6.181: Q&A Labs (PGTBL)

Adam Belay <abelay@mit.edu>

MIT CSAIL

Agenda

1. Review **page table** lab assignment
2. Examples of how real OSes use the features you implemented
3. Answer your questions

Page table lab

- Traditionally a difficult lab
- Debugging can be challenging
    - Bugs in page tables can change code and data layout
- Focus is on features enabled by page tables

Part 1: USYSCALL

- Problem: Kernel transitions have high overheads
- Could we speed up some system calls through shared memory between process and kernel
- Which system calls can be sped up?
  - Must have no side-effects
  - Returns constant value while process runs
  - But value can change after entering kernel (e.g., ticks)

Q: Which system calls in xv6?

Q: Which system calls in xv6?

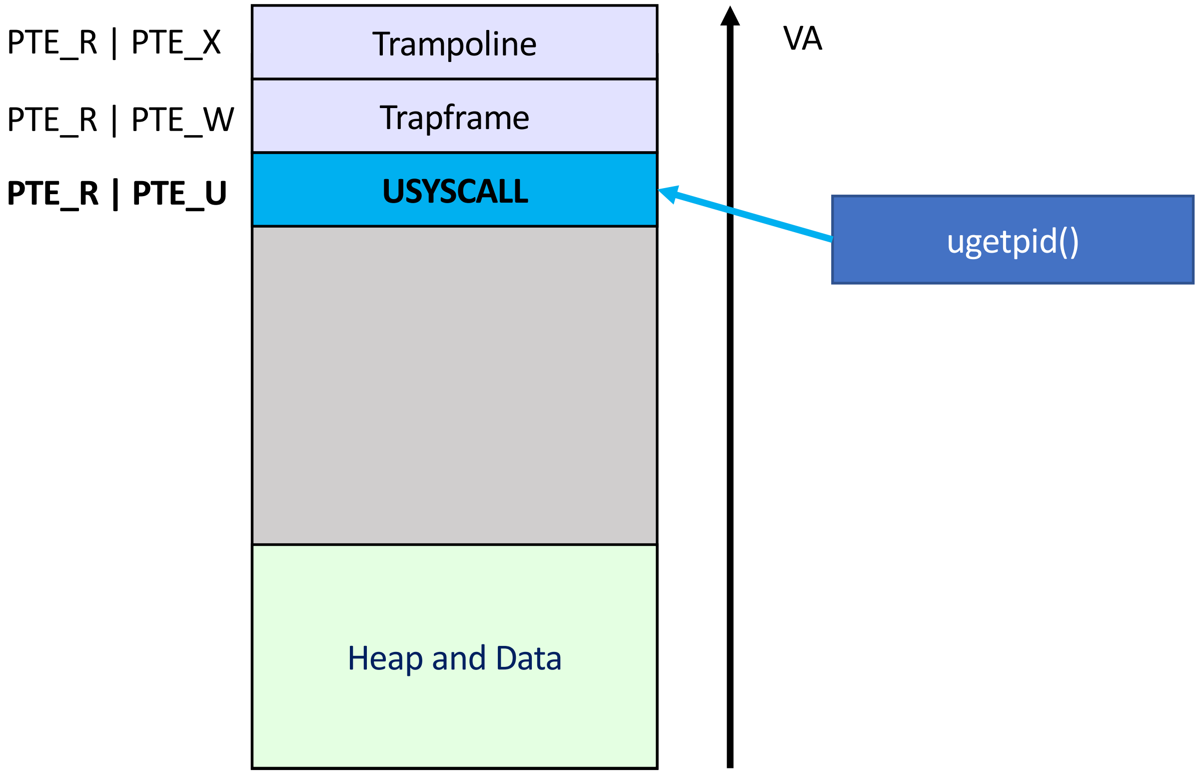
Best options:

- getpid() – constant value, doesn’t ever change
- uptime() – constant until next timer tick
  - Each tick triggers a kernel interrupt, which updates the value

Less likely:

- File system calls if willing to map a lot of state in memory

USYSCALL Mapping



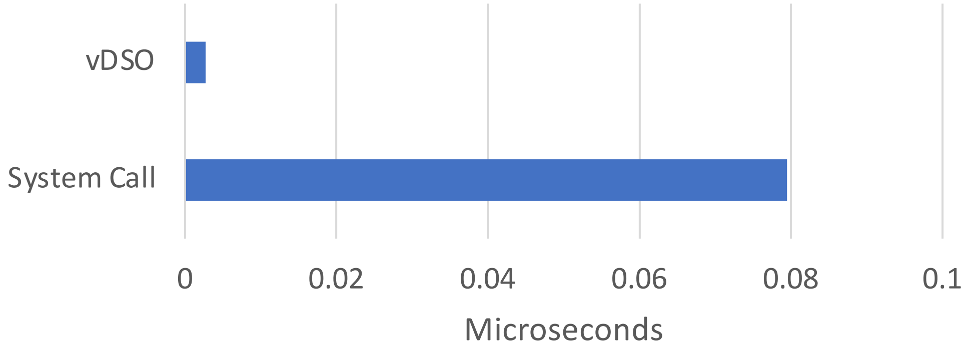
Code walkthrough

How does Linux use USYSCALL?

- A more sophisticated mechanism called **VDSO**
- Idea #1: Read-only, shared memory region
  - Like the lab assignment
- Idea #2: Kernel puts data <u>and</u> code in shared region
  - Code interprets the data in the shared region
  - Allows kernel to change data format over time

Linux VDSO methods + speed

- clock_gettime()
- getcpu()
- getpid()
- gettimeofday()



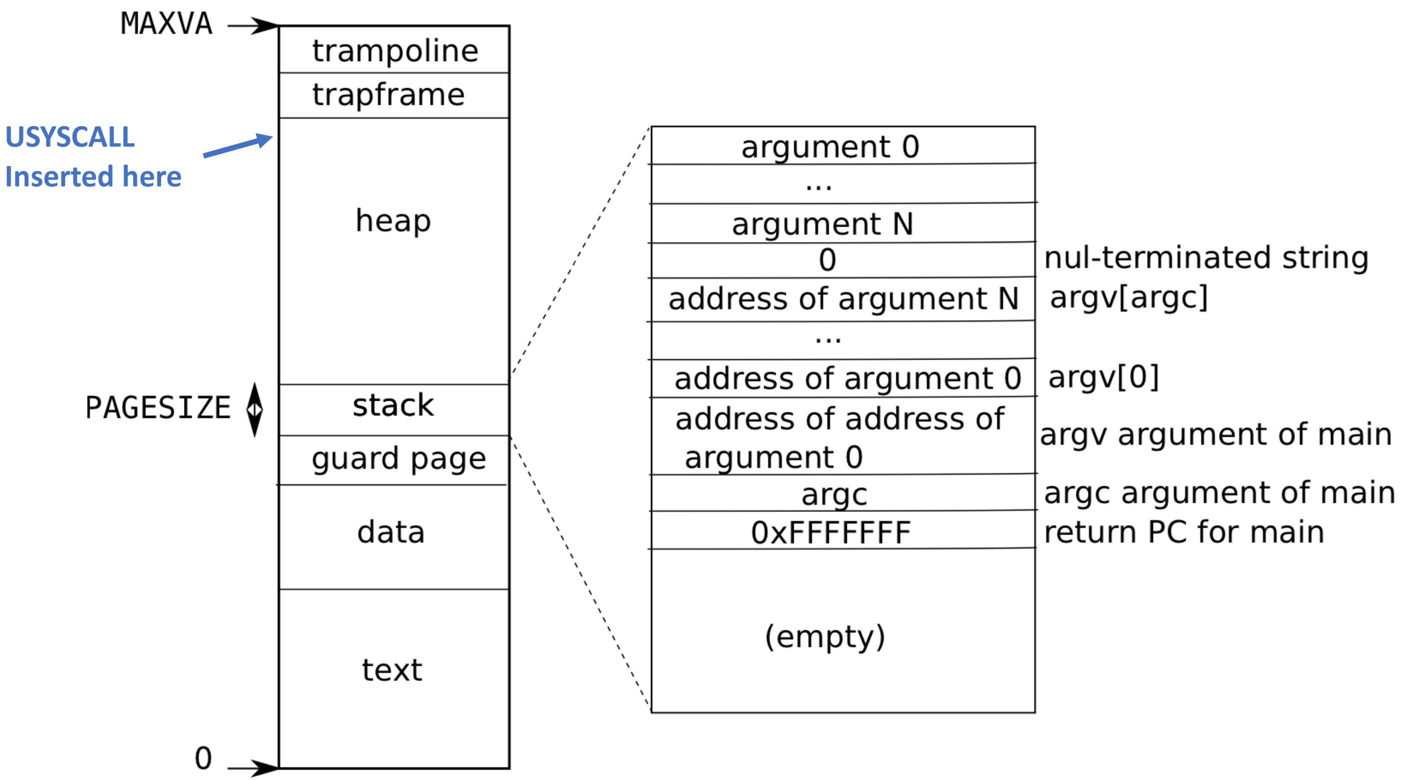
Example use case: clock_gettime()

- 1: Kernel posts time to shared region each time process is entered
- 2: VDSO code adds TSC to posted time

Part 2: Printing a page table

- Goal: Print the contents of the user page table
- Save your code! Useful for debugging future labs

Recall user address layout (fig 3.4)



User page table output

LVL0 LVL1 PTE

```
page table 0x0000000087f6b000
..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. ..0: pte 0x0000000021fd9801 pa 0x0000000087f66000
.. .. ..0: pte 0x0000000021fda01b pa 0x0000000087f68000   Code
.. .. ..1: pte 0x0000000021fd9417 pa 0x0000000087f65000   Data
.. .. ..2: pte 0x0000000021fd9007 pa 0x0000000087f64000   Guard page
.. .. ..3: pte 0x0000000021fd8c17 pa 0x0000000087f63000   Stack
..255: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. ..511: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. ..509: pte 0x0000000021fdcc13 pa 0x0000000087f73000 USYSCALL
.. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000 TRAPFRAME
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000 TRAMPOLINE
```

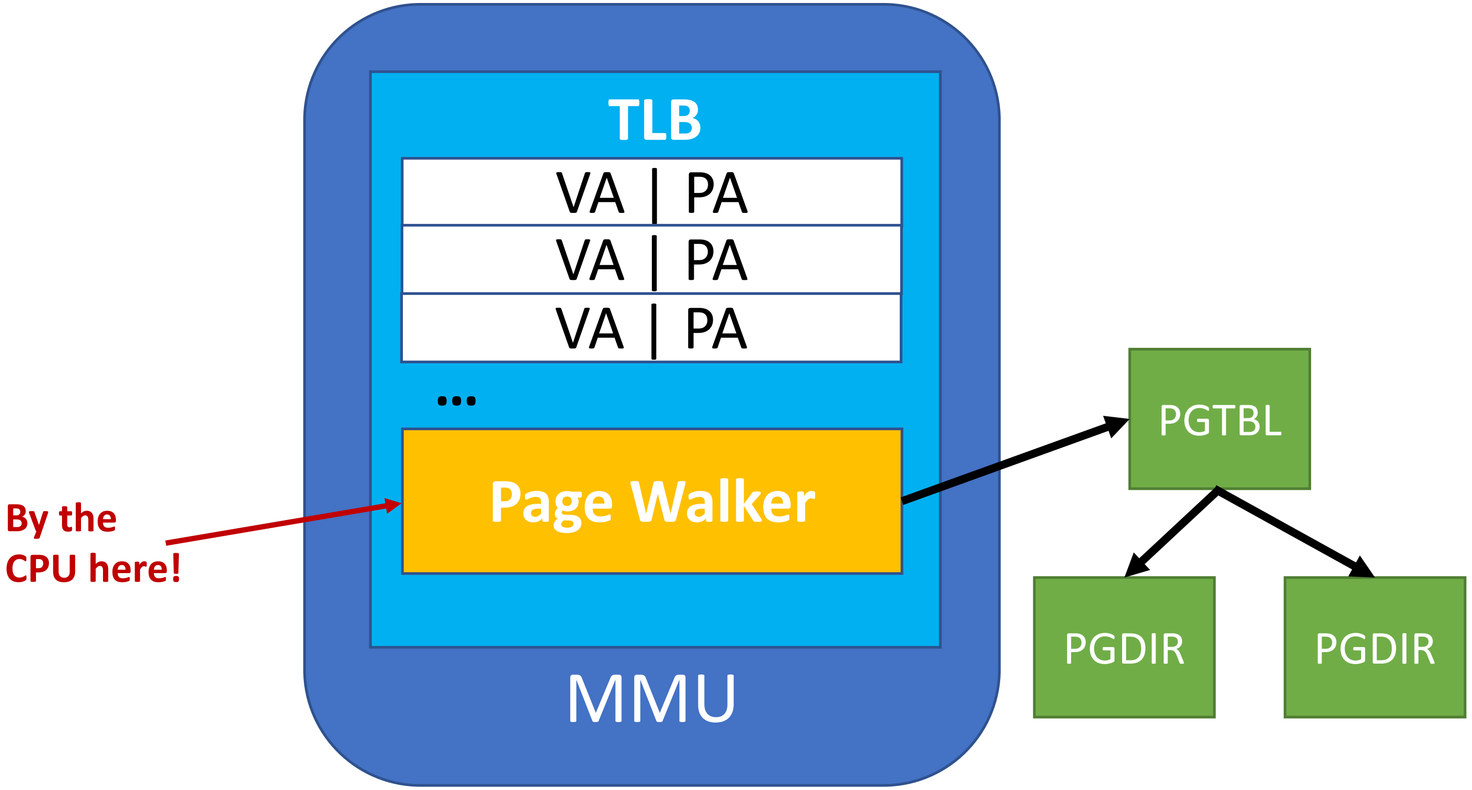
Permission bits

Code walkthrough

Part 3: Access bits

- Goal: Efficiently tell which pages were accessed
- Hardware page walker accelerates this:
  - PTE_A: Was the page accessed (read or write)
  - PTE_D: Is the page dirty (only write)
  - HW marks these bits when walking page table
- This lab: Provide a bitmask indicating which pages were accessed

How is PTE_A set?



Code walkthrough

How does Linux use access bits?

- Used for swapping pages to disk
- CLOCK algorithm: Scan pages, which were accessed (PTE_A marked) since last interval?
- Least accessed pages moved to disk
- PTE_D used to detect if copy on disk is stale
- Linux does not expose this info to userspace!

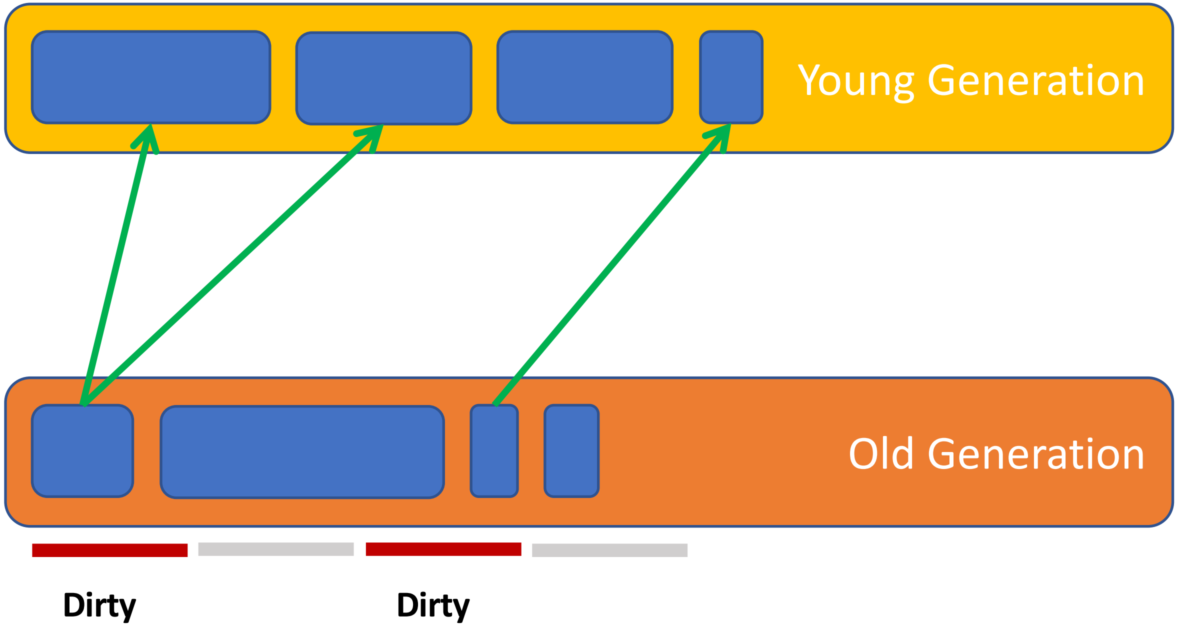
Q: How could you detect page accesses without access bits?

Q: How could you detect page access without access bits?

- Use page faults!
- Clear PTE_V, wait for faults
- In fault handler, record fault, then set PTE_V
- Slow!

Example use case: Garbage collector

- Paging HW tracks which pages were modified (DIRTY)



Q: How does the kernel start running C code?

Q: How important is it to gracefully handle incorrect arguments?

- In general, extremely critical
- Isolation and security often depends on it
- This is one reason OSes are often insecure

Q: Why do kernels copy arguments?

Q: Why do kernels copy arguments?

- In xv6, to verify if the mapping exists and if the permissions are right
- In Linux, there is more concurrency than xv6
- What if an argument is modified after the kernel reads it?
- This is called a time-of-check time-of-use (TOCTOU) attack
  - A serious security problem!
  - Copying prevents the attack

Q: Why is free memory a linked list?

- This is one common strategy for tracking free memory
- Advantage: No extra memory needed for metadata
  - Metadata stored inside free memory
- Advantage: Simple
- Disadvantage: Hard for CPU to prefetch
- Disadvantage: Hard to allocate very large chunks

Q: Can you explain what qemu does?

Q: How does myproc() work?

```
// Return this CPU's cpu struct.
// Interrupts must be disabled.
struct cpu* mycpu(void)
{
  int id = cpuid();
  struct cpu *c = &cpus[id];
  return c;
}
```

```
// Return the current struct proc *, or zero if none.
struct proc* myproc(void)
{
  push_off();
  struct cpu *c = mycpu();
  struct proc *p = c->proc;
  pop_off();
  return p;
}
```

Q: How can I access physical memory?

1. Turn off paging

2. Map physical memory into virtual memory

Q: Which one does xv6 do?

Other questions?