Mar 2, 2005 -- Lecture 18

22C:169
Computer Security

Douglas W. Jones

Department of Computer Science

Examples

The Cambridge CAP System

Wilkes and Needham (project start, 1970; book, 1979)

Microprogrammed CPU

- 16 data registers, hold only data
- 16 capability registers, hold only caps
- Overly expensive multiply indirect caps

Address space = C-list

- *Process hierarchy*
- *Parent has capability for C-list of child*
  - No priviliged mode needed

CAP access rights

For data segments

R - read (into data register)

W- write (from data register)

E - execute (into instruction register)

For C-lists

RC - read (into capability register)

WC - write (from capability register)

Cap with W and RC dangerous!

CAP refine operation

`C' = refine(C, rights, base, size);`

Rights of `C'` reduced from rights of `C`

`C'.rights` *is a subset of* `C.rights`

Segment `C'` is part of segment `C`

can pass cap for part of a segment

Adds serious complexity to system

Cap enter operation

Applicable to enter capabilities

Distinct type of capability (complexity!)

Enter

Pushes old domain on C-stack

Enter creates new domain from:

Object instance C-list

Actual parameter C-list

Code context C-list

Very Complex Semantics

Added complexity in CAP

Capabilities are revokable

A consequence of multiple indirection

Cap allows seal and unseal operations

Sealed objects are a type of capability

Cap File System support for persistancy

inform versus outform capabilities

CAP evaluation

CAP was successful

- *As a university-built one-off system*
- *All security goals of system were met*

CAP was complex

- *Capability unit more complex than CPU*
- *Microprogramming is no longer popular*

CAP relied on custom hardware, microcode

- *Can it be done on conventional CPU?*
- *Can it be done in RISC philosophy?*

The Mach kernel

Work began, 1985 Carnegie Mellon, moved to OSF 1994.

Uses conventional MMU

Easily ported to many modern machines

Basis of: OSF/1, NeXTStep, IBM's OS/2, MacOS X, others

Capability list per multithreaded process

Capabilities used for message passing

Capabilities refer to mailboxes

Client-server model of system construction

Send messages to servers

Await replies from servers

Await exception messages

Problems with Mach

Message overhead

FreeBSD / MacOS X found this too high

Partial abandonment of Microkernel

Integration of programming models

Microkernel has object model

C++, Java, etc have object models

Difficult to make models mesh

The uniform Reference Problem

Reference to an object should be uniform!

local: `result=obj.meth(parm);`

protected:

```
send(obj_cap,rep_cap,
     meth,parm);
await(rep_cap,result);
```

Seriously degrades software development

Can use local agents to hide ugliness

Use of local agents adds overhead