COS 417 Precept 5

Traps + Filesystem

Why Traps?

- The assignment interacts with trapframes in 2 ways:
  - Lazy allocation
  - Copy-on-write
- Traps are the middle ground for syscalls
  - Every syscall is triggered by a trap

Further reading: https://github.com/palladian1/xv6-annotated/blob/main/traps.md

Anatomy of a Trap

- Trap is triggered
  - Interrupt
  - Page Fault
  - Syscall
- Hardware redirects trap to IDT
  - Interrupt Descriptor table, specified in vectors.S
- IDT redirects to trap.asm
- Trapasm.asm sets up trapframe
  - Calls trap in trap.c

```
  # vectors.S sends all traps here.
.globl alltraps
alltraps:
  # Build trap frame.
  pushl %ds
  pushl %es
  pushl %fs
  pushl %gs
  pushal

  # Set up data segments.
  movw $(SEG_KDATA<<3), %ax
  movw %ax, %ds
  movw %ax, %es

  # Call trap(tf), where tf=%esp
  pushl %esp
  call trap
  addl $4, %esp
```

Trap Handler

- trap.c handles various cases
  - Syscall? Trap.c calls syscall()
  - Unknown? Trap.c panics/kills process after recording unrecognized input.
  - Page fault? Assignment problem.

Trap Calls

- Important funcs:
  - lcr3(int) flushes the TLB buffer, updates changes to recent memory
  - rcr2() obtains the relevant memory address in a page fault
- Important trapframe vars:
  - tf->cs&3 returns 0 if trap triggered in kernel mode, 1 otherwise.

```
static inline uint
rcr2(void)
{
  uint val;
  asm volatile("movl %%cr2,%0" : "=r" (val));
  return val;
}

static inline void
lcr3(uint val)
{
  asm volatile("movl %0,%%cr3" : : "r" (val));
}
```

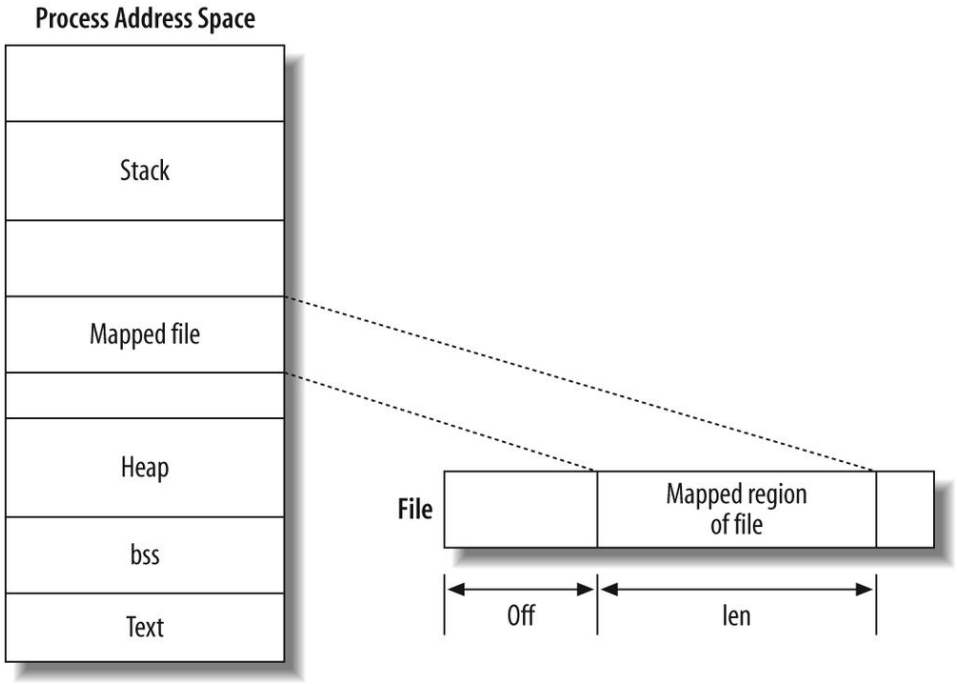
Filesystems!

- One major feature of wmap is capability of writing to file.
  - Need to learn file systems to support this feature
- You will work directly with file systems later in the course.

Purpose of file system

- One major feature of wmap is capability of writing to file.
  - Need to learn file systems to support this feature
- You will work directly with file systems later in the course.

File-based Wmap

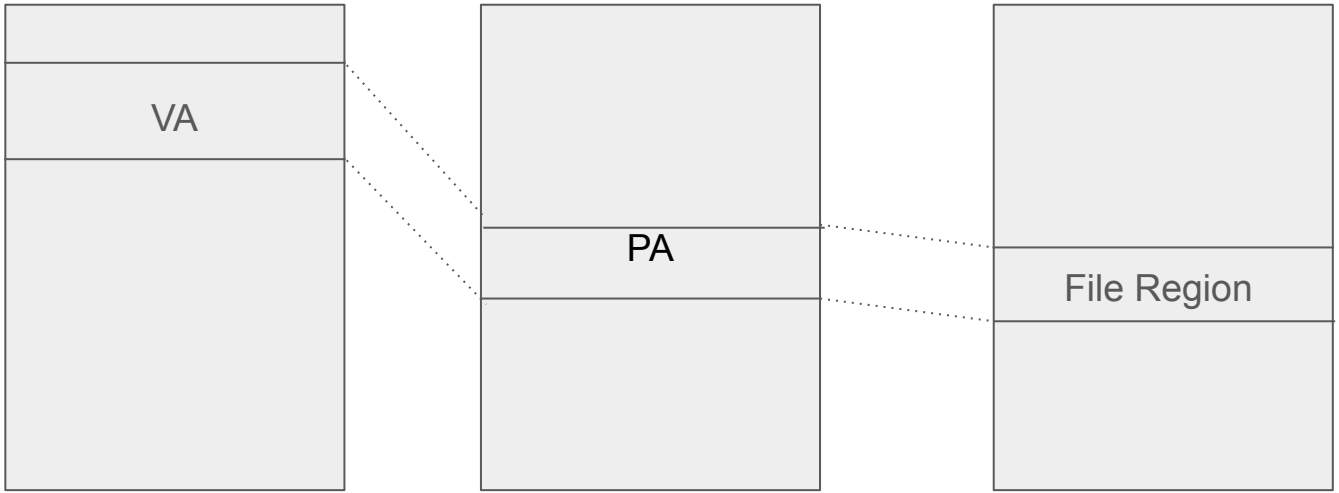


Purpose of file-based wmaps

- Abstraction:
  - Read/Write to file as memory
  - wmap takes care of the rest.
- Performance:
  - Can read from portions large files, without having to seek every time.
  - Multiple processes can access same memory.

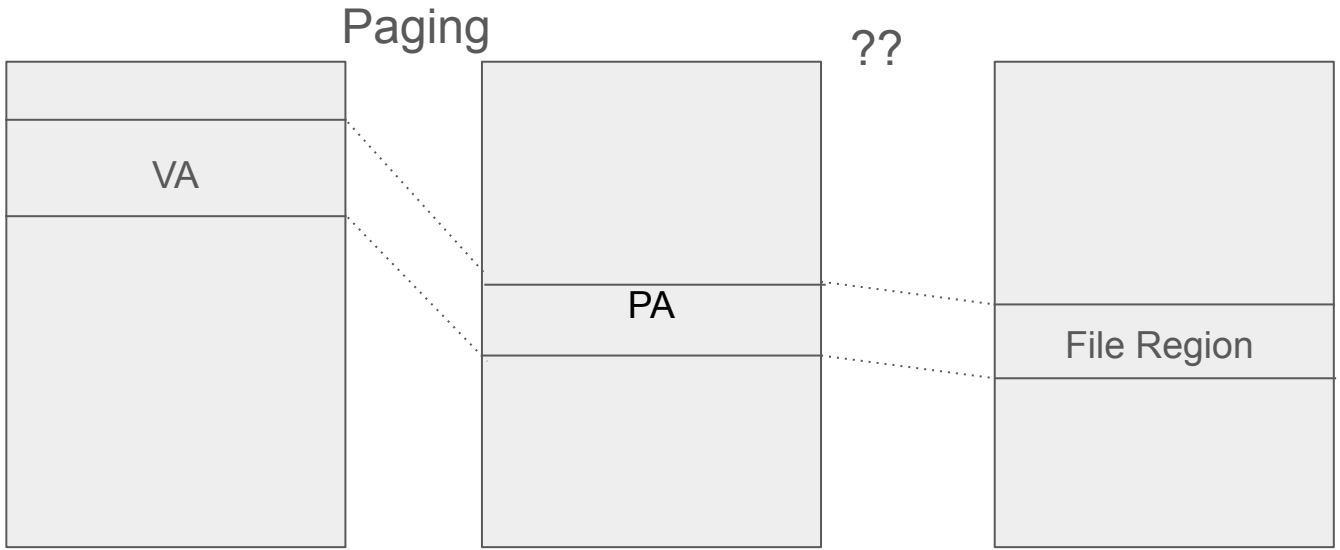
WRT to Assignment

- Implement (limited) version of file-based mapping.

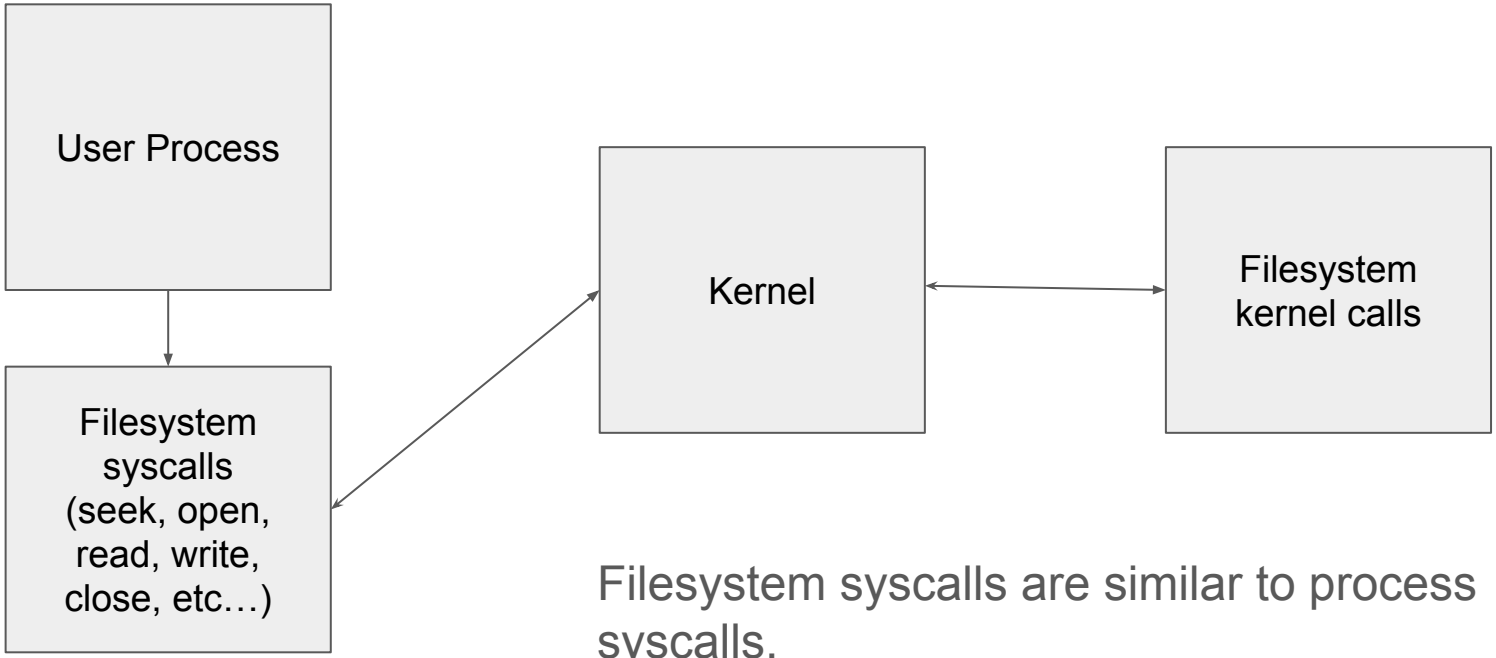


WRT to Assignment

- Implement (limited) version of file-based mapping.



Xv6 Filesystem Intro



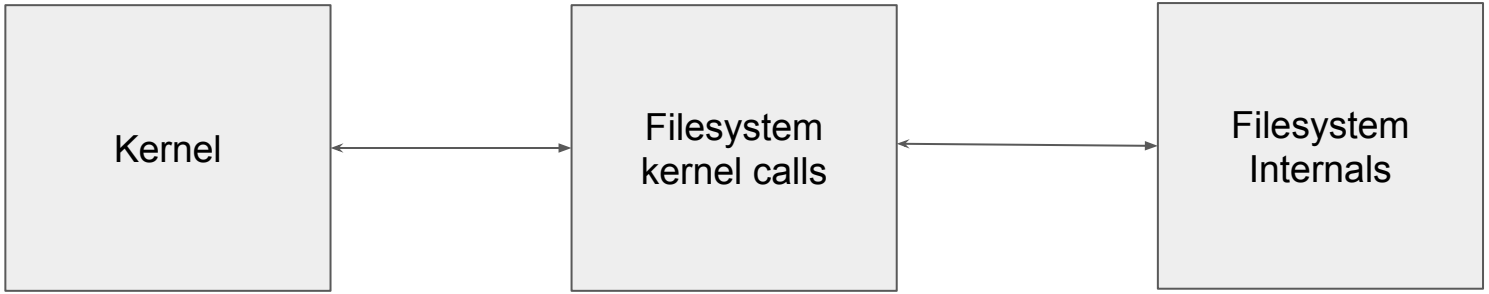
Filesystem syscalls are similar to process syscalls.

Xv6 Filesystem API

Just like process syscalls, filesystem syscalls are wrappers to enter into kernel mode.

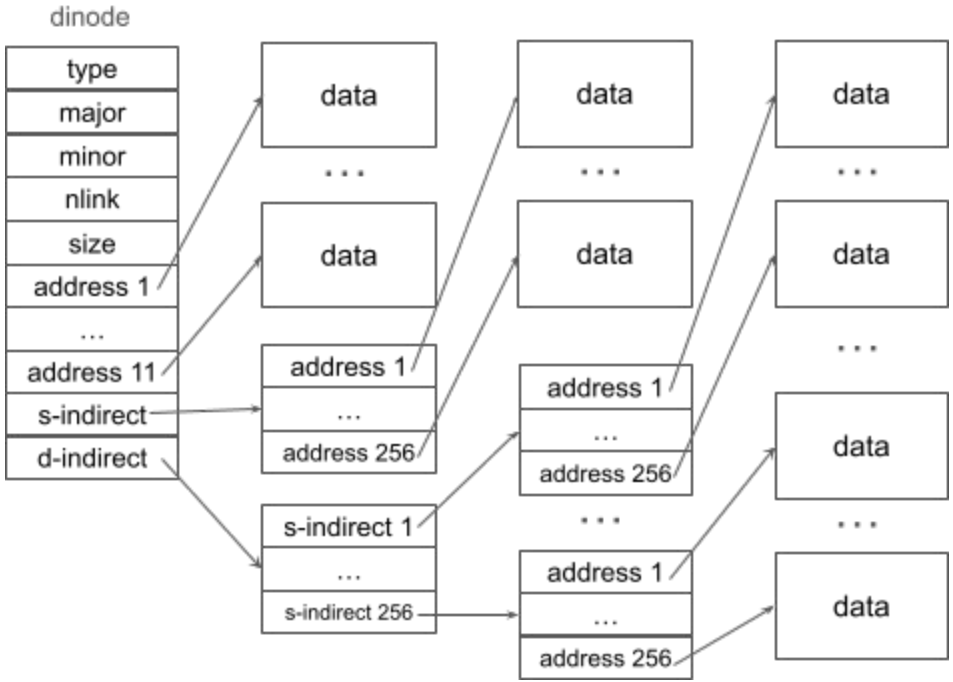
- syscall() redirects the syscall into arch.
- For each user function, there exists an equivalent kernel function.
  - Read vs sysread,
  - Write vs syswrite
  - Etc.

Xv6 Filesystem API



Calls abstract away from
filesystem internals

Xv6 Filesystem Internals



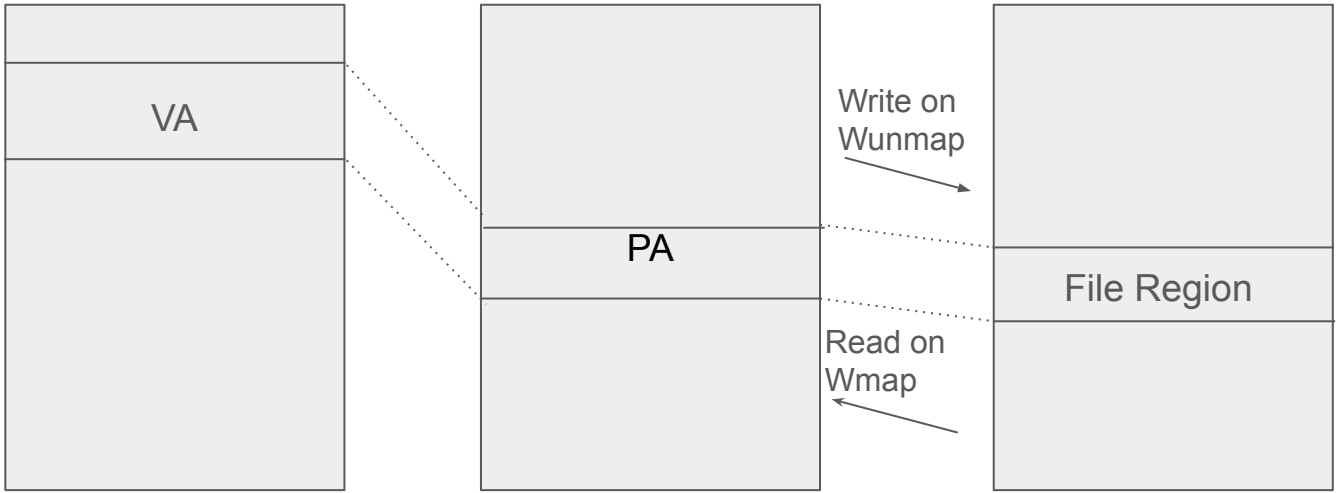
No need to understand this. It’s to show a peek behind the hood.

Xv6 Filesystem Demo

Time to look at the code directly.

File Wmap in practice

- Implement (limited) version of file-based mapping.



Summary

- OS handles exceptions/page faults/syscalls at the trap level
- File-based wmap is useful for both performance and convenience reasons.
- For assignment purposes, file-based wmap reads from files at wmap, and writes to files at wunmap.
    - Implications to killing processes.