

# L4Sys Fault Injection Campaign – User Manual

Martin Unzner (munzner@os.inf.tu-dresden.de),  
Björn Döbel (doebel@os.inf.tu-dresden.de),  
Tobias Stumpf (tstumpf@os.inf.tu-dresden.de)

June 20, 2014

## Abstract

This document describes how to use the L4Sys experiment suite. However, this is not a complete documentation. When in doubt, please read the source code or contact us. Still, we would like you to read this whole document before investigating further.

## 1 Overview

This is the user manual of the L4Sys generic system test framework. The framework builds on Fail\* and provides means to perform fault injection experiments for applications running on top of the Fiasco.OC/L4Re microkernel-based operating system as well as the underlying microkernel.

L4Sys provides two experiment types:

1. *GPRFLIP* simulates bit flips in general purpose registers.
2. *MEM* simulates memory bit flips.

L4Sys currently works for x86/32 running in Fail/Bochs only. This is partly due to some issues with timing — as soon as a valid model of time in the target emulator as well as an assembler/disassembler functionality in the Fail\* framework are established, I would recommend a backend change, as Bochs' reliability is very limited.

## 2 Framework Setup

To prepare a fault injection campaign you will first need to configure and build Fail\* itself. This process is described in `doc/how-to-build.txt`. The following CMake flags need to be set:

- `BUILD_BOCHS = ON`
- `BUILD_X86 = ON`
- `CONFIG_BOCHS_NO_ABORT = ON`
- `CONFIG_EVENT_BREAKPOINTS = ON`
- `CONFIG_EVENT_IOPORT = ON`
- `CONFIG_SR_RESTORE = ON`

- `CONFIG_SR_SAVE = ON`
- `EXPERIMENTS_ACTIVATED = 14-sys`

Enabling `CONFIG_FAST_BREAKPOINTS` may speed up the experiment clients significantly. Enabling `CONFIG_BOCHS_NO_ABORT` is necessary to detect whether Bochs stopped because of a bad instruction (currently not used). Keep in mind that this implies the risk of a deadlock in the campaign system, because packets (i.e. experiment descriptions) are resent if the client does not answer and finally, all clients might fail because they tried to execute the same faulty instruction.

### 3 Emulator Setup

The next step is to prepare a L4Re application setup to run in Bochs. To setup your system, first, you need a dedicated `bochsrc` file. It has proven useful to have a Bochs resource file or an independent Bochs instance with GUI enabled for the initial testing, however the experiments are intended to be conducted without graphical output.

Bochs should be booted using a CD image containing your setup. To obtain this setup, first build Fiasco.OC and L4Re separately as described in their respective build instructions. Make sure your setup is running, e.g., in QEMU. Once this works, create an ISO image using the L4 build system's `make grub2iso E=<entry>` command. Validate that this ISO boots and runs in Bochs.

### 4 Client Setup

Now that we have Fail\* and the L4Re setup running, we can prepare our fault injection campaign. This requires three (+ one optional) steps:

1. *OPTIONAL*: If we want to perform a campaign that only targets a single application, we need to determine this application's address space ID. Choosing a different link address (add `DEFAULT_RELOC = 0x20000000` in your applications Makefile) helps you to figure out your address space by stopping at a unique instruction pointer.
2. *REQUIRED*: We perform an initial run of our setup in Bochs until the point where Bochs is booted and the application in question starts. At this point we take a snapshot of the emulator so that we can skip everything upfront in the remaining runs.
3. *REQUIRED*: The L4Sys campaign uses the number of instructions to determine the set of instructions to inject faults in. We need to perform one run of our setup to determine this number.
4. *REQUIRED*: We need to perform a *golden run* without any fault injections. Later faults are then compared against this run.

All parameters of the L4Sys experiment can be configured via file `experiment.conf` or can be given to `fail-client` as command line parameter.

## 4.1 Constants

Some configuration values are constant throughout all steps of the preparation and also when the workload program is run. The most important configuration parameter is `emul_ipc`, which has to be consistent with your `bochsrc` setting and is used for several timely calculations in the client.

## 4.2 Preparation

First, we need to find the start and end instruction addresses for our workload program and our given experiment. For this purpose, use a disassembler, such as `objdump` or *IDA Pro*. Determine the first and last instruction for your campaign and set `func_entry` and `func_exit` in the configuration file accordingly. Additionally, you can limit the range for FI by setting `filter_entry` and `filter_exit`.

All preparations steps can be done together or step-by-step. To start preparation call `fail-client` with the parameter `-Wf,--step=OPTION`. Use `OPTION=all` for doing all preparations steps together.

### 4.2.1 Step 0: Determine the address space (OPTION=cr3)

If you are not interested in address space filtering, you may set `address_space` to 0. Note that in this case you will probably encounter instruction pointers across various address spaces and may not get the unique results you want.

During this step, the option `address_space` of your configuration file is updated.

### 4.2.2 Step 1: Save the initial state of the machine

Make sure you have set `address_space` accordingly. Now recompile and execute the framework code again, this time with the graphical user interface disabled. The experiment client runs until `func_entry` is reached and then saves the complete configuration.

### 4.2.3 Step 2: Determine the instructions to execute

For this part the filter-file (`filter.list` by default) is read to filter out instruction which are not in any of the specified ranges.

After the program has finished, you will get a summary on the total of instructions executed.

### 4.2.4 Step 3: Determine the correct output

The framework runs the complete program and logs the output. You can check the resulting file (by default `golden.out`), and if it does not comply with your expectations of a valid run, you should correct the entry and exit point, the address space or, in the worst case, your Bochs settings.

#### 4.2.5 Step 4: Fill the database with your experiments

Use the tools `import-trace` and `prune-trace` to import the data and reducing the amount of necessary experiments.

You can call both applications with parameters specifying your database setup or using a default config file (`~/.my.cnf`) defining username, password and database name. For `import-trace` you have to specify the following parameters:

- `--importer MemoryImport|RegisterImporter`
- `-e YourImageName`
- `-t NameOfYourTraceFile` (trace.pb by default)

## 5 Campaign Setup

To start a FI-experiment, you need to start both the campaign server (`l4-sys-server`) and the experiment client (`textttl4-sys-server`) needs a parameter to specify the experiment type.

If `l4-sys-server` and `fail-client` are running on different core then the config parameter `campaign_server` should be added to specify the host name or ip addr. of the machine running `l4-sys-server`.

Each experiment client processes exactly one experiment and exits. To complete your campaign, you should use the `client.sh` script in the `scripts` sub-directory of `Fail*`.

## 6 Get your results

Your results are stored in your database. Look at the table `result_L4SysProtoMsg` to see your results.

## 7 Known bugs

If you need support for more than one processor, you will have to extend the code accordingly: at the moment, when in doubt, it uses the first CPU.