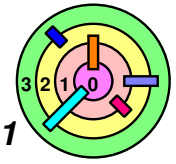
Kernel 2

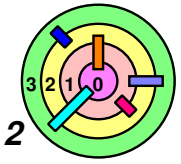
Bill Cheng

http://merlot.usc.edu/william/usc/



New Things in Kernel 2

- Polymorphism - VFS should be able to work with *any* AFS
  - achieved using *polymorphism*
    - the AFS for kernel 2 is `ramfs`
    - read the kernel code to see how it works
- Reference counting
  - whenever you *keep* a reference to an object, you increase the reference count of that object
    - so that the object won't disappear
  - whenever you remove a reference to an object, you decrement the reference count of that object
    - if it reached zero, it means that nothing in the system knows about that object and you should delete/free it
  - this is done for objects in memory
  - this is done for objects inside the file system (which you don't have to worry about)

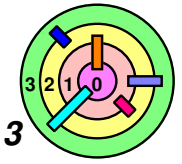


Kernel 2

```
% make nyi
```

```
        main/kmain.c:218                    idleproc_run()      VFS
        main/kmain.c:223                    idleproc_run()      VFS
          fs/vnode.c:460              special_file_read()       VFS
          fs/vnode.c:473             special_file_write()       VFS
       fs/vfs_syscall.c:68                       do_read()      VFS
       fs/vfs_syscall.c:83                      do_write()      VFS
       fs/vfs_syscall.c:97                      do_close()      VFS
      fs/vfs_syscall.c:120                        do_dup()      VFS
      fs/vfs_syscall.c:136                       do_dup2()      VFS
      fs/vfs_syscall.c:168                      do_mknod()      VFS
      fs/vfs_syscall.c:189                      do_mkdir()      VFS
      fs/vfs_syscall.c:214                      do_rmdir()      VFS
      fs/vfs_syscall.c:235                     do_unlink()      VFS
      fs/vfs_syscall.c:263                       do_link()      VFS
      fs/vfs_syscall.c:278                     do_rename()      VFS
      fs/vfs_syscall.c:298                      do_chdir()      VFS
      fs/vfs_syscall.c:320                    do_getdent()      VFS
      fs/vfs_syscall.c:337                      do_lseek()      VFS
      fs/vfs_syscall.c:357                       do_stat()      VFS
             fs/namev.c:45                        lookup()      VFS
             fs/namev.c:72                     dir_namev()      VFS
             fs/namev.c:90                   open_namev()       VFS
              fs/open.c:94                       do_open()      VFS
```

- make sure `DRIVERS=1`, `VFS=1` in `Config.mk`



Kernel 2

Create devices, set current working directory

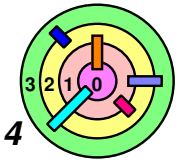
main/kmain.c:218	idleproc_run()	VFS
main/kmain.c:223	idleproc_run()	VFS

Reading from device and writing to device

fs/vnode.c:460	special_file_read()	VFS
fs/vnode.c:473	special_file_write()	VFS

Pathname resolution functions

fs/namev.c:45	lookup()	VFS
fs/namev.c:72	dir_namev()	VFS
fs/namev.c:90	open_namev()	VFS



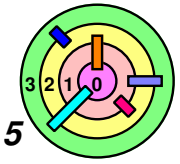
Kernel 2

System calls

fs/vfs_syscall.c:68	do_read()	VFS
fs/vfs_syscall.c:83	do_write()	VFS
fs/vfs_syscall.c:97	do_close()	VFS
fs/vfs_syscall.c:120	do_dup()	VFS
fs/vfs_syscall.c:136	do_dup2()	VFS
fs/vfs_syscall.c:168	do_mknod()	VFS
fs/vfs_syscall.c:189	do_mkdir()	VFS
fs/vfs_syscall.c:214	do_rmdir()	VFS
fs/vfs_syscall.c:235	do_unlink()	VFS
fs/vfs_syscall.c:263	do_link()	VFS
fs/vfs_syscall.c:278	do_rename()	VFS
fs/vfs_syscall.c:298	do_chdir()	VFS
fs/vfs_syscall.c:320	do_getdent()	VFS
fs/vfs_syscall.c:337	do_lseek()	VFS
fs/vfs_syscall.c:357	do_stat()	VFS

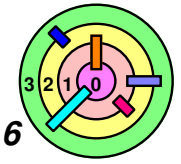
Open

fs/open.c:94	do_open()	VFS



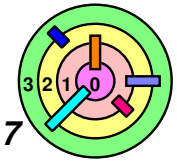
Kernel 2: Where To Start

- *(Phase 1)* Create directories and devices in `idleproc_run()`
  - `do_mkdir()` and `do_mknod()`
    - and whatever you need to support these functions (including *pathname resolution* functions in "`namev.c`")
      - `open_namev()`, `dir_namev()`, `lookup()`
  - maybe get `do_stat()` to work to check your work
- *(Phase 2)* Get kshell to work again in `initproc_run()`
  - kernel 2 kshell is different from kernel 1 kshell
    - kernel 2 kshell uses VFS!
  - need to get a few more functions to work so that you can interact with kshell
    - `do_open()`, `do_write()`, `do_read()`, `special_file_write()`, `special_file_read()`
- *(Phase 3)* Create *kshell command* to run "`vfstest`"
  - pass all tests in "`vfstest.c`"
  - also need to pass tests in "`faber_fs_test.c`"



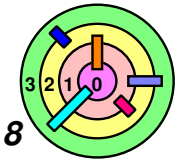
vnode.c

```
/*
 * If the file is a byte device then find the file's
 * bytedev_t, and call read on it. Return what read
 * returns.
 *
 * If the file is a block device then return -ENOTSUP
 */
static int
special_file_read(vnode_t *file, off_t offset,
                  void *buf, size_t count)
{
    NOT_YET_IMPLEMENTED("VFS: special_file_read");
    return 0;
}
```



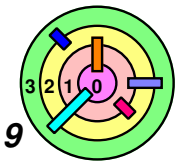
vnode.c

```
/*
 * If the file is a byte device find the file's
 * bytedev_t, and call its write. Return what write
 * returns.
 *
 * If the file is a block device then return -ENOTSUP.
 */
static int
special_file_write(vnode_t *file, off_t offset,
                   const void *buf, size_t count)
{
    NOT_YET_IMPLEMENTED("VFS: special_file_write");
    return 0;
}
```



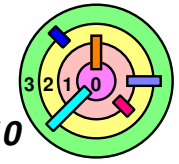
namev.c

```
/*
 * This takes a base 'dir', a 'name', its 'len', and a
 * result vnode.
 *
 * Most of the work should be done by the vnode's
 * implementation specific lookup() function.
 *
 * If dir has no lookup(), return -ENOTDIR.
 *
 * Note: returns with the vnode refcount on *result
 * incremented.
 */
int lookup(vnode_t *dir,
           const char *name,
           size_t len,
           vnode_t **result);
```



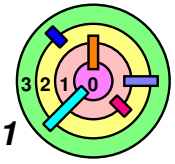
namev.c

```
/*
 * When successful this function returns data in the following
 * "out"-arguments:
 *  o res_vnode: the vnode of the parent directory of "name"
 *  o name: the `basename' (the element of the pathname)
 *  o namelen: the length of the basename
 *
 * For example: dir_namev("/s5fs/bin/ls", &namelen, &name, NULL,
 * &res_vnode) would put 2 in namelen, "ls" in name, and a pointer to
 * the vnode corresponding to "/s5fs/bin" in res_vnode.
 *
 * The "base" argument defines where we start resolving the path from:
 * A base value of NULL means to use the process's current working
 * directory, curproc->p_cwd.  If pathname[0] == '/', ignore base and
 * start with vfs_root_vn.  dir_namev() should call lookup() to take
 * care of resolving each piece of the pathname.
 *
 * Note: A successful call to this causes vnode refcount on *res_vnode
 * to be incremented.
 */
int dir_namev(const char *pathname,
              size_t *namelen,
              const char **name,
              vnode_t *base,
              struct vnode **res_vnode);
```



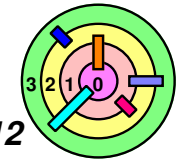
namev.c

```
/*
 * This returns in res_vnode the vnode requested by the
 * other parameters.  It makes use of dir_namev and
 * lookup to find the specified vnode (if it exists).
 * flag is right out of the parameters to open(2); see
 * <weenix/fcntl.h>.  If the O_CREAT flag is specified,
 * and the file does not exist call create() in the
 * parent directory vnode.
 *
 * Note: Increments vnode refcount on *res_vnode.
 */
int open_namev(const char *pathname,
               int flag,
               vnode_t **res_vnode,
               vnode_t *base);
```



vfs_syscall.c

```
int do_close(int fd);
int do_read(int fd, void *buf, size_t nbytes);
int do_write(int fd, const void *buf, size_t nbytes);
int do_dup(int fd);
int do_dup2(int ofd, int nfd);
int do_mknod(const char *path, int mode, unsigned devid);
int do_mkdir(const char *path);
int do_rmdir(const char *path);
int do_unlink(const char *path);
int do_link(const char *from, const char *to);
int do_rename(const char *oldname, const char *newname);
int do_chdir(const char *path);
int do_getdent(int fd, struct dirent *dirp);
int do_lseek(int fd, int offset, int whence);
int do_stat(const char *path, struct stat *uf);
```



open.c

```
/*
 * There are a number of steps to opening a file:
 *      1. Get the next empty file descriptor.
 *      2. Call fget to get a fresh file_t.
 *      3. Save the file_t in curproc's file descriptor table.
 *      4. Set file_t->f_mode to OR of FMODE_(READ|WRITE|APPEND) based
 *         on oflags, which can be O_RDONLY, O_WRONLY or O_RDWR, possibly
 *         OR'd with O_APPEND or O_CREAT.
 *      5. Use open_namev() to get the vnode for the file_t.
 *      6. Fill in the fields of the file_t.
 *      7. Return new fd.
 *
 * If anything goes wrong at any point (specifically if the call to
 * open_namev fails), be sure to remove the fd from curproc, fput the
 * file_t and return an error.
 *
 * Error cases you must handle for this function at the VFS level:
 *      o EINVAL
 *      o EMFILE
 *      o ENOMEM
 *      o ENAMETOOLONG
 *      o ENOENT
 *      o EISDIR
 *      o ENXIO
 */
int do_open(const char *filename, int flags);
```