CS 3214 Spring 2022 Final Exam Solution

May 9, 2022

Contents

Rules

- This exam is open book, open notes, and open Internet, but in a read-only way.
- You are not allowed to post or otherwise communicate with anyone else about these problems. This includes sites such as chegg.com, which will be monitored. The open Internet stipulation does not apply to such sites.
- You are required to cite any sources you use, except for lecture material, source code provided as part of the class material, and the textbook. Failure to do so is an Honor Code violation.
- If you have a question about the exam, you may post it as a *private* question on Discourse addressing the instructors (account names: @Godmar_Back and @djwillia and @Liting_Hu). If your question is of interest to others, we will make it public as a clarification or hint.
- During the test period, you may not use Discourse to post other questions or reply to questions posted there.
- Any errata to this exam or hints given will be published prior to 12h before the deadline, please check Discourse under tag `final`.

1 Networking (30 pts)

1.1 Know Your Internet (8 pts)

Find out if the following statements related to networking are true or false. If true, just write **true**. If false, write **false** and provide the corrected statement.

(a) By separating the business logic of an application from its presentation logic, a multi-tier architecture makes Internet applications much more flexible to changes.

True.

(b) The dominant transport layer protocol today is called HTTP.

False. Today's dominant transport layer protocol is TCP (or TCP/UDP).

(c) When determining the destination socket for a received IP datagram that carries a TCP segment, the receiving host must consider both the sender's IP address and port and the destination's IP address and port.

True.

(d) In TCP, the number of bytes that one party may send before receiving an acknowledgement from the other side is fixed by design.

False. The TCP window size is configurable.

(e) Cryptographically signed tokens allow a server to validate a user's identity even when the user provided their authentication credentials to a different server (called an identity provider.)

True.

(f) The network layer uses the transport layer to ensure the reliable delivery of network packets.

False. The transport layer ensures reliable delivery in the presence of a network layer that provides only unreliable delivery.

(g) If the transition from IPv4 to IPv6 goes according to plan, IPv4 traffic will eventually fade out even before users disable IPv4 on their networks.

True.

(h) The design of custom transport layer protocols such as QUIC requires changes to the OS kernel because all network protocol processing takes place inside the kernel.

False. QUIC, for instance, is implemented at the user level inside a library in the Chrome browser.

1.2 Read It Online (6 pts)

In a blog post[1] designed to attract search engine traffic, an author writes the following about the purported differences between HTTP and TCP:

The Main Differences Between HTTP and TCP

- *HTTP typically uses port 80 – this is the port that the server "listens to" or expects to receive from a Web client. TCP doesn't require a port to do its job.*
- *HTTP is faster in comparison to TCP as it operates at a higher speed and performs the process immediately. TCP is relatively slower.*
- *TCP tells the destination computer which application should receive data and ensures the proper delivery of said data, whereas HTTP is used to search and find the desired documents on the Internet.*

[1]HTTP vs TCP - What's the difference?

- *TCP contains information about what data has or has not been received yet, while HTTP contains specific instructions on how to read and process the data once it's received.*
- *TCP manages the data stream, whereas HTTP describes what the data in the stream contains.*
- *TCP operates as a three-way communication protocol, while HTTP is a single-way protocol.*

In these six bullet points, the author managed to bury several contortions, misrepresentations, or downright misinformation.

Select *two* of these erroneous or misrepresented statements. Briefly explain why they do not make sense and provide a correct explanation.

[Solution.]

- TCP does require the use of port numbers to allow the OS to identify the process(es) that are the endpoints of the connection.
- Because HTTP (usually) runs on top of TCP the statement that "HTTP is faster in comparison to TCP" is meaningless.
- HTTP does not support "search(ing) and find(ing) the desired documents on the Internet" - it is a protocol used to retrieve objects based on their known URL. There are no search facilities built into HTTP.
- HTTP does not "contain specific instructions on how to read and process the data once it's received." (HTTP responses contain a Content-Type header, but the client decides how process content of each type.)
- TCP is not a "three-way communication protocol," it has a three-way handshake used by the two parties during the connection establishment phase.
- HTTP is not "single-way protocol," it involves two parties as well. It is also fairly symmetric, allowing both uploads (via PUT or POST) and retrievals (via GET).

1.3 A ROT13 Server (16 pts)

The ROT13 protocol is a new request/response protocol to retrieve files and send them encrypted over the network using the ROT13 substitution cipher.

Details of the protocol are still under development, but you were able to secure an strace log of a transaction performed by a prototype ROT13 server. You were also able to learn how a ROT13 client works because you witnessed an invocation using the netcat program (nc) and Unix pipes, which looked like this:[2]

[2]Note that `echo` appends a single newline character here.

```
$ echo 'ROT13 secret.txt' | nc hazelnut 20000

Tbbq yhpx jvgu gur svany!
```

Here, `secret.txt` is the name of the file to be retrieved in ROT13 encryption. This file must exist in the directory in which the server ran.

The strace of the server, when run with

```
strace -v -s 1024 -o log ./rot13 20000
```

is shown below:

```
write(1, "calling getaddrinfo\n", 20)   = 20
[... system calls made by getaddrinfo are elided... ]
write(1, "getaddrinfo returned\n", 21)  = 21
socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, {sa_family=AF_INET6, sin6_port=htons(20000), sin6_flowinfo=htonl(0),
  inet_pton(AF_INET6, "::", &sin6_addr), sin6_scope_id=0}, 28) = 0
listen(3, 500)                          = 0
accept(3, {sa_family=AF_INET6, sin6_port=htons(47698), sin6_flowinfo=htonl(0),
  inet_pton(AF_INET6, "::ffff:192.168.5.104", &sin6_addr), sin6_scope_id=0},
  [128 => 28]) = 4
setsockopt(4, SOL_TCP, TCP_NODELAY, [1], 4) = 0
write(2, "Accepted connection from ::ffff:192.168.5.104:47698\n", 52) = 52
read(4, "ROT13 secret.txt\n", 4097)     = 17
openat(AT_FDCWD, "secret.txt", O_RDONLY) = 5
fstat(5, {st_dev=makedev(0, 0x42), st_ino=19682582745, st_mode=S_IFREG|0644,
 st_nlink=1, st_uid=14913, st_gid=16151,
 st_blksize=1048576, st_blocks=8, st_size=27, st_atime=1651687994
 /* 2022-05-04T14:13:14.384810192-0400 */, st_atime_nsec=384810192,
 st_mtime=1651687994 /* 2022-05-04T14:13:14.384810192-0400 */,
 st_mtime_nsec=384810192, st_ctime=1651687994
 /* 2022-05-04T14:13:14.384810192-0400 */, st_ctime_nsec=384810192}) = 0
mmap(NULL, 27, PROT_READ|PROT_WRITE, MAP_PRIVATE, 5, 0) = 0x7f9095f19000
write(4, "\nTbbq yhpx jvgu gur svany!\n", 27) = 27
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

(this strace has irrelevant details elided, you can access the full strace on the class website).

Reconstruct `rot13.c` so that it implements the ROT13 service.
Your implementation should be subject to the following conditions:

- When started, your server should accept a single command line argument which is a string denoting the port number on which the server should listen. (In the example above, the server was invoked with `./rot13 20000`.)
- From clients, your server should accept ROT13 requests for any filename consisting of alphanumerical characters (not including slashes).
- Your server should accept and handle a single client connection and then exit.
- Your server must use `mmap()` to access the file content, but it must not change the file on disk. (Hint: investigate the difference between `MAP_SHARED` and `MAP_PRIVATE` and use the correct flag.)
- To save time, you should use the following routine to perform the ROT13 encryption:

```
// In-place ROT13 cipher
static void
rot13(unsigned char *b, size_t n)
{
    for (size_t i = 0; i < n; i++) {
        switch (b[i]) {
        case 'a'...'z':
            b[i] += 13;
            if (b[i] > 'z') b[i] -= 'z' - 'a' + 1;
            break;

        case 'A'...'Z':
            b[i] += 13;
            if (b[i] > 'Z') b[i] -= 'Z' - 'A' + 1;
            break;
        }
    }
}
```

- You may use `socket.c` and `socket.h`. A copy will be provided on the class website, which is a slightly modified version of the one used in project 4.
- For the purposes of this problem, you may assume a friendly (non-malicious) client, so you do not need to check for protocol violations or IDOR attacks.
- Your server must handle short reads.

[Solution.]

```
// solution

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include "socket.h"

// In-place ROT13 cipher
static void
rot13(unsigned char *b, size_t n)
{
    for (size_t i = 0; i < n; i++) {
        switch (b[i]) {
        case 'a'...'z':
            b[i] += 13;
            if (b[i] > 'z') b[i] -= 'z' - 'a' + 1;
            break;

        case 'A'...'Z':
            b[i] += 13;
            if (b[i] > 'Z') b[i] -= 'Z' - 'A' + 1;
            break;
        }
    }
}

static void *
server(void *_arg)
{
    int client = socket_accept_client(accsoc);
    int accsoc = socket_open_bind_listen((char *)_arg, 500);
    #define MAX 4096
    char buf[MAX+1] = { 0 };
    size_t bread = 0;
    size_t totalread = 0;
    while (totalread < MAX &&
          (bread = read(client, buf + totalread, sizeof buf - totalread)) > 0) {
        totalread += bread;
        if (memchr(buf, '\n', totalread))
            break;
    }

    char *path = NULL;
    sscanf(buf, "ROT13 %ms\n", &path);
    printf("ROT13 `%s'\n", path);
    int fd = open(path, O_RDONLY);
    free(path);
    struct stat st;
    fstat(fd, &st);
    void *addr = mmap(NULL, st.st_size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
    rot13(addr, st.st_size);
    write(client, addr, st.st_size);
    close(client);
    close(accsoc);
    return NULL;
}

int
main(int ac, char *av[])
{
    server(av[1]);
}
```

2 Virtual Memory (18 pts)

2.1 A Case of Cache Busting (10 pts)

The well-known performance engineer Brendan Gregg reports in a blog post from Aug 30, 2021[3] about a performance incident at Netflix.

[3]You can find the full blog post (which includes pictures and graphs) here: Analyzing a High Rate of Paging

Gregg reports:

> A service team was debugging a performance issue and noticed it coincided with a high rate of paging. (...)
>
> The microservice managed and processed large files, including encrypting them and then storing them on S3. The problem was that large files, such as 100 Gbytes, seemed to take forever to upload. Hours. Smaller files, as large as 40 Gbytes, were relatively quick, only taking minutes.
>
> A cloud-wide monitoring tool, Atlas, showed a high rate of paging for the larger file uploads. (...) the iostat(1) tool showed a high rate of disk I/O during a large file upload (...)
>
> There [was] not much memory left, 349 Mbytes, but more interesting is the amount in the buffer/page cache: 48,643 Mbytes (48 Gbytes). This is a 64-Gbyte memory system, and 48 Gbytes is in the page cache (the file system cache).
>
> This shows many cache misses, with a hit ratio varying between 6.5 and 74%. I usually like to see that in the upper 90's. This is "cache busting." The 100 Gbyte file doesn't fit in the 48 Gbytes of page cache, so we have many page cache misses that will cause disk I/O and relatively poor performance.
>
> The quickest fix is to move to a larger-memory instance that does fit 100 Gbyte files. The developers can also rework the code with the memory constraint in mind to improve performance (e.g., processing parts of the file, instead of making multiple passes over the entire file).

Answer the following questions:

(a) (3 pts) When the system experienced the "performance issue," did it likely make full use of its CPU(s)? Justify your answer.

[Answer.] No, it did not make full use of its CPUs.

As a matter of fact, Gregg shows the percentage of time the CPUs were idle in part 2, ranging from 92.18% to 54.11%. The comment "Reads usually have apps waiting on them" indicates this as well - processes are in the blocked state because they are waiting for data to be retrieved from disk.

(b) (5 pts) How did Gregg conclude that the workload in question must have "made multiple passes over the entire file?" Relate your answer to the decisions that Linux's page replacement algorithm must have made.

[Answer.] Based on the diagnostic tools, Gregg observed that the program was spending much of its time being blocked reading data from disk in order to process it. This was considered a "performance issue" so we can rule out that cold misses (bringing the on-disk data into the (page cache's) memory upon first access) was the issue of concern here. Instead, repeated accesses to the data must have occurred, and the data must have been evicted between these accesses (so that it needed to be refetched again). This is consistent with the behavior of a page replacement algorithm that uses a LRU (or LRU-like) replacement policy in the presence of multiple (looping) sequential accesses to the file.

For example, consider an access pattern of 1-2-3-4-1-2-3-4-1-2-3-4 for a memory of size 3: under an LRU replacement policy, it causes 4 cold (for 1-2-3-4) and 8 capacity misses (the access to 4 evicts 1, 1 evict 2, 2 evicts 3, and so on.)

Another way to put this is that the "working set" of this workload exceeded the size of the physical memory, meaning that it is impossible to execute the workload efficiently.

(c) (2 pts) When Gregg talks about "moving to a 100GB" instance, is he referring to the size of virtual memory or the size of physical memory?

[Answer.] This is referring to the size of physical memory. (Or, if this is a virtual machine instance, a guaranteed allotment of 100GB of physical memory.)

2.2 MAP_POPULATE (8 pts)

A CS3214 student was using `mmap` to read from a file residing on a filesystem on a magnetic disk for processing in their program. The file is large, but not so large that it does not fit in main memory. The student, having learned that `man` pages are very useful, began reading about different flags to `mmap`, and found one called `MAP_POPULATE`:

```
MAP_POPULATE (since Linux 2.5.46)
       Populate (prefault) page tables for a mapping.  For a file
       mapping, this causes read-ahead on the file.  This will
       help to reduce blocking on page faults later.  The mmap()
       call doesn't fail if the mapping cannot be populated (for
       example, due to limitations on the number of mapped huge
       pages when using MAP_HUGETLB).  MAP_POPULATE is supported
       for private mappings only since Linux 2.6.23.
```

(a) (4 pts) Suppose the student's program was a data processing program that `mmap`ed the large file and then accessed it sequentially. In theory, should the student specify the `MAP_POPULATE` flag to `mmap`? What effect (if any) would specifying `MAP_POPULATE` have on the performance of the program? Justify your answer.

ANSWER: Yes, if the entire file will eventually fault in, MAP_POPULATE will enable the file to be loaded in without paying page fault costs to

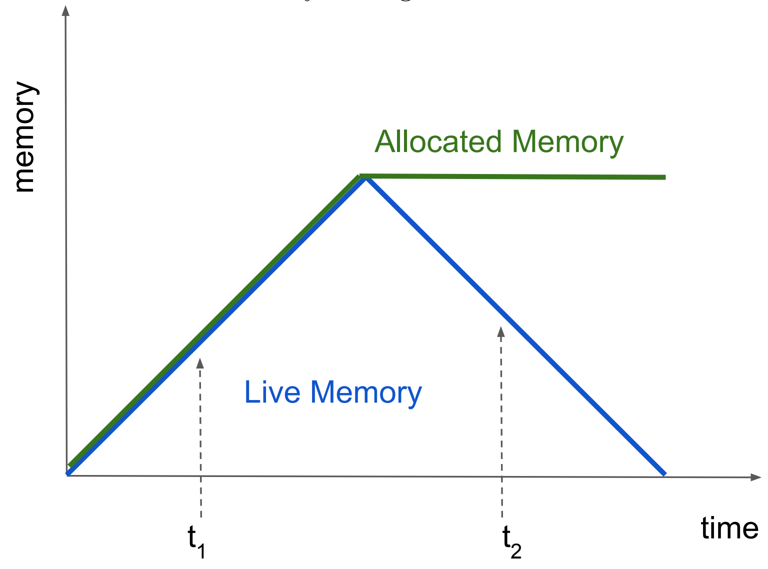
lazily map in every page in the file, thereby enabling a potential speedup of the total runtime of the program (as compared to the case without MAP_POPULATE).

(b) (4 pts) Suppose that instead, the student's program only performed a handful of small, random reads in the large file that was `mmap`ed. In theory, should the student specify the `MAP_POPULATE` flag to `mmap`? What effect (if any) would specifying `MAP_POPULATE` have on the performance of the program? Justify your answer.

ANSWER: No, in this case, MAP_POPULATE would slow down the total runtime of the program because only a relatively small amount of the file actually needs to be faulted in. MAP_POPULATE would force the entire file to be fetched unnecessarily.

3 Automatic Memory Management (12 pts)

Consider the following memory allocation profile for a program in some language that uses a form of automatic memory management:



(a) (6 pts) Write a program in a language of your choice that would produce this memory allocation time profile. Assume that no garbage collection takes place.

[Answer.] There are many ways to write this. In a Java-like language, the object could be allocated and stored in an array, and subsequently become garbage like so:

```
Object [] obj = new Object[N];
for (int i = 0; i < N; i++)
```

```
        obj[i] = new Object();
    for (int i = 0; i < N; i++)
        obj[i] = null;
```

A recursive function should as this one (assuming no optimizations are applied by the compiler) could produce a similar pattern:

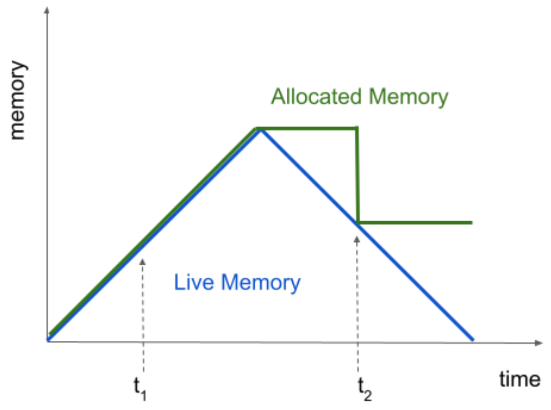
```
void recurse(int n) {
    var obj = new Object();
    if (n > 0) recurse(n-1);
}
```

(b) (3 pts) Now consider the point in time t_1. If garbage collection took place at t_1, would it affect the allocation/time profile and if so, how?

[Answer.] It would not. At time t_1, there is no garbage, so a garbage collector could not free any allocated memory. Garbage is the difference between allocated and live memory.

(c) (3 pts) Now consider the point t_2. If garbage collection took place at t_2, would it affect the allocation/time profile and if so, how?

[Answer.] It would. At t_2, a garbage collector would free all memory that is no longer live, thus the remainder of the green line would shift down to the y coordinate of the blue line at t_2.



4 Heap Overwrites (18 pts)

A programmer has written a secure embedded program to remotely control the door locks on a secure facility. In the text section, the program contains numerous functions, including one called `print_username` which prints a user's name to the screen:

```
void print_username(struct N *n) {
  printf("%s", n->name);
}
```

Another interesting function in the code is called `unlock_door`, which resides at the virtual address of 0x80020300.

The program is using a simple explicit dynamic memory manager that uses implicit lists with boundary tags and immediate coalescing to manage the heap. A first fit selection policy is used. After running the program for some time, the concrete heap state is as shown in Figure 1.

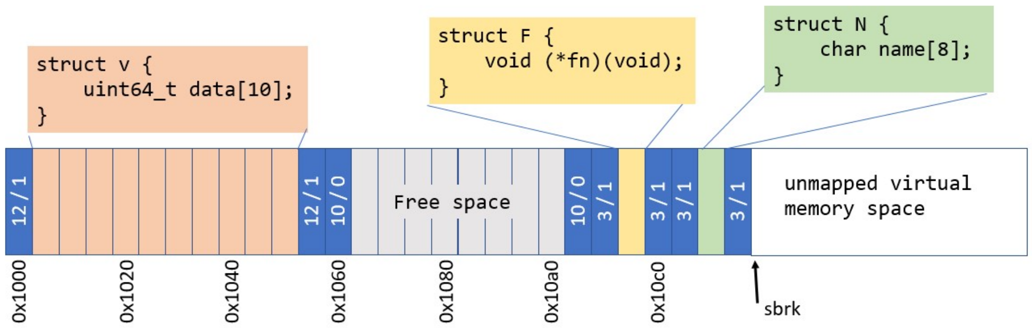


Figure 1: The heap. Each box is 64-bits in size. Boundary tags (in blue) are displayed in "size/alloc" format, with 63 bits forming a size field and one bit specifying whether the chunk is allocated (1) or free (0).

Unfortunately, an attacker has gained the ability to perform an out of bounds write for the array contained in the orange object of type `struct v` on the heap. The vulnerable line is here, where `v` is a pointer to this object:

```
v->data[i] = val;
```

where val is a 64 bit attacker controlled value and, due to poor input sanitization, the attacker also has control of the index i.

Your task is to answer the following questions about different scenarios of out-of-bounds writes on the heap:

(a) (3 pts) What will be the immediate result if the attacker causes the following to execute:

```
v->data[512] = 0xdeadbeef;
```

ANSWER: the program will immediately receive a segfault, because it is writing in memory past the system break onto a page that has not been mapped into the address space.

(b) (3 pts) What will happen on the next call to `malloc(8)` if the attacker causes the following to execute:

```
v->data[11] = {.size=1000, .alloc=1};
```

ANSWER: The chunk that was previously free will now appear allocated and very large, so on implicit list traversal, malloc will jump past the system break and receive a segfault when trying to read the header of the next chunk.

(c) (3 pts) What will happen on the next call to `malloc(8)` if the attacker causes the following to execute:

```
v->data[14] = 0xdeadbeef;
```

ANSWER: Nothing will happen; this write is in the middle of a free chunk. It eventually be overwritten when the newly allocated object is initialized.

(d) (3 pts) What will happen if the program immediately frees the yellow object F after the attacker causes the following to execute:

```
v->data[20] = {.size=20; .alloc=0}
```

ANSWER: Upon immediate coalescing, part of the orange object will be erroneously marked as free with a newly written boundary tag, corrupting the object.

(e) (3 pts) What will happen the next time the program calls `print_username` with a pointer to the green object N if the attacker causes the following to execute:

```
v->data[25] = 0x2144454B434148;
```

Assume that the architecture uses the Little Endian format to store multibyte integers.

ANSWER: The data in the green object will be silently overwritten so that the string printed out from `print_username` will read "HACKED!"

(f) (3 pts) What will happen the next time the function pointer in object F is called if the attacker causes the following to execute:

```
v->data[22] = 0x80020300;
```

ANSWER: The attacker has managed to modify a function pointer in the yellow object. When using the function pointer, the control flow of the program will now call the `unlock_door` function instead of what it was supposed to call!

5 Virtualization and Containers (9 pts)

Find out if the following statements related to virtual machines and/or containers are true or false.

If true, just write **true**. If false, write **false** and provide the corrected statement.

(a) Running virtual machines can migrate from one physical server to another without disturbing normal operations or causing noticable downtime.

True.

(b) Some virtual machine monitors translate a guest kernel's binary machine code instead of directly executing it in deprivileged mode.

True.

(c) Container images are static bundles of files that describe the environment in which a container instance will execute when it is started.

True.

(d) Container designs typically use a virtualized instruction set architecture (ISA) that is different from the ISA associated with the hardware of the machine running the container engine.

False. Containers do not modify the instruction set architecture.

(e) OS kernels often include accommodations that allows them to execute more efficiently when running inside a virtual machine.

True.

(f) Layers in container filesystems refer to the 7-layer design as specified in the Open Systems Interconnection model.

False. The OSI model is about layered design of networks.

(g) Before running your containerized application on another machine such as provided by a commercial container service, you must ensure that the provider of the target system has installed all of the necessary libraries that your application requires.

False. One of the major value propositions of containers is that all library dependencies ship within the container image.

(h) Both virtual machines and containers are equally good tools for facilitating debugging operating system kernels.

False. Virtual machines facilitate debugging operating system kernels because guest kernel crashes do not affect the host kernel. With containers, there is only one kernel; thus kernel crashes take down the host making debugging difficult.

(i) On a per instance basis, containers tend to need fewer physical resources such as memory when compared to virtual machines.

True.

6 Container Woes (13 pts)

A CS3214 student learned about Linux containers and how they have their own private `chroot` filesystem. So, the student decided to play around with containers from scratch, by creating a new directory tree, containing the binary for a bash shell to poke around inside the container with.

Here are the steps the student took. First, they copied `bash` from their local filesystem into a directory that served as a staging area for what would become the container's `chroot` filesystem:

```
$ mkdir -p mychroot/bin
$ cp /bin/bash mychroot/bin/
```

The student then created the following Dockerfile in the working directory and built a container named `mycontainer` using the Docker CLI. (Note: the `docker build` command is taking a *build context*, which is where the Dockerfile and staging directory (`mychroot`) reside. In this case it is specified as the current directory '.')

```
$ cat <<EOF > Dockerfile
FROM scratch
```

```
COPY mychroot/bin/bash /bin/bash
EOF
$ docker build -t mycontainer .
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
STEP 1/2: FROM scratch
STEP 2/2: COPY mychroot/bin/bash /bin/bash
COMMIT mycontainer
--> bcc7380ddae
Successfully tagged localhost/mycontainer:latest
bcc7380ddae7ead1b539c4b2b36e5c38f4d2b4ee1f672d392b5cca9c7ca1ef3d
```

However, when running the container, the student encountered a bizarre error:

```
$ docker run --rm -it mycontainer "/bin/bash"
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
standard_init_linux.go:228: exec user process caused: no such file or directory
```

The student double-checked the mychroot directory from which the container chroot was built, and sure enough `bash` was there, as follows:

```
$ find mychroot/
mychroot/
mychroot/bin
mychroot/bin/bash

$ file mychroot/bin/bash
mychroot/bin/bash: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=0abac065ab1eb79dff27b1278df95da568034daa, stripped

$ ldd mychroot/bin/bash
        linux-vdso.so.1 (0x00007ffee4772000)
        libtinfo.so.6 => /lib64/libtinfo.so.6 (0x00007fab87f64000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007fab87d60000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fab8799b000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fab884af000)
```

(a) (5 pts) Given what you know about dynamically linked executables and the current contents of the student's `mychroot` directory, diagnose the problem. Then, fix the problem, providing the contents of the staging directory (`mychroot`) in `answer.txt` and your modified Dockerfile. For this question, the modified Dockerfile must still use `FROM scratch`, and you may not recompile the bash executable.

ANSWER: The issue is that the libraries were missing from the container chroot. Adding them back in allows bash to run in the from scratch container:

```
$ find mychroot/
mychroot/
mychroot/bin
mychroot/bin/bash
mychroot/lib64
mychroot/lib64/libtinfo.so.6
mychroot/lib64/libdl.so.2
mychroot/lib64/libc.so.6
mychroot/lib64/ld-linux-x86-64.so.2
$ cat Dockerfile
FROM scratch
COPY mychroot/bin/bash /bin/bash
COPY mychroot/lib64/libtinfo.so.6 /lib64/libtinfo.so.6
COPY mychroot/lib64/libdl.so.2 /lib64/libdl.so.2
COPY mychroot/lib64/libc.so.6 /lib64/libc.so.6
COPY mychroot/lib64/ld-linux-x86-64.so.2 /lib64/ld-linux-x86-64.so.2
```

(b) (4 pts) After solving the above problem, the student quickly found that their "from scratch" container was not very interesting. They could not even run `ls`, though `pwd` and `exit` worked:

```
$ docker run --rm -it mycontainer "/bin/bash"
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
bash-4.4# ls
bash: ls: command not found
bash-4.4# pwd
/
bash-4.4# exit
exit
```

Recalling what you know about shells from project 1, provide an explanation why `ls` failed but `pwd` and `exit` succeeded.

ANSWER: Even after adding the libraries in the previous step, the student did not add other executables, such as ls. pwd and exit, on the other hand, are builtins in bash, so for these it was unnecessary to add new files to the container.

(c) (4 pts) After some time, the student added some more tools to the "from scratch" container. They remembered that they had to do something later and quickly jotted down a note before exiting the container:

```
$ docker run --rm -it mycontainer "/bin/bash"
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
bash-4.4# echo "remember to do the thing" > todo.txt
bash-4.4# ls -l todo.txt
-rw-r--r-- 1 0 0 25 May  5 23:10 todo.txt
bash-4.4# exit
```

Later, the student ran the container again to retrieve the note.

```
$ docker run --rm -it mycontainer "/bin/bash"
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
bash-4.4# ls -l todo.txt
```

What output will the student see, and why did they receive that output?

ANSWER: The output will be:

```
ls: cannot access 'todo.txt': No such file or directory
```

What happened is that a new filesystem layer is created atop the base image for each run of the container (and discarded afterwards) so the todo file is now gone!