6.508i: VM for user applications

OS kernel use VM in creative ways

Paper argues user apps can use VM too

- Garbage Collector
- Data compression
- SVM
What primitives?

Trap → alarm handler

Prot → decrease accessibility
R-W → R → not access

ProtN → save TLB flushes

Unprot → increase accessibility

? Dirty

? Map2 → several mmaps
Unix today: `mmap`

map a file

```c
addr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, offset)
```

Next lab

map anonymous
Unix today

mprotect (addr, len, R) → lots

sigalrm

unmap: remove address range

sigaction: signal handler (f)

signal: segfault
VM implementation

AS: page table + Virtual Memory Area (VMA's)

Contiguous range of addresses
Same permission
backed by same object

1000-2000
2100-2200
User-level traps

- PTE marked invalid / RD
  - CPU jumps into kernel
    - Kernel saves state
    - VM asks the VM system what to do?
      - Upcall into user space
        - Run handler -> improper?
        - Handler returns to kernel
        - Kernel resumes interrupted process

VMNs
Example: Huge memorization table

\( f(0) \)
Challenge: Table might be big

Solution:

- Use VM permissions
  - Allocate huge range
  - \( \text{allocate page } \) \[ \text{table}[i] \rightarrow \text{default} \rightarrow f(i) \rightarrow \text{table} \]
- If much memory is in use, free some pages

\( \text{probe1/probe2} \)
Example: garbage collector

A copying garbage collector

(forwarding)

discarded → is now free space
Baker's: real-time incremental GC

**NEW:** forward a few more objects
- Check if in from space
  - forwarding
Use VM:

- Use fault handler

- Scan one page of objects
- Forward from unprotected
- Incremental + the scanned page
- No pointer check (the VM does it for us)
Should we use VM?

Most cases could have been implemented without extra in-depth compilation. Unix supports them.
What has changed 1991?

Many!

Continuous development

Some big changes:

5-levels
ASID
KPTI