16. Virtual Memory

What methods of protecting memory from corruption by bad processes do we have?

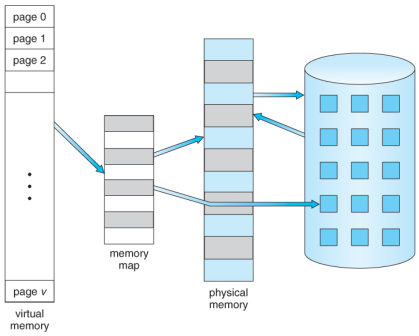
1. Hire better programmers
2. Enable subscript/NULL checking
   a. requires a language that stores array sizes
   b. conditional branches and checking make this very slow
3. Base & Bound Registers
   a. simple: only requires 2 registers
   b. good for batch environments, bad for dynamic
   c. requires contiguous memory per-process
      i. external & internal fragmentation
   d. not conducive to sharing
      i. repetition of code with copying for each process
4. Segmented Memory
   a. a hardware table mapping indices to a (array, index)/(segment#, offset) pair
   b. does not require adjustment of locations
   c. still has external fragmentation

All of these have downsides—a better and the most common one is virtual memory
This is for a few reasons:

1. frees programmers from worrying about physical locations & fragmentation
2. prevents processes from illegal memory access
3. lets processes and threads share memory
4. lets VM be larger than actual memory
   a. while this is technically true, it can cause rapid switching of pages (**thrashing**)

How does it work?

- per thread register %cr3 points to the page table
- page table maps virtual to physical addresses
- Intel keeps the specific contents secret



Pages are supported by the hardware

- x86-64 has a CTRL register that stores page sizes
- How do we choose the size?
  - large pages → small table (saves time & space)
  - small pages → large table (less fragmentation)

Are loops allowed? (virtual→physical→virtual)

- this would allow us to look at the table
- Linux doesn't even let us look at the table to prevent memory cheating

How do we edit the page table then?

```
int mmap(void *vir_addr, size_t len, int prot, int flags, int
fd, off_t offset);
# returns the virtual address of the new page (not necessarily
the one you give)
# flags
  # fixed -- do not give me an address other than what I asked
for
  # private -- other processes are not allowed to see our memo
ry
```

What if the length isn't a multiple of the page size?

~ not allowed; use int getpagesize()

~ this limits pages to INT_MAX, so we use long sysconf()

Are all protections reasonable?

- rw-: ordinary data
- r--: read only data such as variables marked const
- r-x: code (read is given because they are open source)
- rwx: dynamically generated close
  - this allows buffer overflow attacks
  - the stack is often still rwx because of legacy
- ---: useful for catching addressing errors

        - can be used for guard pages on either side of the array
        - can be used to guard a recursive frame from modifying the base pointer in stack
        - IS used on 0 to protect from NULL accesses

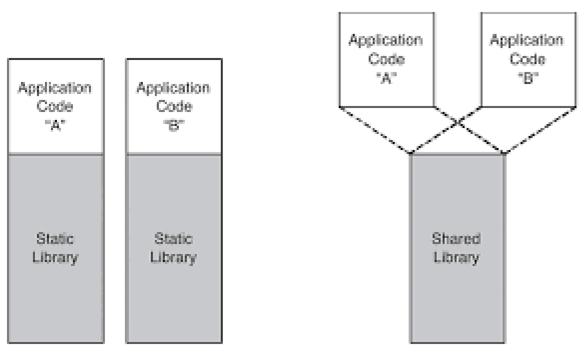
~ x is expensive, so low end chips with no security risk is often combined with r

mmap trivia:

- the most common file to map is /dev/zero
    - this functions as an input only stream of zeroes
        - read ~ succeed & fill page with zeroes
        - write ~ succeed without doing anything
- suppose two processes try to map to the same address at the same
    - this is not allowed!
- if a file has been mapped, it will not be deleted while still mapped
- a mapped page is not intrinsically tied to any file

mmap allows us to use dynamic linking of libraries

```
$ gcc main.c -o executable -lm;
# -lm links to math linbrary
```



Processes share only an image of the code they share, so the virtual addresses do not match

— how do we map virtual addresses to physical addresses?

We use the Global Offset Table

- the GOT is rwx self-modifying code that gives the offset for machine language system calls