Question1 :

File descriptors is consisted of a vnode, which is an abstract representation of the file. An offset, which is the current position of the file, has modes (O\_RDONLY, O\_WRONLY, O\_RDWR) which represent read-only, write-only, and read-write. Lastly, it is consisted of a lock, which protects the file from being accessed by multiple threads at once.

The kernel data structure that is used in our implementation is a file descriptor manager. Its purpose is to manage the file descriptors in kernel mode. This data structure contains an array of file descriptors that it is managing currently. Our file manager has a function to deal with the different default standard files (STDIN, STDOUT, STDERR). The manager has functions to add/remove/return file descriptors and other various functions to modify its contents.

OPEN is used to open files. vfs\_open is called in the beginning to check if the file is openable. If the vfs\_open call was not successful, we return the appropriate error. Otherwise, a file descriptor is created with the file name, file mode, the vnode, and a lock. Then it is added to the file manager.

CLOSE is used to close file. When this is called, we access its file descriptor in the file manager, and remove it. If this is not successful, appropriate errors are returned.

READ is used to read a file. A file descriptor is fetched from the file manager of the current thread. We make sure that the file descriptor’s file mode is set to an appropriate flag to be read. Otherwise, an error will be returned. We create a uio structure to pass data from the kernel space to the user space containing information such as: number of bytes to be read, file description, etc. Since all of the files are managed by the kernel, the file to be read must be passed onto the user space in order for the file to be read. A lock is acquired during the read, such that the file is only being accessed by a single thread at a time. After the file is finished being read, the lock is released so that other threads can access the file. During the read call, the uio is passed onto VOP\_READ so that uio\_resid is updated to match the number of bytes read. Finally we return the (number of bytes to be read – uio\_resid). This value is the actual bytes that have been read.