Modularity through Client-Server

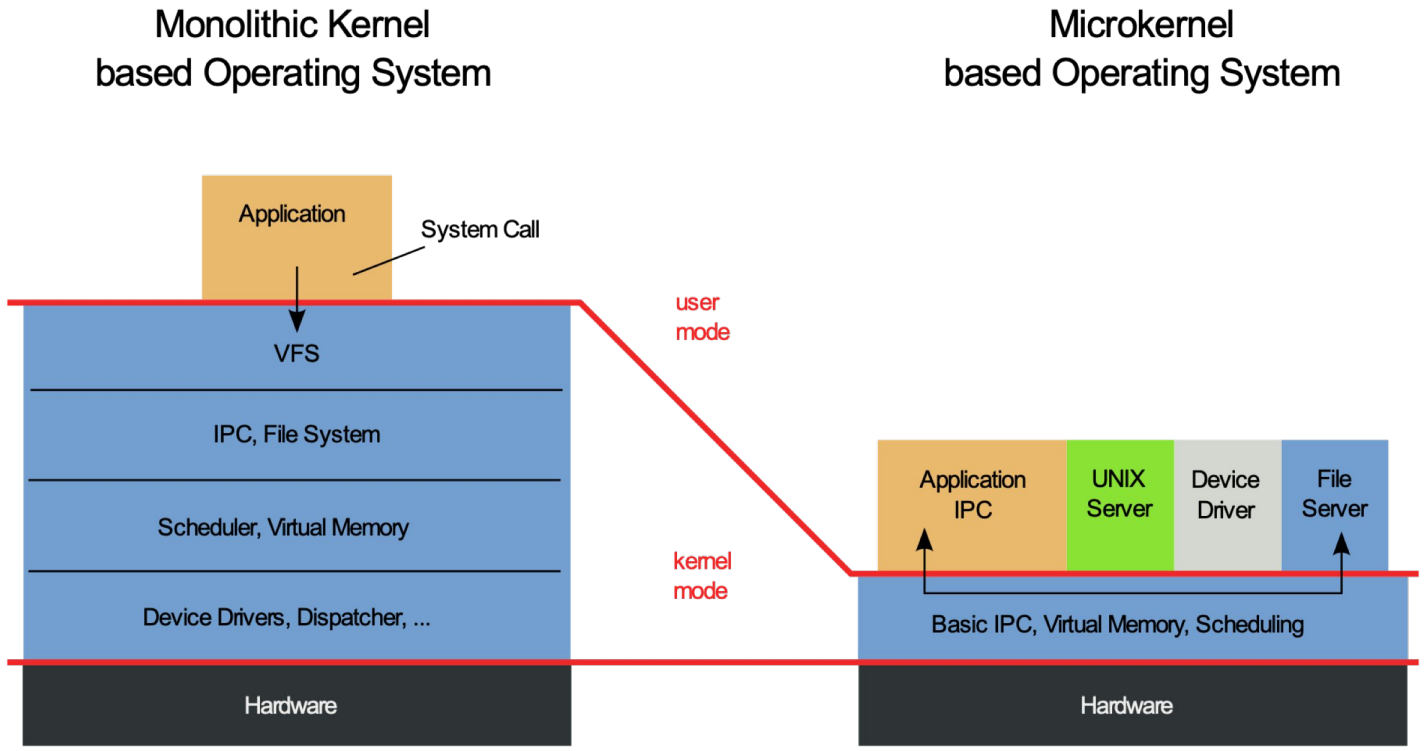
µ-kernels

What is a µ-kernel and what are the advantages of the µ-kernel design?

- Enforcing modular system structure
- Enforced modularity/Fault isolation

On µ-Kernel Construction

- Main Ideas
  - Minimality Principle
    - Address Spaces
    - Threads & IPC
    - Unique Identifiers
  - Misconceptions:
    - Performance



Principles

- Independence

  A programmer must be able to implement an arbitrary subsystem S in such a way that it cannot be disturbed or corrupted by other subsystems S’

- Integrity

  There must be a way for S_1 to address S_2 and establish a communication channel which can neither be corrupted or eavesdropped by S’

Compare the µ-kernel with the exokernel design

“The presented design shows that it is possible to achieve well performing µ-kernels through processor-specific implementations of processor-independent abstractions.”

How does a µ-kernel guarantee integrity and authentication for the IPC?

- Authentication is done through UID. The kernel controls the UIDs, apps/servers cannot forge it.
- Address space enforces integrity. IPC can be done with or without data copying. One way is to have the kernel copying the data across the address spaces of the sender/receiver. The other way is to have shared memory between the sender/receiver, and IPC is done through the shared memory without copies. In both cases, the data is either in the address space of the sender or receiver during the IPC, no other subsystems can eavesdrop the IPC since the data is never passed in their address space.

How do other client-server systems achieve those properties?

- Try to answer the question on your own. The answer is on next slides

How do other client-server systems achieve those properties?

- One way I can think of is to use encryption to ensure integrity of the communication channel and using signature for authentication.

What is the underlying assumption in the paper in order to guarantee the two principles?

- The kernel and the hardware has to be trusted.
- The kernel has to be trusted to be correct and non-malicious since these two principles are enforced by the kernel. For example, if the kernel has a bug, it might forward the IPC msg to a server A to another server B by mistake, which will break the integrity principle.
- The hardware has to be trusted since it also participates in enforcing the principles. For example, the hardware translates the virtual memory address to the physical memory address, if there is a bug in the translation hardware, virtual memory address of app A might be translated to a physical address mapped to another app B, which breaks the independence principle.

Check Reflections on Trusting Trust

What is a trusted intermediary?

- A single service that has multiple clients brings up another technique for enforcing modularity: the *trusted intermediary*, a service that functions as the trusted third party among multiple, perhaps mutually distrusted, clients. The trusted intermediary can control shared resources in a careful manner.
- One example of trusted intermediary is the exokernel. The exokernel serves mutually-distrusted clients, i.e., the libOSes. And it safely multiplexes the shared hardware resources among libOSes.

Who is a trusted intermediary in the µ-kernel case?

- One example is the microkernel itself. It serves mutually-distrusted clients, i.e., the apps/servers. It protects the shared resources such as memory.
- Another example could be a filesystem server. It serves mutually-distrusted users, and it protects the files and only allow users with the right permissions to access files.

Design Project

Design a µ-kernel service (could be implemented as one or multiple servers) that provides networking functionality to the programs running on the system, e.g., a web server. Think about the abstractions that the networking service will expose, the IPC mechanism, and how to achieve integrity.

Some details to think of regarding the IPC: Should you use synchronous or asynchronous IPC? Should the IPC be based on shared-memory or not? How should the program check for data availability, via polling or interrupts?