How to run:

1. Get the linux source, patch with my supplied patch. Build kernel.
2. Boot into kernel, navigate to this directory, run `make`. The rest of the work is handled.

How to verify:

1. Log the output of terminal in a file or a `tee` instance.
2. open with any text-editor and search for the "/dev/random" number. Each "random" number will be written exactly once and retrieved exactly once.

Kernel modifications:

0. Please go through the patchfile as an when required while reading below.
1. Two syscalls were added, one for reading and other for writing. (548, 549). `linux-version/arch/x86/entry/syscalls/syscall_64.tbl`
2. SYSCALL_DEFINE(d) in `linux-version/kernel/sys.c`
3. The writer syscall allows 1 argument, the reader allows none.
4. The writer returns error/succes, the reader returns value/error.
5. The queue, essentially a circular linked list, of maximum size defined by `QSIZE=16`, this value can be modified, but for the sake of the assignment, it is a beautiful number. The queue named as `k_mqueue` is a static pointer reference to `a_node`, it is also defined in `sys.c` along with the mutex variable for synchronization. Also a variable to track the initialization has been made inorder to disambiguate reference allocation/NULLPTR. There are also static refen's to the head and tail of the queue.
6. Both syscalls by-default check if the queue as been initialized or not.
7. EVERYTIME a syscall is used, it will always acquire the MUTEX as uninterruptible, make changes to queue (or not) and then UNLOCK the MUTEX, finally return the control. (ref: kernel.org/MUTEX.

The writer syscall:

- This syscall writes to the queue specified above. The circular linked list is implemented using a struct with the `next` address field as desired. The queue is allocated using `kmalloc` when the writer is called for the first time after boot. Then the initialized variable is set to 1 until `poweroff`. The struct specified contains a field for the `dev-random` variable and a boolean(ish) value to indicate that the dev-random is fresh and was not read by the reader syscall.
- Whenever this syscall is used, the tail of the linked list is queried, if `tail->allocated` is true, the syscall returns with an error since the queue is full. if `tail->allocated` is false, we write to `tail->dev-random` and set tail as `tail->next`. Thus enqueuing the value.

The reader syscall:

- Whenever this syscall is used, the head of the linked list is queried, if `head->allocated` is false, the syscall returns with an error since the queue is empty. if `head->allocated` is true, we read to a temporary variable the value of `head->dev-random` and set head as `head->next`. Thus dequeueing the value. The temporary variable is then returned.

P1 and P2 (testers)

- These are simple tests to verify the synchronous behaviour and data storing/retrieving from the "queue".
- P1 reads from `/dev/random` by popening the file and read first 8 bytes using the `head -c 8 /dev/random` command. head -c 8 means read first 8 bytes from specified file. A loop (which runs only 10k times) is used to syscall the `writer` and write the values. IFF a write fails (buffer full), the same value is sub-looped to be sent to the syscall again and again (i.e. a spinlock (kindof) in userspace).
- P2 just calls the syscall to read (infinite times). It can only receive an error when the buffer is empty.
- Both processes sleep for random amount of time to force race conditions in the kernel, however the race never happens because one of the calls get blocked when trying to acquire a mutex.