<u>Problem 1 Solution Description:</u>

Kernel Producer Consumer problem

My code has the following elements:

1. **Adding the syscall**: I have changed the syscall_64.tbl file present in the directory linux-5.14.3/arch/x86/entry/syscalls and added the name of the system calls as reader and writer at the end i.e (line 448,449 of the stock kernel).
2. **Mutex**: I have defined a global mutex(myqueue_lock) in the file sys.c present in the header file linux/mutex.h.This mutex is initialized using DEFINE_MUTEX(name) and this is used for synchronization in syscalls reader and writer. Both the system before performing any operation of queue locks the mutex using the function mutex_lock() and after performing operations releases the mutex using mutex_unlock(). Mutexes don't allow multiple threads and processes to access the same resource at same time.
3. **Working of queue**: I have implemented a circular queue using a char pointer array of fixed size(5) initialized as Null. Two integer variables head, tail are used to navigate the file. All the variables are initialized as global variables in file sys.c so both the system calls have their access.
4. **Writer**: first writer syscall gets access to the queue using mutexes. To enqueue data firstly memory is provided to the queue using kmalloc() with flag as GFPKERNEL after which it copies the data provided by the user into the queue using copy_from_user() system call. All the data is enqueued at the tail and tail is updated is (tail+1)% size (this provides the functionality of circular queue).
5. **Reader**: first reader syscall gets access to the queue using mutexes. To dequeue data firstly data is copied in user level pointer using copy_from_user() system call and further the entry of the queue is initliazed as null. All the data is dequeued from the head and head is updated is (head+1)% size (this provides the functionality of circular queue).

Nakul Thureja

2020528