A1 and A2 kernel.

Given the choice to implement our A1 and A2 kernel, we consciously decided against doing so. We did not find it necessary to complicate our workspace in such a manner.

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* increase a reference count variable. 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*. One possible flaw with such implementation can occur when *fork* is called: two processes accessing the same *vnode* may create race condition causing an inaccurate offset variable next time the *vnode* is accessed.

Implementation of system calls.

Part of A3 was the implementation of 10 extra 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.h

*int* **sys\_dup2** (*int* oldfd, *int* newfd, *int* \*retval)

1. Initially, method checks that both fd are

* unique
* and valid,

1. checks if newfd points to an open file,
2. copy vnode pointer from t\_entries of oldfd to t\_entries of newfd,
3. 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,

1. gets appropriate *vnode* with given fd,
2. get current offset from fetched *vnode*,
3. sets up *uio* for read,
4. uses VOP\_READ() to read,
5. update retval, which is the original amount minus how much is left in the buffer,
6. 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
  + 1. gets appropriate *vnode* with given fd,
    2. gets offset from the *vnode*,
    3. sets up *uio* for write,
    4. use VOP\_WRITE() to write,
    5. update retval, which is the original size minus how much is left in the buffer,
    6. 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,
    1. seek using VOP\_TRYSEEK(),
    2. and if successful, update file's offset with toOffSet,
    3. 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)

* + 1. get t\_entries with fd
    2. and get offset of current file,
    3. set up *uio* for read,
    4. then pass work to VOP\_GETDIRENTRY() as defined in vnode.h,
    5. update retval, which is original size minus what is left in the buffer,
    6. update offset of file.