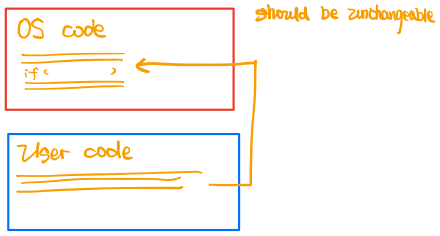


Topic 2: Protecting the OS

Reading: Sections 2.1-2.4

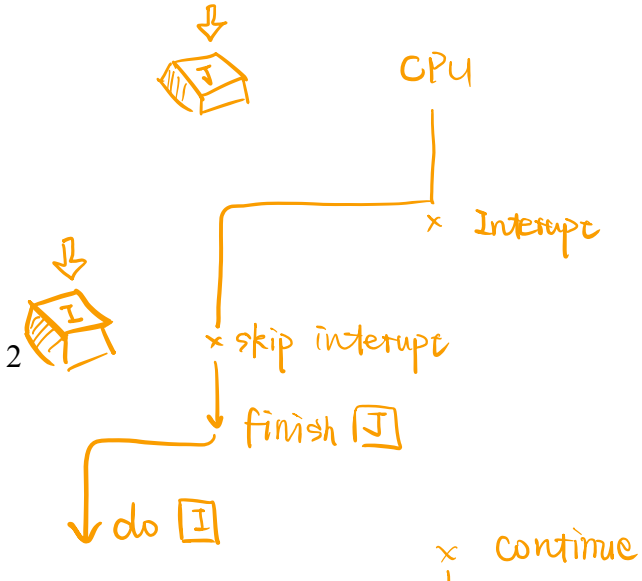
Next reading: same

- Think of the OS as a library that contains routines[1] that everyone needs. Consider a routine that reads a file from the disk. How does the OS ensure that the file is protected properly, e.g. only the owner can read it?
  - One possibility is to check permissions at the beginning of the routine, and return if the read is not allowed.
  - What prevents someone from writing assembly code that jumps into the routine after the protection is checked?
  - Procedure calls offer no protection.
- The OS needs hardware support to protect itself from applications.
  - Use user/supervisor (*kernel*) mode of processor.
  - Certain machine instructions can only be executed while in kernel mode (*privileged instructions*).
  - Some of these instructions allow memory to be made off-limits to user mode. The OS puts itself this memory.

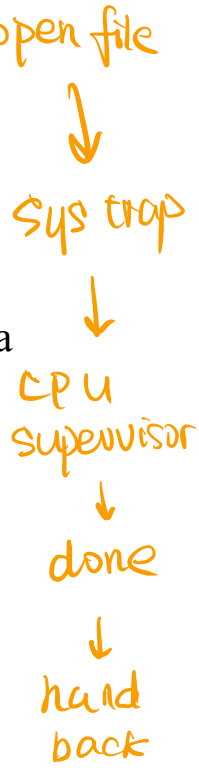
[1] I will use the terms procedure, function, routine, and subroutine interchangeably.

- User programs run in user mode; OS runs in kernel mode. For this reason the OS is often called the *kernel*.
- Processor switches from user mode to kernel mode when an *interrupt* or *trap* occurs.
  - User program cannot switch to kernel mode directly!
- OS uses privileged instruction to switch back to user mode once the interrupt or trap has been handled.
- Processor powers-up in kernel mode.

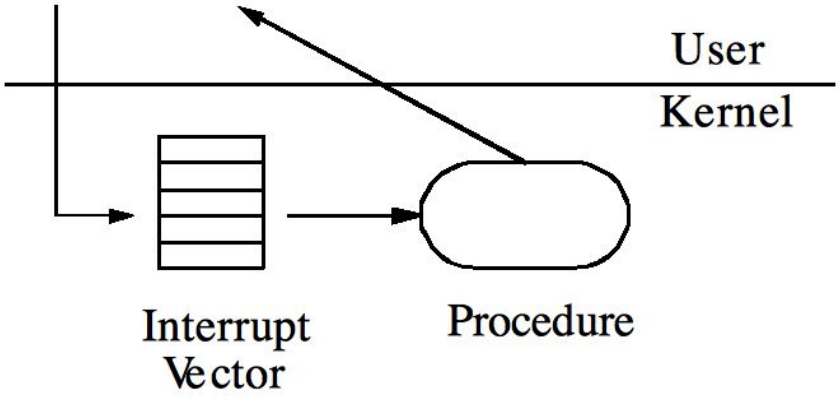
- *Interrupt*: asynchronous event that causes the processor to stop what it is doing and begin executing at a specified address. Has nothing to do with the instruction being executed, e.g. disk interrupt. <KeyBoard Interupt>
  - CPU switches to kernel mode if it is in user mode.
  - Routine located at the address is called an *interrupt handler*.
  - *Interrupt vector* is a table of interrupt handlers, one for each type of interrupt.
  - Processor switches to a separate *interrupt stack* or *kernel stack*.
    - Reliability
    - Security



  - As part of invoking an interrupt handler the processor *masks* interrupts of the same type. OS can change an interrupt's mask.
  - Interrupts sometimes have priorities, depending on the CPU.
    - Higher priority interrupts can interrupt lower-priority handlers.
- *Trap*: synchronous event associated with the execution of the current instruction, e.g. divide-by-zero, overflow.
  - A *system call* is a special kind of trap. It is not an error, but a request by the user program for the OS to do something. Typically processors have a trap instruction that is used to implement system calls.
- OS is interrupt-driven. *The only way to enter it is via an interrupt or trap*.
- User programs can only access OS through system calls (traps), allowing the OS to protect itself.
- Interrupts make the OS *non-deterministic*. It is difficult to deal with so many things going on at one time.
- OS software structure
  - Monolithic (e.g. Linux)



the only way to swith from user to sys (kernel)



- OS is a protected library. User programs invoke system calls via trap. OS services implemented as a bunch of procedures.
- Some OS-related programs never leave the kernel. More on this later.
- Can be messy. Requires good software engineering practices.
- More on other OS architectures later.