Appeared in the Proceedings of SDA 98, Dresden, March 1998

Dresden Realtime Operating System

R. Baumgartl, M. Borriss, H. Härtig, Cl.-J. Hamann, M. Hohmuth, L. Reuther, S. Schönberg, and J.Wolter

Department of Computer Science, Dresden University of Technology, D-01062 Dresden

Abstract. This paper describes the construction of an operating system supporting real-time applications with strong quality of service requirements based on a consequent resource scheduling as well as non-real-time functionality by a conventional Unix personality. The main part of the paper discusses the port of the Linux operating system onto the L4 μ-kernel. We give an architectural overview of the project and subsequently discuss selected real-time components (file system, network component, multimedia accelerator). Modelling aspects are explained introducing a deterministic model for a quantitative description of the work load by means of a parameter set. This allows a uniform description of the data flow between all system components.

1 Introduction

Current research on operating systems is strongly influenced by the increasing number of multimedia applications and various "classical" real-time applications with high demands on Quality of Service (QoS) guarantees, especially concerning latency, transfer bandwidth, buffer requirements, service guarantee over longer periods of time and synchronisation of data streams (e.g., teleconferencing or video-on-demand). This situation demands the systematic construction of operating systems supporting QoS requirements of applications. A main aspect of real-time systems is resource reservation.

In the past, various systems have been designed facing these requirements. Since almost all of them are only extensions of current time-sharing systems, they are not efficient or flexible enough.

For that reason, the operating system research group of Dresden University of Technology develops a new system which is able to handle resource reservation. The project called *DROPS* faces two main goals. Firstly, the system is being developed on top of the highly efficient μ-kernel L4 [9], which leads to a consequent client-server-based system. Secondly, the architecture is being designed in parallel to the development of the mathematical model of a scheduling technique and a general work load description. Using this technique, various system component dependent planning and reservation algorithms can be derived.

This paper describes the current status of the project and gives an overview of the system architecture. Furthermore, it describes the L^{4}Linux server, which is the main development platform and the base for all non-real-time applications. In the next section the design of the following real-time components is explained:

– A file system for continuous media data with QoS support and support for conventional data storage.
– A real-time networking component based on ATM technology.
– A multimedia accelerator based on a Digital Signal Processor (DSP) to remove computationally-intensive tasks from the CPU.

Next, the mathematical modelling is explained, especially the quantitative description of the work load supported by the system and the prognosis model used for resource scheduling. Based on a deterministic model, a set of four parameters is given which enables a uniform description of the data flow between all system components.

The paper closes with a brief outlook on the project's future plans.

2 Overview

2.1 Motivation and Goal

Analysis of current real-time systems [7] reveals that these systems can always be regarded as somewhere in between the extrema pure priority based systems [8] and systems which schedule completely in advance [11]. Similar but less strong, this holds true for systems like Spring or Real-Time Mach.

In the field of reservation-based transfer protocols for high-speed networks research concentrates mainly on the network-related part of the system [1]. The lack of flexibility and efficiency of these systems led to the idea for a radical new design of a μ-kernel-based system. The main goal is the development of a system that guarantees real-time requirements as needed for reservation-based transmission protocols or for multimedia data transfer. It should be emphasized that not only a few but *all* components of the system build a chain of QoS-guaranteeing modules.

2.2 Design

The requirements mentioned above lead to the following general system design principles (figure 1):

– multi-server architecture based on a second-generation μ-kernel;
– system components that guarantee specified parameters independent of the system load;
– all components use techniques for planning in advance and negotiate minimal and required QoS parameters;
– the architecture permits the coexistence of real-time and non-real-time services within a single system by strict separation of these services.

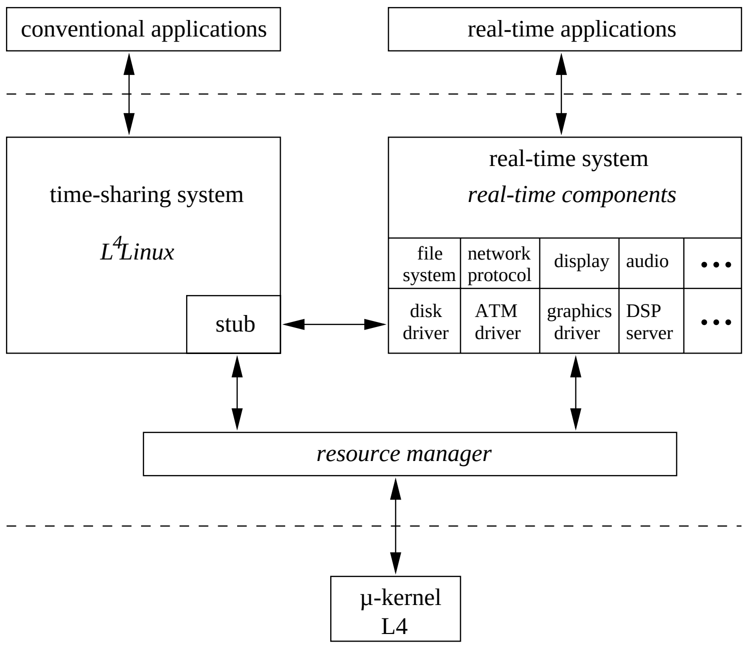

Fig. 1. DROPS System Architecture

We chose the L4 kernel for Intel processors, developed by J. Liedtke [9], for its flexibility and performance. L4 only supports a minimal set of hardware-independent abstractions and services (i.e., address spaces, threads, process communication, round-robin scheduling based on hard priorities). All complex operating system functionality (e.g., device drivers) runs in the processor's user mode. Therefore, L4 enables applications to implement their own memory management strategies and the scheduling of the CPU. The kernel is 12 KBytes in size and implements an exceptionally fast Inter-Process-Communication (IPC).

3 Porting Linux to the L4 μ-Kernel

The main design goals of the port can be described as follows:

– The Linux server runs in user mode.

– The server is fully binary compatible to native Linux, all applications run on L^{4}Linux without recompilation.
– The server is as efficient as possible and provides the full Linux functionality.

3.1 Major Design Decisions

The most important influence factor for the design of the emulation was the interaction between user tasks and the Linux kernel. In classical Unix systems, after a system call, a process simply switches to a privileged mode and continues execution within the same context. In contrast, a μ-kernel environment requires client-server based communication, the appropriate data has to be transferred and a context switch to the server is necessary.

With the exception of the scheduling code the architecturally independent part of Linux was not modified. The adaption of the architecture-dependent part comprises about 12,000 lines of C code.

Following, we discuss selected aspects of modifications applied to the interrupt system, the system call interface, signalling mechanisms, scheduling and the memory management.

Task and Thread structure. From the point of view of the μ-kernel the Linux server has the following structure (figure 2):

– All Linux kernel services are represented by a single L4 thread. Its context can be switched to multiplex the various kernel activities.
– Every interrupt source has its own dedicated thread.
– Another thread handles all non-time-critical parts of the interrupt procedures.
– Linux user applications consist of two threads, one for the primary user activity and another for signalling.

Additional synchronisation is unnecessary because only short critical sections within the kernel are protected against interrupts.

The memory management is based on L4's external pager concept. L^{4}Linux manages its memory hierarchically: a "root pager" handles page faults in the L^{4}Linux kernel whereas page faults raised in user applications are handled by the L^{4}Linux kernel.

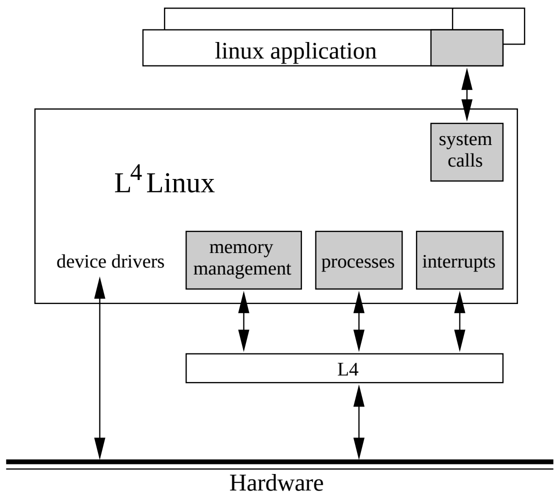

Fig. 2. L^{4}Linux Structure

System Services. L^{4}Linux system calls are implemented using remote procedure calls, that is, IPCs between the user processes and the Linux server. There are three concurrently usable system-call interfaces:

1. a modified version of the standard shared C library `libc.so` which uses L4 IPC primitives to call the Linux server;
2. a correspondingly modified version of the `libc.a` library;
3. a user-level exception handler ("trampoline") which emulates the native system-call trap instruction by calling a corresponding routine in the modified shared library.

The first two mechanisms are slightly faster, and the third one establishes true binary compatibility. Applications that are linked against the shared library automatically obtain the performance advantages of the first mechanism. Applications statically linked against an unmodified `libc` suffer the performance degradation of the latter mechanism. All mechanisms can be arbitrarily mixed in any Linux process.

Signals. Signals are used to asynchronously inform a process about an external event or change of its state. Signal delivery is done by manipulating the user stack when returning from a system call. In original Linux, the timer interrupt periodically forces the current process to enter the kernel. In L^{4}Linux , we implemented a "fake-interrupt-mechanism" instead. A dedicated thread running within the application context accepts signal messages from the kernel and forces the user application to enter the kernel. On return of that "null system call" all pending signals are delivered.

Interrupts. In L4 all hardware interrupts are handled on user level. A thread can attach to a distinct hardware interrupt. If an interrupt occurs IPC is used to inform the attached thread about the interrupt condition which executes the interrupt service routine. The non-time-critical part is implemented by another thread which is synchronized using L4 thread priorities.

Scheduling. Thread scheduling is done in the L4 μ-kernel. The L^{4}Linux `schedule()` routine is restricted to multiplex the Linux server activities to the single L^{4}Linux kernel thread.

3.2 Performance

To demonstrate that μ-kernel-based systems can be implemented very efficiently, we performed an extensive optimization phase after the initial implementation which can be divided into:

– general optimizations: inlining of function calls, compiler-supported optimization, hand-tuning of time-critical parts, exploitation of the Pentium's super-scalar architecture, minimization of cache and TLB influences, and
– optimization of the system call path: analysis of different scheduling strategies, improvement of data alignment and minimization of data accesses.

The performance was measured using the micro-benchmark *lmbench* and the *AIM* application benchmark to compare L^{4}Linux to original Linux and to the Mach-based Linux emulation MkLinux [6]. Under very heavy system load the AIM benchmark revealed an overhead of 7% for L^{4}Linux . On application level the overhead is about 3%.

4 Selected Real-Time Components

4.1 File System

Local operations on a multimedia stream (without transmission over a network) in general include loading a file from an external medium, synchronization of several streams if necessary, and presentation using an appropriate component. At least the presentation needs to be done with an appropriate bit rate.

The main problem for the file system is to make the data available in time under the constraints of limited resources such as buffer space or transfer bandwidth. Conventional systems try to satisfy all requests using a best-effort strategy without time constraints. New file system approaches addressing the real-time problem (e.g., CMFS [10]) are neither compatible to each other nor to classical file systems.

In DROPS a file system is being developed that allows storage and playback of continuous data under real-time constraints and resource restrictions. Additionally, it can be used as conventional Linux file system without QoS guarantees.

The design is close to the common ext2-filesystem of Linux [3]. Meta data such as time-dependent information is stored together with the real-time data. The File server resembles a multi-threaded client-server structure, using L4 IPC. Various block sizes are supported, conventional transfer- and read-ahead buffers are implemented (cf. section 5.3). Parameters of the file system are transformed into requests to the SCSI driver using the model for jitter-constrained streams (cf. section 5.1).

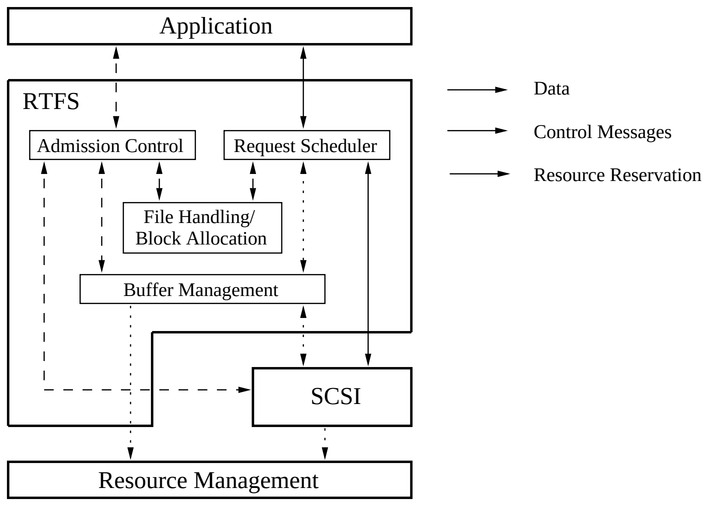

Fig. 3. Structure of the Real-Time File System RTFS

4.2 Network Component

With reference to potential DROPS applications, ATM is the best choice for a network technology. ATM allows QoS support and guarantees high transfer bandwidth and predictability. The job of the operating system is to guarantee sufficient resources like CPU or main memory for the network activities. For DROPS a real-time network component is planned which provides an interface for applications to transmit data, traffic descriptions of data streams and the according QoS parameters.

The ATM component consists of the driver using the PCI adapter cards from FORE (PCA200e) with 155 Mbits/s. The driver is used for both real-time and time-sharing services of L^4Linux . The second layer provides a set of protocols for establishment of connections and QoS negotiation. Furthermore, connection admission control and UNI 3.1 conform signalling is implemented. The main emphasis of the work within the DROPS project is on the reservation of resources (i.e., buffer memory and CPU time) in advance.

4.3 Multimedia Accelerators

The increasing integration of multimedia data into application programs requires immense processing power and transfer bandwidth as well as the consideration of real-time requirements. This situation results in an unacceptable slow-down of the application and high latency. The usual approach of concentrating more and more computing power into the main processor does not provide a consistent solution because at the same time more and more complex multimedia algorithms (data compression, 3d graphics, music synthesis) are being developed. Digital Signal Processors (DSPs) acting as flexible yet powerful accelerators thus reducing CPU load could be an cost-efficient alternative.

One of the goals of our project is to use a DSP as multimedia accelerator and embed it into a flexible subsystem within the DROPS real-time system. This real-time component comprises a DSP, a bus interface to the host, local memory and interfaces to the outside world (e.g., audio codec, network interface). It makes external hardware components of the host like MPEG decoder and sound card obsolete. To identify an efficient interface between CPU and accelerator, a part of the work focuses on the analysis and evaluation of different bus systems (e.g., ISA, SCSI, PCI) and transfer mechanisms. Additionally, a very tight integration (i.e., local bus interface) is taken into consideration.

To accelerate more than one host process at the same time a real-time kernel which is consequently optimized for multimedia stream processing is run on the DSP. The timer-activated context switch is very efficient and needs approximately 110 clock cycles at the moment. The static scheduling provides services such as insertion, start, stop and deletion of tasks at runtime. On the host side a server process is responsible for communication between DSP subsystem and DROPS. It calculates the DSP schedule, controls all DSP tasks and organizes the transfer of data to be processed by the DSP.

5 Modelling Aspects

5.1 Jitter-Constrained Streams

For a wide set of real-time (especially multi-media-based) applications, a load can be characterized as follows. At an interface in the system, events occur periodically with a distance T or a rate of $R = 1/T$. However, they may vary within a given interval: events may occur τ time units too early or τ' time units too late as long as they obey a minimum distance $D < T$ (cf. Figure 4). Hence, T is the average distance between two events. Without restricting of generality we assume $t_0 = 0$.

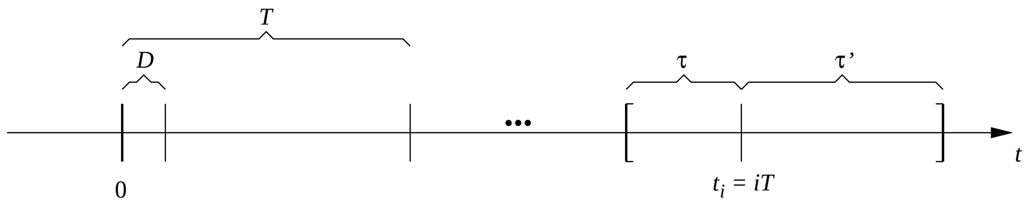

Fig. 4. Jitter Constrained Event Stream with Constant Bit rate

Such a sequence of events, known as "jitter-constrained streams", are described by various partially independent parameters, such as peak rate, average event distance, maximum burst size, or minimal buffer size to avoid loss of data. Examples are the Tenet real-time protocol suite [1], [5] or linear bounded processes in the LBAP model.

The following model allows an uniform description of heterogenous components in a real-time system.

Definition 1. *Let*

$$D, T, \tau, \tau' \in \mathbb{R} \qquad \textit{with} \qquad T > D > 0, \qquad \tau, \tau' \geq 0$$

and

$$i \in \mathbb{N}.$$

A (τ, τ')-jitter-constrained stream with constant period T and minimal distance D is a sequence $(E_i)_{i=0,1,\ldots}$ of events iff the following holds: event E_i occurs at real event time

$$a_i \in \left[iT - \tau,\ iT + \tau'\right] \subseteq \mathbb{R},$$

and the distances

$$d_{i+1} = a_{i+1} - a_i$$

obey the condition

$$d_{i+1} \geq D \qquad \forall i \in \mathbb{N}.$$

We only consider bursts (i.e., streams of events occurring in shortest possible distances), especially the worst-case situations, which are:

- **maximum burst streams:** streams consist of bursts of maximum length,
- **dense burst streams:** events appear in different burst sizes but as often as possible.

In both cases, relevant parameters such as maximum burst size L, earliest and latest starting point of a burst, distances between bursts, minimal buffer size P and number $N(t)$ of received events until time t can be calculated.

5.2 Selected Results

The length L of a maximum burst stream can be obtained by

$$L = 1 + \left\lfloor \frac{\tau + \tau'}{T - D} \right\rfloor . \tag{1}$$

Furthermore, it is obvious that a maximal burst stream is periodic with the period LT, and the jitter parameters τ, τ' can be obtained from the given burst size L:

$$\tau + \tau' \in \Big[(L-1)(T-D),\ L(T-D)\Big). \tag{2}$$

Data arriving too early must be buffered. B is the minimal buffer size required to handle all events without loss and can be obtained by

$$P = \left\lceil \frac{\tau + \tau'}{T} \right\rceil . \tag{3}$$

5.3 Prognosis Model

One of the essential services of a real-time system providing QoS guarantees is the reservation of all resources required by an application. In the special case of a real-time file system this means that requested hard disk blocks are read in time and sufficient buffer space is available. Therefore, all resources such as memory or disk bandwidth need to be allocated in advance. On application startup, the system checks by admission control whether all application requests can be satisfied or not.

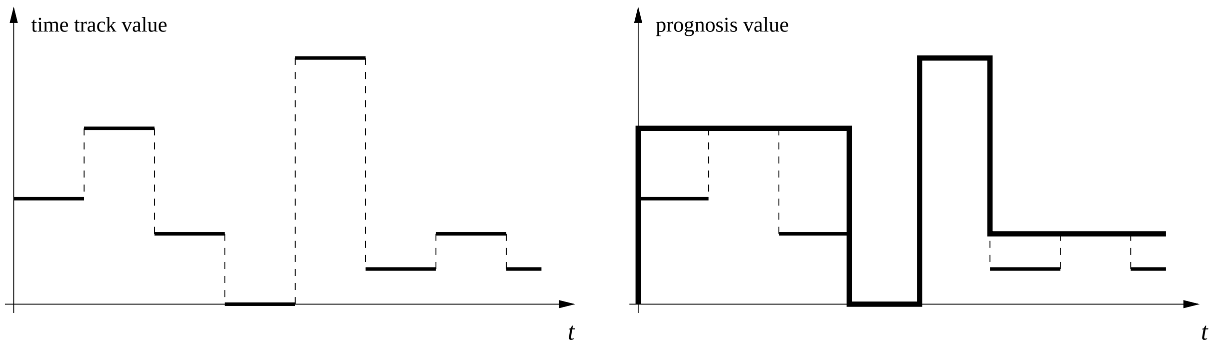

Fig. 5. Time track and prognosis value with envelope of 0th order

The DROPS system uses a time track consisting of information of the form (relative time, size). This track is stored together with every real-time file. The parameter "size" can represent for example bandwidth, block size, or processing time. Based on this time track, the admission control component schedules the application. The size of the time track is proportional to the size of the file. For a one-hour MPEG I video, the time-track's size is expected to be 500KBytes. This information can be compressed easily by approximating an envelope (of 0th or 1st order, i.e. a sequence of line segments) for the time track. The time track is segmented and the value of the envelope for this segment is the maximum value within the interval (cf. figure 5).

The length of the interval and with it the quality of the approximation is controlled by a quality factor G with $0 \leq G \leq 1$. $G = 1$ describes the original time-track and for $G = 0$ the envelope is the maximum of the entire track. Figure 5 illustrates this using $G = 0.6$.

6 Current Status

Currently, L^{4}Linux is a stable and well-running platform for development of real-time applications. The file system serves all non-real-time requests. The ATM Linux driver has been implemented [4], the L4 port and the protocol component is under way. Performance measurements for a DSP connected to a μ-kernel [2], were carried out and a prototype implementation of a DSP server on L4 has been started.

The next stage of the project is the development of tools for the systematic investigation of resource demands of typical real-time applications to obtain real-world parameters for our reservation interfaces.

References

1. A. Banerjea, D. Ferrari, B.A. Mah, M. Moran, D.C. Verma, and H. Zhang. The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences. Technical Report TR-94-059, Univ. of California at Berkeley and ICSI, November 1994.
2. Robert Baumgartl and Hermann Härtig. On the Integration of DSP Hardware into a Microkernel-based Operating System. In *Proceedings of the International Conference On Signal Processing Applications & Technology (ICSPAT'96)*, pages 867–872, Boston, October 1996.
3. M. Beck and et al. *Linux-Kernel-Programmierung, Algorithmen und Strukturen der Version 1.2.* Addison-Wesley, 1995.
4. M. Borriss and U. Dannowski. *Linux support for FORE Systems PCA-200E NIC.* Dresden University of Technology, 1997. Available from `http://os.inf.tu-dresden/project/atm`.
5. ATM Forum Technical Committee. ATM: Traffic Management Specification, Version 4.0, 1996.
6. F. B. des Places, N. Stephen, and F. D. Reynolds. Linux on the OSF Mach3 Microkernel. In *Conference on Freely Distributable Software*, Boston, MA, February 1996. Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111. Available from `http://www.gr.osf.org/~stephen/fsf96.ps`.
7. H. Härtig, C.-J. Hamann, P. Hochgemuth, M. Schalm, and J. Wittenberger. Multimedia-Unterstützung in Betriebssystemen. Technical report, Dresden University of Technology, 1995.
8. Dan Hildebrand. An Architectural Overview of QNX. In *1st USENIX Workshop on Micro-kernels and Other Kernel Architectures*, pages 113–126, Seattle, WA, April 1992.
9. Jochen Liedtke. L4 Reference Manual (486, Pentium, PPro). Arbeitspapiere der GMD 1021, GMD–German National Research Center for Information Technology, Sankt Augustin, September 1996. Available from `ftp://borneo.gmd.de/pub/rs/L4/l4refx86.ps`.
10. D. Makaroff, N. Hutchinson, and G. Neufeld. *The UCB Distributed Continuous Media Filesystem – Interface Documentation.* Department of Computer Science, University of British Columbia, Vancouver, 1996.
11. J. Reisinger. Time Driven Operating Systems – A Case Study on the MARS Kernel, 1992. Position Paper, TU Wien.