Ein praktischer Erfahrungsbericht über Model Checking in L4Linux (A real-world experience talk about model checking L4Linux's internals)

Martin Pohlack
Technische Universität Dresden
pohlack@os.inf.tu-dresden.de

2006-11-29

Symptom: L^{4}Linux "deadlocks"

- After some hours of heavy load (wget ...) L^{4}Linux is locked
- Backtrace of 4 L^{4}Linux kernel threads shows:

```
backtrace (thread 11.04, fp=05025b68, pc=00404333):
  #1 05025b68 00404333 : l4x_global_cli + 0x83
      /home/mp26/src/drops/l4linux-2.6/arch/l4/kernel/tamed.c:232
  #2 05025b7c 0044b299 : kfree + 0x19
      /home/mp26/src/drops/l4linux-2.6/mm/slab.c:3456
  #3 05025b90 00527d03 : skb_release_data + 0x53
      /home/mp26/src/drops/l4linux-2.6/net/core/skbuff.c:318
  ...
```

- Tamer is ready to receive:

```
backtrace (thread 11.03, fp=0063b88c, pc=00403270):
  #1 0063b88c 00403270 : cli_sem_thread + 0x130
      /home/mp26/src/drops/l4linux-2.6/arch/l4/kernel/tamed.c:152
  #2 0063b8cc a00acd68 : l4th_thread_start + 0x98
      /home/mp26/src/drops/l4/pkg/thread/lib/src/create.c:74
  ...
```

History

- A lot of effort went into the tamer implementation in L^{4}Linux, also on the last time:

```
revision 1.7
date: 2006-08-03 00:50:45 +0200;  author: adam;  state: Exp;  lines: +8 -10

- fix (hopefully) a deadlock of the system
- big thanks to AlexW and MLP for joining the extensive debugging sessions!

- the tamer thread was supposed to run atomically and thus at the highest
  priority within the l4linux thread universe but it might happen, due to
  donation, that an interrupt thread of a stub shortly has the same or
  higher priority than the tamer thread which then leads to curruption of
  the tamer state, which then can lead to missing wakeups and thus to
  deadlocks
- we hopefully have lifted the atomicity limitation now so that our testcase
  now runs for ages (well, nearly)
- someone wants to prove that? highly welcome...
```

- Developers have the feeling that: "There is still a race in the code, somewhere."
- I was the one to trigger it, repeatedly, while taking measurements

Solution attempt

- Method of "scharfes Hinsehen" failed for us
  - Could not easily find problem in code
  - Could not find error trace
- Formal modeling of problem to let tool construct error trace
- If we had a model, we could try out fixes and new ideas there first

$\Rightarrow$ Model in Promela and model checking with Spin

Problems

- New language Promela, C not directly usable
- Promela is relatively primitive (e.g., no function support: *"There is no mechanism for defining a hierarchically layered system in Promela, nor is there a good excuse to justify this omission. At present, the only structuring principles supported in Promela are proctype s, inline s, and macros ."* — http://spinroot.com/spin/Man/hierarchy.html)
- Model creation took about two days (until *nearly* bug-free)
- Spent several days trying to reproduce the bug in the model

[show source code]

Problems (2)

⇒ The definition of the never claim is important

- I assumed a deadlock and testet for it

```
never
{
  do
    :: assert( ! ((enqueue_count >= 3) &&
                  (tamer_is_waiting == true )))
  od
}
```

Problems (2)

⇒ The definition of the never claim is important

- I assumed a deadlock and testet for it

```
never
{
  do
    :: assert( ! ((enqueue_count >= 3) &&
                  (tamer_is_waiting == true )))
  od
}
```

- I found a livelock (... some days later)

```
never
{
  do
    :: assert( ! ((enqueue_count + threads_outside >= MAX_THREADS) &&
                  (enqueue_count >= 1) &&
                  (tamer_is_waiting == true ) ||   /* Deadlock */
                  (threads_inside > 1)             /* Safety   */
    ))
  od
}
```

Problems (3)

- It was unclear whether the model represents the semantics of the implementation
  - Michael P. and me found another small model inconsistency later (STI loop)
- Huge resource demands for model checking

Solution

- After fixing the model, we (Michael P. and me) tried out Michaels old idea (sending back information in reply message from tamer)
  - Passed live lock never claim
- Took about 40 minutes

Solution

- After fixing the model, we (Michael P. and me) tried out Michaels old idea (sending back information in reply message from tamer)
  - Passed live lock never claim
- Took about 40 minutes
- Violated safety never claim

[show spin / xspin cycle]

Solution (2)

- Alex had another impl. variant which I could falsify within 10 minutes
- Michael P. came up with a new implementation which passes both never claims (now fully checked, depth ca. 200,000 steps)
    - (Maybe fairness issues in solution)

Conclusion

- Formal approach interesting for such problems
- A bit time intensive for model creation, but pays off
  - Relatively high certainty that certain types of bugs are not in the implementation
  - Error search is simple and cheap
  - Proposed changes can be validated very easily in the model
- Semantic similarity between model and implementation (again, “scharfes Hinsehen” required)
- Somewhat cumbersome model creation, also due to Promela limitations
- Definition of never claim might be tricky (but usually also required for real implementation)

Outlook

- Implement Michael P.'s solution in L^{4}Linux and test it
- Online checking of LTL formulae (˜ never claims)
  - No model needs to be built
  - States defined with events
  - Büchi automaton checked in monitor (LTL2Bu etc.)