Virtual Memory and Caches

ECE598: Advanced Operating Systems – Homework 7

Spring 2018

Due: Thursday, 29 March 2018, 2:00pm

This homework involves virtual memory and caches.
You may work on this homework with a group.

1. **Download the homework code template**

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece598/ece598_hw7_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece598_hw7_code.tar.gz`

2. **Investigate virtual memory support**

   - The provided code implements some very simple ARMv7 virtual memory support. It sets up a large, 1:1 "sections" mapping, so the virtual and physical addresses are the same, with 1MB page granularity (4096 page table entries)
   - Look at the code in `./kernel/memory/armv7-mmu.c` for all of the details. It's a lot of low-level setting up of ARM special registers.
   - The code as provided does not provide memory protection support. Try running the provided command `cause_error kernelread`
     which will try to read operating system memory. Then try running `cause_error kernelwrite` which will try to overwrite your operating system with garbage. Note what happens.
   - Now, edit the file `./kernel/memory/memory.c` and uncomment `enable_mmu()` line that is commented out. Rebuilt, reboot. At bootup you should get some extra boot messages about the MMU.
   - Now try running `cause_error kernelwrite` again. Also try `cause_error kernelread` and `cause_error kernelexec`

3. **Investigate cache performance (5.5pt)**

   Caches are small, fast, memories provided on modern CPUs that "cache" the much larger and slower DRAM. By default these are turned off.

   (a) Try running the `memory_test` executable. This will take at least 15s so don't give up on it.
   This runs `memset()` on large chunks of memory. Byte-at-a-time does a simple byte-by-byte setting of memory, while the 32-bit version tries to write out one 32-bit word at a time.
   Note in the README the performance of each.

   (b) Now, we will enable the caches.
   Edit `kernel/memory/armv7-mmu.c`, find the "Enable caches" line, and change the ifdef from a 0 to a 1. Then recompile and reboot.
   I haven't been able to get this to work reliably on the Pi3 :(

Your Pi will likely crash after a few commands, so after this change be sure to run `memory_test` right away after booting.

Did the cache increase the speed of the memset? Note in the README the performance of each.

4. **Something cool (0.5pts)**

   (a) Our `memset()` routine can be improved. Can you write a better routine that runs faster?

   Report what method you chose/designed/found and its performance in MB/s.

   You can run the test with caches disabled to avoid the crashing issues (but be sure to report if your results are with/without cache enabled).

   Edit the `memory_test.c` file under the userspace directory. Fill in the `memset_custom()` routine.

   To rebuild the userspace disk image you will need to do "make" then "make image" in the userspace directory, followed by a "make" in the kernel directory.

5. **Questions (4pts)**

   In addition to the data requested above, answer these questions in the README file.

   (a) Name one feature found in Linux/UNIX that is made possible (or easier) because of virtual memory.

   (b) What important piece of logic on a modern processor hides a lot of the overhead of page table lookups?

   (c) What is it called when a memory access happens, the CPU walks the page tables, but the virtual memory page requested is not found in the page table? Whose job is it to handle this now that the CPU has given up?

   (d) The operating system has found the page requested by the CPU, but it tries to allocate space for it in physical memory but physical memory is full. How can more room be freed up in physical memory?

6. **Submit your work**

   - Run `make submit` in your code directory and it should make a file called `hw7_submit.tar.gz`. E-mail that file to me as well as the document with the answers to the questions.