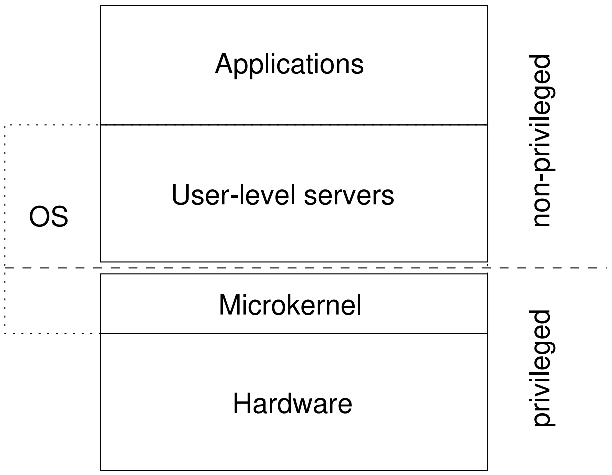
Microkernels and Client-Server Architectures

- early operating systems had very little structure
- a strictly layered approach was promoted by [Dijkstra 1968].
- later OS (more or less) followed that approach

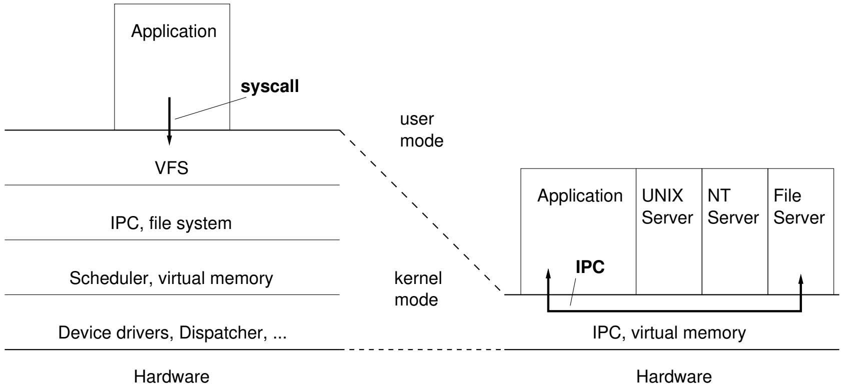
Problems with layered approach

- 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

Idea: break OS up



Monolithic vs. client-server OS structure



Kernel

- contains code which *must* run in supervisor mode
- isolates hardware dependence from higher levels
- is small and fast

⇒ extensible system

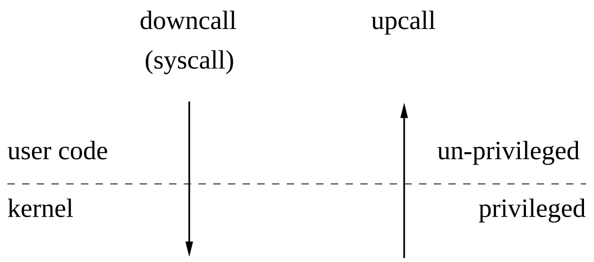
- **kernel** provides *mechanisms*

User-level servers

- are hardware independent/portable
- provide “OS environment”/”OS personality” (maybe several)
- may be invoked from **application**
  via *inter-process communication* (IPC, usually message-passing)
  or from **kernel** (upcalls)
- **servers** implement *policies*

[Brinch Hansen 1970].

Downcall vs. upcall



unprivileged code enters kernel mode	privileged code enters user mode
implemented via trap	implemented via IPC

Early example: Hydra

- separation of mechanism from policy (e.g. protection vs. security)
- no hierarchical layering of kernel
- protection, even within OS
  uses (segregated) *capabilities*,
- objects, encapsuation, units of protection
- unique object *name*, no ownership
- object persistence based on reference counting

[Wulf et al. 1974]

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

We will come back to Hydra several times later in the semester

POPULAR EXAMPLE: MACH

Developed at CMU by Rashid and others from 1984 on, successor of Accent [79] and RIG [75].

Goals:

- *tailorability:* support different OS interfaces
- *portability:* almost all code H/W independent
- *real-time*
- *multiprocessor and distribution* support
- *security*

Coined term *microkernel*.

BASIC FEATURES OF MACH μ-KERNEL

- task and thread management
- interprocess communication (operations on objects are achieved by message-passing)
- memory object management
- system call redirection
- device support
- multicomputer support

[Rashid et al. 1988]

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

Mach IPC: Ports

IPC addressing uses *ports*. A 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
- a task may hold a "send-once" capability to a port

Passing the receive capability to a port to another process implies giving up read access to the port. Kernel detects ports without senders or receiver.

Processes may have many ports (UNIX server has 2000!)
Can be grouped into *port sets*.

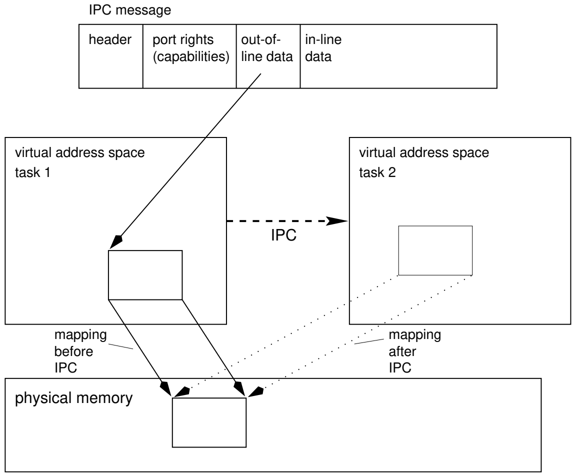
Mach IPC: Messages

Messages may contain

- send capability to destination port (mandatory)
  used by kernel to validate operation
- send capability to reply port
- possibly other capabilities
- "in-line" (by-value) data
- "out-of-line" (by reference) data, using copy-on-write,
  may contain whole address spaces

Capabilities are segregated and threads refer to them via local indices.
Kernel marshalls capabilities in messages.

Mach IPC



Mach virtual memory management

VM is a set of *memory regions*, created by

- explicit allocation (initially empty)
- mapping a *memory object*
- inheriting from "blueprint" (as in UNIX fork),
  may be *shared* or *copied*
- allocated automatically by kernel during IPC
  when passing by-reference parameters

$\Rightarrow$ sparse virtual memory use (unlike UNIX)

Copy-on-write

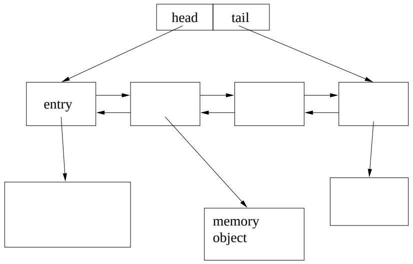
- Copies of data (inherited from “blueprint” or passed by-reference) are initially kept only once
  i.e. the two copies (at different VM addresses) are mapped to the same physical pages
- marked as read-only
- when one copy is modified, a fault occurs
- handling by kernel involves making a physical copy is made,
  the VM mapping is changed to refer to the new copy

⇒ efficient way of sharing/passing large amounts of data

⇒ expensive for small amounts of data (page table manipulations)

Address maps

task address spaces are represented as *address maps*:



Any part of a task’s address space can be mapped to an arbitrary location in a memory object.
Linked list structure supports *sparse* address spaces.

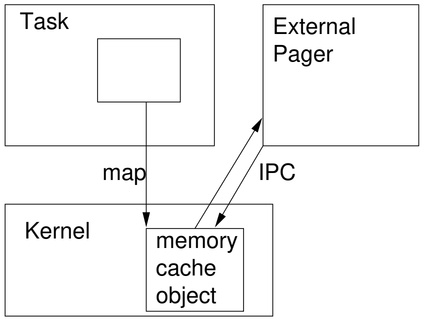
Memory objects

- are an abstraction of secondary storage
- are mapped into virtual memory
- are cached by the kernel in physical memory
- if not cached (i.e. non-resident), pager is invoked
- mapping objects into several tasks’ address spaces allows sharing
- no *files* are supported by the kernel
  (but servers can implement file systems on top of memory objects)

User-level page fault handlers

actual I/O performed by *pager*; can be

- default pager (provided by kernel), or
- *external* pager, running at user-level



HANDLING PAGE FAULTS

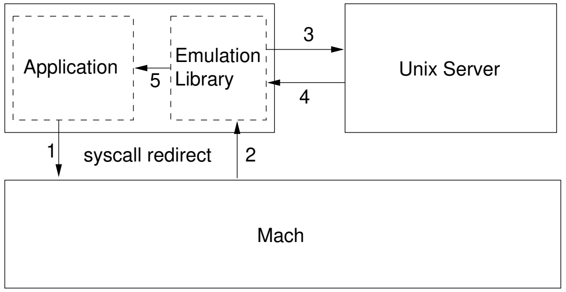
- check protection (from task's address map) and locate memory object
- check cache (using a hashed page table)
  if not cached invoke pager to provide data
- check copy-on-write: if write access to a COW marked page, copy page and map
- enter new mapping into H/W page tables

REMOTE COMMUNICATION

Client A sends message to server B on remote node

- on A's node there is a *proxy port* for B's receive port
  A sends message there
- on A's node there is a user-level *network message server* listening to proxy port
- NMS converts proxy port into (global) *network port*
- sends message to NMS on B's node (possibly after conversion)
- remote NMS converts network port into local port
- sends message to that port

EXAMPLE: MACH UNIX EMULATION



- emulation library and system call redirection used to implement UNIX services (*trampoline mechanism*)

MACH = MICROKERNEL?

- most OS services implemented at user level
- provides mechanisms, not policies
- mostly hardware independent
- 140 system calls
- size: 200k instructions
- performance???

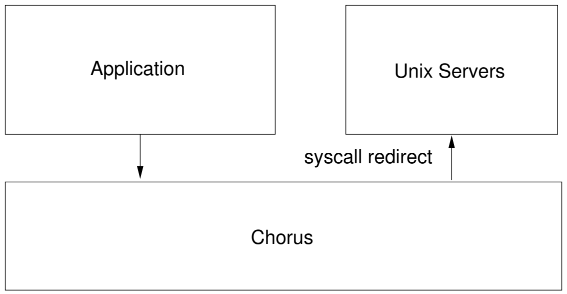
Further information on Mach:
[Young et al. 1987], [Coulouris, Dollimore, Kindberg 1994], [Sinha 1997]

Chorus

Developed at INRIA, France, from 1980 on, commercialised by *Chorus Systèmes* in 1988.

- Basic ideas similar to Mach
- servers can be user-level, or can be dynamically loaded into kernel space (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

Example: Chorus UNIX emulation



- system call redirection to UNIX server(s) used to implement UNIX services

Further information in [Dean, Armand 1992], [Rozier et al. 1990], [Rozier et al. 1992] [Coulouris, Dollimore, Kindberg 1994], [Sinha 1997]

Other client-server systems

The client-server approach has recently been used a lot, with varying success.

Most notable systems in this category:

Amoeba: Mullender, Tanenbaum (early 1980's)
[Tanenbaum, Mullender 1981], [Tanenbaum, Mullender 1984], [Tanenbaum, Mullender 1986]

Windows NT: Microsoft (early 1990's)
[Custer 1993]

We will have a closer look at these systems later on.

References

[BH70] Per Brinch Hansen. The nucleus of a multiprogramming operating system. *Comm. ACM*, 13:238–250, 1970.

[CDK94] George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2nd edition, 1994.

[Cus93] Helen Custer. *Inside Windows NT*. Microsoft Press, 1993.

[DA92] Randall W. Dean and Francois Armand. Data movement in kernelized systems. In *Proc. USENIX W. Microkernels & other Kernel Arch.*, pages 243–261, Seattle, WA, USA, April 1992.

[Dij68] Edsger W. Dijkstra. The structure of the "THE" multiprogramming system. *Comm. ACM*, 11:341–346, 1968.

[MT86] Sape J. Mullender and Andrew S. Tanenbaum. The design of a capability-based distributed operating system. *The Comp. J.*, 29:289–299, 1986.

[RAA+90] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the CHORUS distributed operating system. Technical report CS/TR-90-25, Chorus systeèmes, Montigny-le-Bretonneux (France), April 1990.

[RAA+92] M. Rozier, V. Abrossimov, F. Armand, L. Boule, M. Gien, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the Chorus distributed operating system. In *Proc. USENIX W. Microkernels & other Kernel Arch.*, pages 39–69, Seattle, WA, USA, April 1992.

[RTY+88] Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golub, Robert Baron, David Black, William J. Bolosky, and Jonathan Chew. Machine-independent virtual memory management for paged uniprocessor and multiprocessor

architectures. *IEEE Trans. Computers*, C-37:896–908, 1988.

[Sin97] Pradeep K. Sinha. *Distributed Operating Systems: Concepts and Design.* IEEE Comp. Soc. Press, 1997.

[TM81] Andrew S. Tanenbaum and Sape J. Mullender. An overview of the Amoeba distributed operating system. *Operat. Syst. Rev.*, 15(3):51–64, 1981.

[TM84] Andrew S. Tanenbaum and Sape Mullender. The design of a capability-based distributed operating system. Technical Report IR-88, Vrije Universiteit, November 1984.

[WCC+74] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: The kernel of a multiprocessor operating system. *Comm. ACM*, 17:337–345, 1974.

[YTR+87] Michael Young, Avadis Tevanian, Richard Rashid, David Golub, Jeffrey Eppinger, Jonathan Chew, William Bolosky, David Black,

and Robert Baron. The duality of memory and communication in the implementation of a multiprocessor operating system. In *Proc. 11th ACM Symp. OS Princip.*, pages 63–76. ACM, 1987.