A1 and A2 kernel.

We have imported our implementation for A1 and A2 into A3. A1 allows us to pass arguments for p command, and overwrite the semaphore hack A3 started code came with. We hoped this would help us test our A3 implementation of system calls. However, A2 seemed to be unnecessary.

Open File System.

Our implementation of the open file system supports fork(). We keep track of current file's offset and store it in *vnode* so we can call it using the file descriptor. Modifications can be done on it once our *vnode* is retrieved.

fork implementation.

Our implementation of *fork* increases a reference count variable of the vnode. It also creates a new *filetable* which points to associated *vnodes*. We also defined two locks, one for the *vnode* and for the *filetable* to prevent race condition with *vnode* and *filetable*. As a consequence of this implementation, both the parent and child processes after fork() will alter the same offset value, in the same vnode, creating a race condition (uncertain which one will read/write first).

Implementation of system calls.

Part of A3 was the implementation of 10 file system calls. We will go briefly over the syscalls, their return value, and their purpose.

int **sys_open** (*userptr_t* filename, *int* flags**,** *int* mode**,** *int* *retval)

1. Method takes filename,
2. allocates memory for *char* *fname,
3. copies filename parameter,
4. and opens the file,
5. free the malloced space.

int **sys_close** (*int* fd)

1. Method passes the work to file_close() as defined in file.c

int **sys_dup2** (*int* oldfd, *int* newfd, *int* *retval)

1. Initially, method checks that both fd are
    - unique
    - and valid,
2. checks if newfd points to an open file,
3. copy vnode pointer from t_entries of oldfd to t_entries of newfd,
4. and call VOP_INCREF() to increase reference counter.

int **sys_read** (*int* fd, *userptr_t* buf, *size_t* size, *int* *retval)

1. Initially, Method checks:
    - if size,
    - fd is valid,
2. gets appropriate *vnode* with given fd,
3. get current offset from fetched *vnode*,
4. sets up *uio* for read,
5. uses VOP_READ() to read,
6. update retval, which is the original amount minus how much is left in the buffer,
7. update offset of read file.

int **sys_write** (*int* fd, *userptr_t* buf, *size_t* len, *int* *retval)

1. Initially, method checks if
    - len is valid,
    - fd is valid
2. gets appropriate *vnode* with given fd,
3. gets offset from the *vnode*,
4. sets up *uio* for write,
5. use VOP_WRITE() to write,
6. update retval, which is the original size minus how much is left in the buffer,
7. update offset of written file.

int **sys_lseek** (*int* fd, *off_t* pos, *int* whence, *off_t* *retval)

1. Initially, check whether fd is valid,
2. get appropriate *vnode* to seek through,
3. get the old offset of the file,
4. update
5. deal with the following cases:
    - if SEEK_SET, then set pos as toSetOffSet,
    - if SEEK_CUR, then pos + oldOffSet as toSetOffSet,
    - if SEEK_END, then:
        - get size of file,
        - update toSetOffSet, which is st_size + pos,
6. seek using VOP_TRYSEEK(),
7. and if successful, update file's offset with toOffSet,
8. and set retval to 0.

int **sys_chdir** (*userptr_t* pathname)

1. Allocate memory for new path variable,
2. copy pathname to path,
3. give the work to vfs_chdir() as defined in vfscwd.h.

int **sys___getcwd** (*userptr_t* buf, *size_t* buflen, *int* *retval)

1. Set up *uio* for read,
2. pass work to vfs_getcwd() as defined in vfscwd.h,
3. update retval, which is original size minus how much is left in buffer.

int **sys_fstat** (*int* fd, *userptr_t* statptr)

1. Initially, function checks if fd is valid,
2. get file from t_entries with valid fd,
3. and get file info with VOP_STAT() as defined in vnode.h.

int **sys_getdirentry** (*int* fd, *userptr_t* buf, *size_t* buflen, *int* *retval)

4. get t_entries with fd
5. and get offset of current file,
6. set up *uio* for read,
7. then pass work to VOP_GETDIRENTRY() as defined in vnode.h,
8. update retval, which is original size minus what is left in the buffer,
9. update offset of file.

Tests for system calls

Unfortunately we were not able to fully debug our code. Therefore not all tests passed fully. There's some issue with invalid pointer that we couldn't check and some semaphore locks had their own issues with invalid pointers. In both cases the issue happened in code outside our own, and it has been difficult to pinpoint the cause.

Other Parts

Part 3 and 4 of the assignment (SFS codes) were not implemented.