Andy Mahoney

USERFS Write Up

u\_import() - Check if the file is invalid for any reason, like using stat to see if the file is too large, checking to make sure the file name isn't too large, checking if the file already exists, if it does give a warning. Then check if there's room in the directory and free inodes available. If all of this is fine then we get an inode and fill it with the information of the file to complete the struct found in userfs.h, then we update the bitmap, and stores where those updates were in curr\_inode.blocks and we update the inode. Then we update the superblock’s number of free blocks using u\_quota(). We do it in this order to go from the smallest thing to the larger things, as the later ones depend on the results of the earlier ones. Then we return 1. If any of the checks failed we return 0.

u\_export() - first we check if the file we’re trying to export exists, if it does we continue, if not we return 0. Then we open a read/write handle of the linux file we’re trying to create, if it doesn’t exist we make. Then we set the curr\_inode to the ufile we’re exporting. Then for each block of the file we seek to it on the virtual disk, read it to a buffer and then write that buffer to the linux file. When all this is done we return 1.

u\_del() - first it checks if the file exists in the directory, if not we return 0. If it is found we save its location. Then we free up its spot in the directory, and set the curr\_inode to its inode. Then we use blocks[] to find its locations in the bitmap and use free\_block to free the parts in the bitmap. Then we reset all the data in the inode back to defaults, update the number of free blocks in sb and return 1.

u\_fsck() was not implimented.

Food for thought:

1. BLOCK\_SIZE\_BYTES \* MAX\_BLOCKS\_PER\_FILE = 409600 bytes
2. 100
3. 1
4. 0
5. We would just change BLOCK\_SIZE\_BYTES as listed in 1 and recalculate, so 1024 would be 102400 bytes.