Q1. What are individual contributions of your group for this project? (Explain the roles and work of each member of your group)

Jake was responsible for coding the kernel and modules C programs, Justin was responsible for coding the modules and apps programs, and Cam was responsible for building, compiling, and testing the system.

Q2. How does a system call execute? Explain the steps from making the call in the user-space process to returning from the call with a result.

First, the user-space process requests a system call by invoking a library function which will identify the system call number corresponding to the requested operation.

Then, the system call triggers a CPU instruction to switch the CPU to kernel mode, while saving the current user-space context. The kernel then performs the requested operation, and if it succeeds, it prepares the result, otherwise an error code is generated.

Finally, with the prepared result, the CPU restores the user-space context, switches back to user mode, and passes the result to the application.

Q3. What is SYSCALL_DEFINEx and where is it defined?

SYSCALL_DEFINEx (SYSCALL_DEFINE4 in our case because we had 4 arguments) was defined in sys_calc.c. For this project, it is a macro that returns calc_curr which holds the current version of the calculator (either the regular calculator if we haven't inserted the modulo module or a modulo calculator if we have).

Q4. What synchronization mechanism can be used to access shared kernel data structures safely?

There are multiple synchronization mechanisms that can be used to access shared kernel data structures safely. To name a few, read-copy-update allows readers to access data without acquiring a lock and, specifically regarding this lab, copy_to_user and copy_from_user commands are linux commands that allow us to copy between kernel memory space and user memory space.

Q5. What issues have you faced with the project and how did you resolve them?

Some issues that we faced were lack of experience working in the kernel space and the inability to test our code without the raspberry pi. Regarding the first issue, this was our first time doing a lot of new things like defining system calls and creating Kbuild files. Because of this, we had to research a lot to better understand the right way to do things and we spent a lot of time just learning the basics. Regarding the second issue, the problem was pretty straightforward. In most projects that we've worked on, we've gotten used to having the comfort of compiling and running our code over and over and over again whenever we want in order to test small changes. For this project, however, we didn't all have access to the raspberry pi at all times, and so a lot of the time we had to thoroughly read through our code ourselves instead of relying on the compiler’s error messages.