

L4eRTL: Port of eRTL(PaRTiKle) to L4/Fiasco microkernel

Guanghui Cheng, Nicholas Mc Guire, Qingguo Zhou, Lian Li

Distributed and Embedded System Lab
School of Information Science and Engineering
Lanzhou University
Tianshui South Road 222, Lanzhou, P.R. China
cheng.guanghui@gmail.com

Abstract

L4/Fiasco is a real-time microkernel developed by Dresden Real-Time Operating System Group of Technical University Dresden and it is one of L4-family microkernels with the support of real-time capability. As said microkernel could only provide mechanisms not policy so based on Fiasco L4Env is developed to satisfy the different requirements as a available microkernel-based operating system. For a real-time kernel the posix real-time extension (pse-51) is widely accepted by industry and academic. However, now Fiasco can't support PSE-51 either. eRTL is developed by Real-Time Systems Group of the Universidad Politecnica de Valencia and it is aimed to embedded real-time application and as part of descendent version of RTLinux/GPL 3.0 it could support PSE-51 standard and some new extension only belonged to RTLinux such as `pthread_delete_np`. eRTL is well designed and implemented to be suitable for different environment such as the standard alone eRTL or as a domain in the XtratuM nanokernel from the same group as the eRTL. Therefore, it is reasonable to port eRTL to L4/Fiasco and provide an pse51 interface as the famous RTLinux.

1 Background

1.1 DROPS

The Dresden Real-Time Operating Systems Project shortly named DROPS is a research project in Technical University Dresden aiming at the support of applications with Quality of Service requirements in Technical University Dresden. This project is based on the second generation L4 microkernel named Fiasco written by C++. For the real-time capability Fiasco kernel is preemptible and it could only provide the basic mechanism like management of address space (task), thread, memory, and more importantly the implementation of synchronous fast ipc. The simple DROPS architecture is shown in Figure 1.

Around this microkernel lots of services totally named L4Env are developed aiming at a available real operating system like others. `roottask` is a basic resource manager including physical memory, interrupts, tasks and address spaces; `DDE` is a framework

used to port Linux device driver into DROPS; `DOPE` is a real-time window manager for GUI application. More, the creation, deletion of thread and semaphore is also provided, too. However, it is not posix compatible.

L4Linux plays a shining role in the DROPS and it could provide the same API/ABI as the widely-used Linux. Detailly L4Linux makes use of a pseudo "L4 CPU" as the physical CPU does and pure software "L4 CPU" is made up by lots of components from L4Env. From this feature it is possible to support more than 1 "L4 CPU" at the same time and so in fact they develop a paravirtualization technology based on L4 microkernel against Xen hypervisor.

Normally real-time application is divided into 2 parts: real-time part and non real-time part. Real-time part could run directly on the preemptible fiasco kernel and non real-time part could run an general purpose operating system like Linux. So in the real-time application L4Linux could give the environment for non real-time part with the use of abundant

of Linux libraries and applications. Hence, DROPS could provide the real-time solution as the parallel execution of the real-time task and non real-time task as described in Figure 1.

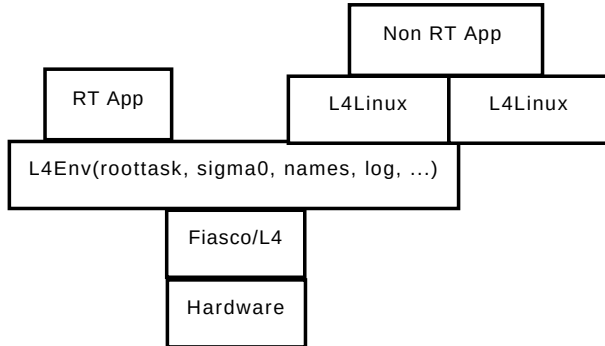


FIGURE 1: *Architecture of DROPS*

1.2 Real-Time in DROPS

IPC is one of the most famous attributions in the all the L4-family microkernels and since Professor Jochen Liedtke lots of researchers focus on fast IPC for many years. But in the real-time system IPC could be the potential cause of priority inversion when the high priority thread is trying to communicate with the low priority thread which is preempted by the middle priority. In the Fiasco the donation mechanism is invented to reduce the side effect of the IPC priority inversion. When the client tries to communicate with the server the client will donate its time quatum and priority to the server, vice versa.

About the resouorce management in the real-time community the separation of real-time part and non real-time part is still used not just in the calcuation task. DOpE (Desktop Operating System Environment) separates the best effort screen updating into two parts: client redraw requests and server redraw operation in order to guarantee the QoS of real-time screen updating request.

1.3 eRTL

Offically eRTL could be considered as the successor of RTLinux/GPL which is too old to track the steps of mainstream linux kernel because of the problems in the architecture. But the interface of RTLinux is so friendly to use that eRTL only keeps the same interface but the architecture is redesigned. For many different kinds of applications with real-time requirements eRTL is designated to be portable and expandable.

PaRTiKle could provide the same real-time interface as RTLinux/GPL. But differently PaRTiKle could run on top of bare hardware as a single address space operating system aiming at the extreme requirements of real-time capability and a few functionality. At present this kind of PaRTiKle could run on top of ARM and x86 to be aplied in the robotics system. But PaRTiKle could also be executed in the different domain separately from Linux kernel based on a nanokernel named XtratuM. This model of PaRTiKle is more robust and safer than the first single address space model but potentially with reduced real-time performance. In PaRTiKle the arch-dependent and arch-indepdent code is separated very well like in the Linux kernel. The directory architecture of PaRTiKle kernel are described below:

XtratuM [13], developed by the UPV (Universidad Politecnica de Valencia) has been rapidly evolving in the last several years. Currently it is at version 2.2 . XtratuM is a nanokernel which mean it does not include IPC mechanims in the kernel core and thus it is smaller than a typical microkernel. Basically XtratuM provides interrupt and timer virtualization, minimum domain management and very simple memory management so it can support concurrent execution of many PaRTiKle. At first XtratuM adheres to Linux like as the traditional RTLinux does. But XM2 has been developed to be dependent from the Linux kernel. Similarly in the implementation of XtratuM the architecture-depdnent part and architecture-independent part are separately very nice so it is so portable to run on X86, MIPS, PowerPC and Sparc.

TLSF(Two-Level Segregate Fit) allocator is a general purpose dynamic memory allocator designed to meet real-time requirements with bounded response time limitation, fast memory allocation and efficient memory usage. TLSF is O(1) cost for memroy malloc, free, realloc and memalign with very low overhead. Now TLSF allocator is also used in many other projects.

2 Design and Implementation of L4eRTL

For L4eRTL, we have two basic solution related to porting PaRTiKle to L4/Fiasco. The first method is to port XtratuM to Fiasco first. This seems easier because we only need to touch the very few XtratuM hypercalls (system calls). But if we port XtratuM on top of Fiasco and we try to support many instances of PaRTiKle we have two solutions like this:

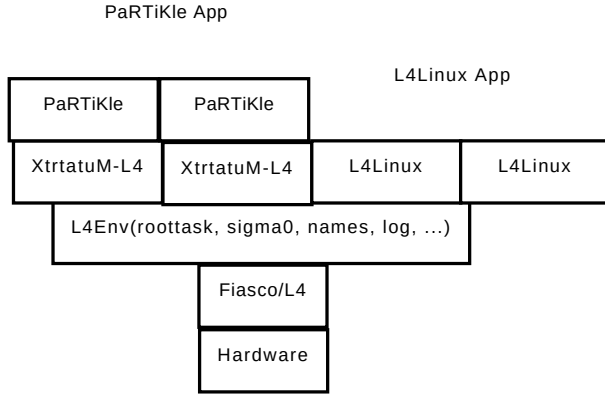


FIGURE 2: *Architecture of L4eRTL:1*

While this is straightforward porting, we can see that there are many XtratuM-L4 instances in the whole system. Of course what we need to do is little but performance loss can be expected.

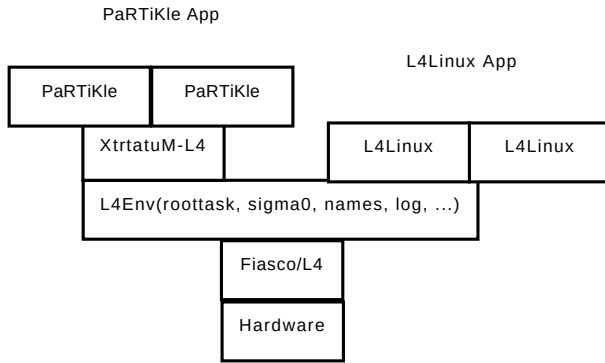


FIGURE 3: *Architecture of L4eRTL:2*

In this style if we design XtratuM-L4 as server, just like other L4 components, in which case we baically need to rewrite the XtratuM layer and generate different kinds of IDL files. In this mode there is the potential for substantial additional IPC related cost between PaRTiKle and XtratuM-L4. Further XtratuM has been moving on to XM2 which has seen a major redesigned of the core, So we can't keep our design bound to the XM 1 code.

Both of these is a departure from our actual target. Our first target is to port a POSIX interface for L4/Fiasco, but in the system the POSIX interface is provided by PaRTiKle not XtratuM.

Finally we choose to port PaRTiKle to L4/Fiasco directly and due to clean layering of the code, only a relatively small part of PaRTiKle source code needs to be reimplement with l4-specific APIs. So the architecture should be like this:

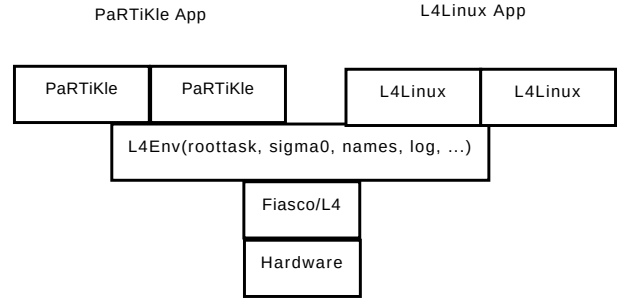


FIGURE 4: *Architecture of L4eRTL:3*

2.0.1 Hints in the Port

There are several aspects to be mentioned about the porting of eRTL to L4/Fiasco:

- time
Just as most operating systems, PaRTiKle provide two kinds of clocks a realtime and a monotonic clock source. The Real-time clock keeps track of current time which is a relative to an absolute (typically external) time based, on most systems 1970/1/1 is used as reference data, this clock can be adjusted and thus can stall or even go backwards. Monotonic clock on the other hand, means the clock value is monotonically increasing, it represents monotonic time since some unspecified starting point - the maching boot time in the most systems. In the X86 platform the real-time clock is derived from the RTC and monotonic clock is based on the TSC.

- timer

Fiasco doesn't provide the timer-virtualization or similar mechanism which could implement the user-space timer because Fiasco itself handles the timer interrupt. One solution about timer emulation in Fiasco is to (missuse) IPC timeout because Fiascos IPC is synchronous and thus when the timeout expires this IPC is cancelled which notifies the calling application. We could call this solution as the software timer, but it can't provide sufficiently accurate timer intervals compared with the hardware because in Fiasco timeouts are calculated by power expression $[\text{man} * 4^{(15-\text{exp})}]$ with an exponent and a mantissa in order to save some register bits, leading to a relatively low resolution. Never the less we are currently using the software timer to emulate the timer interrupt in this implementation. In the following code the `l4_rcv_timeout` converts the absolute

time to a timeout and `l4_ipc_receive` will return when the timeout expires.

```
pint=l4x_kinfo->clock;
//l4x_kinfo is the kernel info
//page which stores all the
//global information including clock.
while(1)
{
    pint += 10000;
    //about 10 milliseconds = 10000
    //microseconds
    l4_rcv_timeout(l4_timeout_abs(pint,
        L4_TIMEOUT_ABS_V1_ms), &to);
    //absolute time into relative timeout
    l4_ipc_receive(l4_myself(),
        L4_IPC_SHORT_MSG, &d1,
        &d2, to, &result);
    //wait for itself for the timeout
    l4_do_IRQ();
    //timer interrupt handler
}
```

Another option is to use real hardware to generate the timer interrupt, a normal PC-compatible machine has three devices available PIT (Programmable Interrupt Timer), RTC (Real-Time Clock) and APIC (Advanced Programmable Interrupt Controller). One of them is exclusively assigned to Fiasco and the two others are free to use for any other applications. In the Fiasco system it is allowed to use the user-space device driver. So the hardware clock for L4eRTL will be the choice in the future to remove the timer related issues found in the first implementation.

- hardware interrupt

In the Fiasco kernel all the hardware interrupts except the timer interrupt are translated into synchronous IPC processed then by the respective interrupt handler thread in user space. Normally `l4io`, which is a user-space resource handler, will manage the interrupts which it got passed on from the `omega0` (basic hardware interrupt multiplexer). And then application can do the actual processing of the interrupt it received from the `l4io` server. Thus hardware interrupts can be configured to be handled by L4Linux or L4eRTL or any user-space applications, though with some overhead due to the noted layered architecture.

- memory

L4eRTL executes in the user space on a L4/Fiasco system, so it only handles virtual

memory. For efficiency a continuous pinned memory block with predefined size is reserved for L4eRTLs main thread exclusively. And then this pool of memory is used to initialize by TLSF allocator. All application (thread) level memory operation in the L4eRTL are then handled via the TLSF allocator allowing dynamic memory resources in the bounded by the initial pool size.

- L4eRTL thread and L4 thread

A L4eRTL thread is different from L4 thread which is scheduled by L4 microkernel: the L4eRTL thread is scheduled by the internal L4eRTL scheduler which is one of the core PaRTiKle functions. For L4 kernel the L4eRTL appears only to be a single task which consists of L4eRTL timer thread, L4eRTL main thread and several L4eRTL interrupt handler thread if necessary. So the L4 kernel doesn't know anything about L4eRTL threads (or generally the internals of a task).

3 Conclusion and Discussion

Currently L4eRTL runs some basic tests-cases using a PSE51 compliant interfaces but the real-time performance of L4eRTL is not acceptable, most notably it is highly dependant on load produced in other tasks i.e. when a L4Linux instance is coexisting with the L4eRTL. Initial analysis suggests that the software timer is one of the main issues for the bad real-time performance so the next step is to fix it by utilizing a hardware timer. Generally the temporal isolation in L4/Fiasco does not seem to be as good as we had expected, though we are not yet sure if this must not be attributed to the current implementation of our L4eRTL. Improving the real-time performance of L4eRTL will thus be the main focus in the near future.

4 Acknowledge

Prof. Nichols Mc. Guire given us some important technology support. DROPS team from Technical University, Germany, gave us some help with this port. Most of work was done when the author was a guest student in the DROPS team, for which we would like to thank Prof Herman Haertig. DSLab offered resources and the appropriate surroundings for this development.

References

- [1] ,DROPS Overview,Hermann Haertig, <http://os.inf.tu-dresden.de/drops/overview.html>
- [2] ,Ten Years of Research on L4-Based Real-Time, Michael Roitzsch, Hermann Haertig, Proceedings of the Eighth Real-Time Linux Workshop, Lanzhou, China, 2006
- [3] ,Probabilistic Admission Control to Govern Real-Time Systems under Overload, Claude-J. Hamann, Michael Roitzsch, Lars Reuther, Jean Wolter, Hermann Haertig Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS 07), July 2007
- [4] ,Principles for the Prediction of Video Decoding Times applied to MPEG-1/2 and MPEG-4 Part 2 Video Roitzsch, Pohlack, Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS), Rio de Janeiro, Brazil , 2006
- [5] ,Fast Component Interaction for Real-Time Systems, U. Steinberg, J. Wolter, H. Haertig Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05), Palma de Mallorca, Spain , July 2005
- [6] , Low-latency Hard Real-Time Communication over Switched Ethernet J. Loeser, H. Haertig Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004), Catania, Italy , June 2004
- [7] ,Rotational-Position-Aware Real-Time Disk Scheduling Using a Dynamic Active Subset (DAS) L. Reuther, M. Pohlack Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), Cancun, Mexico , December 2003
- [8] , Demonstration of DOpE - a Window Server for Real-time and Embedded Systems N. Feske, H. Haertig Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), Cancun, Mexico , December 2003
- [9] , OS-Controlled Cache Predictability for Real-Time Systems, J. Liedtke, H. Haertig, M. Hohmuth Proceedings of the Third IEEE Real-time Technology and Applications Symposium (RTAS'97), Montreal, Canada , June 1997
- [10] , The Performance of uKernel-based Systems, H. Haertig, M. Hohmuth, J. Liedtke, S. Schoenberg, J. Wolter Appeared at 16th SOSP, 1997
- [11] , TLSF: A New Dynamic Memory Allocator for Real-Time Systems ,Miguel Masmano,Ismael Ripoll,Alfons Crespo, ECRTS, 2004
- [12] , Quality-Assuring Scheduling in the Fiasco Microkernel, Udo Steinberg, Diploma Thesis Paper, 2004
- [13] *XTRATUM: AN OPEN SOURCE HYPERVISOR FOR TSP EMBEDDED SYSTEMS IN AEROSPACE*, A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J.J. Metge, 2009