Behavioral Analysis of an I²C Linux Driver

Dragan Bosnacki, Aad Mathijssen, Yaroslav S. Usenko
Eindhoven University of Technology

Introduction

Formal methods for the analysis of system behaviour offer solutions to problems with concurrency, such as race conditions and deadlocks. In this work we employ two such methods that are presently most applied by industry: *model checking* and *static analysis*. We use these techniques to analyse the behaviour of a Linux driver for an I²C (Inter-Integrated Circuit) device. We present some experiences and results that we carried out within the Laboratory for Quality Software (LaQuSo), at the Computer Science Department of the Eindhoven University of Technology. The goal of the project was to analyse the feasibility of the techniques used within LaQuSo, like model checking and advanced static analysis, for industrial scale software

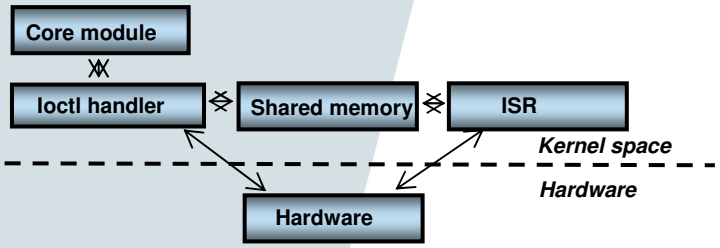

Figure 1: High level structure of the I2C Linux bus driver

The I²C Linux Driver

In general, the Linux 2.6 kernel contains an I²C driver stack that is split up into three layers: chip driver, core module and bus driver, he core module is part of the Linux kernel as are a number of chip drivers and bus drivers. In our case, an I2C bus driver was supplied by the client. The code mainly performs two tasks: handle ioctl calls from user space, offered via the core module, and handle interrupts from the hardware. To find race conditions we focused on the interaction between the two parallel components of the driver: the ioctl handler and the interrupt service routine (ISR) (see Fig. 1).

```
void uno_check(void)
{
 if (uno_state == 0){ //interrupt enabled
    if (select("shmem_var", USE|DEF|REF0,NONE))  //shared memory access
      error("Shared memory reference with enabled interrupts"");
    if (select("int_disable",FCALL, NONE)) //interrupts disabled
      uno_sate = 1;
 }
 if uno_state == 1) {//interrupts disabled
  if (select('int_enable", FCALL, NONE)) //interrupts enabled
     uno_state = 0;
 }
}
```

Figure 3: UNO monitor for erroneous shared memory access

Model Checking with mCRL2

The mCRL2 language and toolset [2] allows users to model and automatically verify the behavior of distributed systems. Systems can be modeled using process algebra enriched with data types. Automated verification is supported by checking temporal properties of all states of the model.

Based on the source code we of the I2C bus driver we created an mCRL2 model consisting of a translation of the ioctl handler and the interrupt service routine and the environment in which these functions occur. For the verification of our model we focused on violation of mutual exclusion of shared memory accesses. Exploration of the complete state space revealed two types of violations: more than 100 concurrent shared memory accesses and one concurrent access to low level functions.

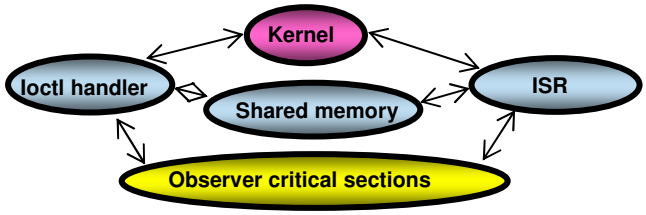

Figure 2: Structure of the mCRL2 verification model

Advanced Static Analysis with UNO

We applied UNO [3] to fins the same violations as reported by the mCRL2 analysis. The mutual exclusion properties needed to be encoded as property automata (Fig. 3). A property automaton monitors the traversal of the control flow graphs of the C functions. UNO produces an error trace in case a violation of the property is found. UNO was able to produce all possible defects that were discovered with mCRL2: the errors of accessing shared memory without previously disabling interrupts and unsafe function calls.

Conclusions

With both model checking and static analysis we found several non-trivial possible defects that were later confirmed by the developers. Furthermore, we provided a verified fix for the found defects. Although in general model checking is a more powerful technique than static analysis, in this case study it seems that they are evenly matched. This is probably due to the low number of parallel components in the checked properties.

References:

[1] D. Bosnacki, A. Mathijssen, Y.S. Usenko, *Behavioural analysis of an I²C Linux Driver,* CS-Report 09/09, Technische Universiteit Eindhoven. 2009
[2] J.F. Groote et al., *Analysis of distributed systems in mCRL2,* in Proc. Alg. for Par. and Distr. Processing, pp. 98-128. Chapman and Hall, 2008.
[3]G.J. Holzmann, *Static source code checking for user defined properties*, in Integrated Design & Proc. Tech., IDPT, 2002.