Improving Support for Multimedia System Experimentation and Deployment[*]

Douglas Niehaus

Information and Telecommunication Technology Center
Electrical Engineering and Computer Science Department,
University of Kansas, Lawrence KS 66045, USA
niehaus@eecs.ukans.edu

Abstract. New applications with increasingly complex complex requirements present significant challenges to existing operating systems, motivating the design and implementation of new system abilities. Multimedia applications are excellent driving problems because they exhibit stringent temporal constraints and non-trivial interactions among applications within and across machine boundaries. This paper describes work done under several projects conducted by the author in the Information and Telecommunication Technology Center at the University of Kansas which had the need for real-time software in common. This common set of requirements motivated the development of real-time extensions to Linux capable of scheduling events with microsecond accuracy, and the creation of a facility called the Data Stream Kernel Interface capable of gathering performance data with sub-microsecond precision. This paper provides an overview of these new system abilities, how we have used them, and discusses their relevance to the design, implementation, and evaluation of multimedia systems.

1 Introduction

A new class of applications with increasingly complex requirements for an increasing range of system capabilities is emerging as the worldwide Internet continues to increase in size and complexity, and the power of the computing platforms in the homes of typical members of the public increases to a level that would have seemed a fantasy even a few years ago. Multimedia applications are excellent examples of this new class of programs because they exhibit most of the important characteristics of class members, including: increasingly stringent temporal constraints, significant interaction among other applications on the same machine, and significant interaction with network resources.

The challenges presented by such applications, which must be met by operating systems in the next few years, are likely to require significant changes in current system architectures. This is an important point that many researchers appreciate and a wide range of research projects are investigating different approaches to fundamental changes in system and computing architectures [1, 6, 9].

[*] This work was supported in part by grants from Sprint Corporation.

Investigators at Microsoft Research, for example, have recently written a position paper advocating their assertion that "it is time to reexamine the operating system's role in computing" [3]. While the work presented here is not as aggressive as a fundamental change in the operating system's role, our experience has indicated that several of the things we have done are useful.

The work discussed in this paper was done under several past and present research projects conducted by the author and his colleagues in the Information and Telecommunication Technology Center at the University of Kansas. Some projects addressed aspects of the design, implementation, and evaluation of next generation telecommunication networks. Other projects considered how emerging distributed applications might use these networks and how distributed software technology can be applied to the design, implementation, and deployment of these next generation networks. In the course of this work, an interesting set of results emerged which are relevant to the problem of improving support for experimentation and deployment of multimedia systems.

Specifically, we have spent the last several years working with the design, development, and evaluation of several software systems which must satisfy a variety of firm real-time execution constraints. We have concentrated on the use of commercial off-the-shelf (COTS) hardware and free, or at least inexpensive, software. Development of application and operating system software which must satisfy fine grain temporal constraints has also required us to significantly improve our ability to monitor events within all layers of the system in support of evaluating our work.

We believe that this work provides a useful approach to important components of experimental and deployment platforms for multimedia. First, it is inexpensive and open source, being based on the Linux OS, GNU software, and COTS hardware. This maximizes both the set of people who can use it and the ease of collaboration. Second, it is capable of scheduling and monitoring events at a significantly finer temporal resolution than most current systems, and certainly finer than current consumer systems. Third, continued development and use of this system has demonstrated that it provides these capabilities at a reasonable cost in overhead, and that it can be used for a wide range of interesting and innovative applications. Finally, preliminary tests support the idea that these capabilities can contribute to the support of distributed multimedia and the integration of network quality of service.

The rest of this paper first discusses system support for increased temporal resolution and scheduling applications with firm real-time constraints in Section 2, and then considers support we have created for evaluating the performance of such applications in Section 3. Section 4 describes several ways in which we have used these facilities in past and current research, and Section 5 presents some experimental results which demonstrate the viability of our approach. Section 6 points out where more detailed discussion of related work in several categories can be found. Finally, Section 7 briefly presents our conclusions and discusses future work.

2 System Support

Experimentation with systems supporting multimedia and other applications with more complex and demanding requirements for system support than conventional systems consider normal requires significant extension to the system support for controlling and monitoring events. We created the KU Real-Time (*KURT*) system in response to the need in several projects for a low cost system with microsecond temporal resolution in both timestamp data gathering and event scheduling. The first application of this ability was the ATM Reference Traffic system, which we use to help evaluate the performance of ATM networks, and which is discussed in Section 4. Later projects motivated us to expand the basic capabilities into a more generally applicable system which is the current version of KURT. This work is relevant for multimedia and quality of service researchers because it significantly increases the capabilities of a low cost and convenient experimental and deployment platform.

KURT is based on the freely available Linux operating system, modified in several important ways. First, by running the hardware timer as an aperiodic device we have increased the system's temporal resolution without significantly increasing the overhead of the software clock. We call this the *UTIME* component of KURT, which alone increases the utility of Linux for soft real-time applications.

The KURT system builds on top of UTIME, adding a number of features. KURT enables the system to switch between three modes: **normal mode**, in which it functions as a normal Linux system, **mixed real-time mode**, in which it executes designated real-time processes according to an explicit schedule, serving non-real-time processes when the schedule allows, and **focused real-time mode**, in which only real-time processes are run. Focused real-time mode is useful when KURT is being used as a dedicated real-time system satisfying the most stringent execution constraints. When real-time applications share a generic desktop workstation, KURT's mixed real-time mode is most often used, since it allows both real-time and non-real-time processes to run. When in either real-time mode, real-time processes can access any of the system services that are normally available to non-real-time processes. It is important to note that the use of different subsystems introduces different levels of scheduling distortion. Our current work includes analysis of how different subsystems create these scheduling distortions and the development of methods to control or eliminate them.

The basic approach to KURT implementation and its current scheduling resolution have been presented elsewhere. Srinivasan describes the original KURT design [14, 15], while Hill describes several modifications of KURT which improve its behavior in a number of ways [7, 8]. The essential points drawn from these sources, include: UTIME modifications increase KURT's temporal resolution to microseconds, events can be scheduled with an accuracy varying from tens to hundreds of microseconds depending on the Linux subsystems in use, and that KURT provides a powerful and flexible experimental platform. The rest of this section will discuss KURT modifications by Hill not addressed in the previous conference paper [14].

2.1 Scheduling Extensions

Kurt provides a simple and well structured interface for implementing different scheduling modules. Hill added two new modes to the KURT real-time process scheduling module to accommodate distributed real-time applications: *event-driven* and *buffered-periodic* [8]. Both of these scheduling modes were evaluated using a simple distributed multimedia system as described in Sections 3 and 4.

The event driven mode waits until a packet arrives, and then decides if the process waiting for the packet can be scheduled. KURT gives preference to processes that are already explicitly scheduled, but if no real-time process is already scheduled to run,the event-driven process is chosen for execution as though it had been explicitly scheduled. If there is already a running or pending explicitly scheduled process, the event-driven process is allowed to run in the next available slot in the real-time schedule.

The mode supporting periodic scheduling with buffering was created in response to the observation that a simple periodic process was able to find and process the intended frame a large percentage of the time, but that the arrival jitter of the network was too large to permit excellent performance. In this mode the system allows one or more frames associated with a periodic KURT process to accumulate before it begins execution. Our experiments in the LAN environment showed significant advantage to a buffer of only a single frame.

Atlas has also recently added a scheduling mode to KURT implementing her Statistical Rate Monotonic Scheduling (SRMS), which is a generalization of the classical RMS results of Liu and Layland for periodic tasks with highly variable execution times and statistical QoS requirements [2]. The SRMS scheduler is a simple, preemptive, fixed-priority scheduler. The paper describes the technical issues that arose while integrating SRMS into KURT Linux and presents the API developed. One of the issues cited, explicit scheduling of device driver bottom halves, is part of our current work to reduce scheduling distortion. We have not yet integrated the SRMS work into our generic KURT release but hope to do so in the near future.

2.2 Networking Extensions

The standard ATM implementation under Linux supports unspecified bit rate (UBR), or best effort, and constant bit rate (CBR) traffic. Data can be accessed directly as either individual cells, termed AAL0, or AAL5 packets. In addition, support is provided for IP over ATM and LANE. Hill extended the ATM implementation to include variable bit rate (VBR) traffic and to use raw AAL5 packets as the transport mechanism to minimize protocol processing overhead [8]. Multimedia data streams are usually compressed in some way, and thus typically exhibit variation in their transmission rate. VBR is thus the most commonly favored QoS for video streams. VBR service can be broken down further into two categories: real-time (rtVBR) and non-real-time (nrtVBR). Real-time VBR provides two QoS parameters: cell-delay variation (CDV) and cell transfer delay (CTD). CDV provides an upper bound on the variation in the delay cells experience while CTD provides an upper bound on the maximum end-to-end delay for cells.

Hill's modifications only support non-real-time VBR due to lack of support in the current version of the ATM switch control software. However, for the purposes of experimentation in a LAN environment, nrtVBR provided sufficient support for multimedia processes with periods in the tens of milliseconds range as arrival jitter was non-trivial but not excessive. Future work includes experiments with multimedia streams across WAN connections, which may reveal the advantage of more stringent rtVBR support.

2.3 Middleware Support

KURT also provides a good platform for our experimental work with CORBA middleware. Gopinath worked on the implementation and evaluation of real-time CORBA end-systems[5]. This work included the implementation of a simple real-time scheduling service on top of KURT, and simple real-time extensions to OmniORB[12]. We are currently using the KURT extensions, in conjunction with data collection capabilities discussed in Section 3, as part of a project creating tools for creating and conducting CORBA performance evaluation experiments[11]. We are extremely interested in the resulting capability to examine the influence of system support on overall ORB performance and to consider the design and implementation of explicit support for CORBA event services and the integration of kernel I/O and ORB level I/O management. This latter work is currently being done in the context of the OmniORB and the ACE ORB[13]. We intend to use a CORBA based version of our simple multimedia system as one of the driving applications as these efforts progress, which will provide for an interesting comparison between CORBA and non-CORBA based performance of essentially the same scenarios.

3 Experimentation Support

Many of the new facilities and possibilities for experimentation and support of applications discussed in Section 2 are interesting, promising, and initial results justify our current interest. The fact remains, however, that they are subjects of research because their ultimate utility, and thus their ultimate inclusion, in system of the future supporting multimedia and other sophisticated applications is as yet only potential. Further work, and further proof of their advantages, is required before potential can become reality.

Yet, the nature of the extended system capabilities means that the methods used to evaluate them must also be extended. For example, evaluating the KURT extensions to Linux requires the ability to gather data for performance evaluation with timestamps having sub-microsecond resolution. This is three or four orders of magnitude finer resolution than current commercial systems commonly provide. A more subtle difficulty is that evaluating management of events at increased temporal resolution is accompanied by a significant increase in the *volume* of performance data collected.

Data collection from internal levels of operating systems has traditionally been done using *ad hoc* and *data source specific* methods that are often of limited utility or unavailable to others interested in similar information. This represents a tremendous waste of time and effort since many problems are solved repeatedly by different investigators. We decided to reduce this waste of effort, while making available the broader range and greater volume of information we required, by creating a platform independent interface to support customizable collection of performance data from the operating system's internal subsystems. The interface which we have developed is called the Data Stream Kernel Interface (DSKI) [4]. The DSKI is currently implemented as a pseudo device driver on two machine architectures, PC hardware running Linux and DEC Alpha hardware running Digital Unix. The choice of the pseudo device driver interface was made to maximize convenience of porting the DSKI to other systems.

The DSKI provides both and *internal* and *external* interfaces. The external interface supports user programs gathering performance data from the operating system. The internal interface is used by kernel subsystems to register events and other information which may then be gathered by these user processes. An *event* is a time-stamped set of data generated when a thread of execution crosses the location of the event in the operating system executable code. The DSKI uses the Pentium Time Stamp Counter, or the equivalent register on the Alpha, to record timestamps at the resolution of the CPU clock. Each kernel subsystem, including device drivers, registers its events with the DSKI during initialization. Note that this works just as well for loadable device drivers.

The number of events within different components of the operating system which are of potential interest is huge, and thus presents a design challenge. We control events in several ways. First, they are conditionally compiled into the system so that their overhead can be eliminated in a production system. Even in an experimental system, each event compiled into the system may be enabled or disabled. This lowers the aggregate overhead since the overhead of a disabled event is $0.05\mu s$ on a 133 MHz Pentium system.

Since the number of potentially interesting events associated with various subsystems is so large, we also provide for grouping the events into sets called *families.* Families of events may be enabled and disabled just as individual events are. The event families, and events within families, registered with the DSKI are presented to the user as a *name space* examined and manipulated using the *ioctl* system call of the DSKI. This is not elegant, but future work includes the relatively simple task of implementing a more graceful API on top of the current functionally complete one.

A process gathering performance data opens the DSKI, enables the sets of events in which it is interested, and then reads the resulting stream of events. Each enabled event is thus associated with at least one process, and perhaps

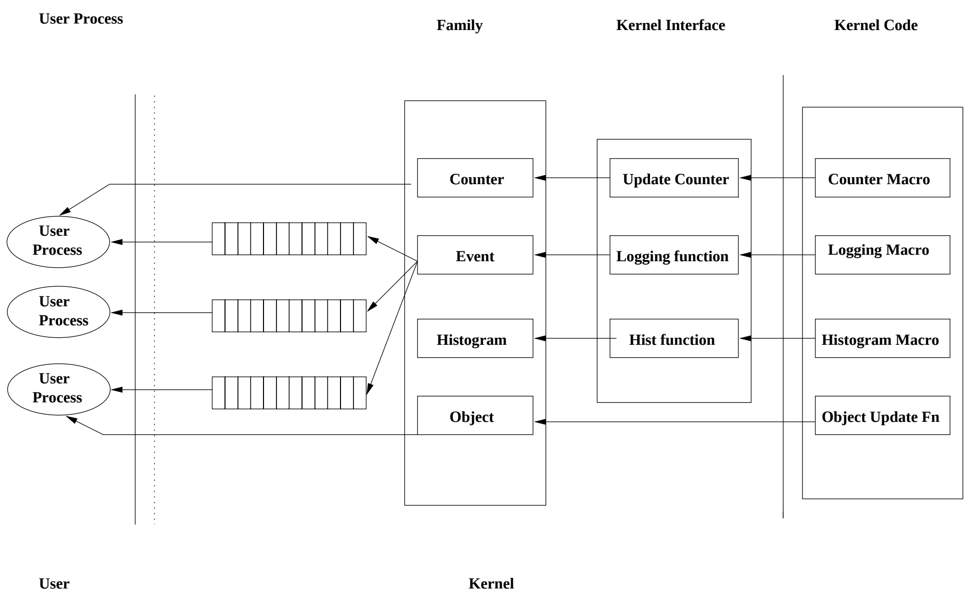

Fig. 1. The Data Stream Kernel Interface Architecture

more. The overhead of logging an event to a single event queue is $3\mu s$ with an additional $2\mu s$ for each additional event queue. In comparison, a simple *kprintf* used to output a timestamp and a long integer to the console required roughly $35\mu s$. More detailed evaluation of the DSKI overhead is given elsewhere [4].

Figure 1 illustrates the basic DSKI architecture. Note that events generated in the kernel flow through the DKSI and into event queues associated with processes. The figure also shows two other elements call the *counter* and the *object*. The *object* is simply a data structure updated by events in the kernel. A process gathering data receives a copy of the data structure's current state each time it chooses to read the DSKI stream attached to the object. The overhead of this approach is significantly lower than that of a datastream actively generating a series of event records, and it is an excellent way of gathering cumulative performance data. The *counter* is a special case of the object with even lower overhead, since the DSKI support for counters is optimized to increment the specified integer with minimal overhead. We have also recently created the *histogram* object which is a parameterized collection of counters. This has been particularly useful recently for gathering distributions of packet inter-arrival times and scheduling distortions.

Consider the differences among the DSKI data stream types applied to evaluation of the TCP/IP protocol stack. If we define a set of active events in the TCP/IP code where we want to gather detailed information, we create an active data stream which collects these events as they are generated by the kernel. We have done this many times and have been able to track individual packets from a socket down through the protocol stack on the sending machine and then up through the protocol stack to the socket on the receiving machine by collecting enough information (e.g. port number and byte sequence number) to uniquely identify the packet. However, if the data we want to collect is adequately represented by a passive object, for example aggregate TCP/IP throughput statistics, then using a passive data stream would incur less overhead. Since this information is not maintained by any specific object in the kernel, we define a data structure for this purpose which is updated by events to count the packets transmitted or received. Finally, if we are interested in the distribution of inter-arrival times of packets or the distribution of the number of packets transmitted or received during N-second periods, we can use the histogram object.

4 Example Driving Applications

Driving applications should, ideally, be software that is desired for its own sake which, at the same time, provides a significant level of practical exercise to a system under development. This section describes three driving applications that have been, and should continue to be, useful challenges to several projects.

The first significant driving application for KURT was the ATM Reference Traffic System (ARTS), which was the main focus of a project funding development of ATM network performance evaluation tools. ARTS can record and generate packet-level ATM traffic streams with microsecond accuracy. ARTS uses KURT's scheduling abilities in its traffic generation module by placing packet transmission events in an explicit KURT schedule [15]. As the application driving initial KURT development, ARTS was re-implemented several times, ending as a KURT real-time module operating in fixed scheduling mode. We avoid all but minor scheduling distortion when we are able to keep the entire packet transmission schedule in main memory, but on a 128MB machine this enables us to hold a schedule with approximately 3 million packet events. Using the disk removes all bounds on the schedule length, but does occasionally distort packet transmission events by delaying them until disk operations are complete. The need to evaluate the performance of ARTS with microsecond accuracy drove the development of the DSKI, as did the need to gather data to assist in optimizing performance and fixing bugs.

Hill extended and refined Srinivasan's work in the areas of Scheduling, I/O subsystems, and network quality of service discussed in Section 2 and in greater detail elsewhere [8, 7]. Hill's driving example was a distributed multimedia application with a simple client-server architecture. Thirty audio and video frames are produced every second, defining a 33 ms period for each frame. The accuracy of the frame period, the packet response latency, and the frame loss rate are obvious performance metrics. The streams of video and audio frames were handled by separate real-time processes, thus making the synchronization of the audio and video processes at the frame level a significant performance metric. The server was capable of generating multiple streams which were synchronized at the frame level.

KURT has also been used to support less obvious applications of real-time capabilities. Murthy used the basic KURT framework to implement a simple ATM switch, and to precisely control the rate at which it processed ATM cells. Similar modifications to host ATM code enabled him to control the rate at which ATM cell streams were produced and consumed. He then used this platform to conduct experiments testing the behavior of Available Bit Rate Service algorithms. His comparisons to the results of conventional simulations, and the execution time of those simulations, indicate that this approach can be used to conduct experiments significantly more quickly than by conventional means[10]. Even if the initial promise of this approach is not realized, Muthy's work shows that the KURT and DSKI provide a powerful and flexible platform for experimentation.

5 Illustrative Results

The results presented in this section illustrate and support the main message of this paper at two levels. First, they demonstrate that the software and methods described here are capable of supporting applications with the finer grain and more complex constraints and requirements typical of emerging multimedia applications. Second, that the evaluation technology and methods has been improved in a way commensurate with the increased capabilities of the systems being evaluated. In particular, the increased temporal resolution of the KURT supports both firm real-time execution of multimedia applications, and the DSKI's ability to gather performance data with microsecond accuracy.

The results presented in this section cannot cover all relevant and interesting aspects of the research described here. Interested readers must refer to the relevant cited works for more detailed results. Some of Hill's results evaluating the end-to-end quality of service provided by KURT and the the ATM LAN to the distributed multimedia application illustrate the relevant points well, and are drawn from his Master's thesis [8].

	Num. Clients	min ms	mean ms	max ms	*Interval within which 'x'% of events fell*			
					95%	99%	99.9%	99.99%
Std. Linux	1	10.036	32.989	113.947	0.419	2.011	22.447	78.935
Event-Driven	1	27.962	32.996	37.156	0.397	1.875	2.901	4.938
Periodic	1	30.229	32.997	35.769	0.543	2.011	2.397	2.673
Std. Linux	2	10.038	32.996	222.709	0.835	4.387	22.958	164.85
Event-Driven	2	13.215	33.004	59.541	0.585	2.348	17.249	24.815
Periodic	2	22.992	32.995	41.905	0.019	0.166	6.556	8.719
Std. Linux	4	10.038	32.997	246.677	1.283	6.394	30.247	172.375
Event-Driven	4	14.557	33.007	69.372	0.332	2.000	13.588	30.327
Periodic	4	20.529	32.998	45.453	0.181	1.317	2.663	12.131

Table 1. Confidence Intervals for Multimedia Client Period with Disk Activity

Table 1 provides information regarding how well the standard Linux scheduler, the event driven scheduler, and the periodic scheduler with buffering maintained the multimedia client's period which was nominally 33 ms. Note that the standard Linux scheduler was tested using the finer temporal resolution and DSKI support features of KURT since standard Linux did not provide the ability to gather this information. These data were taken when non real-time processes generating constant disk activity were active, creating unusually large scheduling distortion. The results show that KURT support maintained the period of the multimedia client significantly better than the standard Linux scheduler, as well as demonstrating that the DSKI supports gathering a large volume of data at a fine temporal resolution.

Table 2 presents the confidence intervals for the packet response latency. This data demonstrates that the performance of the event driven scheduler under KURT performs comparably to standard Linux with respect to this performance metric, which is reassuring but not surprising. The data for the periodic scheduling mode with buffering is interesting because it clearly shows the effect of the 1 packet buffer used by this method, since the latency is approximately 1.5 times the 33 ms packet inter-arrival time and multimedia client period.

	min	mean	max	*Interval within which 'x'% of events fell*			
	μs	μs	μs	95%	99%	99.9%	99.99%
Std. Linux	139	156	10260	182	374	2469	5306
Event-Driven	124	143	9905	161	258	1842	5101
Periodic	46725	48341	54247	49691	49807	50524	53883

Table 2. Confidence Intervals for Packet Response Latency

The DSKI was also used to gather data about another performance metric; packet loss. This metric illustrated a significant difference between standard Linux and KURT support, especially when more than one multimedia stream was being processed. When only one stream was used, standard Linux lost 1 frame out of 3000 while the KURT methods lost 0. However, when 2 multimedia streams were used, standard Linux lost 107 of the 3000 while the event drive KURT method lost 1 and the buffered periodic method lost 0. When 4 streams were used, standard Linux lost 138 of 3000 frames, while the event driven KURT method lost 2 and the buffered periodic method lost 0.

Finally, Table 3 shows the time between when each of the audio and video processes began running to process the paired audio and video frames. The confidence intervals show the interval from the mean difference within which the given percentage of events fell. As the table shows, the buffered periodic real-time processes maintain the closest synchronization throughout the transfer. The maximum deviation experienced is only one quarter of the period. The event-driven process does not perform as well, experiencing some deviations of almost one full period. The standard Linux application experiences the most

	min ms	mean ms	max ms	*Interval within which 'x'% of events fell* 95%	99%	99.9%	99.99%
Std. Linux	0.114	21.374	52.197	6.935	7.895	8.915	15.715
Event-Driven	10.315	20.716	32.157	3.173	4.976	6.912	8.914
Periodic	3.125	5.165	8.525	0.596	0.937	1.375	2.956

Table 3. Confidence Intervals for Audio/Video Stream Synchronization

variance, with some individual variances approaching 2 full periods. Even with disk background activity, four real-time multimedia clients served by a single server process, running at 15 frames per second, remained fully synchronized after 90 minutes of continuous play. This shows the accuracy which the two real-time scheduling modes provides.

6 Related Work

The diversity of the work discussed in this paper brings with it a similar diversity of related work. Srinivasan provides an extensive discussion of how KURT relates to other real-time systems, as well as how the ARTS application compares to related ATM testing methods [14, 15]. Hill discusses a largely separate set of related real-time systems, as well as work in end-to-end quality of service [8]. The DSKI technical report discusses related work in performance data gathering[4], Gopinath discusses real-time support for ORBs[5], and Nimmagadda discusses ORB performance evaluation[11].

7 Conclusions and Future Work

This paper has presented work associated with several projects conducted by the author and his colleagues at the Information and Telecommunication Technology Center at the University of Kansas. The KURT extensions to Linux in support of a variety of firm real-time applications have shown that it can be effective, especially in conjunction with the ability to gather detailed performance data provided by the DSKI. Certainly, the development of these abilities is still relatively primitive, but the work discussed has shown that even modestly enhanced capabilities can make previously infeasible projects possible. The diverse and continuing application of KURT and the DSKI to current and future research projects will continue to drive their development. The two most interesting near-term extensions to KURT are support for explicit scheduling of device driver bottom halves, and integration of Atlas's SRMS and possibly other rate monotonic scheduling modules. The first should significantly decrease scheduling distortions, while the second would provide familiar and popular scheduling semantics.

References

1. T. Anderson, D. Culler, D. Patterson, and the NOW team, "A Case for Networks of Workstations: NOW," *IEEE Micro*, Feb 1995.
2. A. Atlas and A. Bestavros, "Design and Implementation of Statistical Rate Monotonic Scheduling in KURT Linux," Boston University Technical Report 98-013, September 1998.
3. W. Bolosky, R. Draves, R. Fitzgerald, C. Fraser, M. Jones, T. Knoblock, R. Rashid, "Operating System Directions for the Next Millenium," http://research.microsoft.com/sn/Millennium/mgoals.html
4. B. Buchanan, D. Niehaus, R. menon, S. Sheth, Y. Wijata, S. House "The Data Stream Kernel Interface," Technical Report ITTC-FY98-TR-11510-04, Information and Telecommunication Technology Center, Electrical Engineering and Computer Science Department, University of Kansas, 1998.
5. A. Gopinath, "Performance Measurement and Analysis of Real-Time CORBA Endsystems," Master's Thesis, Electrical Engineering and Computer Science Department, University of Kansas, 1998.
6. A. Grimshaw, W. Wulf, and the Legion Team. "The Legion Vision of a Worldwide Virtual Computer." *Communications of the ACM*, 40(1), January 1997.
7. R. Hill, B. Srinivasan, S. Pather, D. Niehaus "Temporal Resolution and Real-Time Extensions to Linux," Technical Report ITTC-FY98-TR-11510-03, Information and Telecommunication Technology Center, Electrical Engineering and Computer Science Department, University of Kansas, 1998.
8. R. Hill "Improving Linux Real-Time Support: Scheduling, I/O Subsystem, and Network Quality of Service Integration," Master's Thesis, Electrical Engineering and Computer Science Department, University of Kansas, 1998.
9. M. Jones, J. Barrera III, A. Forin, P. Leach, D. Rosu, and M. Rosu. An Overview of the Rialto Real-Time Architecture. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, pages 249–256, September 1996.
10. S. Murthy "A Software Emulation and Evaluation of Available Bit Rate Service," Master's Thesis, Electrical Engineering and Computer Science Department, University of Kansas, 1998.
11. S. Nimmagadda, C. Liyanaarachchi, A. Gopinath, D. Niehaus, A. Kaushal, "Performance Patterns: Automated Scenario Based ORB Performance Evaluation," To Appear *COOTS '99*, May 1999.
12. The Olivetti & Oracle Research Laboratory. Omniorb2, http://www.orl.co.uk/omniorb/.
13. D. Schmidt, R. Bector, D. Levine, S. Mungee, and G. Parulkar. TAO: A Middleware Framework for Real-Time ORB Endsystems. In *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.
14. B. Srinivasan, S. Pather, R. Hill, F. Ansari, D. Niehaus "A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software," In *Proceedings of the Real-Time Technology and Applications Symposium*, Denver, June 1998.
15. B. Srinivasan "A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software," Master's Thesis, Electrical Engineering and Computer Science Department, University of Kansas, 1998.