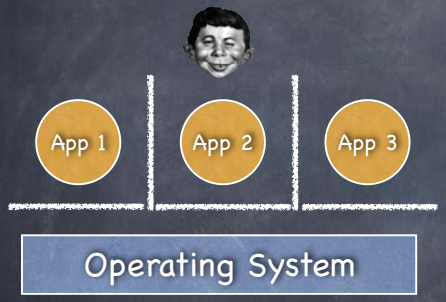
The Kernel

wants to be your friend

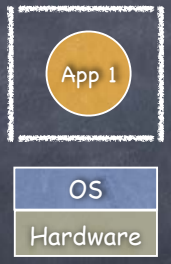
Boxing them in



Reading and writing memory, managing resources, accessing I/O...

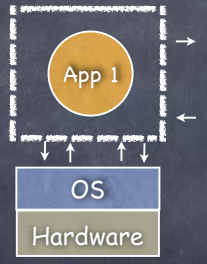
- Buggy apps can crash other apps
- Buggy apps can crash OS
- Buggy apps can hog all resources
- Malicious apps can violate privacy of other apps
- Malicious apps can change the OS

The Process



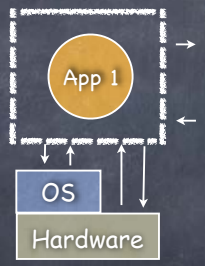
- An abstraction for isolation
  - the execution of an application program with restricted rights
- But there are tradeoffs (there always are tradeoffs!)
- Must not hinder functionality
  - still efficient use of hardware
  - enable safe communication

The Process

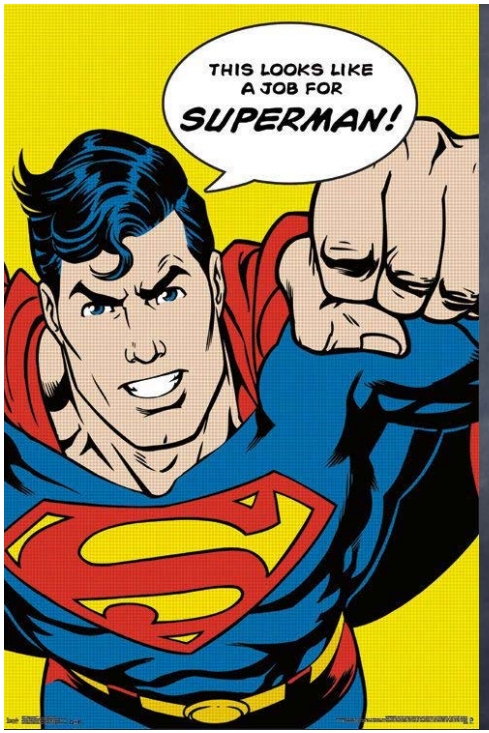


- An abstraction for isolation
  - the execution of an application program with restricted rights
- But there are tradeoffs (there always are tradeoffs!)
- Must not hinder functionality
  - still efficient use of hardware
  - enable safe communication

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



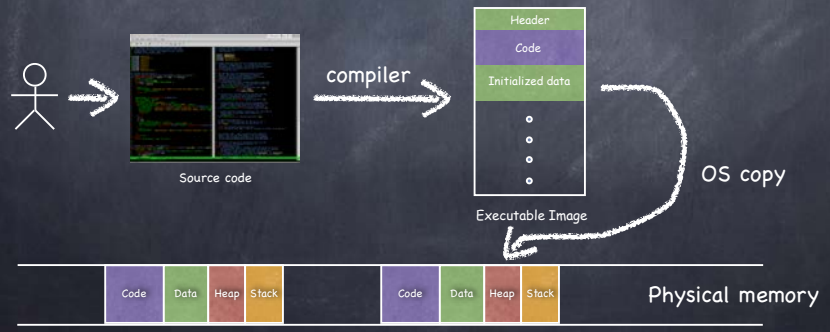
Actually...

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



Keeping track of a process

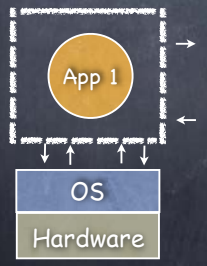
- A process has code
  - OS must track program counter
- A process has a stack
  - OS must track stack pointer
- OS stores state of process in Process Control Block (PCB)
  - Data (program instructions, stack & heap) resides in memory, metadata is in PCB

Process Control Block

PC
Stack Ptr
Registers
PID
UID
Priority
List of open files
Process status
Kernel stack ptr
...

How can the OS enforce restricted rights?

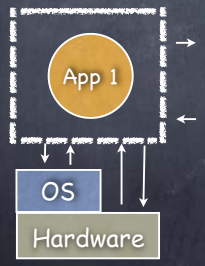
- Easy: kernel interprets each instruction!



- 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

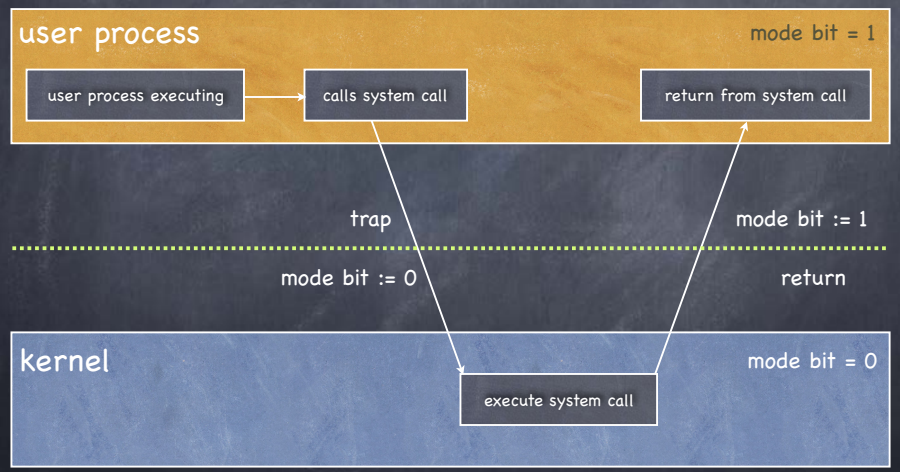
I. Privileged instructions

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

I. Privileged instructions

- 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

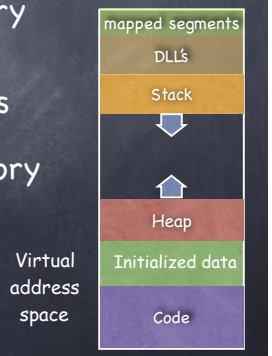
Crossing the line



II. Memory Protection

Step 1: Virtualize Memory

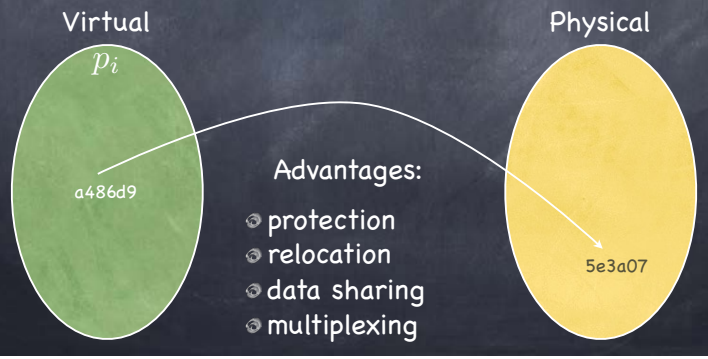
- Virtual address space: set of memory addresses that process can "touch"
  - CPU works with virtual addresses
- Physical address space: set of memory addresses supported by hardware



II. Memory Protection

Step 2: Address Translation

- Implement a function mapping
  $\langle pid, virtual\ address \rangle$ into $physical\ address$

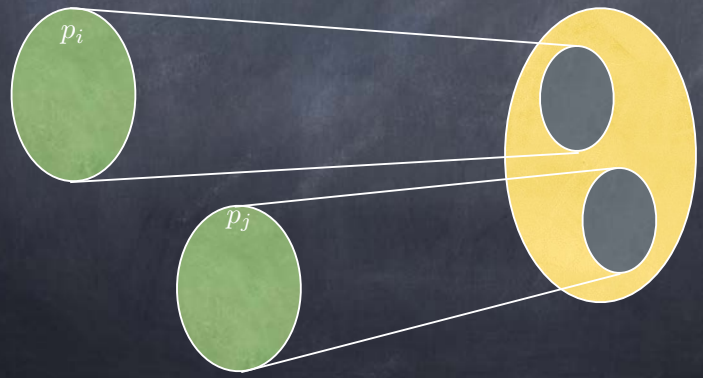


Advantages:

- protection
- relocation
- data sharing
- multiplexing

Isolation

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

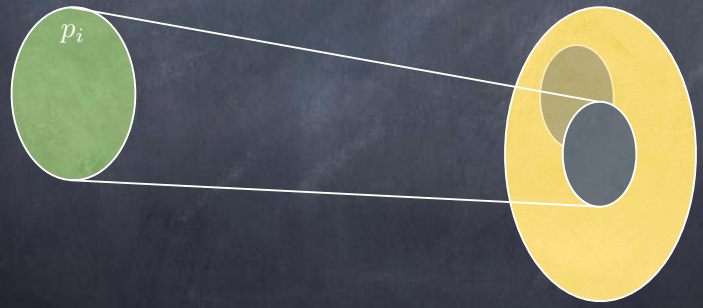


Relocation

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

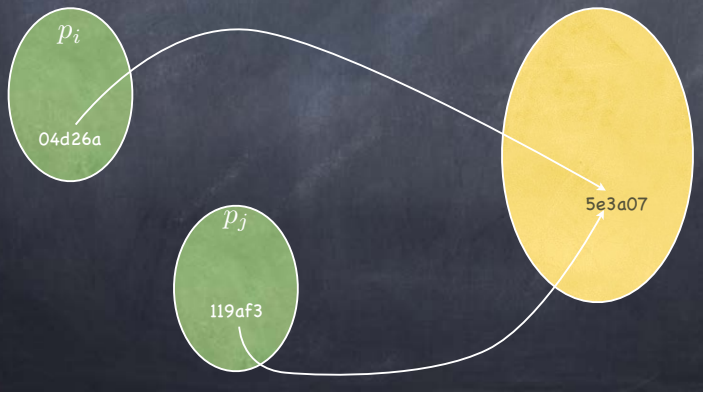
Relocation

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



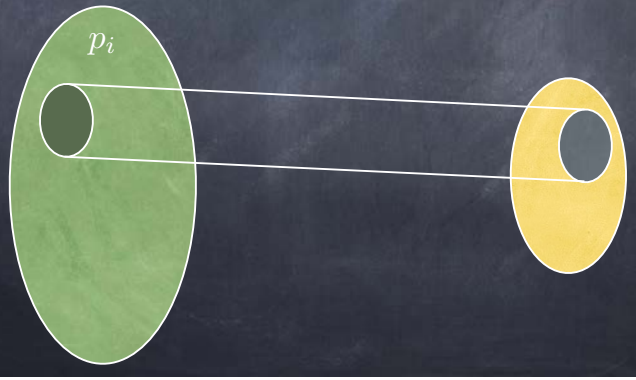
Data Sharing

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



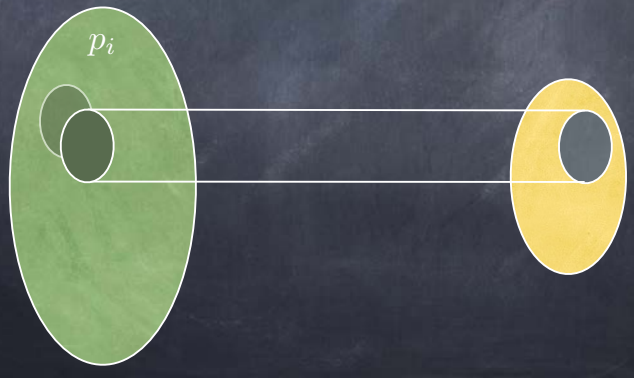
Multiplexing

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



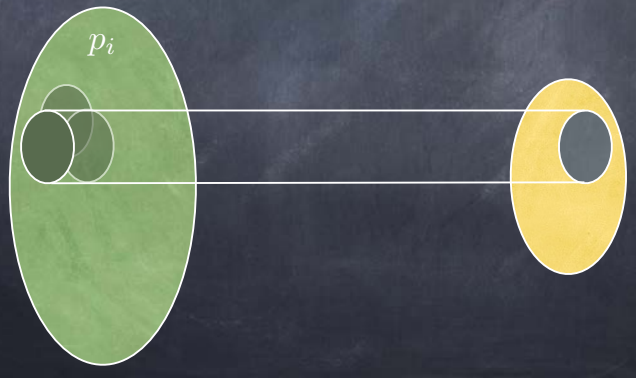
Multiplexing

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



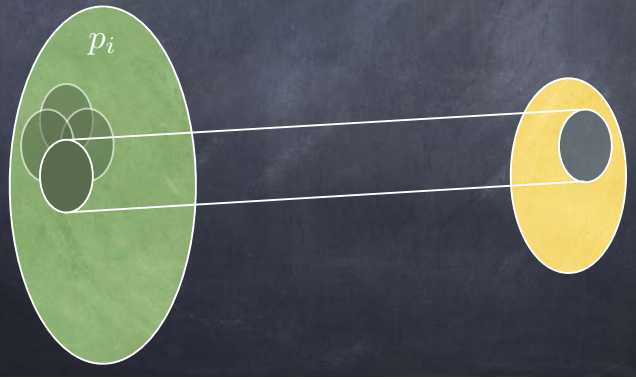
Multiplexing

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



Multiplexing

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



Multiplexing

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