Homework 9

memory mapping Write one program that will be called memmapA and will also be called via a symbolic link memmapB; it will use memory mapping to pass their command line arguments to each other (not standard input). Let the shared memory mapped file be called memfile, and assume that no arguments will be longer than 256 bytes. Each instance of the program should display messages such as

```
sent <argument string here>
rcvd <argument string here>
```

Design Criteria:

- To get memmapA and memmapB: write memmapA.c, compile and link it, and use ln -s memmapA memmapB to generate a symbolically linked version.
- Your program can tell by what name it was invoked by examining argv[0]
- The first program running should get writing privileges and initialize the semaphore; it should write all its arguments and then switch to receive mode.
- The first program to run (ie. who's got the semaphore?) should write a character at byte 255 to insure that the memfile is long enough by using one of the `seek` functions.
- Use two semaphores: when semA is 0, then memmapA may write; when semB is 0, then memmapB may write.
- When memmapA is done writing, it should set semA to 1 and unlock semB (ie. set it to zero).
- When memmapB is done writing, it should set semB to 1 and unlock semA (ie. set it to zero).
- Use the pointer returned from mmap to do the string transfers (*not* file i/o functions such as read or write).
- The last program to receive should perform cleanup (remove memfile, semaphores, etc)
- Don't use file i/o to read the memory mapped file! Use the memory mapped array instead.