6.S081: Q&A Labs

Adam Belay
abelay@mit.edu
Agenda

• Review lab assignments
• Main focus: Page table lab
Page table lab

• Traditionally a difficult lab
• Debugging can be challenging
  • Bugs in page tables can change code and data layout
• New version this year focuses more on features enabled by page tables, less on xv6 VM layout
Part 1: USYSCALL

• Problem: Kernel transitions have 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?

• Getpid() – constant value, doesn’t change
• Uptime() – constant until next tick
  • Each tick triggers a kernel interrupt, can update value

• Fstat() – maybe possible, not likely worth it, too much state
USYSCALL Mapping

- PTE_R | PTE_X: Trampoline
- PTE_R | PTE_W: Trapframe
- PTE_R | PTE_U: USYSCALL

Heap and Data

ugetpid()
Code walkthrough
How does Linux use USYSCALL?

• A more sophisticated mechanism called **VDSO**
• Idea: Read-only, shared memory region
  • Exactly the same as the lab
• Idea #2: Kernel ships code into user program
  • Code interprets the data in the shared region

Powerful: makes time measurement more efficient
• 1: Kernel posts time to shared region on user enter
• 2: VDSO code adds TSC to latest time
Linux VDSO methods

- clock_gettime()
- getcpu()
- getpid()
- getppid()
- gettimeofday()
- set_tid_address()
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

**Permission bits**

<table>
<thead>
<tr>
<th>LVLO</th>
<th>LVL1</th>
<th>PTE</th>
<th>Code + data</th>
<th>Guard page</th>
<th>Stack</th>
<th>USYSCALL</th>
<th>TRAPFRAME</th>
<th>TRAMPOLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000087f6e000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fda801 pa 0x0000000087f6a000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fda401 pa 0x0000000087f69000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fdac1f pa 0x0000000087f6b000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fda00f pa 0x0000000087f68000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fd9c1f pa 0x0000000087f67000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fdb401 pa 0x0000000087f6d000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fdb001 pa 0x0000000087f6c000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USYSCALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fdd813 pa 0x0000000087f76000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TRAPFRAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000021fddc07 pa 0x0000000087f77000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TRAMPOLINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0000000020001c0b pa 0x0000000080007000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Code walkthrough
Part 3: Access bits

• Goal: Efficiently tell userspace 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

• In this lab, provide a bitmask indicating which pages were accessed (PTE_A)
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!
Use Case: Generational GC

• Observation: Most objects die young
• Idea: Maintain separate regions for young and old objects
• Plan: Collect young objects independently and more often
• Performance impact: Avoids tracing overhead of old generation
Generational GC

Young Generation

Promotion

Old Generation
Challenge: How to find live objects in young gen?

• Easy part: Start with roots like registers, stack, and global pointers

• Hard part: What if an old gen object points to a young gen object?
  • We can’t trace the old gen or no speedup!
Challenge: How to quickly find live objects in young gen?

• Old gen may have references to young gen!
Solution: Use virtual memory!

- Paging HW tracks which pages were modified (DIRTY)
Other questions?