CS241 Take home Final Exam (3hrs)
Spring 2022

Welcome to your CS241 Final Project. This is a "take home" open book exam with 3 questions. 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 and spoken words you include must be your own original work that you created during this exam. For example, reading aloud text from a web search result is not "in your own words" - it does not demonstrate *your* understanding and *your* competency.

You may not share this exam, your 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 (it should be 3 hours); it is up to you to track this, you can take pauses etc. Reading this pdf does not start a timer.

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 exam by Wednesday May 11th **11:00pm CT**. Submission details will be posted on Ed and/or email. There will be an approximate 59 minute grace period after 11pm to allow for slow video exports, 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.**

**Program specifications are always inexact & incomplete – review the grading rubric and ignore any remaining edge cases.**

**Do not share or post comments about the content until after grades are published.**

Submission logistics will be posted on/before Wednesday. Be sure to check your submission and allow sufficient time to upload it; we recommend you upload the `rpicheck.c` and `honey.c` files before uploading your video file(s). Your video should be mp4 format or similar variant (e.g. ".mov"). It may be compromised of multiple files e.g. you could answer each question in a separate video recording. Any common encoding video file format is acceptable; no need to transcode.

*If providing a URL link to your video(s), & check that the link(s) works in a different browser when not authenticated as yourself. TAs will grade your files; though we may use automation to short-circuit hand-grading of working code that meets the grading criteria.*

*Good luck with this final, your future interviews & courses, but mostly, in changing the world.* – L.A.

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

*“Don't Panic… Sorry!” a familiar English voice announces, “I'm just adjusting … Aha! Right!”*

*The world shifts into full view. Sensations of touch, smell, and sound flood your mind. You are desk area in 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 57)” 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, world issues– future interviews, people, conversations, all the things that will help you change the world but have not yet happened but will happen in your future. You take a deep breathe to relax.*

*“Okay,” the friendly voice continues, “I set the year back to 2023 – 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 CS 241 - ahem it's now called CS 341 - 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 2023 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 of over 2000 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 2023 office hour video. Here are the 10 things the ~~CS241~~ CS341 students ask. You might want cross them off as you complete them, or record them as separate videos. We recommend a screen cast with voice recording; the point is for you to explain something and in doing so, demonstrate your *own understanding and competency in system programming*. Students walk in. They ask the following,

1. Why does `asprintf` take a pointer to a pointer but sprintf does not? Can you help me understand why and how to use `asprintf`?
2. My server code won't listen on port 80 – why? When I try listening on port 8000 it works once but when I restart it my call to bind() fails- why? *In addition to explaining the most likely reason also explain a fix.*
3. I've heard of `fork-exec-wait` can you write and run some code to explain what is going on in each step?
4. I'm stuck; my heap allocator code keeps crashing; can you give me three suggestions on how to find the problem?
5. What is block coalescing and why do Knuth's boundary tags help?
6. Why do I need condition variables? Why not use mutex locks for all synchronization problems?
7. I understand single level page table, can you help me understand 2-level page tables?
8. I'm creating my own TCP-based protocol to transfer files. If TCP packets arrive out of order, will that scramble my file contents and why?
9. How should I use threads to speed up my filter that processes each pixel value separately?
10. Which programming assignment did you learn the most from and why?

We expect you will want to record a video of your laptop screen with audio and use a text editor as a whiteboard, but you could also record using your phone. 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 *by you* (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. Your spoken words need to be your own words, explanations, ideas and thoughts; reading aloud a Google search result or the course book is not sufficient evidence of your understanding.

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 30- 60(max) minutes of content. We will only grade the first 60 minutes of video content; which means you have average of 6 minutes per item.

There are multiple methods to record a laptop screen to a mp4 file or to the cloud. Protip: *Do a test recording first and review it instead of assuming that it is working*; verify your audio and the text is large enough and legible enough to be gradeable check that it is a recording of your screen not your laptop camera! Ultimately you will be sharing the mp4/mov file or providing a URL to your cloud recording; verify that the link works when not logged in as you. Also, mediaspace.illinois.edu may be another useful way to record and share your video (login then click on your own name to get to your Media items).

Grading Rubric (Outline):

| | |
|---|---|
| 0 | missing |
| 5 | mostly incomplete or misleading or inaccurate |
| 7 | mostly correct but a major error – not fully competent |
| 9 | mostly helpful for the student; some minor errors |
| 10 | helpful; no errors |

Part 2 of 3 (100 points, code, 60 minutes) "Did you try turning it off and on again?"

Ooops - Your RaspberryPi chat server was hacked. You quickly turn it off and mount the Pi's root filesystem on your Linux laptop as '`badrpi`'. Fortunately you also have a week old backup of the same filesystem that you mount as '`backup`'. In theory these should be almost the same. Time to recursively check `badrpi` to find the suspicious changes!

Do not follow any symbolic links to files or directories inside `badrpi`: The hackers have deliberately created symbolic links that point to a parent directories, to `/dev/urandom` etc,. Your code must not follow any symbolic links to any files or directories, but instead print out the path of the link, a colon ":" and the file/directory path contained inside the symbolic link. E.g. if a relative symbolic link "`lookhere`" in `badrpi/etc/` leads to the parent directory your program might print " `badrpi/etc/lookhere:..`"

There should be no files in `badrpi` that have a hard link count > 1. If you find such a file, print out its path e.g. "`badrpi/.trojan/usr/bin/ssh`", and on the next line, its contents as a 7-bit ASCII – up to the first 80 bytes. Print a dot "." in place of the character if the character ASCII value is < 32 or > 126.

Print out the path e.g. "`badrpi/etc/payload.tar.gz`" to regular files (but not directories) that are either i) *only* in `badrpi` with no corresponding file in `backup` or ii) if the file in `badrpi` has a different modification time than the corresponding file in the `backup` mount.

Your code "`rpicheck.c`" will compile without warnings or errors on a standard Ubuntu image (e.g. the CS241 VMs) and you run it from your home directory as root, giving it the two starting directories,

```
clang -O0 -Wall -Wextra -g -std=c99 -D_GNU_SOURCE -DDEBUG rpicheck.c
sudo ./a.out badrpi backup
```

If the two arguments are missing or are not directories then print a usage message and exit.
Include your netid as a comment to the top of your code, e.g., if your netid is angrave, write

```
// author: angrave
```

Upload your code, `rpicheck.c`; it should compile using the above clang options on a standard CS241 VM. It will be hand-graded if it does not pass all automated tests. To test your code you will need to create your own test data (files, sub-directories, symbolic links, hard links).

Rubric*: 10 points each (subject to minor modifications)

- Source code includes netid author comment
- If the program arguments are missing or not directories, print a usage message and exit.
- Uses the directories for the badrpi and backup from the program arguments
- Does not follow or attempt to open the target of symbolic links in `badrpi` to directories
- Does not follow or attempt to open the target of symbolic links in `badrpi` to files
- Prints the filename (preferably the path) of a symbolic link and its contents
- Prints the filename (preferably the path) to new files in `badrpi` not in the backup
- Prints the filename (preferably the path) of files in `badrpi` with a different modification time
- Prints the full path (e.g. `badrpi/subdir1/2/a.txt`) for at least one of the above outputs
- Prints the 80 char line of ascii contents of files with more than one hard link

*Grading: All other behaviors, output, handling missing arguments, files, error conditions etc are *unspecified* and *are not graded*; do not ask about them. Write code comments about your assumptions and then answer accordingly. You may review your own prior work, use man pages, CS241 content and stackoverflow.com - cite your sources (URLs) - but you must type new code. You may not copy-paste any prior code into your `rpicheck.c` file.

Part 3 of 3 (100 points, code, 60 minutes) "A multi-threaded honey pot"

To catch an ongoing web attack you quickly write a multi-threaded program `honey.c` that listens and accepts TCP/IPv4 concurrent connections on a specified port and logs the request header data. Quickly now; the attacker may leave soon! Over the next hour there could be 1000s of total requests but never more than a few at the same time so close file descriptors and file handles when no longer needed.

Your main thread will set up the TCP/IPv4 server and accept new connections. It creates a new thread for each new client connection. Pass only a heap address of a client's fd integer to the new thread. Each thread, *after* reading the HTTP headers from the client, responds with the current time using a simple HTTP response. It also secretly logs all the request header data to a file for later forensic analysis. The log filename should be the client IP, `octet1.octet2.octet3.octet4` (e.g. "`127.0.0.1`"). Append the request header data sent by each client to the log file (read up to 8KB or the first empty line, `\r\n\r\n`).

The response should be a valid HTTP response: Send a "`HTTP/1.0 200 OK`" message, a header to specify a plain text MIME type, an empty line, and the current date & time in a human readable format. Then `close` the connection. Compilation options and example testing on port 8000 are shown below.

```
$ clang -O0 -pthread -Wall -Wextra -g -std=c99 -D_GNU_SOURCE -DDEBUG honey.c
$ ./a.out 8000 &
Logging data to file 192.17.1.23

$ cat 192.17.1.23
GET / HTTP/1.1
User-Agent: curl/7.29.0
Host: honey.fake.com:8000
Accept: */*
```

Meanwhile on 192.17.1.23 …
*$* `curl honey.fake.com:8000`
`Fri May  6 18:06:33 2022`

Start with the code on the next page. It will need some changes ("TODO") and completion. Additional debugging output to `stdout`/`stderr` is allowed. Note `ctime()` and `inet_ntoa()` are not thread-safe; use a mutex lock to protect your critical sections. Also use a mutex lock to prevent two concurrent requests from creating overlapping data in the same log file. You can assume clients send a valid HTTP request and receive for the full response data (i.e. they do not close or shutdown the connection early).

For the given code, insert a comment to explain what differences in behavior/output will happen if the mutex lock was not added to correctly overcome the missing thread-safety of `inet_ntoa`.

You may refer to and copy-paste code from stackoverflow.com, man pages, CS241 materials and your own previously written code but not other student's work, nor solutions to this question. *You must cite your sources/(URLs) for any code that you did not specifically write for this final.*

**Rubric* (subject to minor modification): 10 points each**

- Author line includes your netid.
- Creates a TCP/IPv4 server on the specified port (`REUSEADDR/PORT` not graded).
- Uses pthreads to concurrently accept and process multiple concurrent requests.
- Code comment explains the possible result(s) of a missing lock on `inet_ntoa's` critical section.
- Appends the request headers to a log file.
- Client data is stored in a file where the filename is the connecting client's IPv4 octet address.
- Response is a valid HTTP response with the correct MIME type.
- When tested with a client (e.g. `curl`), client displays the current date & time as the response.
- Critical sections (e.g. `ctime`, `inet_ntoa`, and logging) are protected against race conditions.
- Reads headers (up to 8KB of data) that arrive over multiple calls to `read()`.

```
// Start with the following code
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <pthread.h>
#include <stdlib.h>

// author: TODO-1-YOUR-NETID-HERE
// Cite all your sources! This is the starting code of the Spring 2022 CS241 Final

// "It's dangerous to go alone, take this!" - Legend of Zelda (1986)
pthread_mutex_t duck = PTHREAD_MUTEX_INITIALIZER;

void quit (const char *mesg) {
  perror (mesg);
  exit (1);
}

void* handle_client (void *fd_ptr) {
  pthread_detach (pthread_self ()); // pthread_join not needed

  int fd = *(int *) fd_ptr;
  free (fd_ptr); fd_ptr = NULL;

  struct sockaddr_in address;
  socklen_t addressLength = sizeof (address);

  int ok = getpeername (fd, (struct sockaddr *) &address, &addressLength);

  // TODO-2: inet_ntoa is not thread-safe! So make this code safe using the duck.
  // inet_ntoa may return a pointer to the same memory location
  // TODO-3: Answer below: If no lock was used to protect this critical section
  // what errors/changes would you expect to see in the program's behavior or output?
  //
  //
  char *filename = ok == 0 ? inet_ntoa (address.sin_addr) : "unknownip";
  // you do not need to call free on the filename

  printf ("Logging data to file %s\n", filename);
  FILE *file = TODO-4 open filename for append;

  char buffer[8 * 1024 + 1];
  size_t numreadtotal = 0;

  // maybe useful later,
  // if (strstr (buffer, "\r\n\r\n")) break
  // fwrite (buffer, 1, , )
  // time_t, time() and ctime() are great but look out for thread safety.
  // HTTP request and response headers both use \r\n as a line ending.
  // TODO-5 Prevent a race condition when two threads write to the same log file:
  // - Concurrent requests must not have overlapping content in the log file.
```

*Rubric: All other behaviors, output, handling missing arguments, files, error conditions etc are *unspecified* and *are not graded*; do not ask about them. Instead write a code comment about your assumptions and answer accordingly.