Lab 7, Operating Systems

Jay R Bolton

November 30, 2011

1 Questions

1. **Which register number is used for the stack pointer (sp) in OS/161?**: $29
2. **What bus/busses does OS/161 support?**: LAMEbus
3. **What is the difference between splhigh and spl0?**: `splhigh` sets spl to the highest value, disabling all interrupts. `spl0` sets spl to 0 and enables all interrupts.
4. **Why do we use typedefs like u_int32_t instead of simply saying "int"?**: My ideas: u_int32_t is slightly shorter than "unsigned int." It is also more descriptive of the size of that type (exactly 32 bits).
5. **What does splx return?**: it assigns a new spl value and then returns the old one.
6. **What is the highest interrupt level?**: SPL_HIGH
7. **How frequently are hardclock interrupts generated?**: 100 times every second.
8. **What functions comprise the standard interface to a VFS device?**: these functions are in vfs.h under low-level and mid-level operations.
9. **How many characters are allowed in a volume name?**: 32
10. **How many direct blocks does an SFS file have?**: 15
11. **What is the standard interface to a file system (i.e., what functions must you implement to implement a new file system)?**: These funcs are listed in fs.h
12. **What function puts a thread to sleep?**: `thread_sleep`
13. **How large are OS/161 pids?**: 32 bits

14. **What operations can you do on a vnode?**: all the operations are listed in vnode.h

15. **What is the maximum path length in OS/161?**: 1024

16. **What is the system call number for a reboot?**: 8

17. **Where is STDIN_FILENO defined?**: `/kern/unistd.h`

18. **Is it OK to initialize the thread system before the scheduler? Why (not)?)**: No, the scheduler provides the curthread (not sure on this one).

19. **What is a zombie?** Zombies are threads/processes that have exited but have not been fully deleted yet.

20. **How large is the initial run queue?**: 32

21. **What does a device name in OS/161 look like?**: `lhd0, emu0, somevolume, null, etc`

22. **What does a raw device name in OS/161 look like?**: `lhd0raw`

23. **What lock protects the vnode reference count?**: `lock_acquire(vn->vn_countlock)`

24. **What device types are currently supported?**: "block device" and "character device." Not sure about this one.

2 My Procedure Used to Create a New System Call

1. Created the file `simple_syscall.c` in `/src/kern/userprog/`.

2. Modified kern/arch/mips/syscall.c to have a case for sys_helloworld(). Copied case body from the reboot case.

3. Created kern/userprog/simple_syscall.c with a function called sy _hellworld() that takes no arguments and simply does a kprintf.

4. Included simple_syscalls.c into the build by modifying conf/conf.kern to include the file

5. Modified lib/libc/syscalls.S to append SYSCALL(helloworld, 32) to the end

6. Added "int sys_helloworld();" to kern/include/syscall.h

7. Modified kern/include/kern/callno.h to have "#define SYS_helloworld 32"

8. Modified include/unistd.h to have our prototype without the sys_ (int helloworld();)

9. Finally, I created a test for helloworld in testbin. When I ran it, it worked but looped endlessly.
10. I then added the _exit call by defining sys__exit in userprog, which calls thread_exit and then following the same procedure as above (modifying syscalls.c).
11. With exit implemented the test call to helloworld did not loop, and I took that to mean that it worked.