Technology Transfer: Proving a Separation Kernel

Richard M. Cohen

with

William R. Bevier
J Strother Moore

Computational Logic, Inc.
1717 W. 6th Street, Suite 290
Austin, Texas 78703

(512) 322-9951

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

A Little Context...

This is one project toward technology transfer to our sponsors.

There are technology transfer components to several CLI efforts.

This project is part of the FMTI contract.

Recipients at ISSO and NRL.

Many FMTI contributors within CLI.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Their goals

- Learn how to build & analyze mathematical models
- Explore the value of mathematical models
- Assess how to receive & apply the technology

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

A Task Oriented Effort

We chose to center the effort around a example of interest:

a small separation kernel

This provided an example application to focus our efforts.

It was small enough to be tractable.

It did not compound the difficulties of learning the technology with applying it to a leading-edge research/development project.

Separation Kernel Effort

Motivation: To satisfy interest about assembly-code proofs and separation kernels

Goals: Produce a useful result

Portions of result should be reusable

- generic portions of separation kernel high-level design
- generic portions of separation kernel low-level design
- abstract processor specification

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

A Simple Communications Terminal

Envision a small, portable, communications terminal...

The terminal has...

- a keyboard
- a display
- 2 serial I/O ports

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

User View

The terminal supports a few user commands...

- send a message
- receive a message
- display a message
- edit a message

Additional constraints...

- Each command runs to completion
- No preemptive scheduling
- No multitasking
- No overlapped I/O

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Supporting Hardware

- Based on a fairly vanilla processor model

  Process has generic architectural features.

  Programs could map fairly easily to standard processors.

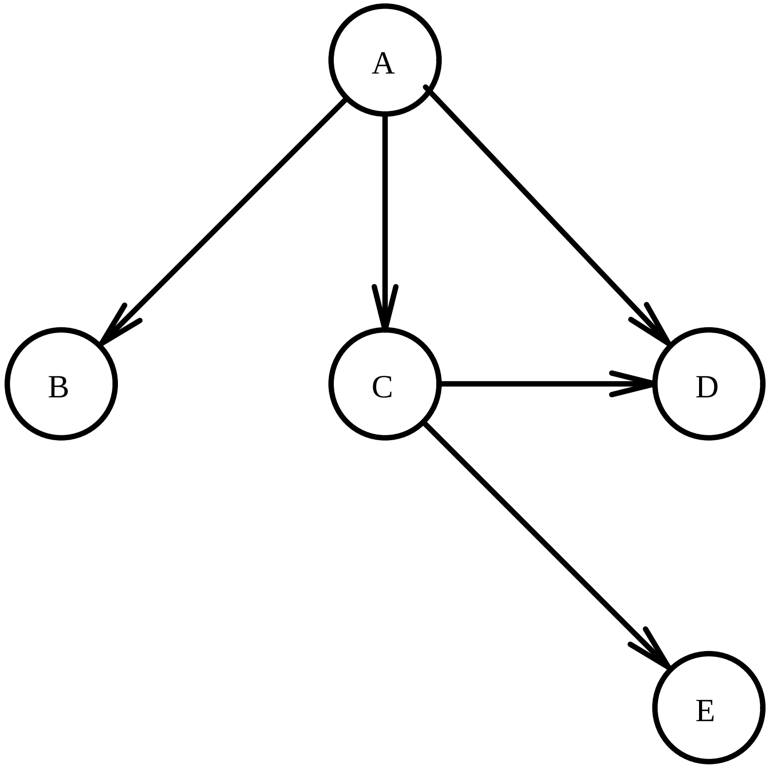
- User/Supervisor mode
- Very simple custom MMU to enforce memory protection

  A set of base/bound registers.

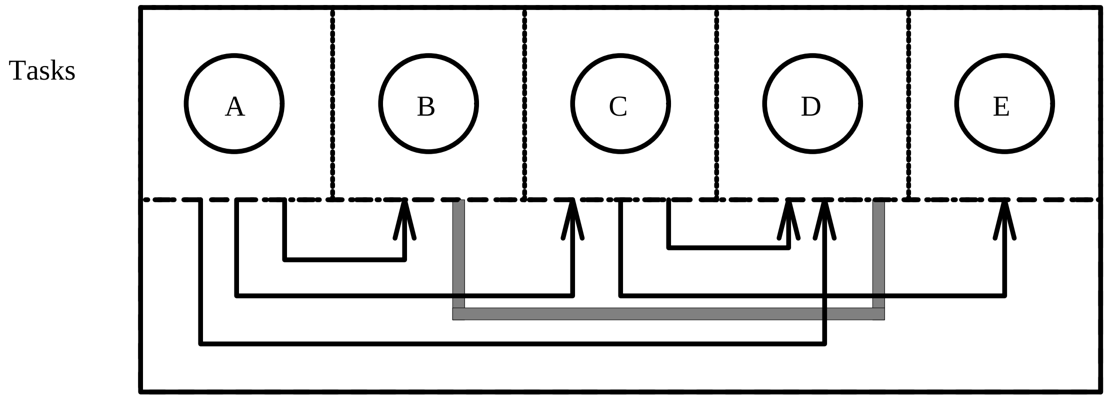
- I/O interrupts

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

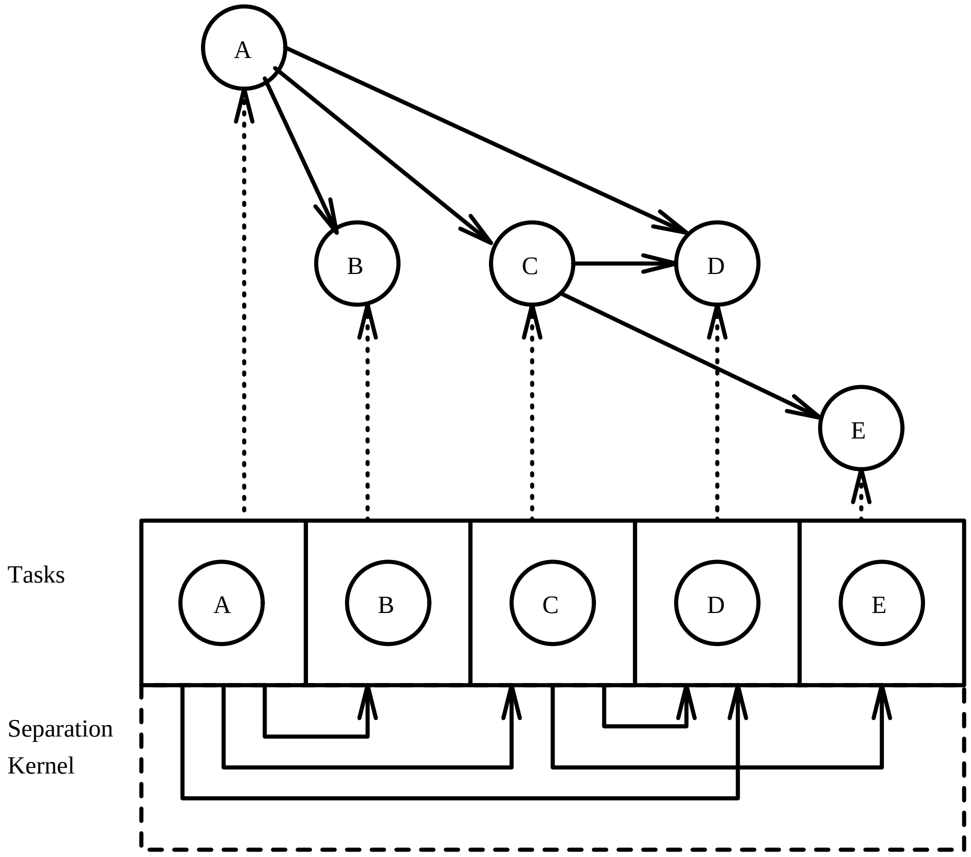
A Network of Processors



Multiple Programs on Single Processor



Multiple Programs + Separation Kernel



Kernel Requirements

- Must enforce process isolation
- Must permit controlled resource sharing
- Basic resources are *memory segments*
- Both executable code and data are represented in segments.
- Segments may have access restrictions
  (e.g., executable, read-only, read-write)
- Variable-length segments may be shared between tasks *as needed*
- At a given time a given task may or may not be permitted access to a particular segment.
- Tasks must be able to share subroutine libraries

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Virtual Call

Analogous to subroutine call *but can cross protection boundaries*.

Strict call/return control flow.

Caller can pass arguments...

  in registers

  in a window into caller's memory

Nothing else in a task's environment can change across a VCALL

CLI COMPUTATIONAL LOGIC INCORPORATED

The Plan

Phase I **Learning about the logic and the problem**

Class on NQTHM logic and proofs.
Lab on separation kernels.
Proof of small assembly-language programs
on J Moore’s "small machine"

Phase II **Exploring Core Problems**

A dozen "challenge problems" relating to
assembly language proofs, and
separation kernels

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

The Plan...

Phase III **Attacking a Larger Problem**

Design of "middle-sized machine"

Design & Specify the separation kernel

- Specify kernel calls as state transitions
- Develop a "legal state" predicate

Prove properties of the specification

Implement the kernel on middle machine.

Prove properties of the implementation.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

The Core-Problem Methodology

This is akin to a "rapid prototype"

Investigate a *tiny example* that exhibits *core aspects* of the real problem.

Supports the early investigation of

- how to formulate the specification
- how to approach the proof.

The small example allows trying out different approaches.

These experiments would be too costly on the full-sized problem.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Piton: From Tiny to Real

Piton: A verified assembler/loader for the FM9001

Tiny version was a 4-instruction prototype.

Exploring specification and proof of the toy took 7 man-months.

Full Piton had 65 instructions and took 9 man-months.

Piton Reports: Piton: A Verified Assembly Level Language, CLI TR#22 (Sept. 1988) [Full Report]

A Mechanically Verified Language Implementation, CLI TR#30 (Sept. 1988) [Summary Report]

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Current Status

Team is currently at end of Phase II.

- Completed the "toy problems"
- Middle-machine processor model
- High-level and low-level specification for kernel

Phase I and II results are considered successful.

Team participants have spent ~ 25% to 50% of their time on this.

Next steps: Implement the kernel on middle-machine
Begin proofs about the abstract kernel

Completion expected this summer.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED