A User Mode L4 Environment

Sebastian Schönberg

Dresden University of Technology
Department of Computer Science, Operating Systems Group
email: schoenbg@acm.org

Abstract

User Mode Linux [3] has become a convenient environment for development and testing of applications and the Linux kernel. Triggered by this idea and our own experiences with the many problems which appeared while developing L4 applications, we propose **UMOL^4**, a user mode implementation of the L4 kernel interface on bare Linux.

Even though L4 and L^{4}Linux are available on IA-32 processors and Linux development features can also be used under L4, a stand-alone UMOL^4 implementation improves the development process and makes porting of L4 application to another platform easier. With UMOL^4, L4 applications can be tested on a standard Linux system without changing the kernel. Testing on a high-speed production server is possible without high risks. We list below many more advantages to motivate the implementation of UMOL^4.

1 Introduction

Over the past years, the L4 microkernel [8] has become a well known standard. The kernel encapsulates the differences of the hardware and provides a small but very powerful interface. It is available on various hardware platforms such as IA-32, Alpha or MIPS. The core functionality of the microkernel provides protection by address spaces, activities by threads, and communication by synchronous inter-process communication (IPC). All other services such as name space management, device drivers are executed in user mode. A modified version of Linux (L^{4}Linux [5]) runs as an operating system personality on L4 and provides a binary compatible interface for Linux applications. Another personality is L4env [9], which combines a collection of services such as a name service, an application loader, and a memory manager. A single-address-space operating system (SASOS) personality on top of the MIPS and Alpha version of L4 is Mungi [6]. In the research area of real-time, the Dresden Real-time Operating System (DROPS) provides support for handling jitter constrained data streams [4].

Since IPC is the fundamental mechanism to exchange data, an Interface Definition Language (IDL) Compiler is available [1]. For simple applications, the OSKIT provides a version of the "libc" and other functions. The complete suite of GNU development tools such as assembler, compiler, and linker are available. For debugging, either kernel internal features such as single stepping or functions to dump the content and mapping of memory pages is available, or applications can be tested with a remote version of GDB. L^{4}Linux-applications have no different behavior than standard Linux applications and can be developed and debugged in the same way. But L^{4}Linux applications additionally can use the L4 microkernel functions.

On the first glance, enough tools and support is available. But despite the continuous effort with all these personalities, development of L4 applications, especially for system parts of a personality is still an adventurous task. As long as no dynamic loading, starting and ending of applications is available, a modification always requires a restart of the entire system. If the system hangs, debugging on the same machine is very problematic or sometimes impossible. Hence, testing always requires an additional machine. For IA-32-based systems, tests can be performed in a virtual machine such as this provided by VMWare [7]. Major drawbacks of a VMWare solution are the relatively high hardware requirements for the "host system."

With introduction of an easily portable version of the L4 kernel [2], development of a kernel for another processor architecture is not a very complex task anymore. Two developers can do a port to a new 32-bit architecture within one or two weeks.[1] But the port of the user mode environments and operating system personalities is still difficult, requires more manpower, and additional hardware for testing. But as often in the early stage of a project, the additional testing hardware is either very expensive or not sufficiently available.

Under the conditions of university research for practical training of students, not every student has a second machine for testing.

As a possible and very effective solution, we propose a User Mode L4 (UMOL^4). UMOL^4 provides an L4 (binary) compatible interface for Linux machines. In contrast to the approach of developing and testing L4 applications on an L^{4}Linux machine, this has some major advantages:

Rapid Prototyping: Application development and testing can be done on the same machine. Rebooting of the test machine is not necessary anymore. By simply ending and starting the UMOL^4 environment, the "virtual

[1] Personal Communication with the L4KA authors

L4 box" is rebooted and the tests can be repeated. With the complex BIOS version of todays hardware, this is a real time saver.

Multiple Users: Since UMOL^4 runs as an application on a standard, unmodified Linux machine, multiple users can test their L4 applications on the same machine.

Increased Stability: L^{4}Linux became more stable on desktop machines, but is still not stable enough to be used on a production server. UMOL^4 does neither modify the Linux kernel nor any critical part of the system. Hence, it can be used without problems on a production server.

Cluster Development: Using L4 as a base for a cluster of machines is one of the possible areas of use. The L4 kernel is the same as on every non-cluster machine, but the Cluster OS personality provides the functionality for inter-machine communication, load balancing, etc. A testbed for such a cluster would require many machines connected to each other. In UMOL^4, one UMOL^4 instance is identical to one machine in the cluster. On a powerful Linux server, hundreds or even thousands of UMOL^4 instances can be started in parallel to test the behavior of the Cluster OS with no additional hardware.

Availability: Linux machines are available almost everywhere. In a new environment, it is sometimes difficult or for security reasons not allowed to change the operating system of machines connected to the internal network. Hence, an L4 or L^{4}Linux machine is not available.

New Platforms: A port of applications to a new architecture now becomes very easy. The UMOL^4 kernel is, with exception of a view parts such as handling the CPU register context, completely architecture-independent. To support a new architecture, only the high-level language bindings for the system calls must be adapted.

2 Design Ideas

In this section we describe some design alternatives and how to map L4 primitives to Linux functionality. Two general approaches are possible.

The first approach makes use of a Linux kernel module. Here, either the complete L4 kernel functionality or only a small stub is implemented in the kernel module. If the full L4 functionality is implemented in the module, no additional support in a Linux user mode application is required. The main advantage is performance since many of the hardware-independent algorithms from the highly optimized L4 kernel can be used unmodified. The main disadvantage is the security problem of Linux kernel modules. Only a superuser can install and remove such modules from the kernel, hence the advantage of availability mentioned in the previous chapter would be given up. This also holds true for a module that implements only an adaptation layer to catch L4 system calls and forward them to a user mode application.

The second approach makes extensive use of the debugging features of Linux. Similar to a debugger, a UMOL^4 core (running as user mode application) catches all exceptions and system calls and performs the appropriate L4 function. The Linux interface (`ptrace` system call) is very similar on all Linux platforms and a port to a new architecture is easy. Figure 1 illustrates the general design idea of UMOL^4 picture.

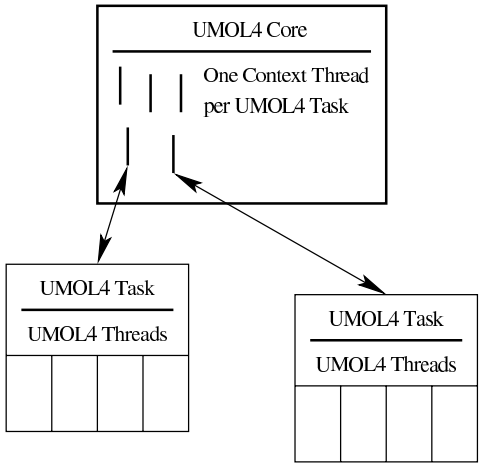

Figure 1: Architecture of UMOL^4

2.1 UMOL^4 Address Spaces

UMOL^4 address spaces are mapped to Linux address spaces. This provides protection and requires no special case handling. A problem may arise, if an address space range used by the Linux kernel (e.g., on IA-32 between 3GB and 3.5GB) is not used and therefore not available for applications (as it would be on a native L4 version). This should not be an existential problem. We are not aware of any application that cannot be linked to a different virtual address.

2.2 UMOL^4 Threads

UMOL^4 Threads are represented by Linux threads. They can be created by the Linux `clone` system call. This requires some code executed in the context of the thread's UMOL^4 address space. The minimal version performs a system call instruction (i.e., on IA-32 an `int $0x80`). The registers are set to the appropriate values by the calling `ptrace` system call.

Writes or reads of the thread's register such as the instruction pointer or stack pointer are mapped to the Linux `ptrace` (PTRACE_GETREGS, PTRACE_SETREGS) system call.

2.3 Memory Management

To implement a memory management which behaves (almost) identical to the original L4 implementation, a Sigma0 server is required which manages all possible memory. Therefore, the UMOL^4 core creates an zero-filled file (MemFile) and maps it with the `mmap` system call. Since all UMOL^4 address spaces are created by `fork` from the UMOL^4 core they all have access to the file handle of the memory mapped file.

The Sigma0 server is the first UMOL^4 task. It receives a complete mapping of the MemFile.

Unmapping of pages can be implemented either by `munmap` or `mprotect`. Problems may arise, if the number of mappings of all UMOL^4 tasks exceed the number of Linux' `mmap` sections. This can be solved as following: to implement the L4-flush function, the UMOL^4-core must already keep track of all mapping relations in a mapping database. If we run out of available Linux `mmap` sections, we free an adequate number of areas mapped in UMOL^4 tasks, but keep the entries in the mapping database. On accessing these regions, we can on demand re-establish the appropriate mappings.

2.4 UMOL^4 System Calls

An UMOL^4 thread which tries to execute an L4 system call raises an exception in the assigned Linux thread in the UMOL^4 core. The core obtains the register with the `ptrace` system call and executes the appropriate function. Upon return, the results are written back to the UMOL^4 thread and execution is resumed. Since UMOL^4 threads are not intended to call Linux system calls, there is no conflict between Linux and UMOL^4.

2.5 Interrupts and Hardware

For development and testing of software components that manage hardware, access to devices and interrupts must be possible. This conflicts with the security idea of the UMOL^4 environment which includes that an application cannot tear down the host machine. In a first approach, interrupts and hardware access is not possible for UMOL^4 applications. An improved version can completely virtualize the most important devices such as Interrupt Controller (PIC) or DMA Controller. Access to ports or memory mapped I/O that belongs to devices is automatically intercepted and the UMOL^4 core is notified.

3 Conclusion and Future Work

All core L4 functionality can be mapped relatively straight forward to Linux functionality. The `ptrace` system call is the instrument of choice to control UMOL^4 threads.

In the next step, we will implement UMOL^4 for IA-32 Linux and do performance measurements to compare UMOL^4 with versions of native L4 such as the original L4 assembly language implementation, Fiasco or L4KA.

References

[1] Ronald Aigner. An IDL Compiler for Microkernel Systems. Master's thesis, TUD, 2001.

[2] Uwe Dannowski and Volkmar Uhlig. L4ka - a portable l4 kernel. private communication.

[3] Jeff Dike. User-mode linux. Technical report, 2001.

[4] H. Härtig, R. Baumgartl, M. Borriss, Cl.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter. DROPS: OS support for distributed multimedia applications. In *Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[5] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The performance of μ-kernel-based systems. In *16th ACM Symposium on Operating System Principles (SOSP)*, pages 66–77, Saint-Malo, France, October 1997.

[6] G. Heiser, K. Elphinstone, S. Russell, and G. R. Hellestrand. A distributed single address-space operating system supporting persistence. SCS&E Report 9302, Univ. of New South Wales, School of Computer Science, Kensington, Australia, March 1993.

[7] VMWare Inc. VMWare, A virtual machine for IA-32 processors.

[8] J. Liedtke. L4 reference manual (486, Pentium, PPro). Arbeitspapiere der GMD No. 1021, GMD — German National Research Center for Information Technology, Sankt Augustin, September 1996. Also Research Report RC 20549, IBM T. J. Watson Research Center, Yorktown Heights, NY.

[9] Frank Mehnert and Lars Reuther. l4env - an enviroment for l4 applications. Technical report, TUD, 2001.