CS241 Final Exam / Final Project
Spring 2021

Welcome to your CS241 Final Project. This is a "take home" open book exam. To complete CS241 you will be demonstrating *your own competency* in CS241 skills and knowledge. This means you may not collaborate, work with, or get help from current or former CS241 students or search for solutions. This final project is open book (you may access and reuse general system programming online materials) but the code you include is expected to be your own original work that you created during this exam.

You may not share code, hints, test code or anything else that you created as part of this exam. Publishing your work that is part of this final exam or sharing your work with another student directly/indirectly is a violation of academic integrity and may affect not just your CS241 grade but your standing as a student at UIUC.

Keep the total time spent on this to less than 5 hours (ideally 3 hours); it is up to you to track this.

We expect most of you will have access to a Linux system (e.g., WSL on Windows, Virtual Machine, ssh access to your CS241 VM, or the VM-in-a-browser). However in principle you could complete this using only a smart phone, pen and paper. Complete this final project by Wednesday May 12th **11:00pm CT**. Submission details will be posted on Piazza and/or email. There will be an approximate 59 minute grace period after 11pm to allow for slow uploads and IT issues.

Questions? Other than submission questions, we're not going to respond to questions about the actual content of the exam; use your best efforts to interpret the instructions.

If you believe it is ambiguous state your assumptions and answer accordingly in your exam responses.

Out of fairness, do not share or post comments about the content until after grades are published.

Be sure to check your submission and allow sufficient time to upload it.

Part 1 of 3 (100 points, video, 60 minutes) “My First Office Hours”

"Um. Don't Panic. Um. Sorry!" a familiar English voice announces, "I'm just adjusting the ... Aha! There we go."

The world shifts into full view. Sensations of touch, smell, and sound flood your mind. You are in a small office inside Siebel and a light breeze against your face awakens you out of a dreamy slumber. There's some tables, a whiteboard, bookcase and chairs. The sensations feel real except the scene is clearly fake the words "History Simulation Test (bug fix 52)" scroll in and blinks slowly in orange and blue letters in the bottom left corner of your vision in Courier New font, confirming that none of this is actually real. Yet the perceptual sensations feel more tangible than the breakfast you had earlier, and definitely more meaningful. A sense of spreading future connections overwhelms you (like a 400-level course on graph theory where the expanding nodes and edges escaped from the paper and leapt into the wild) – connections to people, events, things – future interviews, people, conversations, things that will help you change the world but have not yet happened but will happen in your future.

"Okay," the friendly voice continues, "I set the year back to 2022 – which explains the odd fashion choices you're about to see! In about 1 minute your office hours will start and the first student will walk in. This first test is about whether you can explain to a student some 241 topics, to demonstrate that you actually understand these topics and can be a valuable member of the course staff. So be ready to do some live programming demos, explain the concepts, whatever you need to help the student thrive in this class.

We're not sure how students learned back in 2022 but the computational-socio-geologists believe they were a social bunch who really enjoyed talking and explaining things to each other; they were a strong community that wanted a meaningful life by changing the world and by their connections with each other. Don’t forget to pay it forward!

Continued...

Record your office hours video about synchronization primitives that introduces the concepts and how to use POSIX versions of them. You should cover both theory and practice. There are 10 things the student asks about. (You might want cross them off as you complete them)

1. What is a critical section?
2. What is a race condition?
3. Conceptually, what is a mutex lock – when is it be useful?
4. How do I create pthread mutex lock, lock it and unlock it?
5. What is the definition of deadlock and what are the conditions for deadlock?
6. If I use 2 or more mutex locks in my program how should I prevent deadlock?
7. If I have a data-structure how do I use mutex lock(s) to make it threadsafe? (Give an example)
8. Conceptually, what, is a condition variable and why/when would it be useful?
9. How do I use pthread_cond_wait, cond_signal and cond_broadcast?
10. Conceptually, what is basic idea of the reader-writer problem?

We expect you will want to record a video of your laptop screen with audio and use a text editor as a whiteboard. However any method of recording will be acceptable (e.g. phone pointed at piece of paper). We are looking for *demonstrations of system programming understanding and competency* (i.e. things that a CS225 student would not be able to explain but a CS241 student can), such that your office hour is effective and useful.

You can record multiple parts of a video if you need a break. However don't worry about fluffs, mistakes and restarts; imagine this was a real office hours – just say oops and carry on! We expect most students will create approximately 40(ideal) - 60(max) minutes of content. We will only grade the first 60 minutes of video content; which means you only 6 minutes per item.

You can use a web search to find an easy way to record your screen and audio on your system. There are multiple methods to record a laptop screen to mp4 file or cloud (e.g. Protip: Do a test recording first and review it instead of assuming that it is working; make sure your text is large enough and legible enough to be gradeable). Ultimately you will be sharing the mp4 file or providing a URL to your cloud recording (be sure to check that the link works when not logged in as you). The university website, https://mediaspace.illinois.edu may be another useful way to record and share your video

Grading Rubric Outline:

For each of the 10 student questions,

0	missing
5	mostly incomplete or misleading or inaccurate
9	mostly helpful for the student; some minor errors
10	helpful; no errors

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 *BigCorp* finds you; time for a fast exit. You've put all of your valuable secrets (leaked GME stock prices for next year, unfinished malloc ideas, ideas for a new solar panel fabrication process, new Meme format) into a single file (it could be a text file, zip file; whatever), 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 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, if it is renamed to `encrypt` it creates an encrypted version of your secrets with every byte xor'd with a random value -

```
./encrypt mystuff.txt 1.bin 2.bin
```

It opens the file `mystuff.txt` (using `open` and `mmap`) for reading and writing, and creates two new files `1.bin` `2.bin`. As it processes the bytes of `mystuff.txt` it also overwrites the contents of `mystuff.txt` with zero bytes so that the file's original content is no longer easily recoverable. Output file `1.bin` is a sequence of random bytes (read from `/dev/urandom`). File `2.bin` contains the bytes from the original file encrypted by xor-ing each byte with the corresponding byte stored in file `1.bin`. Upon completion both output files will be the same length as the original file, the length of the original file is unchanged.
For example if the first byte of `mystuff.txt` was 0x51 and the first random value read from `/dev/urandom` was 0x42 then the "`1.bin`" would start with 0x42, "2.bin" would start with 0x13 and the first byte of `mystuff.txt` would be overwritten with 0x00.

You head out of town, pausing only to donate your old laptop. You post `1.bin` and `2.bin` – now on separate USB sticks, together with your program – from two separate locations. Your simple exclusive-or encryption using a one-time pad will still be secure if only one of the files is intercepted. Three days, two flights and a row boat later, you rename your program to `decrypt`, and recover your secrets -

```
./decrypt output.txt 1.bin 2.bin
```

This reads the two files "`1.bin` and `2.bin` (using `mmap`) and creates the output file "`output.txt`" (with contents identical to the original `mystuff.txt` file).

Upon completion it also deletes the files `1.bin` and `2.bin` from the filesystem.

For example, if the first byte of `1.bin` was 0x42 and first byte of `2.bin` was 0x13 then the first byte of the output file would be xor(0x42,0x13) = 0x51. The 2nd byte of output is recovered by xor-ing the 2nd bytes of `1.bin` and `2.bin` etc.

Write one program. 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 3 filename arguments are not provided, then innocently print "`Hello World`" and exit with value 0.
In encryption mode, use `stat` to verify that the output files do not exist. If not print "`Hello World`" and exit with value 0.
In decryption mode, use `stat` to verify that the two input files exist and are the same size. If not print "`Hello World`" and exit with value 0.

Use `open` and `mmap` to read and modify the contents of original file (encryption mode) and read the two random files 1.bin 2.bin (decryption mode).
Use `fopen` and `fputc` to create the output files in both encrypt and decrypt modes.
Use `stat`/`fstat` to verify if the files exist and/or to determine their size.
The `unlink` call will also be useful.
All other behaviors, output, handling error conditions etc. are unspecified and are not graded.

Hint: The `hexdump` program may be useful.

Grading Rubric & Submission

Add your netid as a comment to your code. For example, if your netid is 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-

10 Correctly uses the program's arguments to change behavior between hello-world, encrypt and decrypt modes basd on the process' name.
10 Zeros the original file contents so the file is the same length but the content is destroyed
10 Random bytes are sourced from `/dev/urandom`
10 Implements encryption xor-protocol and generates desired output
10 Original file content can be recreated after encrypting and decrypting
10 Encrypted files are removed from the filesystem after decryption processing is complete
10 Correctly uses `mmap` for the input file(s)
10 Correctly uses `fopen` & `fputc` for output
10 The encrypt mode verifies input files exist before processing
10 The decrypt mode verifies input file sizes are equal before processing

Part 3 of 3 (100 points, code, 60 minutes) “Saving Demo Day (my partner owes me big time)”

It's demo day but you partner’s JavaScript/C++20/python program keeps randomly crashing due to a segfault. There's no time to debug. You need a way to keep it running by automatically restarting it when it crashes.
Create a C99 Linux program `restart.c` using the same clang options as part 2 that automatically restarts the given program if it exits due to a segfault. However if it exits normally with any exit value, it should not restart. Also if the user presses CTRL-C both programs should exit.
In the example below, your restart program searches the current `PATH` environment to find the `python3` executable which is then executed as "`python3`" and passed arguments "`MyCode`" and `ABC`

```
./restart python3 MyCode ABC
```

Use `fork` and an appropriate version of `exec`. If `fork` or `exec` fails print a message to `stderr` and exit with value 1.
Hints: The `ps` and `killall` programs may be useful. It is not necessary to parse the arguments; simple pointer arithmetic is sufficient. A suitable choice of `exec` will search `$PATH` for the program for you.

Grading Rubric & Submission

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

```
// author: angrave
```

Upload your code `restart.c`; it should compile using the clang options in Part 2 on a standard CS241 VM. It will be graded on the following functionality -

10	Uses an `exec`-function to start the given program
10	Does not create unnecessary processes e.g. if the executable cannot be found
10	Prints to `stderr` and exits if `fork` or `exec` fails
10	Passes multiple (arbitrary number of) arguments to the target program
10	Correctly uses wait/waitpid and the wait macros
10	Only a segfault causes the program to be restarted; other reasons result in both programs finishing
10	Uses `fork` to start a new child process
10	Pressing `CTRL-C` causes the buggy program and the restart program to finish
10	Searches `$PATH` for the executable
10	Correctly sets the process name of the child process

That’s it - Thank you! We will provide submission details on or before Wednesday. Expect to upload your mp4 video file(s) (or provide a working link), plus `restart.c` `helloworld.c` files. If providing a URL link to your video, check that the link works in a browser when not authenticated as yourself.