

Formal Verification of General Purpose Microkernel Implementation



Presented by
Shashank Chaudhary

Contents

- Introduction
- Challenges in Verifying Microkernel
- Models Studied
- Formal Verification Methodology
- Results
- Observations
- Conclusion

Introduction

- Kernels are complex and important part of the Operating System.
- Any application can only be as dependable as its operating system on which it runs.

Introduction

- I studied some of the projects which carried out formal verification of general purpose microkernel's implementation.
- Formal verification of microkernels and kernels in general is challenging.
- Some projects have successfully verified parts of microkernel implementation and paved the way to develop a fully verified general purpose microkernel.

Challenges in Verifying MicroKernels

- Microkernels primarily designed for performance over safety.
- No explicit difference between consistent state and inconsistent state.
- Lack of system programming language with formal semantics

Challenges in Verifying MicroKernels

- Most formal verification tools are not programmer friendly.
- Formal methods is a steep learning curve for system programmers.
- Verification is costly and considered secondary.

Models Studied

- EROS/COYOTOS
- L4 Fluke/seL4
- Verification of microkernel's implementation not only involves formal verification but substantial changes to the model of the microkernel itself.

Formal Verification

- Different Projects follow different methodology for verifying the implementation of the microkernel.
- Verification can be either done by Model Checkers or Theorem Provers.

Formal Verification : COYOTOS

- The COYOTOS project along with verifying general functional and security properties also aim at establishing total correctness to verify that the processes terminate and they maintain the invariant.
- It draws its abstract specification from previous work done in developing a formal system model of the kernel where key system operations were formalized in an operational semantics.

Formal Verification : COYOTOS

- The project identifies the lack of formally specified languages which can provide low level detail for writing a kernel. So the project looks at developing a formally specified language which can specify the low level details sufficient enough to implement the kernel.
- The project looks forward in developing the BitC language which is based on Scheme and C.

Formal Verification: BitC

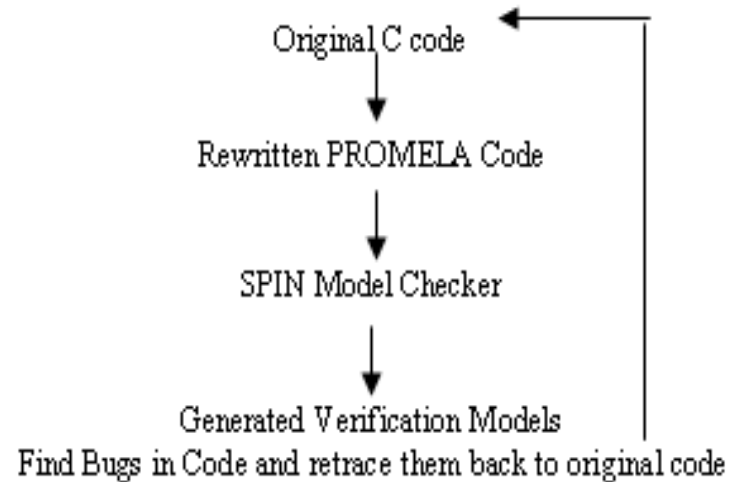
- BitC is developed by extending the Scheme with machine level representation types and C-style structures.
- Several modifications are suggested to make the language directly emitted to C.

Formal Verification: Fluke

- The Fluke project only looks at verifying the IPC subsystem of the kernel
- It uses a model checker namely SPIN to model the system code and use the verification model to trace the bugs back to the implementation.

Formal Verification: Fluke

- Project rewrites the code of the kernel in PROMELA so that it can be checked in the SPIN model checker and verification models generated.

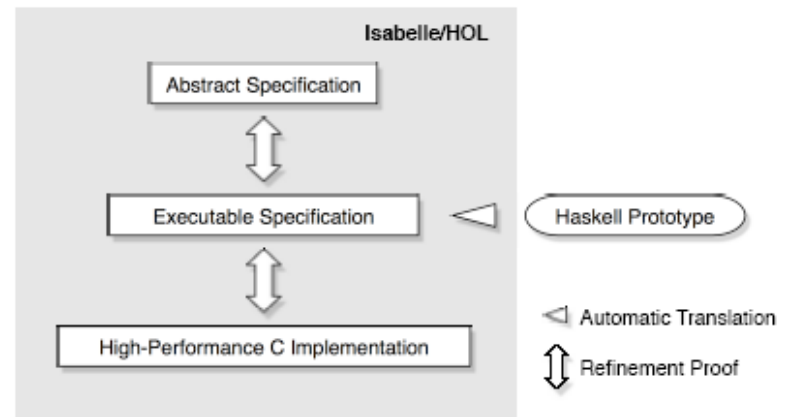


Formal Verification: seL4

- It defines a formal abstract specification for the Kernel Model. It does not model the Compiler code of any of the assembly code and assumes that they are correct.
- Using this defined abstract model it writes the implementation specifications in a strong functional language Haskell. It uses the Isabelle theorem prover for refinement.

Formal Verification: seL4

- It then finally translates this proved Haskell code into C code for implementation.



Results

- Verification of the implementation proves the code to be bug free. Changes to microkernel may be thought of impairing performance. But experiments show performance is not degraded.
- Keeping the system verified as it evolve is difficult especially if it involves fundamental changes into the system

Observations

- Model Checking v/s Theorem Prover for verifying the implementation. Whereas Model checking can't scale for entire kernel due to state space explosion problem. Theorem provers can be used to validate the entire kernel.
- A more efficient functional language is necessary for modeling the code and its automatic translation to implementation code.
- Open Source can offer a way for validating the entire kernel or OS.

Conclusion

- The above projects showed that verifying the implementation of microkernel is possible.
- Work needs to be done in both microkernel design and formal verification tools and functional languages to bring down cost and complexity, so that it can be used commercially.



Thank You

Questions