**The Kernel**

wants to be your friend

---

**Boxing them in**

- Buggy apps can crash other apps
- Buggy apps can crash OS
- Buggy apps can hog all resources
- Malicious apps can violate privacy of other apps
- Malicious apps can change the OS

---

**The Process**

- An abstraction for isolation
  - the execution of an application program with restricted rights
- But there are tradeoffs
  (there always are tradeoffs!)
- Must not hinder functionality
  - still efficient use of hardware
  - enable safe communication

---

**The Process**

- An abstraction for isolation
  - the execution of an application program with restricted rights
- But there are tradeoffs
  (there always are tradeoffs!)
- Must not hinder functionality
  - still efficient use of hardware
  - enable safe communication
The Process

- An abstraction for isolation
- the execution of an application program with restricted rights
- But there are tradeoffs (there always are tradeoffs!)
- Must not hinder functionality
- still efficient use of hardware
- enable safe communication

Actually…

Special

- Part of the OS
  - all kernel is in the OS
  - not all the OS is in the kernel
  - (why not? robustness)
  - widgets libraries, window managers, etc

Process: Getting to know you

- A process is a program during execution
- program is a static file
- process = executing program = program + execution state

Source code

Compiler

Executable Image

OS copy

Physical memory
Keeping track of a process

- A process has code
  - OS must track program counter
- A process has a stack
  - OS must track stack pointer
- OS stores state of process in Process Control Block (PCB)
  - Data (program instructions, stack & heap) resides in memory, metadata is in PCB

How can the OS enforce restricted rights?

- Easy: kernel interprets each instruction!
- Slow
- Many instructions are safe: do we really need to involve the OS?

How can the OS enforce restricted rights?

**Dual Mode Operation**

- Hardware to the rescue: use a mode bit
  - In **user mode**, processor checks every instruction, to make sure it is allowed
  - In **kernel mode**, unrestricted rights
- Hardware to the rescue (again) to make checks efficient

Efficient protection in dual mode operation

- **Privileged instructions**
  - In user mode, no way to execute potentially unsafe instructions
- **Memory isolation**
  - In user mode, memory accesses outside a process’ memory region are prohibited
- **Timer interrupts**
  - Kernel must be able to periodically regain control from running process

Efficient mechanism for switching modes
I. Privileged instructions

- Set mode bit
- I/O ops
- Memory management ops
- Disable interrupts
- Set timers
- Halt the processor

But how can an app do I/O then?

- System calls achieve access to kernel mode only at specific locations specified by OS

Executing a privileged instruction while in user mode (naughty naughty…) causes a processor exception….

...which passes control to the kernel

II. Memory Protection

Step 1: Virtualize Memory

- Virtual address space: set of memory addresses that process can “touch”
- CPU works with virtual addresses
- Physical address space: set of memory addresses supported by hardware
II. Memory Protection

Step 2: Address Translation

- Implement a function mapping \((\text{pid}, \text{virtual address})\) into physical address

Advantages:
- protection
- relocation
- data sharing
- multiplexing

Isolation

- At all times, the functions used by different processes map to disjoint ranges

Relocation

- The range of the function used by a process can change over time
**Data Sharing**

- Map different virtual addresses of different processes to the same physical address

```
π_i
01a26a

π_j
11a9f3

5e3a07
```

**Multiplexing**

- Create illusion of almost infinite memory by changing domain (set of virtual addresses) that maps to a given range of physical addresses

```
π_i
```

**Multiplexing**

- The domain (set of virtual addresses) that maps to a given range of physical addresses can change over time

```
π_i
```

**Multiplexing**

- The domain (set of virtual addresses) that maps to a given range of physical addresses can change over time

```
π_i
```
The domain (set of virtual addresses) that map to a given range of physical addresses can change over time.