Participants:
James Flemings
Luka Spaic

Assignment 3

For the third assignment we were able to successfully implement a file system using fuse. In our design, we implemented our memory_block struct, which is essentially the building block of everything, and then all the other structs in the program, such as file_block and inode_struct_file, are what essentially go on top of memory_block. Once we allocate the amount of memory needed, we retrieve a pointer to the start of the file system. We then have a struct called super_block that stores the magic number, the first free memory, the size, and the root directory. We utilize the get_handle function to get the super block. The super block then points to the first free memory block. We utilize the pointer at the start of the file system, with the offset of each memory block, to be able to successfully get around the memory block. We do this by adding the pointer to whatever offset value the block we are wanting to get to has. We perform this process by the two functions called ptr_to_offset and offset_to_ptr. These functions implement the process of getting the pointer to the start of the file system, and then using the offsets to get the other pointers using pointer arithmetic. When looking for a free memory block, we use get_memory_block to traverse the linked list of free memory blocks until we find a block that is suitable to whatever we're wanting to store. We also have add_to_free_memory function that implements our process of freeing a memory block. It firstly goes through a for loop that loops through the linked list of free blocks. This loops from the start of the free memory to the first free memory block after the block that we are wanting to free. We do this in order to correctly change up the pointers once the memory block is freed, as the freed block will now point to the next free memory block after it. Then the possible merging of memory blocks is done. We also have a reallocate_memory function that handles the situation where more space is needed and we need more memory or a bigger memory block. We also implemented our process of retrieving data with the get_path function. Starting from the root directory, it'll keep searching until it finds the file or directory that is being requested, or it'll return NULL if not located. That is about it for all of the helper functions and our conceptual idea that we used for this assignment.

We were able to implement all thirteen functions necessary for the file system. The functions work successfully on trivial (toy) examples. However, the implementation isn't perfect as there are known bugs. Due to time constraints and sanity, these bugs weren't resolved in time. The bugs known are the following:

- If you fill up the file system with a lot of files, then try to remove all of them, it's possible that the operation will either fail, succeed with no issues, or succeed but the memory available gets corrupted.

- When the file system is full and you try to create a file or directory, the appropriate error message will show. However, if you run “ls”, the file system could crash.
- Most operations invoked after the file system is filled could result in a crash. There are checks in place for each function to determine if there is sufficient memory for allocation, but there is a bug somewhere in the helper functions that bypass this check.

I’m sure there are more bugs in the system that we aren’t aware of. I think most of them derive from our freeing memory helper function, but this isn’t certain.