

# VAMOS Microkernel: Formal Models and Verification

Jan Dörrenbächer

—

Joint work with: E. Alkassar, M. Daum, M. Hillebrand, Th. In der Rieden, S. Knapp, D. Leinenbach, W. Paul

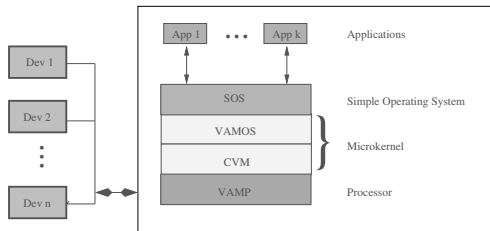
Institute for Computer Architecture,  
Saarland University

August 8, 2006

# Outline

- 1 System Stack
- 2 Formal Models
- 3 Verification
- 4 Conclusion

# System Stack



- microkernel provides interface between hardware and operating system
- provides fundamental functionality to implement basic operating system services
- comprises 2 internal layers:
  - CVM: hardware-dependent, with portions of assembler
  - VAMOS, higher level, pure C0 implementation

# Communicating Virtual Machines (CVM)

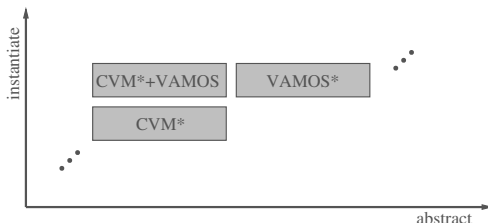
- implements low-level functionality in so-called *CVM primitives*
- provides:
  - virtual processor abstraction
  - allocation/deallocation of memory
  - I/O and interrupt mechanism
  - copying data
- delivers programming environment for simple microkernels

# VAMOS

- completely written in C0
- accesses low-level functionality via CVM primitives
- provides set of kernel calls for:
  - process management
  - memory management
  - device driver support
  - scheduling mechanism
  - privileges and rights management
  - inter-process communication (IPC) with handles and rights
- delivers programming environment for applications

# Formal Models

- formal models for each system layer in Isabelle/HOL
- sequence of models:
  - instantiate model of underlying layer (plug in implementation)
  - abstract functionality of instantiation (specification)
  - show properties of abstraction/specification
- formal models for devices in Isabelle/HOL



# Devices

- input/output devices for non-volatile storage, communication and user-interaction
- devices are integrated as memory-mapped devices, which may generate interrupts
- no usage of direct-memory access (DMA)
- devices are modeled as finite Mealy automata that may interact with processor and external environment (e.g. network)
- device integration leads to concurrency already at VAMP processor layer

# Devices, cont.

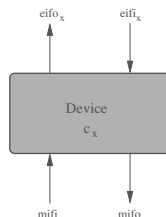
- currently supported devices:
  - UART16550A-compatible serial interface
  - ATA/ATAPI-compatible hard disk
  - FlexRay like interface (automotive project)
- formalized in Isabelle/HOL

# Device Automaton

- device  $x$  is modeled as finite transition system with a configuration  $c_x \in \mathcal{C}_x$  and a transition function

$$\delta_x : (MifiT \times EifiT_x \times \mathcal{C}_x) \rightarrow (\mathcal{C}_x \times MifoT \times EifoT_x)$$

- in each step, the automaton takes input from processor and external environment, updates its state, and may react by producing output



# Device Automaton, cont.

- inputs and outputs to external environment are device-specific
- external input is used for
  - *real* input, like key presses, and
  - device behavior that is not modeled explicitly (e.g. timing)
- same memory interface for each device:

$$MifiT = \{rd : bool, wr : bool, portaddr : nat, din : nat\}$$

$$MifoT = nat$$

- interrupts are signaled by predicate  $is\_irq_x : C_x \rightarrow bool$

# Model: CVM\*

- formalizes computation of a generic abstract operating system kernel interacting with a fixed number of user processes
- in every step of computation either the kernel or one user process can make progress
- introduces notion of separate processes
- uses C0 semantics to model computation of the kernel and virtual machines to model computation of user processes

# Model: CVM\*, cont.

- abstract kernel provides:
  - scheduling, memory management, IPC, interrupt delivery, etc.
- kernel can alter configuration of user process only by means of CVM primitives
- user processes can use interrupts (traps) to call the kernel
- user processes are virtual machines, i.e. VAMP processors with access to (non-shared) virtual memory
- execution of different assembler programs is simulated as if executed on separate processors

# Model Definition and Configuration of CVM\*

- model given through:

$$CVM^* = (\mathcal{C}_{cvm}, \mathcal{C}_{cvm}^0, MifoT, MifiT, \omega_{cvm}, \delta_{cvm})$$

- configuration represented as:

$$\mathcal{C}_{cvm} = \{kernel : \mathcal{C}_{c0}, up : pidT \rightarrow \mathcal{C}_{asm}, cp : pidT, statusreg : regT\}$$

- output function  $\omega_{cvm}$  takes a CVM configuration and returns a memory interface input:

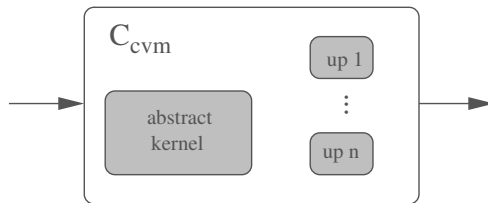
$$\omega_{cvm} : \mathcal{C}_{cvm} \rightarrow MifiT$$

# Model Definition and Configuration of CVM\*, cont.

- $CVM^*$  semantics of local computations are defined by a transition function

$$\delta_{cvm} : C_{cvm} \times intsT \times MifoT \rightarrow C_{cvm}$$

- $\delta_{cvm}$  is composed of transition functions of:
  - the C0-machine
  - the CVM primitives
  - the virtual machines



# CVM\* with Devices

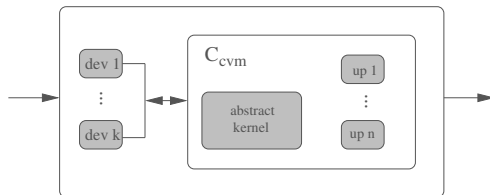
- connects  $CVM^*$  to concrete devices
- $\omega_{cvm}$  delivers output to devices and memory interface output and device interrupts deliver input for  $CVM^*$
- configuration is extended by device configurations

$$\mathcal{C}_{cvm+dev} = \{c_{cvm} : \mathcal{C}_{cvm}, devs : devnumT \rightarrow \mathcal{C}_x\}$$

- global transition function for single device or CVM step

$$\delta_{cvm+dev} : (devnumT \cup \{\epsilon\}) \times EifiT \times \mathcal{C}_{cvm+dev} \rightarrow (\mathcal{C}_{cvm+dev} \times EifoT)$$

## CVM\* with Devices, cont.



- (concurrent) computation parameterized over sequence

$$seq : nat \rightarrow (devnum T \cup \{\epsilon\})$$

- computation in step  $t$ :
  - $seq(t) \neq \epsilon$ : external device  $seq(t)$  performs a step
  - $seq(t) = \epsilon$ : interaction with a device or normal CVM step

# Correctness of CVM implementation

- to show:
  - CVM-implementation on hardware simulates *CVM\**
    - the implementation of the CVM primitives is correct
    - page fault handler with processor simulates virtual machines
- imports:
  - C0 compiler correctness with inline assembler

# Model: VAMOS\*

- models interaction of separate user processes with the VAMOS kernel
- obtained by instantiation of the abstract kernel in *CVM\** with one concrete C0 machine:

$$CVM^* + VAMOS = CVM^*[kernel := VAMOS-impl]$$

- in higher layers, the concrete C0 code is not of interest
- only semantical effects are important
- abstract *CVM\* + VAMOS* to get *VAMOS\**

# Model Definition and Configuration of VAMOS\*

- model given through:

$$VAMOS^* = (\mathcal{C}_{vamos}, \mathcal{C}_{vamos}^0, MifoT, MifiT, \omega_{vamos}, \delta_{vamos})$$

- configuration is represented as:

$$\mathcal{C}_{vamos} = \{kds : kdsT, up : pidT \rightarrow \mathcal{C}_{asm}\}$$

- kernel datastructures:

$$kdsT = \{schedds : scheddsT, priodb : pidT \rightarrow prioT, \\ rightds : rightdsT, sndstatdb : pidT \rightarrow bool, \\ devds : devdsT\}$$

- output function  $\omega_{vamos}$  takes a VAMOS configuration and returns a memory interface input:

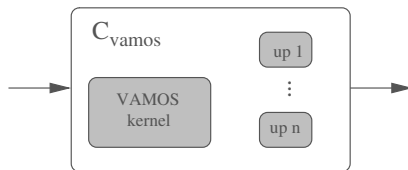
$$\omega_{vamos} : \mathcal{C}_{vamos} \rightarrow MifiT$$

# Model Definition and Configuration of VAMOS\*, cont.

- *VAMOS\** semantics for local computations are defined by a transition function

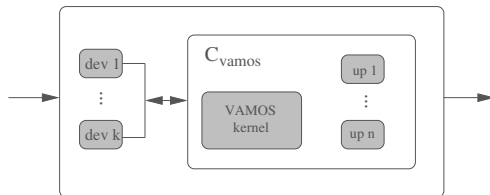
$$\delta_{vamos} : \mathcal{C}_{vamos} \times intsT \times MifoT \rightarrow \mathcal{C}_{vamos}$$

- $\delta_{vamos}$  is composed of transition functions of:
  - the virtual machines
  - the kernel calls
  - the scheduler
  - the external interrupt delivery



# VAMOS\* with Devices

- devices in *VAMOS\** and *CVM\** are modeled in the same way
- $\omega_{vamos}$  delivers input for the devices
- memory interface output and interrupts of devices provide input for *VAMOS\**
- define  $\delta_{vamos+dev}$  similarly to  $\delta_{vamos+dev}$



# Correctness of VAMOS implementation

- to show:
  - $CVM^* + VAMOS$  simulates  $VAMOS^*$
  - invariants about the kernel
- imports  $CVM^*$

# Simulation

To simplify matters we only consider local computations, i.e. no device steps or interaction with devices.

- $CVM^* + VAMOS$  operates on configuration  $C_{cvm}$
- $VAMOS^*$  operates on configuration  $C_{vamos}$
- define simulation predicate  $SIM : C_{vamos} \times C_{cvm} \rightarrow bool$
- $SIM(c, s)$  holds if datastructures *match*

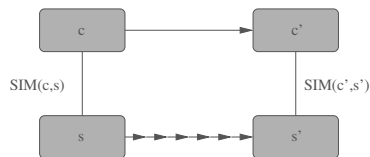
# Simulation Theorem

Simulation Theorem:

$\forall c \in \mathcal{C}_{vamos}, s \in \mathcal{C}_{cvm} :$

$SIM(c, s) \implies SIM(c', s')$

with  $c' = \delta_{vamos}(c)$  and  $s' = \delta_{cvm}^*(s)$



- simulation between two models is based on C0 code verification
- done in the verification environment for Isabelle/HOL (N. Schirmer) providing:
  - language model for sequential imperative programs, and
  - Hoare logics in typical form:  $\{P\} p \{Q\}$
- configuration of  $CVM^* + VAMOS$  is not represented as  $\mathcal{C}_{cvm}$
- equivalence proofs between semantics required

# Code Verification – General Approach

Let  $c \in \mathcal{C}_{vamos}$  and  $s \in \mathcal{C}_{cvm}$  be configurations and  $f_{abs}()$  the abstract counterpart of the implemented function  $f_{conc}()$ .

- pre-state:
  - $SIM(c, s)$  holds
  - conditions which are given implicitly by implementation
- execute the implemented function  $f_{conc}()$  in pre-state
- post-state:
  - we have a new configuration  $s'$
  - $c' = f_{abs}(c, \dots)$
  - show:  $SIM(c', s')$  holds

We show that the effect of the execution of  $f_{conc}()$  simulates  $f_{abs}()$ .

# Property Translation

- state properties/invariants of system in terms of formulas like  $c \models form_{abs}$
- function  $concf_{SIM}(form_{abs}) = form_{conc}$  translates formula into concrete system

Simulation theorem implies

$$\forall c \in \mathcal{C}_{vamos}, s \in \mathcal{C}_{cvm} : \\ (SIM(c, s) \wedge c \models form_{abs}) \implies s \models conf_{SIM}(form_{abs})$$

Proof is straightforward and done by induction on the property language.

# Conclusion

- we defined device models
- we introduced the computational model *CVM\** on top of the hardware, which encapsulates the hardware-dependent parts of a microkernel
- based on *CVM\** we use the model *VAMOS\** to describe the whole semantics of the VAMOS microkernel
- all models are formalized in Isabelle/HOL
- integration of the devices into formal models (true concurrency) is almost finished
- formal verification is work in progress

Thank You!