parts of an object, such as the straightedge of a table, it is desirable to use the lowest resolution, so that computation time is minimized and sample frequency is fastest, and the robot can follow the edge quickly. As the object becomes more interesting, such as the rounded corner of the table, it is desirable to increase the resolution of the tactile sensor. In doing so, the computational time required to process the data increases, and the frequency of data samples must be decreased (and not necessarily linearly).

The *rate monotonic* (RM) real-time scheduling algorithm does not support such dynamic systems very well. 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. This paper proposes a new real-time scheduling algorithm, called *maximum-urgency-first* (MUF). It combines the advantages of the RM, EDF, and MLF algorithms. 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 three types of timing failures, and call failure handler routines for tasks which fail to meet their deadlines.

Section 2 briefly describes the RM, MLF, and EDF algorithms, and Section 3 describes our new MUF scheduling algorithm. Section 4 describes our implementation of the MUF scheduler as the default scheduler of the CHIMERA II Real Time Operating System[8]. It is being used to control several sensor-based robotic systems at Carnegie Mellon University and elsewhere. The flexibility of the MUF algorithm provides many new possibilities in real-time scheduling of sensor-based control systems. A brief discussion in Section 5 is included to stimulate the reader's interest in the MUF algorithm, and to present a few ideas for further research.We also show that RM, EDF, and MLF are special cases of the MUF algorithm.

2 Related Work

Liu and Layland presented the rate monotonic algorithm as an optimal fixed priority scheduling algorithm, and the earliest-deadline-first and minimum-laxity-first algorithms as optimal dynamic priority scheduling algorithms.[4] Two disjoint scheduling philosophies emerged: static priority scheduling and dynamic priority scheduling. The former consists of using RM, while the latter uses either EDF or MLF as the baseline scheduling algorithm.

2.1 Rate Monotonic Algorithm (RM)

The *rate monotonic algorithm* is a fixed priority scheduling algorithm which consists of assigning the highest priority to the highest frequency tasks in the system, and lowest priority to the lowest frequency tasks. At any time, the scheduler chooses to execute the task with the highest priority. By specifying the period and computational time required by the task, the behavior of the system can be categorized *apriori*.

One problem with the rate monotonic algorithm is that the schedulable bound is less than 100%. The *schedulable bound* of a task set is defined as the maximum *CPU utilization* for which the set of tasks can be guaranteed to meet their deadlines. The CPU utilization of task P_i is computed as the ratio of worst-case computing time C_i to the period T_i. The total utilization U_n for n tasks is calculated as follows:

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

For the RM algorithm, the worst-case schedulable bound W_n for n tasks is

$$W_n = n\left(2^{1/n} - 1\right) \tag{2}$$

From (2), $W_1 = 100\%$, $W_2 = 83\%$, $W_3 = 78\%$, and in the limit, $W_\infty = 69\%$ (*ln* 2). Thus a set of tasks for which total CPU utilization is less than 69% will always meet all deadlines. All tasks will be guaranteed to meet their deadlines if $U_n \le W_n$. If $U_n > W_n$, then the subset of highest-priority tasks S such that $U_s \le W_s$ will be guaranteed to meet all deadlines, and will thus form the *critical set*. Note that the worst case values are pessimistic, and it has been shown that for the average case $W_\infty = 88\%$[3].

Another problem with RM is that it does not support dynamically changing periods very well, a feature required by some sensor-based control systems. For example, a task set with three tasks P_1, P_2, and P_3, of periods T_1 = 30ms, T_2 = 50ms, and T_3 = 100ms would have the following fixed priority assignment (from highest to lowest): P_1, P_2, P_3. Suppose the period of P_1 changes to T_1 = 75ms. Under the RM algorithm, we would require that the priorities of each task be reassigned to the ordering P_2, P_1, P_3, which violates the condition that priorities are static.

The problems with RM encourage the use of dynamic priority algorithms. Although many such algorithms exist, we restrict our attention in this paper to EDF and MLF.

2.2 Earliest-Deadline-First Scheduling Algorithm (EDF)

As the name implies, the *earliest-deadline-first* algorithm uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority, while the task with the latest deadline has the lowest priority. One advantage of this algorithm is that the schedulable bound is 100% for all task sets. Secondly, because priorities are dynamic, the periods of tasks can be changed at any time.

A major problem with the EDF algorithm is that there is no way to guarantee which tasks will fail in a *transient overload* situation. In many systems, although the average case utilization is less than 100%, it is possible that the worst-case utilization is above 100%, leaving the possibility of one or more tasks failing. In such cases, it is desirable to control which tasks fail and which succeed during such a transient overload. In the RM algorithm, low priority tasks will always be the first to fail. However, no such priority assignment exists with EDF, and thus there is no control of which task fails during a transient overload. As a result, it is possible that a very critical task may fail at the expense of a lesser important task.

2.3 Minimum-Laxity-First Scheduling Algorithm (MLF)

Our purpose in describing the *minimum-laxity-first* algorithm in this section is not to compare it to RM or EDF, but rather to introduce it as a basis for the *maximum-urgency-first* algorithm proposed in this paper. The minimum-laxity-first algorithm assigns a *laxity* to each task in a system, then selects the task with the minimum laxity to execute next. Laxity is defined as follows:

$$\text{laxity} = \text{deadline\_time} - \text{current\_time} - \text{CPU\_time\_still\_needed} \quad (3)$$

Laxity is a measure of the flexibility available for scheduling a task. A laxity of t_l means that even if the task is delayed by t_l time units, it will still meet its deadline. A laxity of zero means that the task must begin to execute *now* or it will risk failing to meet its deadline.

The main difference between MLF and EDF is that MLF takes into consideration the execution time of a task, which EDF does not do. Like the earliest-deadline-first algorithm, MLF has a 100% schedulable bound, but there is no way to control which are guaranteed to execute during a transient overload. In the next section, we present the MUF algorithm, which allows the control of task failures during transient overload, while maintaining the flexibility of a dynamic scheduler, and 100% schedulable bound for the critical set.

3 Maximum-Urgency-First Algorithm (MUF)

The *maximum-urgency-first* scheduling algorithm which we have developed is a combination of fixed and dynamic priority scheduling, also called *mixed priority* scheduling. With this algorithm, each task is given an *urgency*. The urgency of a task is defined as a combination of two fixed priorities, and a dynamic priority. One of the fixed priorities, called the *criticality*, has higher precedence over the dynamic priority. The other fixed priority, which we call *user priority*, has lower precedence than the dynamic priority. The dynamic priority is inversely proportional to the laxity of a task.

The MUF algorithm consists of two parts. The first part is the assignment of the criticality and user priority, which is done *apriori*. The second part involves the actions of the *MUF scheduler* during run-time

The steps in assigning the criticality and user priority are the following:

1. As with RM, order the tasks from shortest period to longest period.
2. Define the critical set as the first *N* tasks such that the total worst-case CPU utilization does not exceed 100%. These will be the tasks that do not fail, even during a transient overload of the system. If a critical task does not fall within the critical set, then *period transformation*, as used with RM,[6] can also be used here.
3. Assign *high* criticality to all tasks in the critical set, and *low* criticality to all other tasks.
4. Optionally assign a unique user priority to every task in the system.

The static priorities are defined once, and do not change during execution. The dynamic priority of each task is assigned at run-time, inversely proportional to the laxity of the task. Before its cycle, each task must specify its desired start time, deadline time, and worst-case execution time.

Whenever a task is added to the ready queue, a reschedule operation is performed. The MUF scheduler is used to determine which task is to be selected for execution, using the following algorithm:

1. Select the task with the highest criticalness.
2. If two or more tasks share highest criticalness, then select the task with the highest dynamic priority (i.e. minimum laxity). Only tasks with pending deadlines have a non-zero dynamic priority. Tasks with no deadlines have a dynamic priority of zero.
3. If two or more tasks share highest criticalness, and have equal dynamic priority, then the task among them with the highest user priority is selected.
4. If there are still two or more tasks that share highest criticalness, dynamic priority, and highest user priority, then they are serviced in a *first-come-first-serve* manner.

The optional assignment of unique user priorities for each task ensures that the scheduler never reaches step 4., thus providing a deterministic scheduling algorithm. We have yet to investigate the best method for assigning the user priorities.

To demonstrate the advantage of MUF over RM and EDF, consider the task set shown in Figure 1. We assume that the deadline of each task is the beginning of the next cycle. Four tasks are defined, with a total worst-case utilization of over 100%, thus in the worst-case, missed deadlines are inevitable. Figure 1(a) shows the schedule produced by a static priority scheduler when priorities are assigned using the RM algorithm. In this case, only P_1 and P_2 are in the critical set, and are guaranteed not to miss deadlines. Expectably, both P_3 and P_4 miss their deadlines. When using the EDF algorithm, as in Figure 1(b), tasks P_1 and P_2 fail. However, any task may have failed, since with EDF there is no way to predict the failure of tasks during a transient overload of the system.

With the MUF algorithm, all tasks in the critical set are guaranteed not to miss deadlines. In our example, the combined worst-case utilization of P_1, P_2, and P_3 is less than 100%, and thus they form the critical set. Only task P_4 can miss deadlines, because it is not in the critical set. Figure 1(c) shows the schedule produced by the MUF scheduler. Note the improvement over RM: because of a higher schedulable bound for the critical set, task P_3 is also in the critical set and thus does not miss any deadlines. Also, unlike EDF, we are able to control that the only task that may fail is P_4.

The choice of using MLF to calculate the dynamic priority instead of EDF enables the scheduler to detect missed deadlines. There are three failures which the MUF scheduler can detect:

1. A task has not completed its cycle when the deadline time has been reached;
2. A task was given as much CPU time as was requested in the worst-case, yet it still did not meet its deadline;
3. The task will not meet its deadline because the minimum CPU time requested cannot be granted. This case also requires that the minimum amount of CPU time required by a task is specified.

The first case is the standard notion of a missed deadline. The second case will detect bad worst-case estimates of execution time. The third case allows the MUF scheduler to make the most of its CPU time, and it will not start executing a task if that task has no possibility to finish before its deadline, thus providing the early detection of missed deadlines. Instead, the CPU time can be reclaimed for ensuring that other tasks do not miss deadlines, or to call alternate, shorter threads of execution.

4 Implementation

One concern of the MUF scheduler is the overhead that would be required during each reschedule operation. The overhead of the MUF scheduler can be reduced by encoding the algorithm into a single *urgency* value, hence the name of the algorithm. Figure 2 shows an *n*-bit urgency value, which was encoded using *c* bits for criticality, *d* bits for the dynamic priority, and *u* bits for the user priority. With such an encoding, the range of criticalities, dynamic priorities, and user priorities are 0 to 2^c-1, 0 to 2^d-1, and 0 to 2^u-1 respectively. The MUF scheduler must then only calculate a single dynamic priority for each task, then select the task with the maximum urgency. This encoding scheme can be used to implement the MUF algorithm as long as *c*, *d*, and *u* are all greater than or equal to $\log_2$*(max number of tasks in system).* Such encoding allows the maximum urgency scheduler to be implemented efficiently.

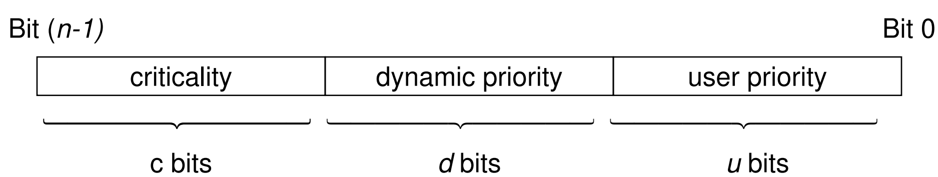


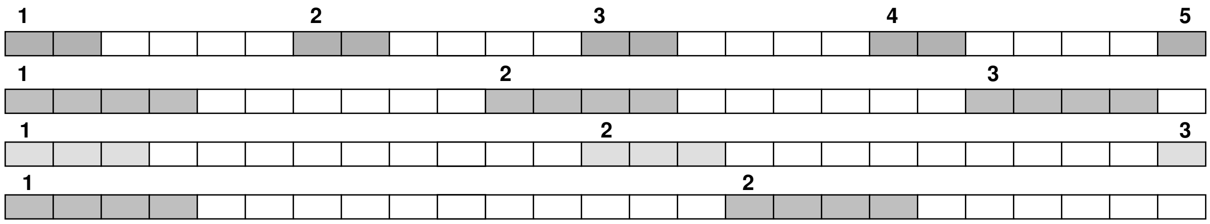
Figure 2: Encoded *n*-bit *Urgency* Value

We have implemented the MUF scheduler as the default scheduler of the CHIMERA II Real-Time Operating System [8]. CHIMERA II is being used both at Carnegie Mellon University and elsewhere, on a variety of sensor-based control systems, including the CMU Direct Drive Arm II [2] and the CMU Reconfigurable Modular Manipulator System [5].

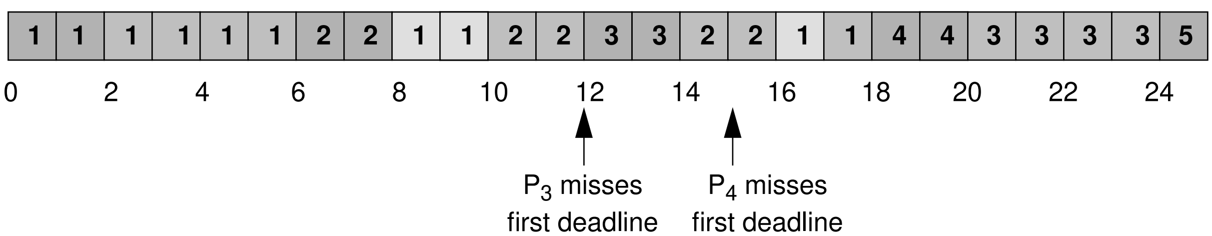
On an Ironics IV3220 Single Board Computer, with a 20 MHz M68020 processor, a reschedule operation with four ready tasks (excluding context switch time), takes 28 microseconds. The context switch takes another

Task	Priority(RM)	Criticality(MUF)	Period	CPU time	Utilization	Legend
P_1	High	High	6	2	33%	
P_2	Med High	High	10	4	40%	
P_3	Med Low	High	12	3	25%	
P_4	Low	Low	15	4	27%	

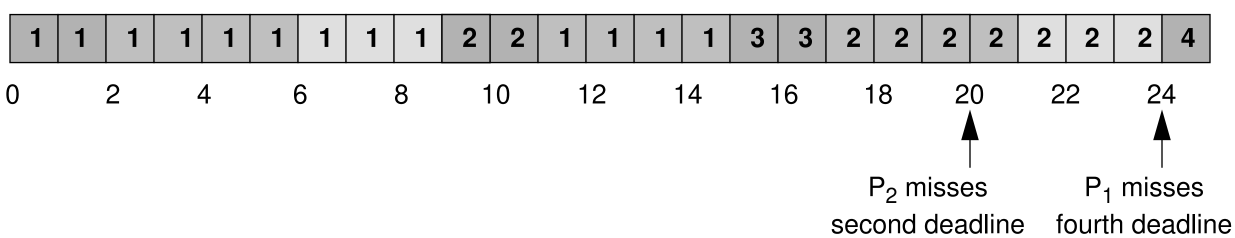
CPU time requested by each task (deadline is beginning of following cycle):



(a) Schedule generated when using *Rate Monotonic* algorithm:



(b) Schedule generated when using *Earliest-Deadline-First* algorithm:



(c) Schedule generated when using *Maximum-Urgency-First* algorithm:

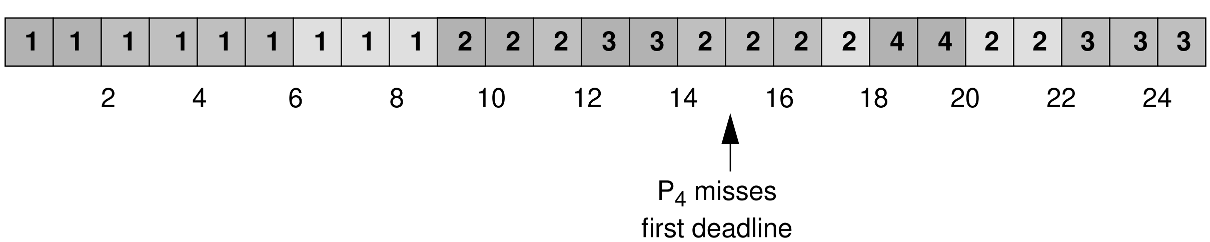


Figure 1: Example comparing RM, EDF, and MUF algorithms

66 microseconds, for a total of 94 microseconds. With a 1 millisecond clock, we maintain over 90% CPU utilization, while with a 10 millisecond clock we maintain over 98% utilization. This type of performance allows the scheduler to be used with sensor-based control applications that have tasks with frequencies as high as 1000 Hz.

Our implementation also offers deadline failure handling. Whenever a task fails to meet its deadline, an optional failure handler is called on behalf of the failing task. The failure handler can be programmed to execute either at the same or different criticality and user priority than the failing task. Such functionality is essential in predictable and fault-tolerant systems. Much emphasis in hard real-time systems has gone into ensuring that critical tasks always meet their deadlines. However, very little has been said about what to do about those tasks which fail to meet their deadlines during a transient overload. Possible actions include the following: aborting the task and preparing it to restart the next period; sending a message to some other part of the system to handle the error; modifying the priority of the task, and continuing its execution; performing emergency handling, such as a graceful shutdown of part of the system or sounding an alarm; maintaining statistics on failure frequency to aid in analyzing the system; in the case of iterative algorithms, returning the current approximate value regardless of precision. Any of these actions and other user-defined actions can be implemented using the deadline failure handling available with our MUF scheduler.

Estimating the execution time of tasks is often difficult. For example, most commercially-available hardware is geared towards increasing average performance via the use of caches and pipelines. Such hardware is often used to implement real-time systems. As a result, the execution time cannot necessarily be predicted accurately. Under-estimating worst-case execution times can create serious problems, as it is possible that a task in the critical set also fails. The use of deadline failure handlers is thus recommended for all tasks in a system, and not only those tasks which are not guaranteed. Our MUF scheduler provides this ability.

5 Discussion of MUF Algorithm

There are still many issues to be addressed with regards to the MUF algorithm. This section presents those issues, with possible approaches, which should be investigated further.

Aperiodic Events: The presentation of the MUF algorithm in this paper assumed only periodic tasks. Most real-time systems also have aperiodic events. Because MUF is a dynamic scheduler, aperiodic events can readily be included in the system without changing the basic MUF scheduler. However, such events must not cause tasks from the critical set to fail. Several methods have been adopted with the RM algorithm, including the *sporadic server* [7]. Similar methods can possibly be used with the MUF algorithm. For example, an aperiodic server can be given a criticality higher than the critical set. Its CPU utilization is included in the computation of the critical set, and calculated such that no critical tasks will miss deadlines if the aperiodic server does not use more CPU time than it is allotted. As with any periodic task, a deadline and maximum execution time is specified. If the server uses up all its time, then the failure handler is called, which replenishes the server's execution time, or blocks the server until its CPU time can be safely replenished

Task Synchronization: Real-time tasks are usually not independent. The sharing of limited resources, and the communication between tasks require appropriate synchronization or scheduling. With the RM algorithm, *priority ceiling protocol* [6] semaphores are often used for ensuring critical tasks still meet their deadlines in the presence of task dependencies. For the dynamic scheduling algorithms, both *dynamic priority ceiling protocol* semaphores [1] and resource scheduling [9] have been proposed. Adaptation of one or more of these methods to the MUF algorithm may be possible.

Varying Time Constraints: In the introduction of this paper we gave an example of dynamically changing timing constraints that may be encountered in sensor-base control systems. The MUF algorithm supports such tasks. Because the MLF algorithm is used to schedule tasks within the critical set, their frequencies and worst-case execution times can change dynamically. In order to guarantee tasks in the critical set in a dynamically changing environment, the worst-case utilization U_P for every task P is defined as $U_P = \max\,(C_{Pc}/T_{Pc})$, which is the maximum utilization of task P during any one cycle. Any combination of period and CPU execution time can then be used, as long as $C_{Pc}/T_{Pc} \le U_P$ for every cycle Pc. This is a signficant improvement over RM, where a change in period and CPU execution time may cause the critical set to change, even though utilization remains constant. When defining the MUF algorithm in Section 3, we first ordered tasks from shortest to longest period. This step can be relaxed, and MUF will still perform properly, but at the cost of non-critical tasks possibly failing unnecessarily.

Modular Design: In developing modular systems, it may be desirable to specify timing constraints on a per-module instead of per-task basis. For example, a module may consist of two dependent tasks, such that the combined worst-case CPU utilization is less than the sum of the utilization of the two tasks. In assigning priorities using RM, the frequency of the tasks plays an important role. However, with the MUF algorithm, only the utilization plays a role. By taking advantage of combined utilizations, it is possible to have a critical set in which the sum of the utilizations of all tasks within the set is *over* 100%, but the worst-case utilization for any one time slice is still less than 100%.

RM, EDF, and MLF as Special Cases of MUF: Without any modification, the MUF scheduler can also be used to schedule task sets using either the RM, EDF, or MLF algorithm. For example, instead of assigning criticalities according to the MUF algorithm, assign criticalities to tasks in the same way as priorities are assigned using the RM algorithm. Every task thus has a different criticality, and MUF behaves as a static highest priority scheduler. Deadline and execution times can still be specified to the MUF scheduler, even though they will not be used in the selection of which task to execute. This allows the MUF scheduler to still detect deadline failures, even though the RM priority assignment is used. Most fixed priority schedulers do not have such capabilities. If all tasks are given the same criticality, then the MUF scheduler behaves as an MLF scheduler. If the tasks all specify *zero* as the worst-case execution time, then the MUF scheduler reduces to an EDF scheduler, since the urgency of the task reduces to a function of deadline time. Note that in the latter case, early detection of deadline failures and failures due to under-estimating worst-case execution times cannot be detected.

6 Acknowledgments

The research reported in this paper is supported, in part, by U.S. Army AMCOM and DARPA under contract DAAA-2189-C-0001, by the Department of Electrical and Computer Engineering, and by The Robotics Institute at Carnegie Mellon University. Partial support for David B. Stewart is provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through a Graduate Scholarship. Special thanks also goes to Donald E. Schmitz, with whom numerous discussions eventually led to the development of some of the ideas presented in this paper.

7 References

[1] Chen, M.-I., and K. J. Lin, "Dynamic Priority Ceilings: a Concurrency Control Protocol for Real-Time Systems," Univ. Illinois at Urbana-Champaign, IL, Tech Report UIUCDCS-R-89-1511, April 1989.

[2] Kanade, T., P.K Khosla, and N. Tanaka, "Real-Time Control of the CMU Direct Drive Arm II Using Customized Inverse Dynamics," in *Proceedings of the 23rd IEEE Conference on Decision and Control*, Las Vegas, NV, December 1984, pp. 1345-1352.

[3] Lehoczky, J., L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proceedings 10th IEEE Real-Time Systems Symposium*, Santa Monica, CA, December 1989, pp. 166-171.

[4] Liu, C. L., and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment," *Journal of the Association for Computing Machinery*, v.20, n.1, January 1973, pp. 44-61.

[5] Schmitz, D. E., P. K. Khosla, and T. Kanade, "The CMU Reconfigurable Modular Manipulator System," in *Proceedings of the International Symposium and Exposition on Robots* (designated 19th ISIR), Sydney, Australia, Nov. 1988, pp. 473-488.

[6] Sha, L., J. P. Lehoczky, and R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemtive Scheduling," in *Proceedings 10th IEEE Real-Time Systems Symposium*, Santa Monica, CA, December 1989.

[7] Sprunt, B., L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *Journal of Real-Time Systems*, v.1, n.1, Nov 1989, pp. 27-60.

[8] Stewart, D. B., D. E. Schmitz, and P. K. Khosla, "Implementing Real-Time Robotic Systems using CHIMERA II," in *Proceedings of 1990 IEEE International Conference on Robotics and Automation*, Cincinnatti, OH, May 1990, pp. 598-603.

[9] Zhao, W., K. Ramamritham, and J. A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems", *IEEE Transactions on Software Engineering*, v.SE-13, n.5, May 1987, pp. 564-577.