PI Names: Krithi Ramamritham, John A. Stankovic, Victor Yodaiken
PI Institution: University of Massachusetts
PI Phone Number: 413 545-0196 or 413 545-0720
PI E-mail Address: krithi@nirvan.cs.umass.edu, stankovic@cs.umass.edu
Grant or Contract Title: Dependable Real-Time Systems
Grant or Contract Number: N00014-85-k-0398
Reporting Period: 1 Oct 87 - 30 Sep 91

1. Productivity Measures:

Refereed papers submitted but not yet published: 12
Refereed papers published: 37
Unrefereed reports and articles: 21
Books or parts thereof submitted but not yet published: 9
Books or parts thereof published: 3
Patents filed but not yet granted: 0
Patents granted: 0
Invited presentations: 42
Contributed presentations: 52
Honors received:
Technical Society Appointments: 1
Conference Committee Roles: 44
Editorships: 4
Prizes or awards received:
Promotions obtained: 1
Graduate students supported >=25% of full time: 14
Post-docs supported >=25% of full time: 1
Minorities supported (include Blacks, Hispanics, American Indians and other native Americans such as Aleuts, Pacific Islanders, etc., Asians, and Indians): 7
Best Available Copy
2. Summary of Accomplishments

2.1 Our Approach and its Advantages

Most current real-time operating systems contain the same basic paradigms found in timesharing operating systems and are simply stripped down and optimized versions of timesharing operating systems. For example, while they stress fast mechanisms such as a fast context switch and the ability to respond to external interrupts quickly, they retain the main abstractions of timesharing operating systems. In addition, very often today's real-time kernels use priority scheduling. Priority scheduling is a mechanism which provides no direct support for meeting timing constraints. For example, the current technology burdens the designer with the unenviable task of mapping a set of specified constraints on task executions into task priorities in such a manner that all tasks will meet their deadlines. Thus, when using the current paradigms together with priority scheduling it is difficult to predict how tasks, dynamically invoked, interact with other active tasks, where blocking over resources will occur, and what the subsequent effect of this interaction and blocking is on the timing constraints of all the tasks. Basically, currently used scheduling policies are inadequate for three main reasons: (1) they do not address the need for an integrated cpu scheduling and resource allocation scheme, (2) they don't handle the end-to-end scheduling problem, and (3) they are not used in a planning mode, thereby containing a myopic view of the system capabilities.

Because of these reasons, even though some commercial and research operating systems have bounded execution times and hence are predictable in a low-level sense, they do not provide the techniques or tools to support application level predictability. This is what our approach attempts to do.

Specifically, our approach is based on categorizing the types of tasks that occur in a real-time system and then providing the necessary support for them in an integrated fashion. Tasks found in real-time applications can be categorized on the basis of their interaction with and impact on the environment. This gives rise to two main criteria: importance and timing requirements. This leads to the identification of three types of tasks: critical tasks, essential tasks, and non-essential tasks. Our kernel then treats the different classes
of tasks differently thereby reducing the overall complexity. The technical underpinnings of the Spring paradigm can be summarized as follows:

- Tasks are part of a single application with a system-wide objective. The types of tasks that occur in a real-time application are known a priori and hence can be analyzed to determine their characteristics (such as their importance, as well as their timing and resource requirements).

The scheduling and allocation schemes adopted in Spring use this information in pre-allocation and for on-line guarantee of timing constraints.

- Predictability should be ensured so that the timing properties of both individual tasks and the system can be assessed (in other words we have to be able to categorize the performance of tasks and the system with respect to properties such as timing and fault tolerance).

In Spring, predictability is achieved by a combination of schemes, including resource segmentation/partitioning, functional partitioning of application tasks, executing system support tasks on a separate processor, and the use of integrated scheduling algorithms.

- Flexibility should be ensured so that system modifications and on-line dynamics are more easily accommodated.

  Flexibility/adaptability is improved in Spring by dynamic (decentralized) task scheduling, and the use of meta-level control.

- The value of tasks executed should be maximized, where the value of a task that completes before its deadline is its full value (depends on what the task does) and some diminished value (e.g., a very negative value or zero) if it does not make its deadline. Fairness and minimizing average response times are not important metrics for tasks with hard timing constraints. We should note that while some of the work at CMU captures task values through (arbitrary) value functions, for efficiency reasons, we capture values through task categorization and importance factors. In addition we explicitly account for resource requirements of tasks.

  Spring maximizes the value of tasks executed through resource preallocation for critical tasks and the use of dynamic scheduling algorithms (that take task importance values into account) for essential and non-essential tasks.

An important aspect of our paradigm is that the system maintains significant semantic information concerning the application in on-line data structures. Consequently, task group information, resource requirements, importance, timing constraints, and fault tolerance requirements are available to the system algorithms such as the dynamic guarantee algorithm or recovery algorithms. These algorithms can then make use of this information to directly support the application.

It is important to note that the system structures and algorithms in the kernel are generic; it is just the specific task information which is in these data structures that is application
dependent. However, we are very interested in showing how the generic kernel can be used in domain specific applications such as process control, avionics, mobile robotics and command and control.

2.2 Specifics of the Accomplishments

Based on the paradigm provided in the previous section, we (1) made great strides in developing a sound approach to scheduling tasks in complex real-time systems, (2) developed a real-time operating system kernel, a preliminary version of which is in operation, and (3) we have developed a modal primitive recursive arithmetic for specifying and verifying real-time systems.

To study different scheduling approaches and to evaluate different real-time architectures, we have developed a simulation testbed with a graphical front-end that allows a system developer to select the task management approaches appropriate for a given application. Using such a testbed, the efficacy of the scheduling algorithms has been carefully demonstrated via extensive simulation studies. Also, we have conducted mathematical analysis of the algorithms to understand their worst-case behavior. Finally, and most importantly, some of the algorithms have also been implemented as part of the kernel.

We now provide a brief summary of our work on the scheduling algorithms, on the Spring kernel, and on formal aspects of real-time systems.

2.2.1 Scheduling Strategies for Complex Real-Time Systems

To date, the main results of our scheduling work include:

- **The adoption and development of guarantee, a notion fundamental to predictable scheduling.** A task is guaranteed by constructing a plan for task execution whereby all guaranteed tasks meet their timing constraints. A task is guaranteed subject to a set of assumptions, for example, about its worst-case execution time, and the nature of faults in the system. If these assumptions hold, once a task is guaranteed it will meet its timing requirements.

- **Algorithms for the guarantee of dynamically arriving tasks.** Our guarantee algorithm has the potential to deal with tasks that have deadlines, resource requirements, precedence relationships among tasks, tasks with varying importance levels, tasks with different fault tolerance requirements, and tasks that are preemptable. Many versions of the algorithms exist each dealing with a subset of these task characteristics. As mentioned earlier, one of our goals is to produce comprehensive schemes that can deal with tasks having all these complex characteristics.

- **Analysis of the quality of the schedules produced by the algorithm.** Both the ability to generate feasible schedules and the quality of the generated feasible schedules, expressed
in terms of the schedule length, are important metrics for scheduling algorithms. Our theoretical analysis identified several ways in which the basic algorithm can be improved even further.

- **Reclaiming unused time and resources when tasks complete early.** The guarantees are based on worst case computation times, but when a task finishes early, as would typically be the case, the unused CPU and resource time may be reclaimed and used.

- **Distributed scheduling and meta-level control.** A suite of distributed scheduling algorithms has been developed and evaluated. We have also hypothesized the usefulness of a meta-level controller that can select the heuristic appropriate for a given system (state). We have developed a local area network architecture for communication in real-time systems that incorporates the abstractions of real-time datagrams and real-time virtual circuits. We have also developed several real-time communication protocols for multiple-access channels. We have yet to integrate these protocols with the distributed scheduling algorithms.

- **Static allocation and scheduling of safety-critical tasks.** We have developed an algorithm that is suitable for the static allocation and scheduling of complex, safety-critical periodic tasks. Besides periodicity constraints, tasks handled by the algorithm can have resource requirements and can possess precedence, communication, as well as fault tolerance constraints. Extensions necessary to accommodate dynamic aperiodic arrivals have also been worked out.

The notion of guarantee is aimed at achieving predictability. Adaptability and flexibility are supported by a collection of dynamic task management techniques. These include dynamic guarantees with task characteristics identified at invocation time, distributed scheduling, meta-level control, and resource reclaiming.

It is important to mention that we have also been able to apply our ideas from real-time scheduling algorithms to develop scheduling and concurrency control techniques for real-time database systems. This work has produced significant results. It is also important to point out that this work represents the first set of results for a real-time database in an experimental testbed environment. All other results we are aware of in this area are based on simulation studies.

### 2.2.2 The Spring Real-Time Operating System Kernel

The Spring kernel stresses the real-time and flexibility requirements, and also contains several features to support fault tolerance. Our approach to supporting this new paradigm combines the following ideas resulting, we believe, in a flexible yet predictable system:

- resource segmentation/partitioning,
- functional partitioning,
- selective preallocation,
- *a priori* guarantee for critical tasks,
• an on-line guarantee for essential tasks,
• integrated cpu scheduling and resource allocation,
• use of the scheduler in a planning mode,
• the separation of importance and timing constraints, e.g., a deadline,
• end-to-end scheduling, and
• the utilization of information about tasks at run time including timing, task importance, fault tolerance requirements, etc. and the ability to dynamically alter this information.

Based on these principles, we are currently developing the kernel for distributed real-time system. The current configuration is composed of a network of multiprocessors each running the Spring kernel. Each multiprocessor contains one (or more) application processors, one (or more) system processors, and an I/O subsystem. Application processors execute relatively high level application tasks which have been previously guaranteed. System processors offload the scheduling algorithm and other OS overhead from the application tasks both for speed, and so that external interrupts and OS overhead do not cause uncertainty in executing guaranteed tasks. The I/O subsystem is partitioned away from the Spring kernel and it handles I/O and interactions with sensors.

Not surprisingly, the main components of the kernel can be grouped into task management and scheduling, memory management, and intertask communication. While this sounds similar to many other kernels, the abstractions supported are quite different and represent a new paradigm for real-time operating systems. To enhance predictability, system primitives have capped execution times, and some primitives execute as iterative algorithms where the number of iterations it will make for a particular call depends on its capped execution time and on other state information including available time.

Nodes in the Spring distributed system are connected via Ethernet (for non-real-time traffic) as well as via a predictable replicated memory based on SCRAMNET (developed by SYSTRAN) - a fiber optic register insertion ring (for real-time traffic). The development of software required to support SCRAMNET is currently underway.

2.2.3 Formal Approaches to Real-Time Systems

In the area of formal approaches to real-time systems, our work has attempted to find new techniques for describing real-time systems as finite state machines. We have developed methods of specifying the behavior and structure of large-scale state machines symbolically, without enumerating states.

We use state-dependent integer functions that implicitly reference a state machine. Each boolean function specifies the family of state machines which cause it to evaluate as true. This approach allows for very compact, parameterizable specifications of state machines which are beyond the reach of traditional methods. We can conveniently compose these functions to specify the behavior of systems composed from concurrent subsystems. Composite specifications also specify finite state machines and can be used as parts of more complex specifications. Our composition method has a particular advantage over other methods in
that it does not require any assumptions about how components are interconnected or scheduled and the composite specification also specifies a finite state machine. Specifications are just integer-valued functions, thus verification takes place in the familiar context of integer mathematics.

The compactness of our specifications of state machines allows us to model the behavior of real-time systems with state machines that change state in response to the passage of physical time, e.g., every nano-second. The flexibility of our composition method allows us to describe real-time systems containing subsystems which change state at differing rates, and which may contain clocks that drift apart at different rates.

We have applied our formalism to specify real-time priority queues, a fault-tolerant broadcast protocol, and the arbitration mechanism of Futurebus+. 
3. Summary

For the most part, we accomplished what we outlined in the proposal submitted at the beginning of the ONR initiative. Many versions of the dynamic and static scheduling problems have been studied and solutions have been evaluated and in some cases even implemented on the Spring kernel. In addition, a few new and interesting problems came to our attention as a result of our endeavor to implement our scheduling algorithm on the kernel. These include the resource reclaiming problem and the problem of predictable and bounded synchronization in a shared-bus multiprocessor system. In addition, our work on formal aspects of real-time systems has matured through its application to many nontrivial examples.
4. Publications, Presentations, and Reports:

4.1 Publications and Reports:


PI Names: Krithi Ramamritham, John A. Stankovic, Victor Yodaiken
PI Institution: University of Massachusetts
PI Phone Number: 413 545-0196 or 413 545-0720
PI E-mail Address: krithi@nirvan.cs.umass.edu, stankovic@cs.umass.edu
Grant or Contract Title: Dependable Real-Time Systems
Grant or Contract Number: N00014-85-k-0398
Reporting Period: 1 Oct 87 - 30 Sep 91

4.2 Presentations, Etc.:

1. Panelist (Ramamritham):

   Conference on Very Large Databases, Barcelona, Spain, Sep 91. (chair)
   9th and 10th International Conference on Distributed Computing Systems, June 1989
   and 1990.
   9th International Conference on Distributed Computer Systems, June 1989.

2. Panelist (Stankovic):

   NSF External Review Panel for initiation grants
   COINS Industrial Affiliates meeting, May 4-5, 1990.
   Workshop on Real-Time Operating Systems, (2 panels), May 1989; EuroMicro Work-
   shop on Real-Time, Como, Italy, June 1989.

3. Conference Committees, etc. (Ramamritham):

   Program Chair:
   Program Co-Chair:

   Member, Conference Program Committee:
Member, Program Committee, 8th DCS, 4th and 5th International Conference on Data Engineering.

Conference Session Chair:

4. Conference Committees, etc. (Stankovic):
Organizing and Program Committee, Workshop on Artificial Intelligence in Real-Time, April 1991.
Program Committee, AIDA '90, November 1990.
IEEE Steering Committee for Distributed Computing.

5. Conference Presentations (Ramamritham):
Conference on Very Large Databases, Barcelona, Spain, Sep 1991.
Program Co-Chair: International Conference on Distributed Computing systems, June 1990, Paris, France.

6. Conference Presentations (Stankovic):
Conference on Very Large Databases, Barcelona, Spain, Sep 1991.
EuroMicro Workshop, Denmark, June 1990.
Invited speaker, ARTEWG meeting, four-hour presentation of Spring Project results, May 1990.
8th DEC Conference, San Jose, CA June 1988
Real-Time Systems Symposium, December 1987
Large Grain parallelism Workshop, October 1987
Real-Time OS Workshop, July 1987
IEEE CS Speaker, Distinguished Visitor Program, 1987-88.

7. Other Presentations (Ramamritham):

IEEE Chapter, Bangalore, India, August 1991.
Oregon Graduate Institute, OR, January, 1991.
University of Pennsylvania, PA, October, 1990.
IEEE Workshop on Dextrous Robot Hands, April 1988; Technical University, Vienna, Austria, June 1988.

8. Other Presentations (Stankovic):

As part of CRICCS preparation, traveled to California and gave three presentations at Stanford, Rockwell and FMC and visited JPL.
Northeastern University, Distinguished Lecture Series, May 1989.
Purdue University, CS Dept, Seminar, February 1989.
University of Michigan, Distinguished Lecture Series, February 1989.
University of Maryland, Seminar, November 1989.
Texas Instruments, Seminar program, Dallas, August 1988
8th DCS Conf. panel, San Jose, CA, June 1988
University of Maryland, CS Seminar, March 1988
University of Connecticut, CS Seminar, November 1987

9. Board Memberships, etc. (Ramamritham):

Visiting Fellowship, Science and Engineering Research Council, UK September 1987 

10. Board Memberships, etc. (Stankovic):

ACSIOM Board of Directors, 1989-present.
Ready Systems, Senior Scientific Advisory Board, current.
Advanced System Technologies, Senior Technical Advisory Board, current.

11. PhD Students - Thesis Chairman (Ramamritham):

12. PhD Students - Thesis Chairman (Stankovic):
K. Arvind, (Co-Chair with Ramamritham), Protocols for Distributed Real-Time Sys-

13. Awards (Stankovic):
IEEE Computer Society's Meritorious Service Award, 1991
International Advisory Committee, *Journal of Computer Science and Informatics*,
Computer Society of India, May 1990.
Full Professor, Dept. of Computer and Information Science, September 1, 1990.
5. Research Transitions and DoD Interactions:

Indirect Research Transitions:

Stankovic has been working with Applied Technology Systems for The Naval Surface Warfare Center, White Oak Lab on integrating real-time scheduling results from Spring and other research groups into a Knowledge Based DOS Assistant.

Ramamritham has been collaborating with the researchers at AT&T Bell Labs, Murray Hill and has completed extending Concurrent C with constructs for specifying real-time constraints. These incorporate the notion of guarantee as well.

We are also working on a new tutorial text entitled, Advances in Real-Time Systems. The text has been approved by the IEEE.

The Spring project formed one of the focal points of the department’s Science and Technology Proposal to NSF. It was to provide the necessary systems support for complex real-time systems. As part of this proposal, interactions with several industrial participants have been initiated. One of these was Texas Instruments. Some of their real-time work has been influenced by Spring scheduling algorithms and we continue to work with them.
6. Description of Software and Hardware Prototypes:

We further developed the software simulation testbed for distributed real-time systems. Documentation has been completed, and the new graphics front-end which animates the scheduling algorithms we develop has also been developed for this testbed. We further enhanced our Software Generation System which enables the specification of task requirements and the subsequent creation of the Spring Kernel with this information embedded in the runtime data structures. Subsequent enhancements are being planned for this tool in order to handle task groups.

RT-CARAT, a distributed database testbed, has many real-time transaction protocols implemented and running. Further enhancements, such as real-time buffer management, incorporation of abstract data type objects, and provision for structured transactions are in progress.

A prototype of Real-Time Concurrent C is nearing completion and should be available for distribution within the next few months.

Development of the distributed version of the Spring Kernel is currently in progress using Systran's SCRAMNet, a replicated shared memory architecture that uses a fiber optic register insertion ring, running at 150 MBits/sec. Each node has 2 MB of shared common memory, and writes to this memory are broadcast (circulated) about the ring. We are currently implementing the IPC primitives with SCRAMNet. SCRAMNet presents some attractive features for implementing distributed scheduling and for realizing fault tolerance. These are currently under investigation.