

cse

Microkernels and Client-Server Architectures

I'm not interested in making devices look like user-level. They aren't, they shouldn't, and microkernels are just stupid.
Linus Torvalds

THE UNIVERSITY OF NEW SOUTH WALES 1

cse

Motivation

- Early operating systems had very little structure.
- A strictly layered approach was promoted by [Dijkstra, 1968].
- Later OS (more or less) followed that approach (e.g., Unix).

PROBLEMS WITH LAYERED APPROACH

- Widening range of services and applications
 ⇒ OS bigger, more complex, slower, more error prone.
- Need to support same OS on different hardware.
- Like to support various OS environments.
- Distribution
 ⇒ impossible to provide all services from same (local) kernel.

THE UNIVERSITY OF NEW SOUTH WALES 2

cse

Idea: Break Up the OS

Applications
 User-level servers
 Microkernel
 Hardware

OS

non-privileged
 privileged

THE UNIVERSITY OF NEW SOUTH WALES 3

cse

Monolithic versus Client-Server OS Structure

Application
 VFS
 IPC, file system
 Scheduler, virtual memory
 Device drivers, Dispatcher, ...
 Hardware

user mode
 kernel mode

Application
 UNIX Server
 NT Server
 File Server
 IPC, virtual memory
 Hardware

THE UNIVERSITY OF NEW SOUTH WALES 4

cse

KERNEL:

- Contains code which *must* run in supervisor mode;
- Isolates hardware dependence from higher levels;
- Is small and fast
 ⇒ extensible system;

Kernel provides mechanisms.

USER-LEVEL SERVERS:

- Are hardware independent/portable,
- Provide "OS environment"/"OS personality" (maybe several),
- May be invoked:
 - from **application** (via message-passing IPC)
 - from **kernel** (upcalls);

Servers implement policies [Brinch Hansen, 1970].

THE UNIVERSITY OF NEW SOUTH WALES 5

cse

Down Call vs. Upcall

downcall (syscall)
 upcall

user code
 kernel

un-privileged
 privileged

- unprivileged code enters kernel mode
- implemented via trap
- privileged code enters user mode
- implemented via IPC or callback

THE UNIVERSITY OF NEW SOUTH WALES 6

cse

Early example: Hydra

- Separation of mechanism from policy
 - e.g. protection vs. security
- No hierarchical layering of kernel.
 - Viewed layering as an unnecessary restriction
 - Protection, even within OS.
- Uses (segregated) *capabilities*.
- Objects, encapsulation, units of protection.
 - Objects have a *type* associated with them.
 - Includes a *procedure* type, which can be associated with object
 - Created indirectly via the representative of that type
 - Can create new types by creating a new representative
- Unique object *name*, no ownership.
- Object persistence based on reference counting [Wulf *et al.*, 1974].

THE UNIVERSITY OF NEW SOUTH WALES 7

cse

Hydra...

- can be considered the first *object-oriented* OS;
- has been called the first *microkernel* OS;
- has had enormous influence on later operating systems research;
- was never widely used even at CMU because of
 - poor performance,
 - lack of a complete environment.

THE UNIVERSITY OF NEW SOUTH WALES 8

cse

Popular Example: Mach

- Developed at CMU by Rashid and others [Rashid *et al.*, 1988] from 1984
- successor of Accent [Fitzgerald and Rashid, 1986] and RIG [Rashid, 1988].

GOALS:

- *Tailorability*: support different OS interfaces.
- *Portability*: almost all code H/W independent.
- *Real-time capability*.
- *Multiprocessor and distribution* support.
- *Security*.

Coined term *microkernel*.

THE UNIVERSITY OF NEW SOUTH WALES 9

cse

Basic Features Of MACH Microkernel

- Task and thread management;
- interprocess communication (asynchronous message-passing);
- memory object management;
- system call redirection;
- device support;
- multiprocessor support.

THE UNIVERSITY OF NEW SOUTH WALES 10

cse

MACH Basic Abstractions

The diagram illustrates the basic abstractions of the MACH microkernel. A task is shown as a container with a text region (containing threads and a program counter) and a data region. A message and port are connected to the task. A port set is also shown. A memory object is connected to the task and is linked to secondary storage.

THE UNIVERSITY OF NEW SOUTH WALES 11

cse

MACH Tasks And Threads

- Task consists of one or more threads.
- Task provides *address space* and other environment.
- Thread is active entity (basic unit of CPU utilisation) .
- Threads have own stacks, are kernel scheduled.
- Threads may run in parallel on multiprocessor.
- "Privileged user-state program" may be used to control scheduling.
- Task created from "blueprint" with empty or inherited address space.
- Activated by creating a thread in it.

THE UNIVERSITY OF NEW SOUTH WALES 12

cse

MACH IPC: Ports

- Addressing based on ports:
 - port is a mailbox, allocated/destroyed via a system call;
 - has a fixed-size message queue associated with it;
 - is protected by (segregated) capabilities;
 - has exactly *one* receiver, but possibly *many* senders;
 - can have "send-once" capability to a port.
- Can pass the *receive capability* for a port to another process
 - give up read access to the port.
- Kernel detects ports without senders or receiver.
- Processes may have many ports (UNIX server has 2000!)
- Ports can be grouped into *port sets*.
 - Allows listening to many ports (like select())
- Send blocks if queue is full
 - except with send-once cap (used for server replies)

THE UNIVERSITY OF NEW SOUTH WALES 13

cse

MACH IPC: Messages

- Segregated capabilities:**
 - threads refer to them via local indices.
 - kernel marshalls capabilities in messages.
 - message format must identify caps
- Message contents:**
 - Send capability to destination port (mandatory)
 - used by kernel to validate operation;
 - optional send capability to reply port
 - for use by receiver to send reply
 - possibly other capabilities;
 - "in-line" (by-value) data;
 - "out-of-line" (by reference) data, using copy-on-write,
 - may contain whole address spaces;

THE UNIVERSITY OF NEW SOUTH WALES 14

cse

MACH Ports and Messages

THE UNIVERSITY OF NEW SOUTH WALES 15

cse

MACH Message Transfer

THE UNIVERSITY OF NEW SOUTH WALES 16

cse

MACH Virtual Memory Management

- Address space constructed from *memory regions*
 - initially empty
 - populated by:
 - explicit allocation
 - explicitly mapping a *memory object*;
 - inheriting from "blueprint" (as in Linux clone()),
 - inheritance: *not*, *shared* or *copied*;
 - allocated automatically by kernel during IPC
 - when passing by-reference parameters;
- ⇒ sparse virtual memory use (unlike UNIX).
- 3 page states:
 - unallocated,
 - allocated & unreferenced,
 - allocated & initialised

THE UNIVERSITY OF NEW SOUTH WALES 17

cse

Copy-on-write In MACH

- When data is copied ("blueprint" or passed by-reference):
 - source and destination share single copy,
 - both virtual pages are mapped to the same frame.
- Marked as read-only.
- When one copy is modified, a fault occurs.
- Handling by kernel involves making a physical copy is made,
 - VM mapping is changed to refer to the new copy.
- Advantage:
 - efficient way of sharing/passing large amounts of data.
- Drawbacks:
 - expensive for small amounts of data (page table manipulations)
 - data must be properly aligned

THE UNIVERSITY OF NEW SOUTH WALES 18

cse

MACH Address Maps

- Address spaces represented as *address maps*:
- Any part of AS can be mapped to (part of) a memory object
- Compact representation of *sparse* address spaces
 - Compare to multi-level page tables?

THE UNIVERSITY OF NEW SOUTH WALES 19

cse

THE UNIVERSITY OF NEW SOUTH WALES 20

cse

Memory Objects

- Kernel doesn't support file system
- Memory objects are an abstraction of secondary storage:
 - can be mapped into virtual memory
 - are cached by the kernel in physical memory
 - pager invoked if uncached page is touched
 - used by file system server to provide data
- Support data sharing
 - by mapping objects into several address spaces
- Memory is only cache for memory objects

THE UNIVERSITY OF NEW SOUTH WALES 21

cse

User-Level Page Fault Handlers

- All actual I/O performed by *pager*; can be
 - default pager (provided by kernel), or
 - *external pager*, running at user-level.
- Intrinsic page fault cost: 2 IPCs

THE UNIVERSITY OF NEW SOUTH WALES 22

cse

Handling Page Faults

1. Check protection & locate memory object
 - uses address map
2. Check cache, invoke pager if cache miss
 - uses a hashed page table
3. Check copy-on-write
 - perform physical copy if write fault
4. Enter new mapping into H/W page tables.

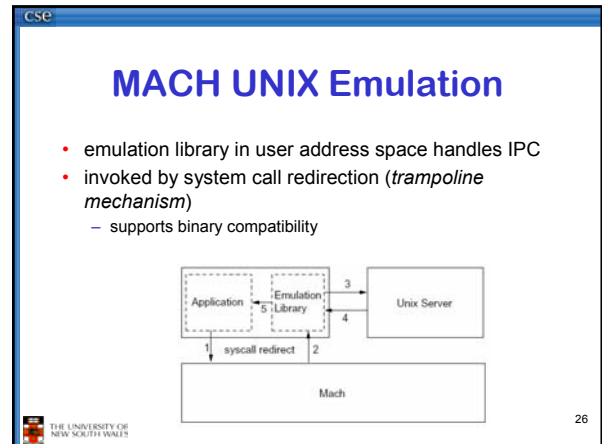
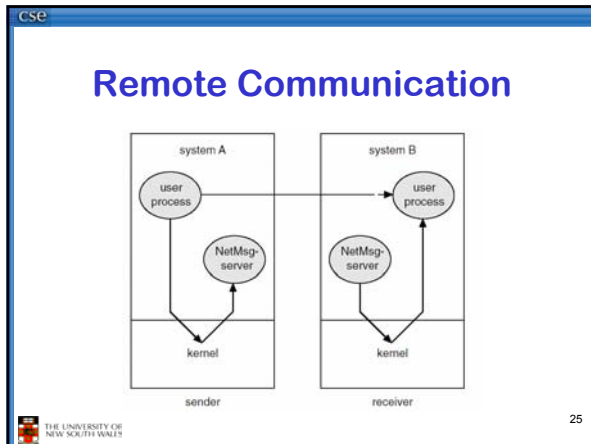
THE UNIVERSITY OF NEW SOUTH WALES 23

cse

Remote Communication

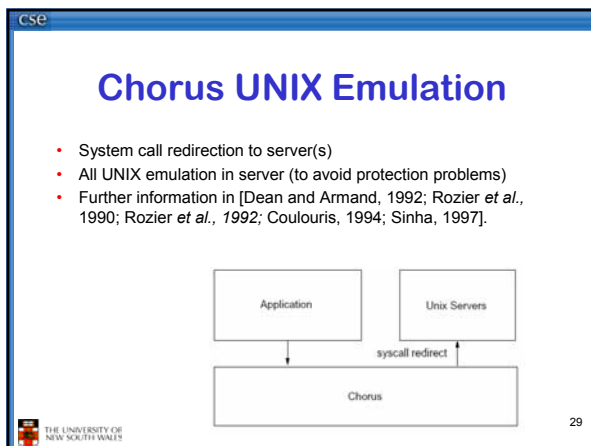
- Client A sends message to server B on remote node.
 1. A sends message to local *proxy port* for B's receive port
 2. User-level *network message server* receives from proxy port
 3. NMS converts proxy port into (global) *network port*.
 4. NMS sends message to NMS on B's node
 - may need conversion (byte order...)
 5. Remote NMS converts network port into local port (B's).
 6. Remote NMS sends message to that port.

THE UNIVERSITY OF NEW SOUTH WALES 24



- ## MACH = Microkernel?
- Most OS services implemented at user level
 - using memory objects and external pagers
 - Provides mechanisms, not policies.
 - Mostly hardware independent.
 - 140 system calls.
 - Size: 200k instructions.
 - Performance???
 - tendency to move features into kernel
 - Served as basis for OSF/1, MacOS X...
 - Further information on Mach: [Young *et al.*, 1987; Coulouris, 1994; Sinha, 1997]

- ## Chorus
- Developed at INRIA, France, from 1980 on.
 - Commercialised by *Chorus Systèmes* in 1988.
 - Basic ideas similar to Mach
 - Servers can be:
 - user-level,
 - dynamically loaded into kernel (to save system call costs).
 - Support for group communication (multicast, any of a group).
 - Like Mach, kernel threads, port groups.
 - Uses password capabilities
 - but servers use ACL based protection.
 - Uses copy-on-write, but receiver controls placement of data.



- ## Other Client-Server Systems
- Lots.
 - Most notable systems:
 - Amoeba:** Mullender, Tanenbaum (early 1980's)
[Tanenbaum and Mullender, 1981; Tanenbaum and Mullender, 1984; Mullender and Tanenbaum 1986].
 - Windows NT:** Microsoft (early 1990's)
[Custer, 1993].