Mar 4, 2005 -- Lecture 19


# 22C:169
# Computer Security

Douglas W. Jones

Department of Computer Science

## Examples: Mach

# 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*

## Original Mach kernel is very small

Minimal MMU support

- *Map-unmap pages in address space*
- *Page fault sends message to server*

Almost symmetric message passing

- *Send message to mailbox via cap*
- *Receive message from mailbox via cap*
- *Messages may contain caps and data*
- *Only one process at a time may receive*

## Design Goal (and source of trouble)

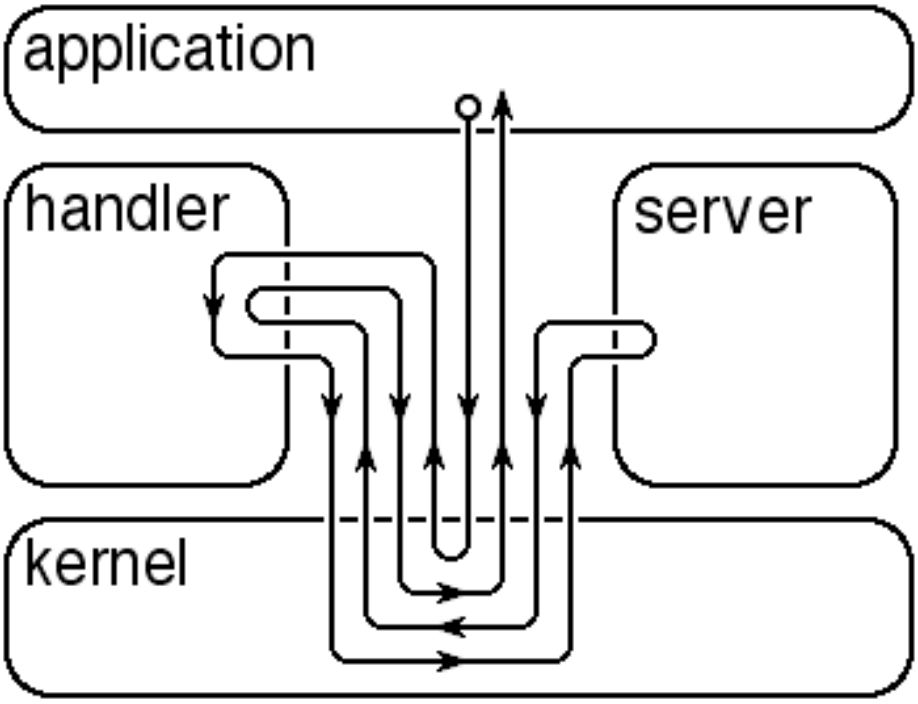
Object-code compatibility

- *Mach kernel sits under UNIX layer*
- *Users unaware of presence of Mach*

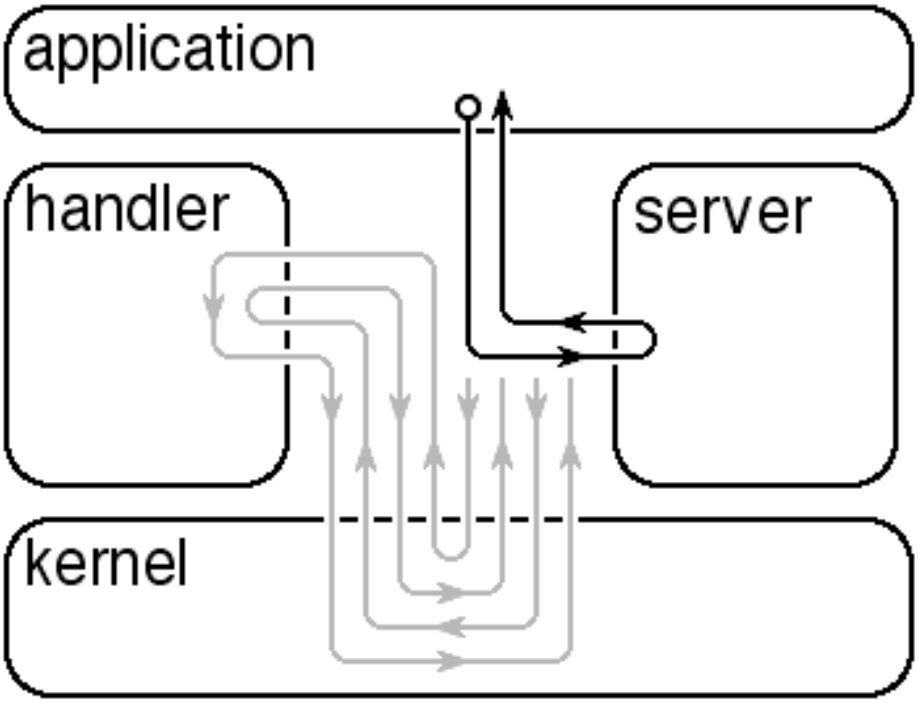
System Call in compatable object code

- *Typically done by trap mechanism*
- *Trap handler sends exception notice*
- *Exception handler implements API*
- *Exception handler communicates*

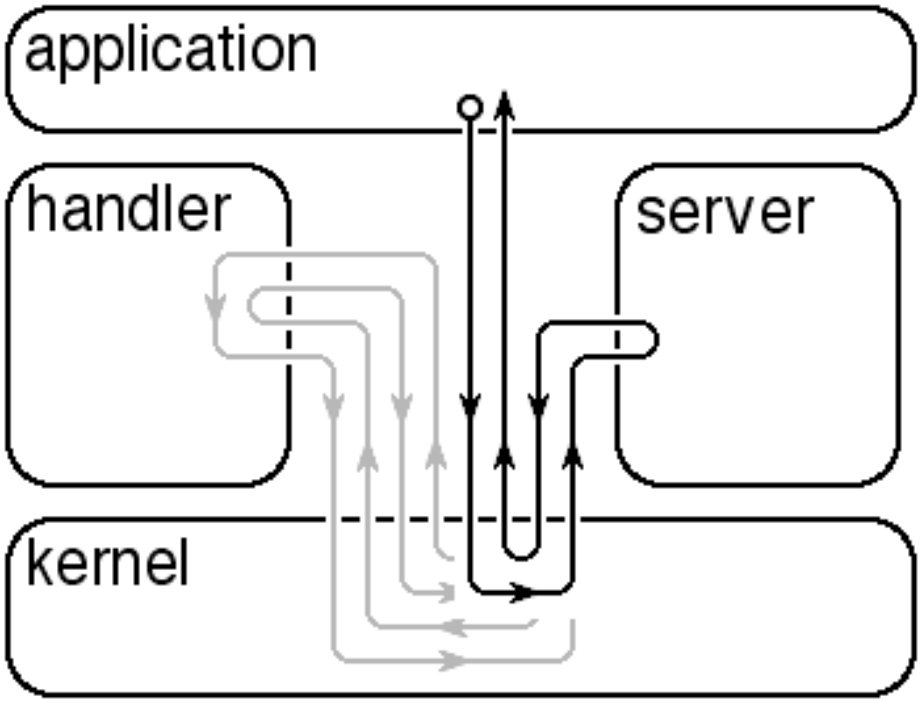
**One system call = 8 context switches!**



# What the User Wanted



# What the user could have had



## Assessment of Cost

Part of this cost is unavoidable

*cost of compartmentalized OS*

Half this cost could be avoided by

*abandoning object code compatibility*
*user code makes direct kernel calls*

Compatible cost reduction strategy

*move functionality into kernel*
*merge handler and server functions*

Potential loss of security

## 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*

## The uniform reference problem is old

Unix "Kernel" calls suffer

*modern languages want*

```
file f = open( ... );
char c = f.get();
```

*Unix gives us*

```
int f = open( ... );
char c;
read( f, c, 1 );
```

We hide this behind layers of middleware

The cost of these layers is real

## Hierarchic rules in a Capabilty System

Idea: objects labeled with classification

*Add support for this to kernel*

*Orthogonal to capabilities*

Worse Idea: security kernel

*Distinct from system kernel*

*Auxiliary code to check all object access*

*Can work at open time*

All IPC paths checked when opened