Part 2 of 3 (100 points, code, 60 minutes) “Memory mapped secrets”

It's no longer safe and you have only minutes to spare before *BigMichicanCorp* finds you; time for a fast exit. You've put all of your valuable secrets into a single file (it could be a text file, zip file etc), now you just need to get out of here without anyone being able to discover or decrypt its valuable contents. Fortunately you have your CS241 skills and a VM. You quickly create and use a Linux C99 program `helloworld.c` using the following -

```
clang -O0 -Wall -Wextra -Werror -Wno-error=unused-parameter -g -std=c99
-D_GNU_SOURCE -DDEBUG helloworld.c
```

When your program is run it seems to be the simplest program; it just prints `Hello World`

```
./a.out
```

(prints "`Hello World`" without the quotes to standard-out and exits with value 0)

However, later if `a.out` is renamed to `encrypt` it can create an encrypted version of the contents of your file with every byte xor'd with a random value and store it in a new output file

```
mv a.out encrypt
./encrypt mystuff.zip cats.jpg
```

It opens the existing file `mystuff.zip` in binary (using `open` and `mmap`) for reading and writing, and creates one new file `cats.jpg`. As it processes the bytes of `mystuff.zip` it also overwrites as soon as possible the contents of `mystuff` with `0xff` bytes so that the file's original content is no longer recoverable. The first half of `cats.jpg` is a sequence of random bytes (read from `/dev/urandom`), the same length of the input file. The second half contains the bytes from the original file encrypted by xor-ing each byte with the corresponding byte stored at the first half of the file. Upon completion the output file will be the twice length as the original file while the length of the original file is unchanged but all of original contents will have been overwritten.
For example if `mystuff.zip` contained two bytes, `0x51,0x52` and the first two random values read from `/dev/urandom` were `0x41 0x03` then the new `cats.jpg` would contain with `0x41,0x03,0x10,0x51` and `mystuff.txt` would become `0xff,0xff`.

You head out of town, pausing only to donate your old laptop. Three days, two flights and a row boat later, you rename your program to `decrypt`, and recover your secrets -

```
mv encrypt decrypt
./decrypt cats.jpg output.zip
```

This reads the two file `cats.jpg` (using `mmap`) and creates the output file `output.zip` (with contents identical to the original `mystuff.zip` file).

Upon successful completion it deletes `cats.jpg` from the filesystem.

For example, using the file contents above it could recover the original two byte file (0x51 0x52)

Write one program that uses the process name to determine behavior when executed. If the program name is `./encrypt` then encrypt the file, if it is `./decrypt` then decrypt the given files, as

described above. If neither is true, or 2 filename arguments are not provided, then innocently print "`Hello World`" and exit with value 0.

Use `argv[0]` to check the process name.
Use `open` and `mmap` to read and modify the contents of original file (encryption mode) and to read the input file in decryption mode.
Use `fopen` and `fputc` to create the output files in both encrypt and decrypt modes.
The `unlink` and `stat` calls may also be useful.
The `hexdump` utility may be useful.

Grading Rubric & Submission

Add your netid as a comment to your code. For example, if your netid was angrave, write

```
// author: angrave
```

Upload your code, `helloworld.c`; it should compile using the above clang options on a standard CS241 VM. It will be graded on (subject to minor modifications)

10	Correctly uses the program's arguments to change behavior between hello-world, encrypt and decrypt modes based on the process' name
10	Overwrites the original file contents with `0xff` bytes as soon as each byte has been read.
10	Random bytes are sourced from `/dev/urandom`
10	Does not crash; exits normally with value 0 when given valid arguments
10	Encoding generates desired encrypted output file
10	The original file's content can be recreated after encrypting and decrypting
10	Encrypted file is removed from the filesystem after decryption processing is complete
10	Uses `mmap` to read the given input file
10	Uses `fopen` and `fputc` for output
10	Includes your netid in the author comment at the top of the code

All other behaviors, output, missing arguments, missing files, handling error conditions etc. are *unspecified and are not graded*.