CSE 451: Operating Systems
Hard Lessons Learned

Windows
RtlZeroMemory

Gary Kimura

Zero Memory

- What can be simpler?
  - Zero a register and do a lot of stores.
- Make is faster by picking a large register.
  - Floating-point registers are pretty big
- Same optimization can be used in Copy Memory.

Make interrupt handling fast

- Save only those registers needed by the device drivers.
- What device driver in their right mind would do any floating point arithmetic?

My Sad Story

- Everyone in the Windows team ran nightly stress tests of each new build.
- A piece of the file system started bug checking every night on multiple test machines.
- A Showstopping bug was assigned to me.
- Examination of the code didn't reveal any obvious problems. It was code that was working fine for a long time.
- Finally in desperation I added an assert that after calling RtlZeroMemory() checked that the memory was indeed all zeros.
- My check caught a lot of machines that night…

Now the fun begins

- My boss's boss had optimized interrupt handling to not save the floating-point registers.
  - Because no one needs to do floating point arithmetic in an interrupt handler…
- RtlCopyMemory and RtlZeroMemory had also been optimized to use the larger floating-point registers.
  - Because is requires fewer instructions…
- Another software engineer started calling RtlCopyMemory in an interrupt handler.
  - Just because…

Fateful Sequence of Events

- I call RtlZeroMemory from the File System (in Kernel mode but not in an interrupt handler)
- While RtlZeroMemory is zeroing out memory an interrupt occurs
- The interrupt device handler calls RtlCopyMemory
- When control returns to me the floating-point register is no longer zero, but contains what was used in RtlCopyMemory
- RtlZeroMemory continues doing stores, but now with a nonzero floating-point register. How did this happen?
- Someone had to tell my boss's boss that his optimization didn't work...

Moral of the Story

- Many seemingly good optimizations have unforeseen consequences.
- OS development work is full of such examples. Where modifying one piece of code can have unforeseen consequences in unrelated modules.
- While I was just the innocent victim of the bug. I was also tasked with chasing it down.

About Alignment

- With respect to physical memory there is natural data alignment (char, short, long, longlong)
- How does the hardware handle aligned and unaligned loads and stores

- These details are usual hidden from application writers, because most compliers and linkers will naturally align the data
- But some generic functions e.g., Zero and Copy Memory might stumble upon this.
- Let’s see what we can do to make life easier and more efficient…

Now comes the tradeoffs

- Let’s consider Zero and Copy memory

- One option, let the hardware handle it all.
- Another option, force the user to only make “well” aligned requests.
- Yet another option, let the hardware handle it all and educate the user that using aligned buffers increases performance.
- And yet another option, we determine the alignment of the buffer and special case how to handle it.
- The last option, tell the user to zero (copy) their own buffer