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 Torwalds*

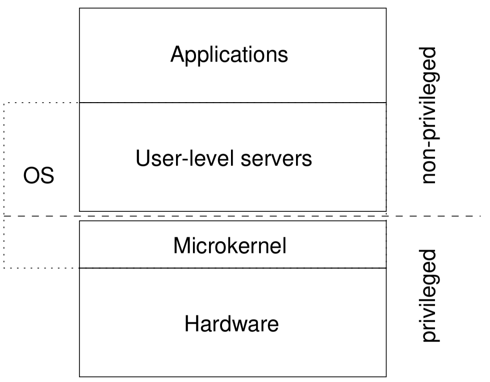
Motivation

- 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).

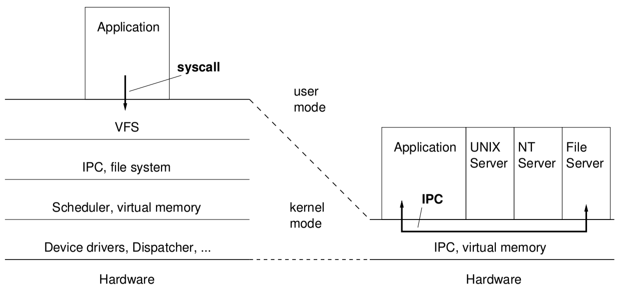
Problems with layered approach

- 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.

Idea: break up the OS



Monolithic vs. client-server OS structure



Kernel:

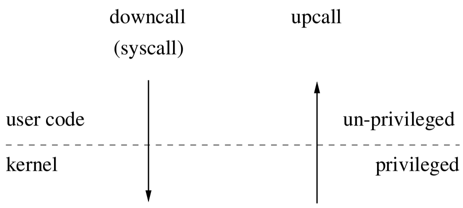
- Contains code which *must* run in supervisor mode;
- Isolates hardware dependence from higher levels;
- Is small and fast

⇒ 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 message-passing IPC)
  ➜ from **kernel** (upcalls);
- **Servers** implement *policies* [BH70].

Downcall vs. upcall



downcall	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 [WCC+74].

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.

Popular Example: Mach

- Developed at CMU by Rashid and others [RTY+88] from 1984
- successor of Accent [FR86] and RIG [Ras88].

GOALS:

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

Coined term *microkernel*.

BASIC FEATURES OF MACH μ-KERNEL

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

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.

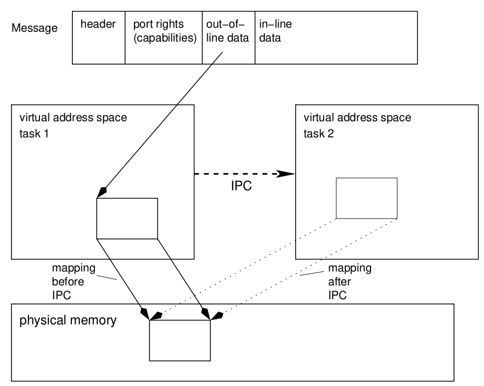
MACH IPC: PORTS

- 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)

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;

MACH IPC



MACH VIRTUAL MEMORY MANAGEMENT

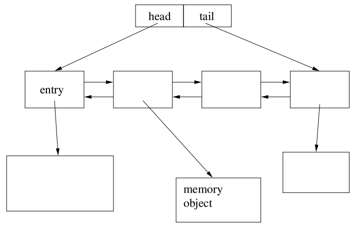
- Address space constructed from *memory regions*
  - – initially empty
  - – populated by:
    - ➜ explicit allocation
    - ➜ explicitly mapping a *memory object*;
    - ➜ inheriting from “blueprint” (as in Linux clone()),
      - ➜ inheritance: *not*, *shared* or *copied*;
    - ➜ allocated automatically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

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

MACH ADDRESS MAPS

- Address spaces represented as *address maps*:



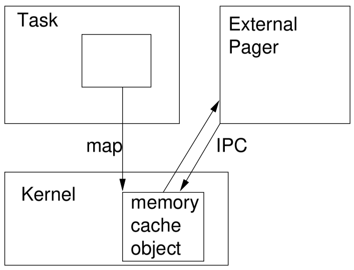
- Any part of AS can be mapped to (part of) a memory object
- Compact representation of *sparse* address spaces
  ➜ Compare to multi-level page tables?

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

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

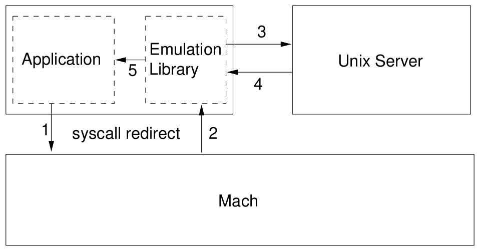
HANDLING PAGE FAULTS

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.

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.

MACH UNIX EMULATION



- emulation library in user address space handles IPC
- invoked by system call redirection (*trampoline mechanism*)
  ➜ supports binary compatibility

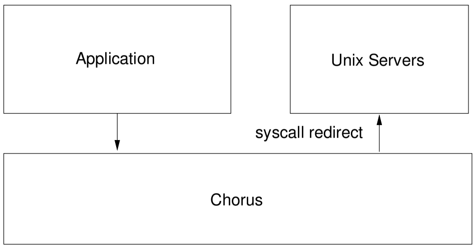
MACH = MICROKERNEL?

- 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: [YTR$^+$87, CDK94, Sin97]

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.

CHORUS UNIX EMULATION



➜ System call redirection to server(s)

➜ All UNIX emulation in server (to avoid protection problems)

➜ Further information in [DA92, RAA+90, RAA+92, CDK94, Sin97].

Other client-server systems

- Lots.
- Most notable systems:
  Amoeba: Mullender, Tanenbaum (early 1980's) [TM81, TM84, MT86].
  Windows NT: Microsoft (early 1990's) [Cus93].

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. W. Microkernels & other Kernel Arch.*, pages 243–261, Seattle, WA, USA, Apr 1992.

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

[FR86] Robert Fitzgerald and Richard Rashid. The integration of virtual memory management and interprocess communication in Accent. *Trans. Comp. Syst.*, 4:147–177, 1986.

[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), Apr 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. W. Microkernels & other Kernel Arch.*, pages 39–69, Seattle, WA, USA, Apr 1992.

[Ras88] Richard F. Rashid. From RIG to Accent to Mach: The evolution of a network operating system. In Bernardo A. Huberman, editor, *The Ecology of Computation*, Studies in Computer Science and Artifical Intelligence, pages 207–230. North-Holland, Amsterdam, 1988.

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

[Sin97] Pradeep K. Sinha. *Distributed Operating Systems: Concepts and Design*. 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, Nov 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 SOSP*, pages 63–76, 1987.