Filesystems

ECE598: Advanced Operating Systems – Homework 10 Solution

Spring 2018

1. **Download the homework code template**

2. **Investigate multicore support**

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?
   Other core should print a message when poked.

   (b) Run the printa command in the background `printa &`. Quickly, run `core_poke 2`. Does core2 respond?
   Yes.

   (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?

   Some sort of race condition with text interleaving could/should happen.

   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).

   printf() uses a write() system call. The system call handler runs in SVC mode with interrupts disabled which prevents things from conflicting on core0.

   Since we are sending the IPI from the shell on core0, and printa is running on core0, printa and the IPI being sent will never happen at the same time. It might still be possible to race if the IPI takes long enough but it would be unlikely.

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?
      Lock at Point B. Need to protect the find free routine or there can be a race where two threads both find the same find block. The earlier code can run per-thread without conflict

      ii. At which points (A, B, C, D, E) would you put an unlock instruction? Why? Unlock at Point D. At this point the blocks we have found are marked reserved so other threads cannot accidentally take them.
      Also need to Unlock at Point C, otherwise on error the function is exited without clearing the lock which can cause a deadlock

```
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.
This can be a large number of things:

- FIFO – mknod()
- pipes – pipe()
- Message Queues – msgset()
- Shared memory – shmget()
- Locks – semget()
- Sockets – socket()
- Netlink Socket – socket()
- Inotify – inotify()
- signals – signal(),sigaction()
- file I/O – open()/read()/write()

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

- buffer overflow – better bounds checking
- brute-force guessing of passwods – hide the password hashes from users
- shellcode/privledge escalation – KASLR (address space randomization), marking pages as non-executable
- setuid binary not properly dropping permissions – code auditing
- hardware issues (rowhammer, meltdown) – buy better hardware?

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.