Solution approaches towards verified µ-Kernel

Danny Ziesche

August 25, 2017

RheinMain University of Applied Sciences

Outline

Motivation

Motivation

- kernels should have a high reliability
- in comparison to monolithic kernels small enough to make verification process worthwhile
- trusted codebase
- security concerns

Methods

Dig into the past

- search for (partly) verified µ-Kernel

Dig into the past

- search for (partly) verified µ-Kernel
- research which parts are verified and why

Dig into the past

- search for (partly) verified µ-Kernel
- research which parts are verified and why
- how does the verification process work

Dig into the past

- search for (partly) verified μ-Kernel
- research which parts are verified and why
- how does the verification process work
- compare verifications

Learn about formal methods

- firm understanding about the fundamentals

Learn about formal methods

- firm understanding about the fundamentals
- used methods by the μ-kernel?

Learn about formal methods

- firm understanding about the fundamentals
- used methods by the µ-kernel?
- do we benefit from it?

Definitions

Theorem Prover

- assist in formalising proofs

Theorem Prover

- assist in formalising proofs
- no automated process

Theorem Prover

- assist in formalising proofs
- no automated process
- human guidance and skill needed

Theorem Prover

- assist in formalising proofs
- no automated process
- human guidance and skill needed
- example theorem prover is isabelle with resolution based on higher-order unification

Linear Temporal Logic

- temporal reasoning

Linear Temporal Logic

- temporal reasoning
- derived from FOPL with new temporal operators:

Linear Temporal Logic

- temporal reasoning
- derived from FOPL with new temporal operators:
    - $\Box$ Always

Linear Temporal Logic

- temporal reasoning
- derived from FOPL with new temporal operators:
  - $\Box$ Always
  - $\bigcirc$ Next

Linear Temporal Logic

- temporal reasoning
- derived from FOPL with new temporal operators:
  - $\Box$ Always
  - $\bigcirc$ Next
  - $\Diamond$ Eventually

Model Checkers

- let M be a state-transition graph

Model Checkers

- let M be a state-transition graph
- let f be a formula of temporal logic

Model Checkers

- let M be a state-transition graph
- let f be a formula of temporal logic
- find all states s of M such that $s \models f$

Results

RUBIS

- verified only the IPC

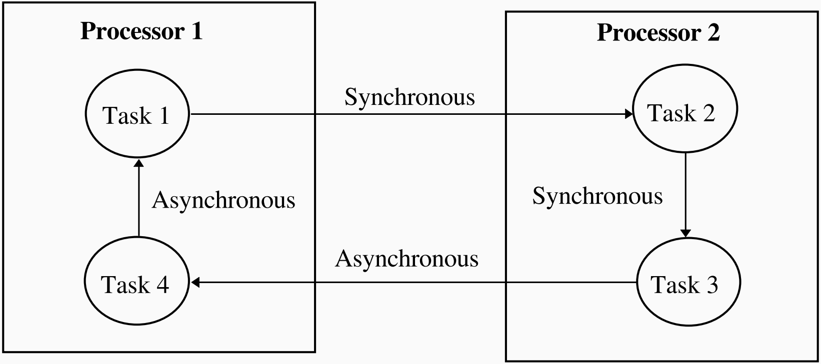

Figure 1: RUBIS Mixed Synchronous and Asynchronous Communication

LTL Property Example

$$\Box(P(0) \rightarrow Q(0) \wedge (P(1) \rightarrow Q(1)) \wedge \ldots \wedge (P(m) \rightarrow Q(m)))$$

- ports need sound state before reusing
- property expressed as LTL
- $P(p) = (\text{Port\_State}[p] = \text{CREATED})$
- $Q(p) = (\text{empty}(\text{Port}[p].\text{messages}))$
- also expressed as promela definition

RUBIS Results

- lots of errors related to return codes
- memory management errors

Fluke

- verified only the IPC

Fluke

- verified only the IPC
- IPC is important and highly concurrent with a complex implementation

Fluke

- verified only the IPC
- IPC is important and highly concurrent with a complex implementation
- makes it worthy target for formal methods

Fluke Formal Methods

- uses spin
- uses subset of C

Fluke Results

- found mutex bugs

Fluke Results

- found mutex bugs
- found race condition

Fluke Results

- found mutex bugs
- found race condition
- scaling problems

Fluke Results

- found mutex bugs
- found race condition
- scaling problems
- maintenance problems

seL4

- interactive machine-assisted and machine-checked proof
- proven over 150 invariants
- discovered about 140 bugs
- revealed 150 problems within the specification
- uses theorem prover isabelle/hol
- tries to offload problematic code to userspace (memory management)
- executable specification in haskell subset
- implementation in a C subset

Refinement Layers

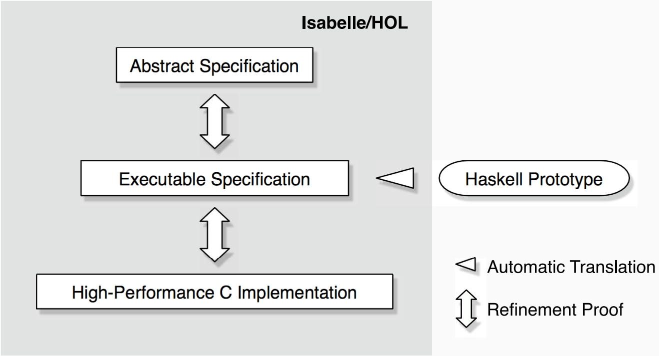

Figure 2: Refinement layers in the verification of seL4

seL4 Results

- claims to have no nullpointer access (the kernel itself)
- functional correctness for the c kernel implementation
- proof maintenance

Conclusions

Conclusions

- IPC is obviously an important component for µ-kernel

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language
- existing code proven with model checker

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language
- existing code proven with model checker
- model checker have a short learning curve

Conclusions

- IPC is obviously an important component for μ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language
- existing code proven with model checker
- model checker have a short learning curve
- non-existing code proven with theorem prover

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language
- existing code proven with model checker
- model checker have a short learning curve
- non-existing code proven with theorem prover
- in my estimation seL4 did the most and best job so far

Conclusions

- IPC is obviously an important component for µ-kernel
- IPC is a high candidate for verification
- agreement on a subset of standard language
- existing code proven with model checker
- model checker have a short learning curve
- non-existing code proven with theorem prover
- in my estimation seL4 did the most and best job so far
- $\Rightarrow$ seems to be a general pattern to µ-kernel verification

Open Questions

Languages

- languages with built-in mechanisms for formal verification
- languages which are designed to make verification easier
- verification of compilers

Famous Quote

”Beware of bugs in the above code; I have only proved it correct, not tried it.“

— Donald E. Knuth

Questions?