Strategies for Memory Accounting of Cooperating Pi-Calculus Processes

Matej Košík and Jiří Šafařík
kosik@fiit.stuba.sk

Faculty of Informatics and Information Technology
Slovak University of Technology in Bratislava

November 15, 2008

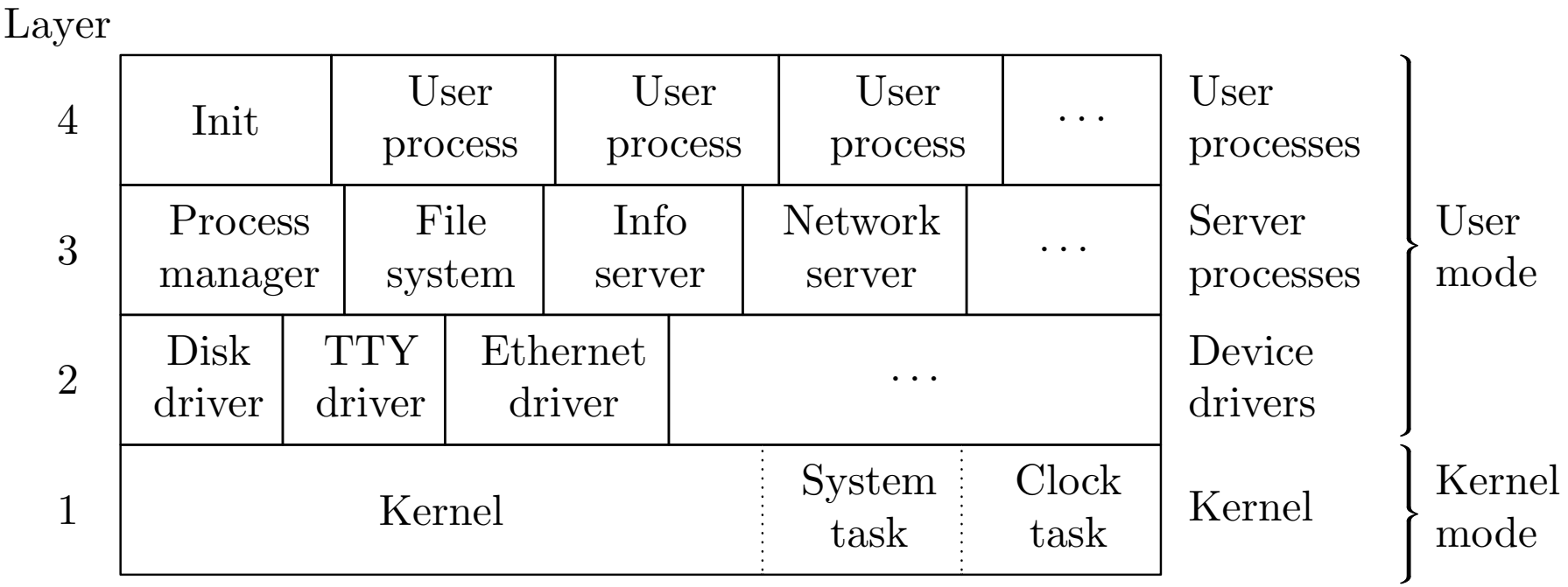
Contents

Topics of interest

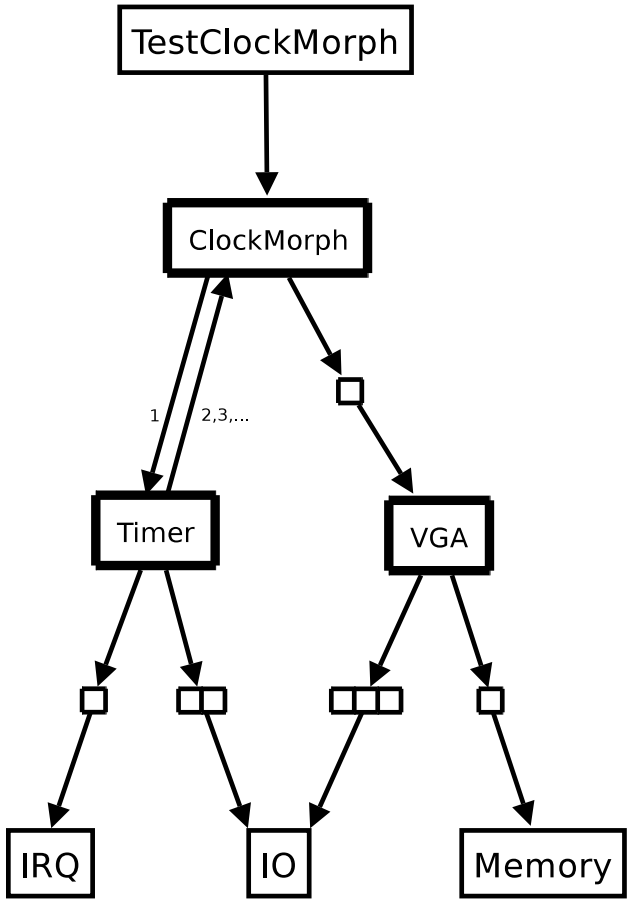
Randy Pausch's Last Lecture:
"Fundamentals, fundamentals, fundamentals. You've got to get fundamentals down because otherwise the fancy stuff isn't going to work."

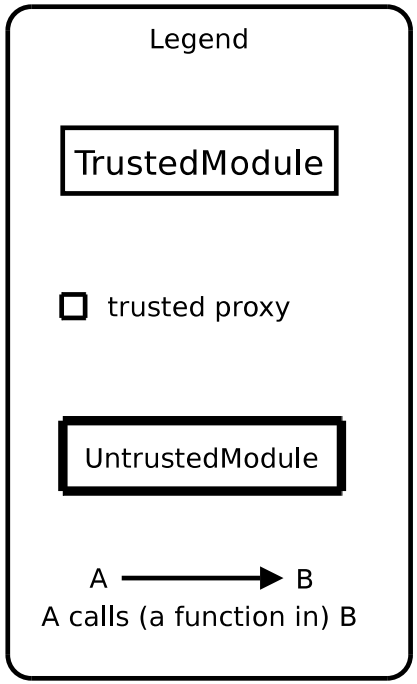
- Properties:
  - robustness
  - how not to trade security for functionality (and vice versa)
- Techniques:
  - microkernel architecture [4]
  - object-capability programming languages [2, 3]
- Applications:
  - kernel-space (mainly)
  - user-space (also)

Minix—operating system designed as a microkernel



Backwater—kernel written in an object-capability language





Two approaches compared

	μ-kernel approach	object-capability approach
communication primitives	some	excellent
TCB size	**X** LOC	**X**/**10** LOC
memory management	solid	missing
CPU management	solid	missing

Object-capability security paradigm

the only way how a subject can get a capability on some object is by:

- initial conditions
- introduction
- parenthood
- endowment

access control design patterns:

- Caretaker (David D. Redell)
- Powerbox !!! (Mark Miller, Marc Stiegler, . . .)
- Membrane (Mark Miller)
- Horton (Mark Miller)

Security policies here are expressed as programs (algorithms or processes), not as as a tuple of values we can set via some configuration options.

Object-capability security paradigm

the only way how a subject can get a capability on some object is by:

- initial conditions
- introduction
- parenthood
- endowment

access control design patterns:

- Caretaker (David D. Redell)
- Powerbox !!! (Mark Miller, Marc Stiegler, . . .)
- Membrane (Mark Miller)
- Horton (Mark Miller)

Security policies here are expressed as programs (algorithms or processes), not as as a tuple of values we can set via some configuration options.

Contents

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

cancer![]
```

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

( cancer![] | cancer![] )
```

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

( cancer![] | cancer![] | cancer![] | cancer![] )
```

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

( cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
)
```

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

( cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
)
```

Cancer

```
def cancer [] =
    ( cancer![] | cancer![] )

( cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
| cancer![] | cancer![] | cancer![] | cancer![]
)
```

Cancer

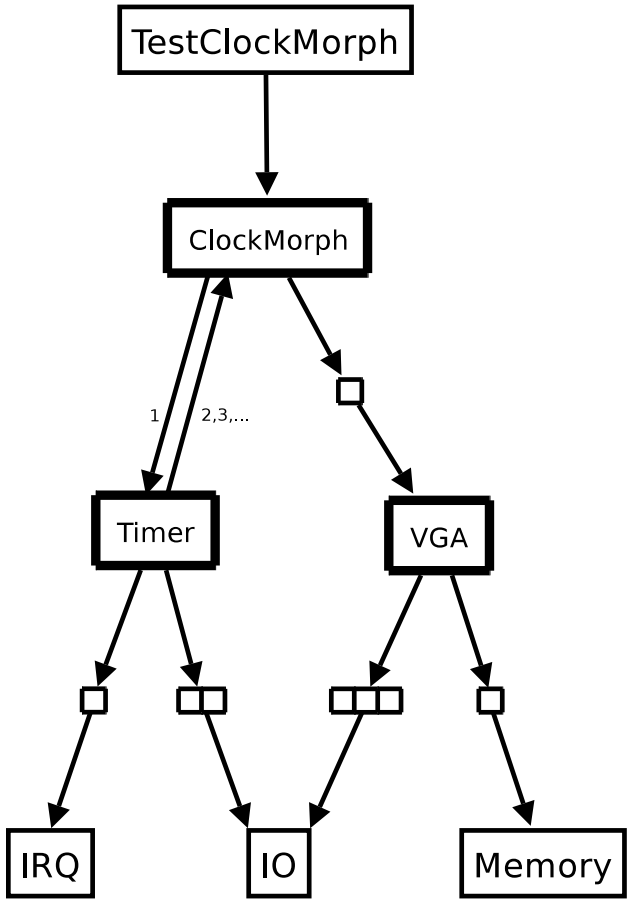
. . . . . .

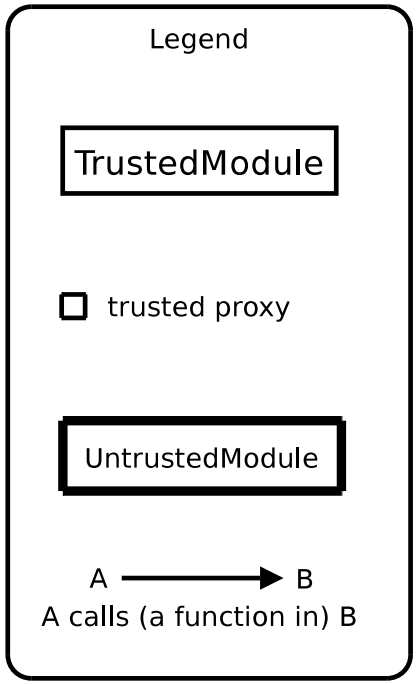
Cancer

STOP

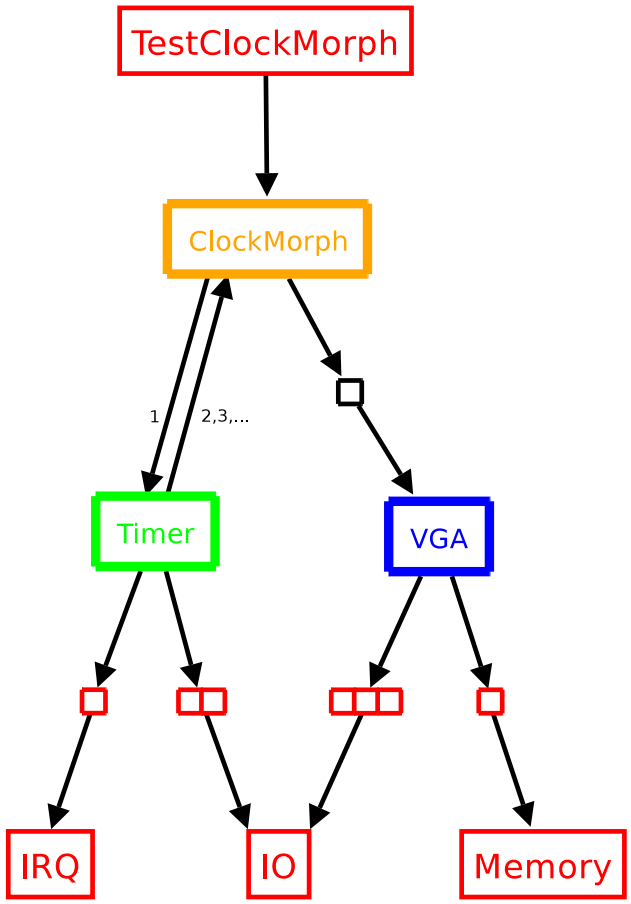
Contents

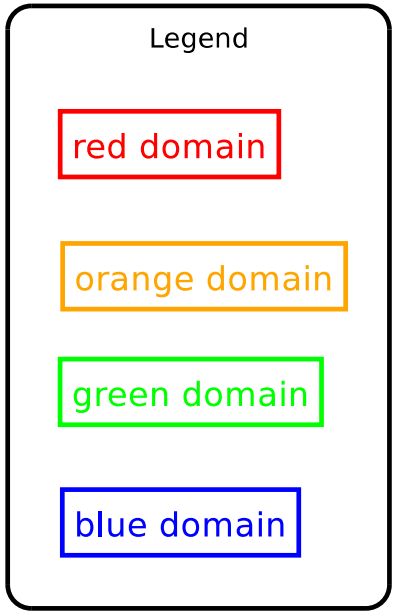
Instead of:


TestClockMorph
ClockMorph
1
2,3,...
Timer
VGA
IRQ
IO
Memory


Legend
TrustedModule
trusted proxy
UntrustedModule
A
B
A calls (a function in) B

we would like to have something like:





The plan

- enhance original Pict programming language
- then I decided to define sufficiently interesting subset of the Pict programming language and enhance this subset
- the chosen subset (I call it Sahara [1]) is based on the π-calculus with i/o-types
- and then add proper new constructs to it

How can that be achieved [1]

- new syntactic constructs
  (for domains)
- update the type system; add new type operators
  (for specifying ownership policy)
- update the operational semantics of the language
  (it should capture changing colors of existing objects and determining color of new objects)

References

Matej Košík.

Sahara programming language web page, 2007.
http://altair.sk/mediawiki/index.php/Sahara, Accessed 23.10.2008.

Mark Samuel Miller.

Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.
PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

Marc Stiegler.

Emily: A high performance language for enabling secure cooperation.
In *C5*, pages 163–169. IEEE Computer Society, 2007.

Andrew S. Tanenbaum and Albert S. Woodhull.

Operating Systems: Design and Implementation.
Pearson Prentice Hall, 2006.

Thank you.