Filesystems

ECE598: Advanced Operating Systems – Homework 10

Spring 2018

Due: Friday, 27 April 2018, 2:00pm

This homework involves multi-core concerns.

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_hw10_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece598_hw10_code.tar.gz`
2. **Investigate multicore support**
   - This adds the beginning of multi-core support. I was sadly unable to get full multi-core going for this homework.
   - First was to add support for locks, found under
     `kernel/lib/locks.armv7.s`
   - Next was to add code to bring up the secondary cores, this can be found in
     `kernel/boot/smp_boot.c` and `kernel/boot/start_smp-armv7.s`
   - Support was added for sending IPI (inter-processor interrupts) so that core0 can trigger an interrupt on the other cores. On pi2/pi3 this is done using the mailbox interface, as described in the `QA7_rev3.4.pdf` file. This allows the central timer tick to be used to indicate a re-schedule on all cores. The code can be found in:
     `kernel/interrupts/ipi.armv7.c`
   - The entire kernel must be audited to be sure we lock data structures that have critical sections. The most pressing would be memory allocation, console printing, and the scheduler.
   - The final step would be to update the current_process pointer to be per-core, and then update the scheduler to have per-core scheduling queues. Then invoke the scheduler on each core, and they will be up and participating in the running of the OS.
3. **Test out the multicore support (5pts)**
   (a) Boot the kernel. Try using the `core_poke` command to poke one of the other cores (it takes the core# as a command line argument). How does the other core respond?
   (b) Run the printa command in the background `printa &`. Quickly, run `core_poke 2`. Does core2 respond?

(c) You can edit `kernel/drivers/console/console_io.c` and disable the locking and rerun the printa/poke test.

What behavior might you expect in an example where two cores are writing to the serial console without locking?

On our OS though that doesn't happen. Why is that?

(Hint, the `printf()` from userspace goes through a different path than a `printk()` from an interrupt handler).

4. **Questions (5pts)**

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

(a) Look at the following memory allocate code from our operating system.

i. At which points (A, B, C, D, E) would you put a lock instruction? Why?

ii. At which points (A, B, C, D, E) would you put an unlock instruction? Why?

```
void *memory_allocate(uint32_t size) {

        int first_chunk,num_chunks,i;
// POINT A
        if (size==0) size=1;
        num_chunks = ((size-1)/CHUNK_SIZE)+1;
// POINT B
        first_chunk=find_free(num_chunks);
        if (first_chunk<0) {
                printk("Error!\n");
// POINT C
                return NULL;
        }
        for(i=0;i<num_chunks;i++) memory_mark_used(first_chunk+i);
// POINT D
        memset((void *)(first_chunk*CHUNK_SIZE),0,num_chunks*CHUNK_SIZE);
// POINT E
        return (void *)(first_chunk*CHUNK_SIZE);

}
```

(b) Name one inter-process communication (IPC) method found in the Linux kernel, and what syscall is involved in using it.

(c) Name one type of security vulnerability that can affect an operating system, and one way to mitigate this problem.

5. **Submit your work**

- Run `make submit` in your code directory and it should make a file called `hw10_submit.tar.gz`. E-mail that file to me as well as the document with the answers to the questions.