# Assignment 0

## Code Walkthrough Questions

A page for students to agree on answers to the questions posed in the code walkthrough. Perhaps those of us who are bored or scared will fill it out.

*I've added the answers we came up with. Maybe some others can correct any mistakes we made - Nathan Wilson and Ivor Popovic*

***Question 1: What is the vm system called that is configured for assignment 0?***

**Answer 1:** dumbvm

*Refer to: kern/arch/mips/conf.arch, implemented in kern/arch/mips/vm/dumbvm.c*

***Question 2. Which register number is used for the stack pointer (sp) in OS/161?***

**Answer 2:** $29

*Refer to: kern/arch/mips/include/kern/regdefs.h*

*Particular line:* `#define sp  $29 /* stack pointer */`

***Question 3. What bus/busses does OS/161 support?***

**Answer 3:** LAMEbus

*Refer to: kern/arch/sys161/include/bus.h*

***Question 4. Why do we use typedefs like uint32_t instead of simply saying "int"?***

**Answer 4:** To specify that a value is an unsigned 32 bit int. Different architectures handle 'int' as different sizes. Specifying like this allows us to make sure that it will be a consistent number of registers/memory each time - always 32bits, unsigned, and thus allows for simplifying compatibility across platforms. It also allows us to get more mileage out of a variable that we know will never go negative.

***Question 5. What function is called when user-level code generates a fatal fault?***

**Answer 5:** static void kill_curthread(vaddr_t epc, unsigned code, vaddr_t vaddr)

*Refer to: kern/arch/mips/locore/trap.c*

***Question 6. How frequently are hardclock interrupts generated?***

**Answer 6:** 100 times per second.

*Refer to: kern/include/clock.h*

*Particular lines:*

切换行号显示

```
/* hardclocks per second */
#define HZ  100

```

***Question 7. How many characters are allowed in an SFS volume name?***

**Answer 7:** The maximum is 32 characters including the null termination character, so only 31 characters are allowed in the name.

*Refer to: kern/include/kern/sfs.h,*

*Particular line:*
```
#define SFS_VOLNAME_SIZE 32 /* max length of volume name */
```

*Refer to: kern/include/kern/sfs.h,*

*Particular line:*
```
char sp_volname[SFS_VOLNAME_SIZE]; /* Name of this volume */
```

*Refer to: userland/sbin/mksfs/mksfs.c*

*Particular line:* `if (strlen(volname) >= SFS_VOLNAME_SIZE) {`

***Question 8. How many direct blocks does an SFS file have?***

**Answer 8:** 15 direct blocks

*Refer to: kern/include/kern/sfs.h*

*Particular line:*
```
#define SFS_NDIRECT       15           /* # of direct blocks in inode */
```

***Question 9. What is the standard interface to a file system (i.e., what functions must you implement to implement a new file system)?***

*Defined in: kern/include/fs.h*

切换行号显示

```
struct fs_ops {
        int           (*fsop_sync)(struct fs *);                    // Flush all dirty buffers to disk.
        const char    *(*fsop_getvolname)(struct fs *);             // Return volume name of filesystem.
        int           (*fsop_getroot)(struct fs *, struct vnode **); // Return root vnode of filesystem.
        int           (*fsop_unmount)(struct fs *);                 // Attempt unmount of filesystem.
};
```

To return a useful root vnode in fs_getroot you also need to implement all the filesystem specific operations on the vnode, i.e. implement the functions defined in kern/include/vnode.h. For SFS, these are defined in kern/fs/sfs/sfs_vnops.c

***Question 10. What function puts a thread to sleep?***

`void wchan_sleep(struct wchan *wc, struct spinlock *lk);` as defined in *kern/include/wchan.h*

Specifically, a thread is switched to S_SLEEP state by calling `thread_switch(S_SLEEP, wc, lk);` as in the implementation in `wchan_sleep()` in *kern/thread/thread.c*

==

*Isn't this in thread.c? Found by tracing through thread.h > struct threadstate_t > thread_switch (thread.c) > wchan_sleep*

***Question 11. How large are OS/161 pids?***

Pid in OS/161 is of type pid_t as defined in *kern/include/types.h* , which is then defined as a **32 bit signed integer** in *kern/include/kern/types.h* .

According to *kern/include/limits.h and kern/include/kern/limits.h* , a maximum value for process id is set as 32767 and user-level processes can only take a pid that is greater or equal than 2. However, these are not enforced anywhere in current implementation of OS/161.

Particular lines:

切换行号显示

```
// in kern/include/kern/types.h:77
typedef __i32 __pid_t;      /* Process ID */

// in kern/include/types.h:127
typedef __pid_t pid_t;

// in kern/include/limits.h:44
#define PID_MIN         __PID_MIN
#define PID_MAX         __PID_MAX

// in kern/include/kern/limits.h:76
/* Min value for a process ID (that can be assigned to a user process) */
#define __PID_MIN       2

/* Max value for a process ID (change this to match your implementation)
*/
#define __PID_MAX       32767

```

***Question 12. What operations can you do on a vnode?***

切换行号显示

```
struct vnode_ops {
  unsigned long vop_magic;      /* should always be VOP_MAGIC */

  int (*vop_eachopen)(struct vnode *object, int flags_from_open);
  int (*vop_reclaim)(struct vnode *vnode);


  int (*vop_read)(struct vnode *file, struct uio *uio);
  int (*vop_readlink)(struct vnode *link, struct uio *uio);
  int (*vop_getdirentry)(struct vnode *dir, struct uio *uio);
  int (*vop_write)(struct vnode *file, struct uio *uio);
  int (*vop_ioctl)(struct vnode *object, int op, userptr_t data);
  int (*vop_stat)(struct vnode *object, struct stat *statbuf);
  int (*vop_gettype)(struct vnode *object, mode_t *result);
  bool (*vop_isseekable)(struct vnode *object);
  int (*vop_fsync)(struct vnode *object);
  int (*vop_mmap)(struct vnode *file /* add stuff */);
  int (*vop_truncate)(struct vnode *file, off_t len);
  int (*vop_namefile)(struct vnode *file, struct uio *uio);


  int (*vop_creat)(struct vnode *dir,
       const char *name, bool excl, mode_t mode,
       struct vnode **result);
  int (*vop_symlink)(struct vnode *dir,
         const char *contents, const char *name);
  int (*vop_mkdir)(struct vnode *parentdir,
       const char *name, mode_t mode);
  int (*vop_link)(struct vnode *dir,
      const char *name, struct vnode *file);
  int (*vop_remove)(struct vnode *dir,
        const char *name);
  int (*vop_rmdir)(struct vnode *dir,
       const char *name);

  int (*vop_rename)(struct vnode *vn1, const char *name1,
        struct vnode *vn2, const char *name2);


  int (*vop_lookup)(struct vnode *dir,
        char *pathname, struct vnode **result);
  int (*vop_lookparent)(struct vnode *dir,
            char *pathname, struct vnode **result,
            char *buf, size_t len);
};
```

*Refer to: kern/include/vnode.h*

***Question 13. What is the maximum path length in OS/161?***

切换行号显示

```
/* Longest full path name */
#define __PATH_MAX      1024

```

Refer to *kern/include/limits.h:42 and kern/include/kern/limits.h:63*

Maximum path can be of length 1024 excluding null-terminator as used in *kern/main/menu.c:203*

***Question 14. What is the system call number for a reboot?***

I think it is 119

Particular line: `#define SYS_reboot       119`

Refer to *kern/include/kern/syscall.h:197*

***Question 15. Where is STDIN_FILENO defined?***

It's defined as 0 in *kern/include/kern/unistd.h:34* .

Particular line: `#define STDIN_FILENO  0      /* Standard input */`

Hint: Use Ctrl+Shift+F in vscode can help you find everything you want in the whole code base.

***Question 16. What does kmain() do?***

Seems to be the kernel main function, ie the first thing that happens when you run the os.

Kernel main. Boot up, then fork the menu thread; wait for a reboot request, and then shut down.

Refer to *kern/main/main.c:212*

***Question 17. What is the difference between splhigh and spl0?***

- spl0() sets IPL to 0, enabling all interrupts.
- splhigh() sets IPL to the highest value, disabling all interrupts.

IPL = Interrupt Priority level I believe

Refer to *kern/include/spl.h:94 and kern/thread/spl.c:128*

***Question 18. What does splx return?***

An int, representing the old interrupt state/spl level for the current thread.

Refer to *kern/thread/spl.c:128*

***Question 19. What is a zombie?***

Zombies are threads that have exited but still need to have thread_destroy called on them.

Refer to *kern/include/thread.h:64*

***Question 20. What does a device name in OS/161 look like?***

```
        /*
         * The name of a device is always just "device:". The VFS
         * layer puts in the device name for us, so we don't need to
         * do anything further.
         */
```

The bit after device: is

```
 * kd_name     - Name of device (eg, "lhd0"). Should always be set to
 *               a valid string.
```

Refer to *kern/vfs/device.c:281 and kern/vfs/vfslist.c:50*

- vfs_adddev - Add a device to the VFS named device list. If
  - MOUNTABLE is zero, the device will be accessible as "DEVNAME:". If the mountable flag is set, the device will be accessible as "DEVNAMEraw:" and mountable under the name "DEVNAME". Thus, the console, added with MOUNTABLE not set, would be accessed by pathname as "con:", and lhd0, added with mountable set, would be accessed by pathname as "lhd0raw:" and mounted by passing "lhd0" to vfs_mount.

***Question 21. What does a raw device name in OS/161 look like?***

```
 *
 * kd_rawname - Name of raw device (eg, "lhd0raw"). Is non-NULL if and
 *               only if this device can have a filesystem mounted on
 *               it.
```

Refer to *kern/vfs/vfslist.c:53*

***Question 22. What lock protects the vnode reference count?***

A spinlock that is a member of `struct vnode` as defined in *kern/include/vnode.h:51* .

Particular lines:

切换行号显示

```
struct vnode {
        int vn_refcount;                /* Reference count */
        struct spinlock vn_countlock;   /* Lock for vn_refcount */
        // ...
};

//====================

// in kern/vfs/vnode.c:84
spinlock_acquire(&vn->vn_countlock);
vn->vn_refcount++;
spinlock_release(&vn->vn_countlock);
```

***Question 23. What device types are currently supported?***

I think its:...

A device is either a "block device" or a "character device".

Since the documented return value of
`static int dev_gettype(struct vnode *v, mode_t *ret)` is:

- Return the type. A device is a "block device" if it has a known
- length. A device that generates data in a stream is a "character
- device".

Refer to *kern/vfs/device.c:195*

Asst0 (2019-02-21 21:20:00由MichelleQiu编辑)