National Yang Ming Chiao Tung University
Computer Architecture & System Lab

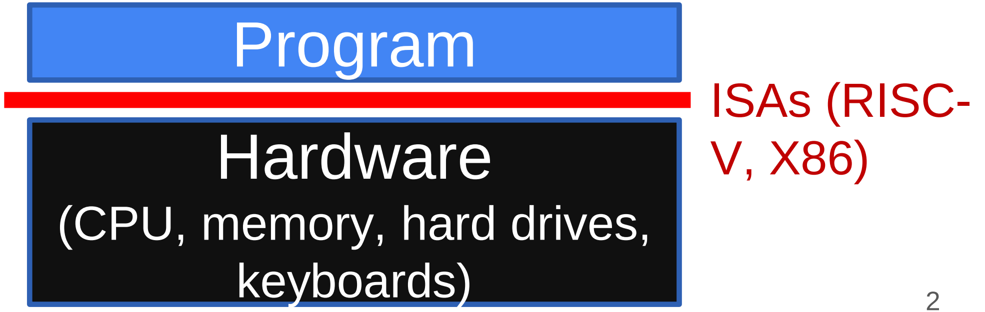
IOC5226 Operating System Capstone

Lecture 1: Course Overview

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao Tung University

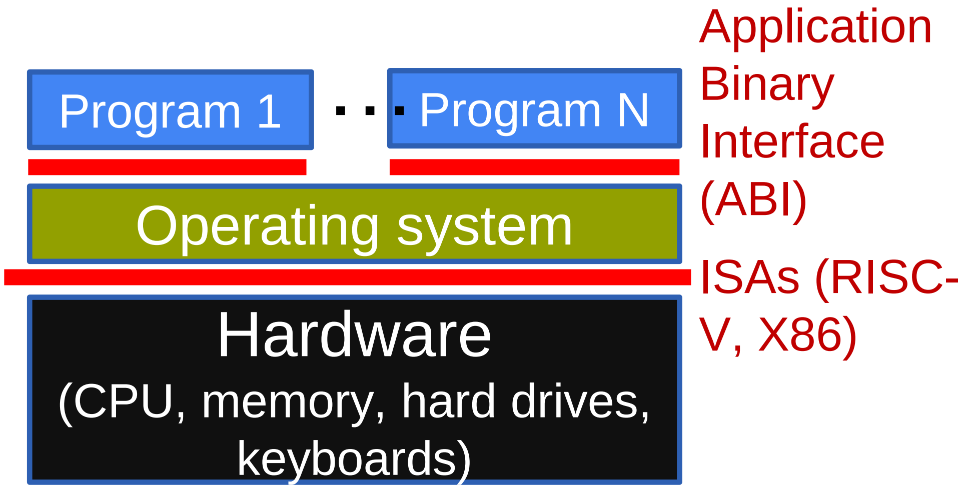
Single-User Machines

- Hardware executes a single program
- The program can access directly all hardware resources in the machine
- The instruction set architecture (ISA) is the interface between software and hardware
- However
  - Most computer systems aren't work like this !



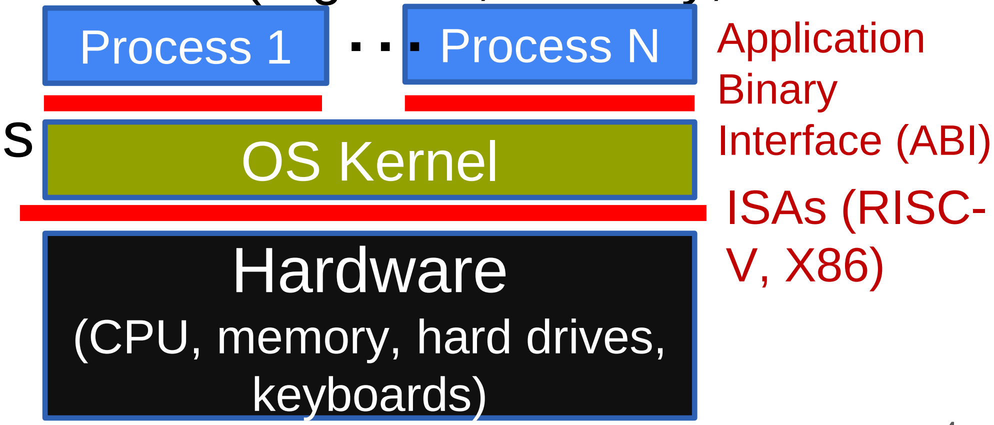
Operating systems

- Multiple executing programs share the machine
- Each program cannot access hardware resource directly
- An operating system (OS)
  - Control these programs how they share hardware
- The application binary interface (ABI) is the interface between programs and the OS



Process vs. Program

- A program is a collection of instructions
- A process is an instance of a program that is being executed
  - Include program code + other state (registers, memory, and other resources)
- The OS kernel is a process with special privileges

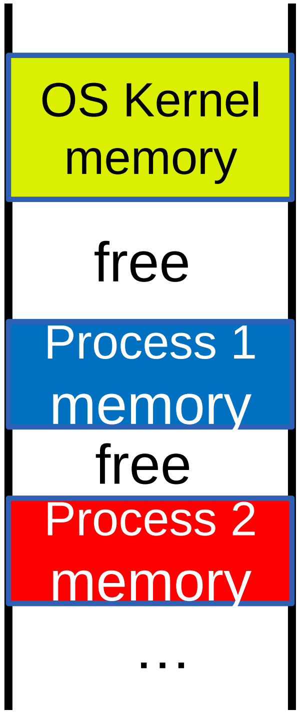


Goals of operating systems

- An operating system is to support several activities at once
  - Many running program as **processes**
- **Protection and privacy**
  - Process multiplexing
  - Processes cannot access each other’s data (isolation)
- **Abstraction**
  - OS hides details of underlying hardware
  - Hardware resource manager

Operating systems: The big picture

- The OS kernel provides a private address space to each process
    - Each process is allocated space in physical memory by the OS
    - A process is not allowed to access the memory of other processes
- The OS kernel schedules processes into the CPU
    - Each process is given a fraction of CPU time
    - A process cannot use more CPU time than allowed
    - Context switch

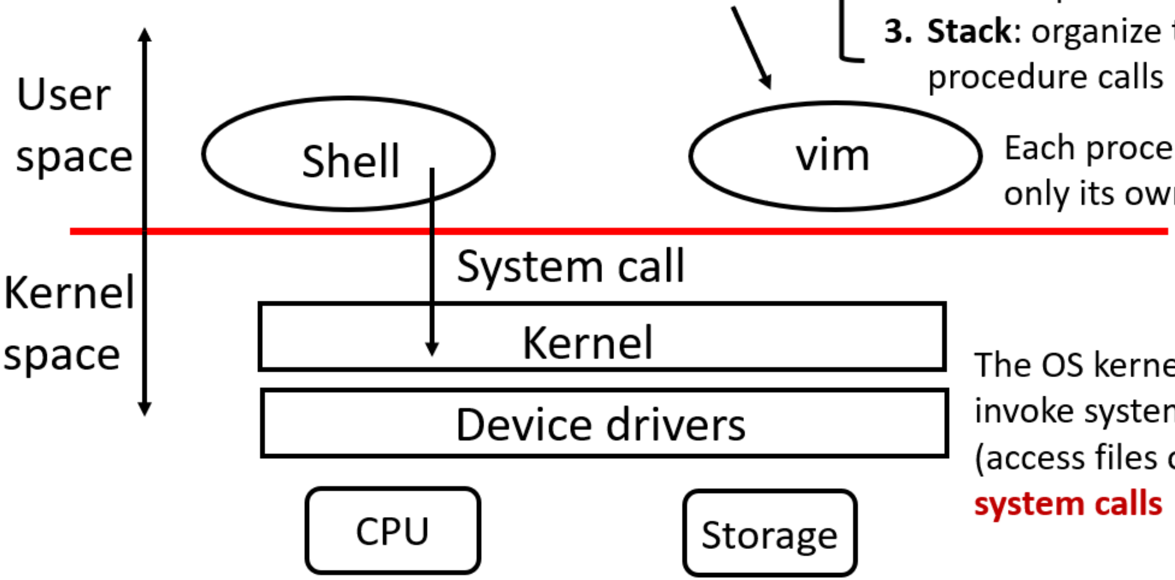


Physical memory

How does OS work?

Each running program is called **process**

Process
1. **Instructions**: implement the program's computation
2. **Data**: the variables on which the computation acts
3. **Stack**: organize the program's procedure calls



Each process can access only its own memory

The OS kernel let processes invoke system services (access files or network) via **system calls**

Implementing an OS

- The OS works as a **virtual machine (VM)** to each process
  - Each process believes it runs on its own machine
- Virtual machines can be implemented entirely in software, but at a performance cost
  - For instance, python programs are 10 – 100x slower than native Linux programs because python interpreter overheads
- We want to support operating systems with minimal overheads
  - Need hardware support for virtual machine

User and kernel mode

- Two modes of execution: user and kernel (supervisor)
  - OS kernel runs in supervisor mode
  - All other processes run in user mode
- **In the kernel mode**
  - Privilege instructions and register are available
  - Interrupts and exceptions to safely transition from user to supervisor mode
- **Virtual memory**
  - Provide private address spaces and abstract the storage resources of the machine

What services does an OS kernel provides?

- Processes
- Memory allocation
- File systems
- Security
- Others: users, networking, terminals, etc..

Process and thread

- Each process has a thread of execution
  - The state of a thread (local variables, function call return address) is stored on the thread's stacks
  - Each process has two stacks: a user stack and a kernel stack

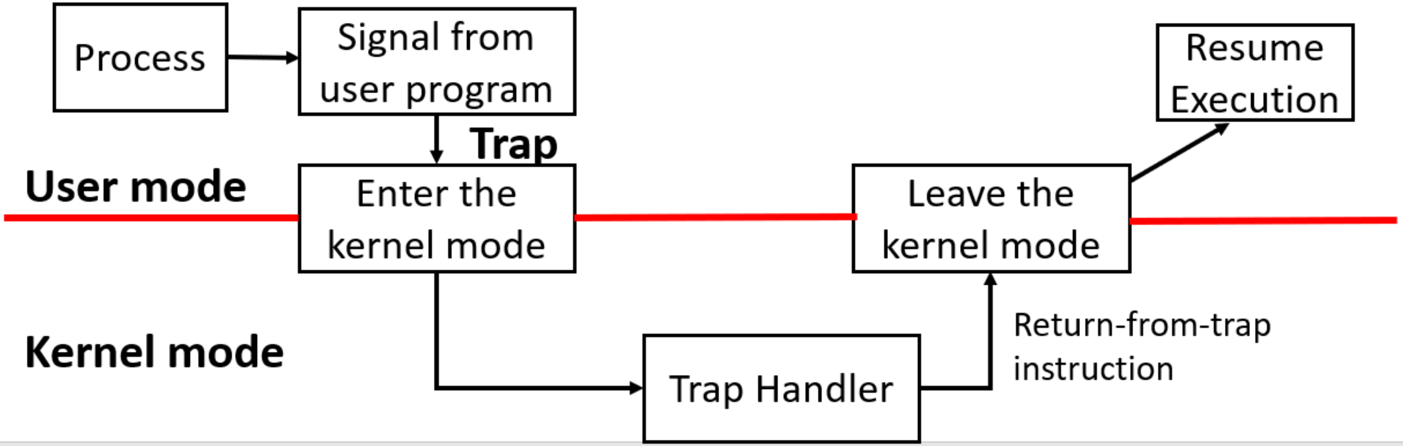
Process	Thread
Process is any in-execution program	Thread is the segment of a process
Process is isolated	Thread share memory
Process has its own process control block (PCB) and address space	Thread has parent's PCB, its own TCB, stack, and address space
Process takes more time for creation	Thread takes less time for creation

https://www.geeksforgeeks.org/difference-between-process-and-thread/

System call

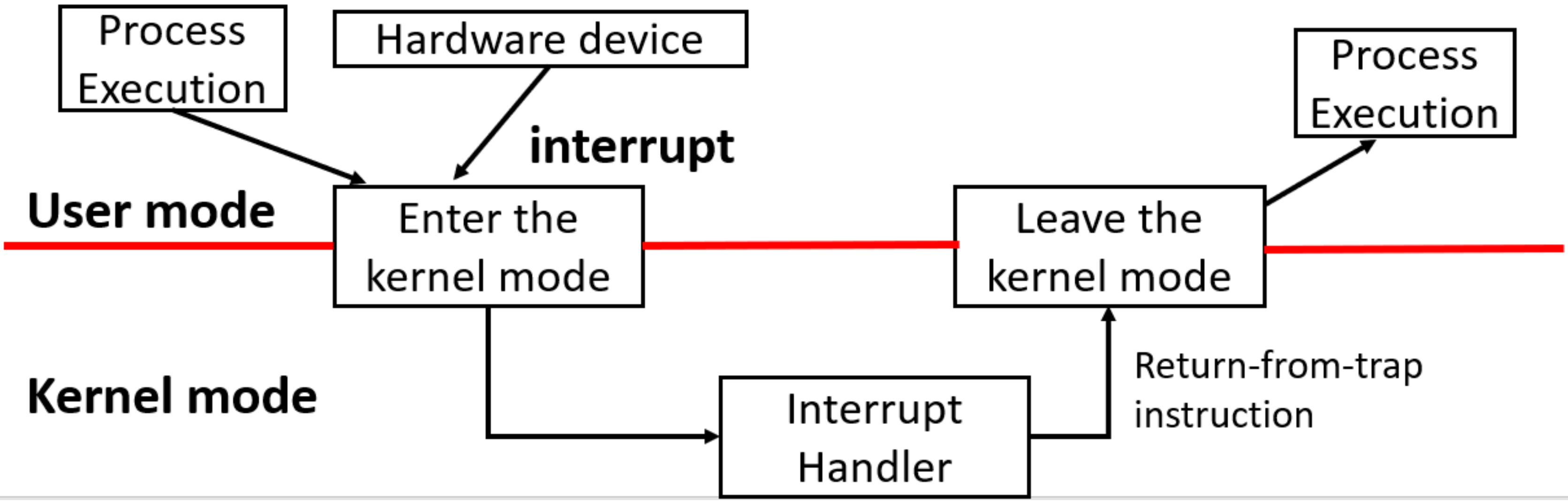
1. During a trap, the execution of a process is set as high priority compared to user code
2. When the OS detects a trap, it pauses the user process
3. The OS resumes the execution when the system call is completed

- Using a **trap** that is a synchronous interrupt triggered by an exception in a user process to execute functionality.



What is the interrupt in the OS ?

- An interrupt is a hardware or software signal and notifies the processor that a critical process needs urgent execution
- Using to interrupt present working process



What is the interrupt in the OS ?

- **Interrupt Service Routine (ISR)**
  - A specific bus control line handles interrupts in I/O devices
- A CPU contains a specific interrupt pin known as INT pin for the interrupt
  - The INT pin connects hardware devices such as keyboards, NIC cards
  - OS can invoke the keyboard interrupt handler routine to do interrupt
  - **Multiple hardware devices share a single INT pin** using an interrupt controller
  - To determine which device produced the interrupt, the processor contacts the interrupt controller.

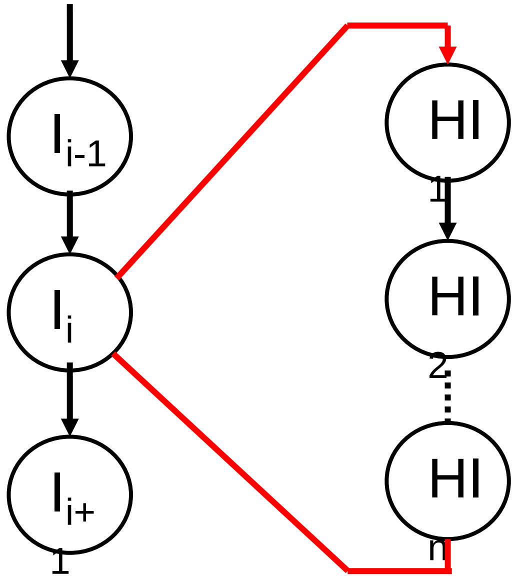
Difference between the trap and interrupt?

Trap	Interrupt
A signal emitted by **a user program**	A signal emitted by **a hardware device**
Synchronous process	Asynchronous process
Can occur only from software device	Can occur from a hardware or a software
Only generated by a user program ISA	Generated by an OS and user program ISA
Traps are subset of interrupts	Interrupts are superset of traps
Execute a specific functionality in the OS and gives the control to the trap handler	Force the CPU to trigger a specific interrupt hander routine

https://www.baeldung.com/cs/os-trap-vs-interrupt

Exceptions

- Exception: Event that needs to be processed by the OS kernel. The event is usually unexpected or rare



Exception handler
(in OS kernel)

Causes for exceptions

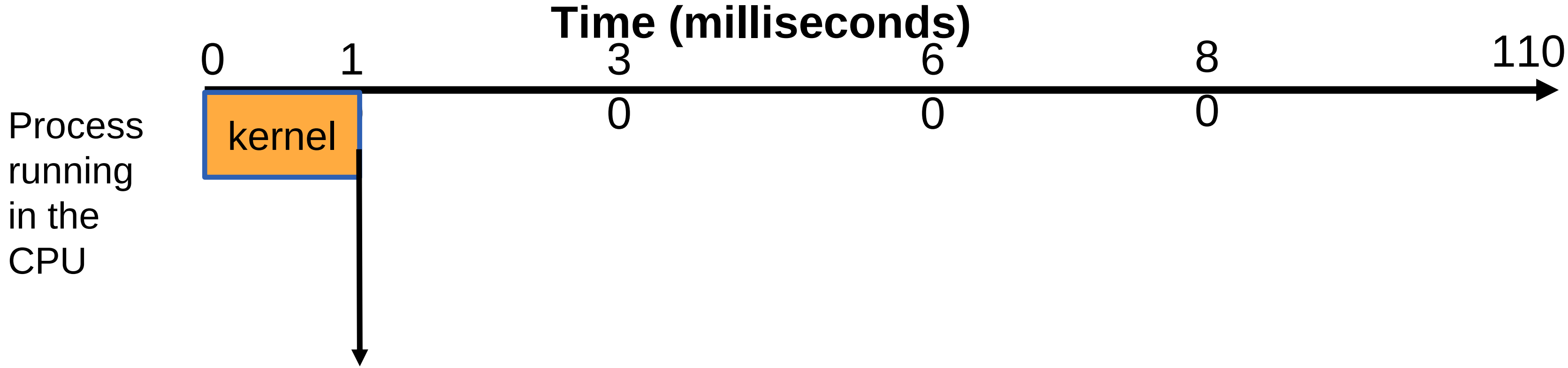
- **Exceptions**
  - Synchronous events generated by the process itself
  - E.g. illegal instructions, divide-by-0, illegal memory address
- **Interrupts**
  - Asynchronous events generated by I/O devices
  - E.g. timer expired, keystroke, packet received, disk transfer complete

Handling exceptions

- When an exception happens, the processor
  - Stop the current process at instruction I_i, completing all the instructions up to I_{i-1} (precise exceptions)
  - Saves the PC of instruction I_i and the reason for the exception in special (privileged) register
  - Enable supervisor mode, disable interrupts, and transfers control to a pre-specified exception handler PC
- After the OS kernel handles the exception, it returns control to the process at instruction I_i
  - Exception is transparent to the process
- If the exception is due to an illegal operation by the program that cannot be fixed, the OS aborts the process

Case study 1: CPU scheduling

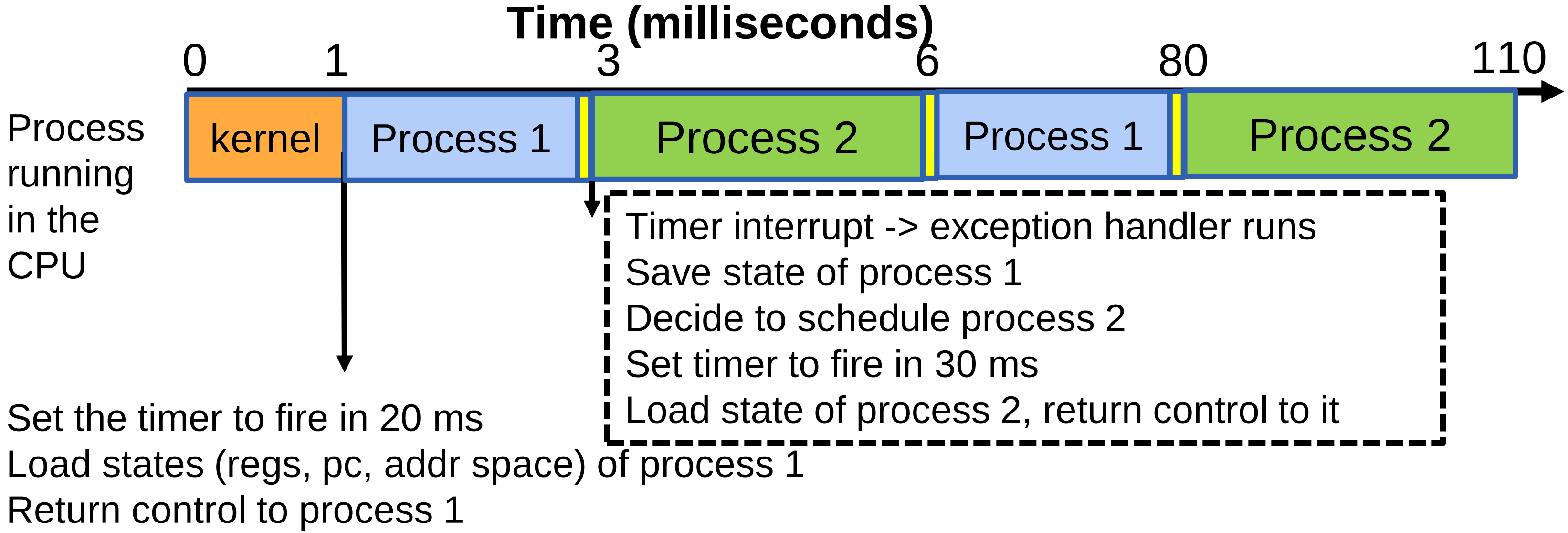
- The OS kernel schedules processes into the CPU
  - Each process is given a fraction of CPU time
  - Enabled by timer interrupts
  - Kernel sets timer, which raises an interrupt after a specified time



Set the timer to fire in 20 ms
Load states (regs, pc, addr space) of process 1
Return control to process 1

Case study 1: CPU scheduling

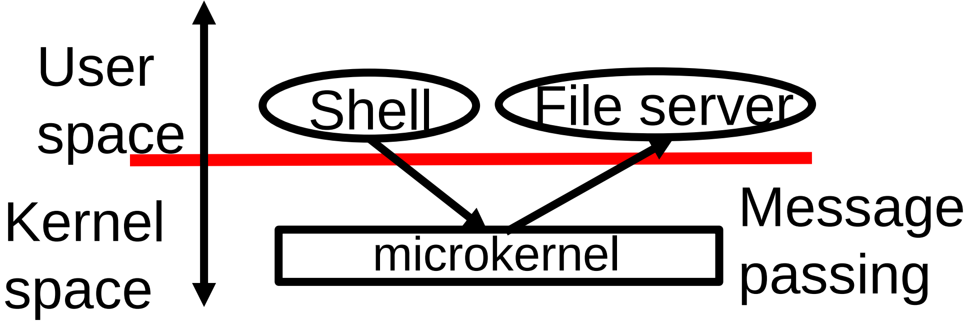
- The OS kernel schedules processes into the CPU
  - Each process is given a fraction of CPU time
  - Enabled by timer interrupts
  - Kernel sets timer, which raises an interrupt after a specified time



Kernel organization

- **Monolithic kernel**
  - Entire operating system resides in the kernel space
  - The implementations of all system calls run in kernel mode
  - E.g. Unix, Linux
- **Good**
  - Easy for subsystems to cooperate
  - One cache shared by file system and virtual memory
- **Bad**
  - Interactions are complex
  - Mistake is fatal because an error in kernel model will result in the kernel to fail
  - No isolation within kernel

Kernel organization



- **Microkernel**
  - Move most OS functionality to user-space
  - Kernel can be small, mostly IPC
  - The hope:
    - Simple kernel can be fast and reliable
- **Microkernel wins:**
  - Fast IPC
  - separate services force kernel developers to think about modularity
- **Microkernel losses:**
  - kernel can't be tiny: needs to know about processes and memory
  - it's hard to split the kernel into lots of service processes!