# Developing with L4 – Overview and Pointers

Carsten Weinhold

11. April 2012

## 1 Introduction

### 1.1 Scope of this Document

This document aims at helping developers who are new to L4 with their first steps. In particular, it describes how to get started with the L4/Fiasco microkernel, the L4Env user-level packages, and L4Linux and it discusses the various aspects and requirements of the development environment. As there already exists a lot of documentation for the individual components, this document will merely give an overview and provide pointers to the more detailed documentation and tutorials.

### 1.2 Names and Terminology

**L4** is the name of a second-generation microkernel originally developed by Jochen Liedtke. However, nowadays it is also a specification for various implementations of the L4 API. One of these implementations is **Fiasco**, which is being developed at TU Dresden. It is sometimes referred to as **L4/Fiasco** to emphasize this relation. **L4Env** comprises a set of user-level services and libraries that make using L4/Fiasco easier and provide a basis for an operating-system built on top of L4.

As L4/Fiasco is real-time capable, it also the foundation of the **Dresden Real-Time Operating System (DROPS)**. It supports parallel execution of real-time and time-sharing applications. The latter also include **L4Linux**, a para-virtualized version of Linux, and unmodified Linux programs running on top of it. An implementation of the **Nizza Secure-System Architecture**, formerly known under the name **uSINA Architecture**, is based on L4/Fiasco as well. It allows executing security-critical applications in parallel to untrusted applications and one or more L4Linux instances.

## 2 Development Environment

### 2.1 Development Platform and Tools

The development platform for L4/Fiasco, L4Env, and L4Linux is GNU/Linux on an x86 host. In order to build all software components from source, you will need standard Linux development tools such as the GNU Compiler Collection (GCC) and GNU make. So, before you get started, you should make sure that you have all the required tools installed on your development machine. Table 1 lists all development tools and the required version numbers. In addition to that you will of course need your favorite text editor.

You do not necessarily need a separate partition on your hard disk, however, you will need some free disk space. This depends on how many software components you want to check out, however **2 GB** should be enough for a start.

| Development Tool | Required Version |
|---|---|
| GCC | 3.x |
| GNU make | >= 3.77 |
| Binutils | >= 2.13.90 |
| GNU bash | >= 2.05 |
| GNU find | |
| awk | POSIX compatible |
| GNU sed | |
| GNU perl | >= 5.6.1 |
| flex | |
| byacc / bison | |
| Doxygen (recommended) | >= 1.2.15 |
| autoconf | >= 2.59 |
| automake | >= 1.4 |
| ncurses-dev(el) | |
| python-curses | |

**Tabelle 1:** Development tools required for building Fiasco, L4Env, and L4Linux from source.

## 2.2 Test-Bed Infrastructure

### 2.2.1 Running L4/Fiasco and User-Level Components

Once you started development with L4-related components, you will need to test them. You can choose among various setups depending on your needs:

**Real Hardware** Of course, you can run your L4 components natively on real hardware. As rebooting your development machine for testing is extremely time consuming, you should have a dedicated test box available.

**Virtual Machines** Another option is to run the microkernel and the user-level components inside a virtual machine, such as VMware or QEmu, that is hosted on your development machine. This solution can save a lot of time in some use cases, however, you should be aware that the emulation also reduces performance.

**Fiasco-UX** Fiasco-UX is a user-mode port of the Fiasco microkernel. It allows you to run Fiasco as a normal Linux program. You can run L4 applications and even L4Linux on top of Fiasco-UX, however, you will have only a limited set of "hardware" devices available. When using Fiasco-UX, it is possible to access the frame buffer of a virtual graphics card. Using special versions of some of the L4Env servers, you can also use the real-time clock, networking, and you get limited file-system support trough a UX-aware implementation of the File-Provider interface.

### 2.2.2 Boot Methods

Fiasco-UX can directly use the program binaries in the host file system that were created when compiling L4 packages. However, if you use a separate test box or an emulator, you will need some infrastructure in order to provide binaries and data files. To boot a machine with L4/Fiasco, we use a modified version of the GRUB boot loader. GRUB is capable of booting from either a local storage device such as a hard disk or network. Booting from hard disk is inconvenient in most cases, as you need to transfer all files to the test machine, which probably needs to be booted into Linux for copying.

The best option is to set up your machines for network boot. GRUB retrieves all files from a TFTP server, which is running either on the development machine or on a dedicated server. In the latter case it is useful

to be able to mount the exported TFTP boot directory via NFS, so that all binaries can be transferred to it easily during the normal build process. To inform GRUB about the IP address of the TFTP server, it is very handy to have a DHCP server running on the same network. This DHCP server needs to be configured to provide this IP address as the "next server" address. For example, the configuration file for the commonly used DHCP server of the Internet Software Consortium could contain an entry similar to the one below:

```
host testbox {                          # Hostname of the machine
 hardware ethernet aa:bb:cc:dd:ee:ff; # Hardware address of the
                                        #   machine
 fixed-address     192.168.0.2;        # IP address of the machine
 next-server       192.168.0.1;        # TFTP server address, can be
                                        #   omitted if same machine
                                        #   as DHCP server
}
```

In this case, `192.168.0.1` is the IP address of the TFTP server. For more detailed information on how to set up a DHCP server you should refer to the according documentation.

## 3 Getting the Sources

There are two ways to get the source code of Fiasco, L4Env, and L4Linux:

- **Download a release tarball.**

  Release tarballs are considered stable. We provide tested and up-to-date snapshots for our project partners. Currently, we do this within the context of the OpenTC project. You can find download links and build instructions at the following location:

  `http://os.inf.tu-dresden.de/opentc/`

- **Check out the sources from remote CVS.**

  The remote CVS is always up to date and expected to compile. However, things may still break because of incompatible changes and incomplete features. You can find Instructions on how to get access to our remote CVS repository at the following location:

  `http://os.inf.tu-dresden.de/drops/download.html`

  Additional information for partial checkouts is provided in Section 3.1 of the DROPS Building Howto:

  `http://os.inf.tu-dresden.de/l4env/doc/html/building-tut/index.html`

  `http://os.inf.tu-dresden.de/l4env/doc/html/building-tut/tut.pdf`

## 4 Configuring and Building

### 4.1 How to get it running?

Instructions for building L4/Fiasco, the L4Env user-level packages, and L4Linux can be found in the following documents. You should start with the first one, the *DROPS Building Howto*, as it allows you get an overview of all the components. Then consult the Fiasco and L4Linux documentation to find out how to change the default configuration.

- **DROPS Building Howto**

  The DROPS Building Howto contains a quickstart guide as well as more detailed documentation regarding configuration and building of L4 components:

http://os.inf.tu-dresden.de/l4env/doc/html/building-tut/index.html

http://os.inf.tu-dresden.de/l4env/doc/html/building-tut/tut.pdf

- **Building and Using Fiasco and Fiasco-UX**

  http://os.inf.tu-dresden.de/fiasco/use.html

  http://os.inf.tu-dresden.de/fiasco/ux/building.shtml

  http://os.inf.tu-dresden.de/fiasco/ux/using.shtml

- **Building and Using L4Linux**

  http://os.inf.tu-dresden.de/L4/LinuxOnL4/build-2.6.shtml

  http://os.inf.tu-dresden.de/L4/LinuxOnL4/use-2.6.shtml

### 4.2 Writing own L4 packages

If you want to write your own user-level packages for the L4 environment, you should be familiar with the build system: the **Building Infrastructure for DROPS (BID)**. The BID is being used for building all user-level packages except for L4Linux, which uses its own build system. The BID documentation is split into two parts:

- **BID Tutorial**

  To create new L4 packages, you can follow the examples in the BID Tutorial:

  http://os.inf.tu-dresden.de/l4env/doc/html/bid-tut/index.html

  http://os.inf.tu-dresden.de/l4env/doc/html/bid-tut/tut.pdf

- **BID Specification**

  The BID spec describes BID in greater detail and is very useful for reference:

  http://os.inf.tu-dresden.de/l4env/doc/html/bid-spec/index.html

  http://os.inf.tu-dresden.de/l4env/doc/html/bid-spec/spec.pdf

The *BID Tutorial* and the *DROPS Building Howto* mention **DICE**, the IDL compiler used for L4Env-based packages. DICE generates C or C++ glue code from IDL files, which specify communication interfaces for L4 components. DICE and IDL processing are integrated into BID. Comprehensive documentation including a user manual can be found at the following location:

http://os.inf.tu-dresden.de/DICE/

### 4.3 Documentation for L4Env Packages

Most of the L4Env packages also have documentation covering their APIs and sometime even internal interfaces. You can it at the L4 documentation page:

http://os.inf.tu-dresden.de/l4env/doc/index.xml

## 5 Debugging L4 components

To be able to debug L4 components, you will need a debugging tool. L4/Fiasco contains the kernel debugger **JDB**, which is not only useful for hunting bugs in the microkernel itself, but also for debugging user-level programs. JDB is available in both Fiasco-UX and native Fiasco. When being used on real hardware, JDB will be controlled via serial console. That is, you have to link your development machine and your test box

with a serial cable (also known as null-modem cable). You can then interact with JDB using a terminal program such as `minicom`.

JDB is extensively documented in the *Fiasco Kernel Debugger Manual*, which can be found at the following location:

`http://os.inf.tu-dresden.de/fiasco/doc/jdb.pdf`

The manual also includes a description of JDB-related command-line options for Fiasco that you need to specify in the configuration file of your boot loader (i.e, TUD-enhanced GRUB, as mentioned in Section 2.2.2). For example, an entry in GRUB's configuration file `menu.lst` could look as follows:

```
title L4/Fiasco with JDB
kernel $(R)/bin/bootstrap -serial
modaddr 0x02000000
module $(R)/fiasco -nokdb -nowait -serial -comport 1 -serial_esc
module $(R)/bin/sigma0
...
```

In this case, the command-line options `-serial`, `-comport 1`, and `-serial_esc` will allow you to control JDB via a null-modem cable connected to the test box' first COM port. Finally, the following command allows your development machine to connect the test box with the other end of the cable being plugged into COM port 2:

```
#> minicom /dev/ttyS1
```