A Case Study of Microkernel for Education

ZHOU Qingguo[1,2], DING Ying[1,2], Nicholas McGuire[1,2*], LI Canyu[1,2], CHENG Guanghui[1,2], Bin Hu[2,3]

1. Distributed & Embedded System Lab (DSLab), Lanzhou University, China;
2. Engineering Research Center of Open Source Software and Real-Time Systems, Ministry of Education, Lanzhou University, China
3. Department of Computing, Birmingham City University, UK
{zhouqg, *nicho}@lzu.edu.cn; {dingy09, cheng.guanghui, tezuka158}@gmail.com

Abstract

The paper will introduce a new operating system architecture microkernel for education. For its safe, small, and flexible feature, micorkernel is becoming widely use in various aspects. Taking it to education, will from different design architecture to make learner get deeper understanding of operating system mechanisms. Fiasco is a real, second-generation microkernel whose source code is less than 15,000 lines, and it is freely available under the GNU General Public License. In this paper, based on Fiasco, the microkernel operating system will be introduced especially in its kernel mechanism.

1. Background

Today's operating system could be divided into monolithic kernel, microkernel [1], nanokernel or others by its architecture. What we most familiar is the monolithic kernel, like Linux. It is efficient, and for it is an open source operating system, Linux is widely used in education. However, now people turning more attention to system security and flexibility. Though these aspects are being enhanced in Linux which restricted by its system architecture. It can't achieve the goal relative to microkernel.

Comparing with monolithic kernel, the microkernel has minimal OS kernel which only leave inevitable mechanism in kernel mode [2]. That makes the kernel be less error prone and suitable for verification. The most system services of microkernel are implemented as user-level servers, which make the system more flexible and extensible. The microkernel system is also designed with protection between individual components, therefore, systems could get more secure from it inter-component protection. Because when one of the components crashed, it does not crash the whole system.

The follow sections will take L4/Fiasco [3] as an example to introduce the core mechanism in microkernel, and through a simple instance to display programming model on fiasco, together with its booting menu and results. It will illustrate alternative design concepts and give another visual angle for operating system education.

2. L4/Fiasco Introduction

L4 is the name of a second-generation microkernel originally developed by Jochen Liedtke. Based on L4 there are many different implementations developed separately by different Universities or organizations such as L4.Fiasco, L4.Pistachio, L4.OKL4, etc [4].

Fiasco has been developed by the Fiasco Team in Technical University Dresden (TUD, Germany) since 1999. It is as a base for DROPS (The Dresden Real-Time Operating Systems Project) [5][6] system which supports running real-time and time-sharing applications concurrently on one computer. It is sometimes referred to as L4/Fiasco to emphasize the relationship with the L4 specification. Fiasco running on x86 PCs intended to be compatible with the L4 microkernel for x86, and it is a preemptible real-time kernel supporting hard priorities.

However, just as Jochen Liedtke said: "A microkernel does no real work."[7]. Microkernel provides inevitable mechanisms, but does not enforce policies, most functionality implemented in user mode. In L4/Fiasco system, there is L4 Environment [8], short L4Env, is a programming environment for applications that should run on the L4/Fiasco kernel. And the whole L4/Fiasco system architecture is as Figure 1.

978-1-4244-3930-0/09/$25.00 ©2009 IEEE

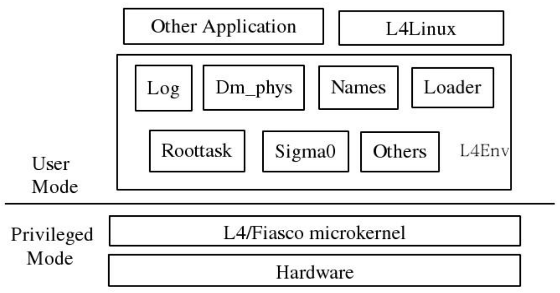

Figure 1. L4/Fiasco System Architecture

As Figure 1 show, the application is running based on the L4Env. The L4Linux [9] is a special application of L4. It is a port of the Linux kernel to the L4 microkernel API. It runs in user-mode on top of the microkernel, side-by-side with other microkernel applications such as real-time components.

3. L4/Fiasco Technologies

In this section, the inevitable technologies [10] of L4/Fiasco kernel will be introduced. There are address spaces, threads, communication, mapping, system calls, and scheduling.

3.1. Address spaces

Address space is the unit of memory management, and it provides spatial isolation. It contains all the data which are directly accessible by a thread. An address space is a set of mappings from virtual to physical memory. Address spaces in L4 can be recursively constructed. A thread can map common memory parts of its address space into another thread's address space and thereby share the data mapped by that region of the address space.

The concept of a task is essentially equivalent to that of an address space. In L4, a task is the set of threads sharing an address space. Creating a new task creates an address space with one running thread.

3.2. Threads

Threads are important mechanism, which reflect the CPU resource. In L4/Fiasco, it is abstraction of code execution and unit of scheduling, which also provides temporal isolation. A thread has an address space which shared with the other threads belonging to the same task, a UID, a register set, a page fault handler, and an exception handler.

Threads can be implemented in user-land. User-level applications need to allocate stack memory, provide memory for application binary, and find kernel entry point to make thread can enter kernel space, then kernel maps user-level threads to kernel-level threads, usually a 1:1 mapping.

In L4/Fiasco one kernel-level thread per logical CPU and 1 address space can contain up to 128 threads. Threads are extremely light-weight and cheap to create, destroy, start and stop. The lightweight thread concept, together with very fast IPC, is the key to the efficiency of L4 and OS personalities built on top.

3.3. Communication

Why do we need to communicate? For IPC is the heart mechanism in a microkernel-based system. IPC is used to:

- exchange data by value or by reference
- synchronize threads, wakeup-calls
- pager invocation
- exception handling
- interrupt handling
- device control

For microkernel, “IPC performance is the master.” as Liedtke said [11][12]. There are two types of IPC have used: Asynchronous and Synchronous IPC. Asynchronous IPC was used in first generation microkernel (e.g., Mach), in this manner, kernel will buffers message until receiver gets ready to receive, that caused data copied twice, So, Mach's performance loss much in this way.

The second generation microkernel (e.g., L4) adopted synchronous IPC between threads. In this manner, first IPC partner blocks until second one gets ready, threads direct copy messages between sender and receiver, no in-kernel buffering. So, synchronous IPC satisfied efficient implementation necessary.

Through the data transmit manner, the IPC be divided into short, direct long, indirect long IPC. And from function, it have send, receive, receive_any, call, reply_and_wait 5 types.

3.4. Mappings

Mapping is a very important mechanism for L4 system exchange data. Threads can map pages from their address space to other address spaces. Page is a manager of user memory, the while kernel manage page tables. This is achieved by adding a flexpage descriptor to the IPC message buffer. A flexpage is a contiguous region of virtual address space. How do the flexpages describe mapping? The flexpage descriptor involves the location and size of memory area, receiver's rights which is read-only or read-writable, and the mapping type to be one of the memory, IO, communication capability mapping. After mapping has finished, microkernel needs to adapt page tables accordingly.

3.5. Syscalls

The microkernel provides a total of seven system calls, in other words, microkernel rule the world with only 7 system calls. These calls provide some very rudimentary OS functionality. Everything else must be built on top, implemented by server threads, which communicate with their clients via IPC.

In L4/Fiasco, the descriptions of these calls are like follow:

- Address spaces
  - l4_task_new – create/delete tasks
- Threads
  - l4_thread_ex_regs – create/modify threads
  - l4_thread_schedule –setup scheduling parameters
  - l4_thread_switch – switch to another thread
- IPC
  - l4_ipc – perform IPC
  - l4_fpage_unmap – revoke flexpage mapping
  - l4_nchief – find next chief

3.6. Scheduling

The scheduling of L4/Fiasco is preemptive priority-based scheduling. In Fiasco, round-robin scheduling among threads of the same priority level, and priority inheritance via helping and IPC time donation mechanisms.

4. L4/Fiasco Programming Model

L4/Fiasco programming model is based on client/server architecture. To write a client/server application, except the implementation of the function, the most important part is communication among them. Writing IPC code manually is inefficient. We can use IDL (Interface Definition Language) and Dice (DROPS IDL Compiler) [13] to automate this part. We just need to write a description of the server's interface using an IDL, then Dice will generate IPC code.

Next, a simple client/server example will be introduced, including its implement structure, boot menu, and result. All of these demonstrate the process of building and running applications on L4/Fiasco.

4.1. A Client/Server Example

The implement structure of this example is as Figure 2.

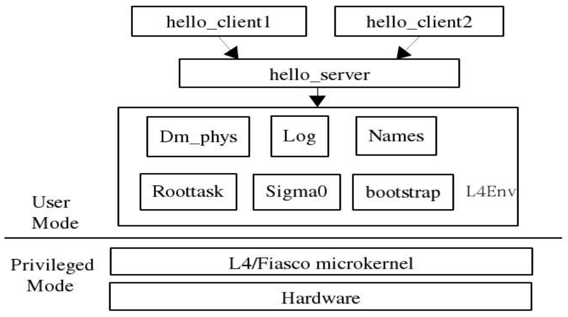

Figure 2. The implementation structure

This example has one server and two client applications. The server has the function of printing “Hello World!”, and two clients call the servers respectively to make it work. They are all running in independent address space.

The IDL file of server's interface as follow:

```
library cs_hello {
  interface hello {
       void hello();
  };
};
```

Compiled this file with DICE, which will generate function interface in relative server.h file, and we need to implement it function body as follow.

```
 Void cs_hello_hello_hello_component
(CORBA_Object _dice_corba_obj,
   CORBA_Server_Environment *_dice_corba_env)
  {
   LOG("Hello World!");
  }
```

After building link between server.h and main function of server, the client could access to this interfaces.

4.2. Boot Menu

Microkernel is the components independent system. And only with Fiasco kernel nothing could be done. Now, let us have a look how this simple example runs on a machine and what essential L4Env modules be needed?

The booting menu.list of Hello:

```
title Hello
kernel /boostrap -serial -modaddr 0x2000000
module /fiasco -serial_esc -comport 1
module /sigma0
module /roottask
module /names
module /log
module /dm_phys
module /hello_server
module /client1
module /client2
```

When machine start, the boostrap is used to boot fiasco kernel, and the modaddr argument tells to bootstrap where to copy the multiboot modules. Sigma0 manages the user space memory. And roottask start the all basic services. Next the log will register and store applications information so as to other applications could find them in names, but before application register in, system use names to find where is log, so the names and log modules are needed. Loading applications in memory, system will allocate data space for them and need data space manager dm_phys to manage it.

All these modules communicate by IPC. It is no access to components besides its interface, which make the system safer and more isolate.

4.3. Results

The running results show as follow. First, client 1 calls the server, and the server prints "Hello World!", then client 1 returns. In the same way, client 2 carries out the same function.

```
client1 | main(): This is client 1 call server to print:
cs_hello| cs_hello_hello_hello_component(): Hello World!
client1 | Main function returned.
client2 | main(): This is client 2 call server to print:
cs_hello| cs_hello_hello_hello_component(): Hello World!
client2 | Main function returned.
```

5. Conclusion

In this paper, in order to present a different architecture for operating system education and raise attention of system safety in operating system study, the microkernel based mechanism was introduced. The Fiasco kernel is a free and flexible system, which is not only suitable for big and complex systems, but also for small, embedded applications. At present, some L4 based systems have being used to mobile phone for their outstanding advantage of safety and virtualization technique. Therefore, taking it for education will broaden learner's view and it is meaningful.

6. References

[1] http://os.inf.tu-dresden.de/L4

[2] Microkernel-Based Operating Systems Lecture Documentation, http://os.inf.tudresden.de/Studium/KMB/WS2008/

[3] Fiasco, http://os.inf.tu-dresden.de/fiasco/

[4] Cheng Guanghui, Nicholas Mc Guire, "L4/Fiasco/L4Linux Kickstart", Distributed & Embedded Systems Lab, Lanzhou University, December, 2006.

[5] Dresden Real-Time OPerating System, http://os.inf.tu-dresden.de/drops/overview.html, 2002.

[6] H. Härtig, R. Baumgartl, M. Borriss, Cl.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, J. Wolter, "DROPS — OS Support for Distributed Multimedia Applications", *Proceedings of the Eigth ACM SIGOPS European Workshop* *(EW 98)*, Sintra, Portugal , September 1998.

[7] Jochen Liedtke, "On u-kernel construction", *In Proceedings of the 15th ACM Symposium on OS Principles*, pages 237-250, Copper Mountain, CO, USA, December 1995.

[8] Operating System Research Group, "— L4Env — An Environment for L4 Applications", Operating Systems Research Group Technische Univerität Dresden, June, 2003.

[9]Adam Lackorzynski, "L4Linux on L4Env", Dresden University of Technology Department of Computer Science Operating Systems Group, December, 2002.

[10] Alan Au, Gernot Heiser, "L4 User Manual Version 1.14", The University of New South Wales, Sydney 2052, Australia: School of Computer Science and Engineering, March, 1999

[11] Jochen Liedtke, "Improving IPC by kernel design", In Proceedings of the 14th ACM Symposium on OS Principles, page: 175-88, Asheville, NC, USA, December 1993.

[12] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, J. Wolter, "The Performance of μ-Kernel-based Systems", Appeared at 16th SOSP; also in ``Wiss. Beiträge zur Informatik", TU Dresden, Fakultät Informatik, Heft, 1997

[13] Ronald Aigner, "DICE Version 3.1.1 User's Manual", Technische University, Dresden, Germany, January 19, 2007.