CS 3214 Fall 2023 Final Exam Solutions

December 13, 2023

1 Networking (35 pts)

1.1 Know Your Internet (10 pts)

Determine if the following statements related to networking are true or false.

(a) Routing protocols are distributed protocols whose goal is to populate each participating router's forwarding table.

[✓ true] / ☐ false.

(b) Internet core routers must track TCP connections (connection establishment and teardown) in order to know when associated resources can be released.

☐ true / [✓ false].

(c) Wide Area Networks (WAN) have higher transmission delay than Local Area Networks, but the propagation delay is generally smaller.

☐ true / [✓ false].

(d) Both IPv4 and IPv6 addresses are grouped based on shared prefixes of their bitwise representations.

[✓ true] / ☐ false.

(e) Internet Service Providers are required to budget for enough upstream bandwidth such that all customers can make full use of the available bandwidth of the "last leg" link connecting them to their ISP.

☐ true / [✓ false].

(f) Transport layer protocols such as TCP or UDP include a container identifier to identify to which container incoming packets should be delivered.

☐ true / [✓ false].

(g) As an Internet link approaches saturation, the latency of packets sent across this link will increase.

[✓ true] / ☐ false.

(h) When multiple TCP connections traverse across the same link in a network and if that link becomes saturated, the connections will obtain a roughly equal amount of bandwidth.

[✓ true] / ☐ false.

(i) Multiple established TCP connections can be multiplexed over the same socket.

☐ true / [✓ false].

(j) Because of the difference in their transport layer representation, HTTP/2 connections cannot be proxied across HTTP reverse proxies unless the origin server also supports at least HTTP/2.

☐ true / [✓ false].

1.2 Visiting a Webpage (25 pts)

(a) Fill in the blanks.

When a user types a URL into their browser's address bar, the browser must first resolve the hostname or domain name to an IP address before it can create a transport layer connection to the server. It uses a protocol called DNS (Domain Name Service) for the resolution step, and then it uses a transport layer protocol called TCP to create a connection. Creating such a connection requires the exchange of 2 packets, adding roughly one roundtrip time (RTT) of delay before the first bit of information can be sent.

The browser will send a HTTP message, which consists of a start line, an optional set of headers followed by a blank line and an optional body. In the case being considered (a user typing into the address bar), the start line will contain the HTTP verb GET.

Once the server receives the request, it will parse it to extract the verb, which is followed by the path and the HTTP version. It retrieves the object requested from the file system or a database and prepares an HTTP response which includes a 3-digit status code, an optional set of headers and a body with the actual object. The size of the object is described by the Content-Length header. The server should also specify the type of the content being sent, which for most websites is HTML (or text/html).

Once the client receives the object, it will parse it and will now learn which other assets are needed before the page can be displayed to the user. Such assets may include Javascript files, stylesheets, or image files, to name 3 examples.

Before retrieving a needed asset, the browser will check in its cache to see if the object was already retrieved before. If not, and if the asset is provided by the same server as the webpage, the client can request it via the already established connection if the server supports an HTTP/1.1 feature called persistent connections.

The server may also send code to the client which is written in the Javascript language. Such code, when run, can make requests to the server. The server will respond to those requests by sending objects represented as, for instance, JSON (or XML or HTML). (1 example suffices.)

Recent browsers/servers may use HTTP/3, which uses the transport layer protocol QUIC, which runs on top of UDP and implements its own connection management and congestion control (Give 2 examples).

2 Virtual Memory (33 pts)

2.1 Fill in the Blanks (13 pts)

In modern computer systems, the memory system uses virtual addresses which are translated to physical addresses. Memory is divided into equal-sized ranges called pages which on the x86 architecture are typically 4KB large.

This translation process is performed by the memory management unit (MMU), which uses a data structure called a page table to that end. The content of this data structure is controlled by the OS (or kernel). Once a translation from this data structure has been retrieved, it is stored in the TLB (Translation Lookaside Buffer) for fast access should the same translation need to be performed again.

If there is a data access but no matching translation was entered into the data structure, a page fault is said to occur. When that happens, the OS will be entered and it may take one of two actions: first, it may terminate the (offending) process. Or, it may allocate physical memory and provide the expected data to facilitate access. The second case is commonly referred to as on-demand paging (or lazy loading, or page-in). Examples of a process's memory areas that are subject to this mechanism include heap memory, mmap'd files, shared libraries, stack, text segment, data segment, or bss segment. (Needed only 3 examples.)

The OS also has the ability to rededicate physical memory for different uses. To that end, it uses a page replacement (or just replacement) policy to decide which memory should be rededicated when needed. Overall, virtual memory is a mechanism that is nearly entirely transparent (or invisible) to users, except for a slight impact on performance.

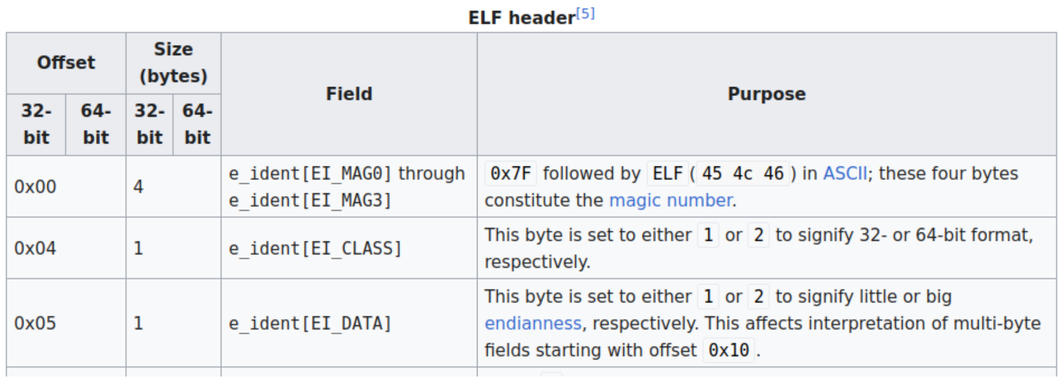
2.2 Classifying ELF files (6 pts)

In exercise 4, you studied how to use `mmap` to extract information from files using the executable and linking format (ELF). In this problem you are asked to complete a program that detects whether a given file appears to be an ELF file or not, and whether the format is 32 or 64 bits and its byte order/endianness.

Once completed, your `elfdetector` program should work like this:

```
$ gcc -o elfdetector elfdetector.c
$ ./elfdetector elfdetector
64-bit Little-Endian ELF
$ gcc -o elfdetector32 -m32 elfdetector.c
$ ./elfdetector elfdetector32
32-bit Little-Endian ELF
$ ./elfdetector elfdetector.c
not an ELF file
```

WikiPedia provides this documentation about the ELF header which is at the beginning of an ELF file:


ELF header[5]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		e_ident[EI_MAG0] through e_ident[EI_MAG3]	0x7F followed by ELF (45 4c 46) in ASCII; these four bytes constitute the magic number.
0x04		1		e_ident[EI_CLASS]	This byte is set to either 1 or 2 to signify 32- or 64-bit format, respectively.
0x05		1		e_ident[EI_DATA]	This byte is set to either 1 or 2 to signify little or big endianness, respectively. This affects interpretation of multi-byte fields starting with offset 0x10 .

Complete `elfdetector.c`:

```
// headers elided for brevity

int
main(int ac, char *av[])
{
    int fd =
    assert (fd != -1);

    void *addr = mmap(NULL, 4096, PROT_READ, MAP_PRIVATE, fd, 0);
    assert (addr != MAP_FAILED);

    char *elf =
13      bool is64 =
14
15      bool isLittleEndian =
16
17      bool isELF =
18
19      if (isELF)
20          printf("%d-bit %s-Endian ELF\n", is64 ? 64 : 32,
21                 isLittleEndian ? "Little" : "Big");
22      else
23          printf("not an ELF file\n");
24      return 0;
25  }
```

Hint: you need to complete lines 6, 12, 13, 15, and 17. For simplicity, you may assume that if the file is an ELF file, then the EI_CLASS and EI_DATA fields will have the values 1 or 2. You may also assume that the program will be invoked with non-empty files.

```c
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdio.h>
#include <assert.h>

int
main(int ac, char *av[])
{
    int fd = open(av[1], O_RDONLY);
    assert (fd != -1);

    void *addr = mmap(NULL, 4096, PROT_READ, MAP_PRIVATE, fd, 0);
    assert (addr != MAP_FAILED);

    char *elf = addr;
    bool is64 = elf[4] == 2;
    bool isLittleEndian = elf[5] == 1;
    bool isELF = strncmp(elf, "\x7f""ELF", 4) == 0;

    if (isELF)
        printf("%d-bit %s-Endian ELF\n", is64 ? 64 : 32,
               isLittleEndian ? "Little" : "Big");
    else
        printf("not an ELF file\n");

    return 0;
}
```

2.3 To Understand Recursion (6 pts)

"To understand recursion, you must first understand recursion" is a well-known self-referential joke among computer programmers. You're trying this out on your machine and run the following program after compiling it with the `-O1` switch.

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

(a) (2 pts) The program crashes with a segmentation fault. You investigate the assembly code of the program:

```
0000000000401106 <main>:
  401106:       48 83 ec 08                 sub    $0x8,%rsp
  40110a:       e8 f7 ff ff ff              callq  401106 <main>
  40110f:       48 83 c4 08                 add    $0x8,%rsp
  401113:       c3                          retq
```

The crash you observed is colloquially called a "stack overflow" - but which of the program's instructions caused the segmentation fault and why?

The callq instruction caused the segmentation fault because it is the only instruction that writes to memory. Each call allocates stack space, but it is callq that writes the return address to the stack, causing the OS to expand the stack whenever a new page boundary is reached and a page fault is caused. Eventually, the OS will hit the stack limit and refuse to allocate memory to expand the stack further, terminating the process instead.

(b) (2 pts) You repeat the command with the perf stat utility:

```
$ perf stat -e page-faults ./recursion
./recursion: Segmentation fault

 Performance counter stats for './recursion':

             1,068      page-faults
```

Based on this information, provide an estimate of the stack limit in MB that was in effect in the environment in which you ran your program. Assume a standard x86 64-bit Linux environment as found on our rlogin cluster. Ignore any effects caused by the program's startup (before `main()`).

The stack limit was set to 4MB.

(c) (2 pts) Stack limits help us catch programs that fail due to infinite recursion, but what is a potential downside or risk to having a stack limit?

The problem with a stack limit is that it may terminate legitimate programs with large stack consumption, such as programs making use of deep recursion or programs that allocate their data in large local variables.

2.4 Working Sets (8 pts)

Alice and Bob (neither of whom had taken 3214) work at a hot new startup where they are tasked with running a program that processes data. They have been asked to run the program on two machines that are attached to the startup's internal network. The program will pull about ∼17GB of data from the network, then perform a proprietary computation — running entirely on the CPU (no I/O, network, GPU, etc.) — that iterates over all of the data multiple times before producing an output.

Alice's machine is configured with

```
8 CPUs at 2.8GHz
32 GB RAM
512 GB disk (w/ 4 GB swap partition)
```

Bob's machine is configured with

```
8 CPUs at 2.8GHz
16 GB RAM
512 GB disk (w/ 4 GB swap partition)
```

1. (2 pts) Bob didn't think the program would run on his machine because he only has 16GB RAM where the workload computes over ∼17GB but was surprised to see it succeed when he tried the workload. Why was the workload able to succeed despite his machine only having 16GB RAM?

   Some of the data will be written to swap space, essentially increasing the amount of memory available to the program. When the data is accessed again, it is read from swap space back into main memory.

2. (2 pts) Alice and Bob run some measurements to compare the performance on each machine. Which one will complete faster? Why?

   Alice's machine will complete faster because the working set fits into memory (no disk I/O required).

3. (4 pts) Alice and Bob notice some additional hardware around the lab including some hard drives that have higher throughput and lower latency than the ones currently installed. Ignoring program load time and instead concentrating on the time the proprietary computation takes to run, would using the new hard drives increase Alice's performance? What about Bob? Explain why.

   - Alice's performance would not increase, as the hard drive does not influence the performance.
   - Bob's performance: should increase as the "swapping overhead" will decrease (the time to write data to/from disk

3 Dynamic Memory Management (10 pts)

Mimalloc is a memory allocator developed by researchers at Microsoft Research. It provides the standard `malloc()`/`free()` API. In this question, let us observe some of its behavior.

(a) (2 pts) Consider this program:

```
int main()
{
    void *ref = malloc(8);
    printf("malloc(%d)  -> %p\n", 8, ref);
}
```

when run with mimalloc a few times, it outputs:

```
$ ./mimalloc1
malloc(8)  -> 0x2c6ee010018
$ ./mimalloc1
malloc(8)  -> 0x4c9ce010018
$ ./mimalloc1
malloc(8)  -> 0x5a31e010018
$ ./mimalloc1
malloc(8)  -> 0x3be5a010018
```

Why does this program's output a different number each time it is run?

The system uses address space layout randomization (ASLR), a safety technique that make certains attacks more difficult.

(b) (2 pts) Now consider this program

```c
int main()
{
    void *ref = malloc(8); // create reference point
    for (size_t sz = 16; sz <= 128; sz += 16) {
        for (int n = 1; n <= 2; n++) {
            void *p = malloc(sz);
            printf("malloc(%d)[%d] -> 0x%x\n", sz, n, p-ref);
        }
    }
}
```

When run, it outputs this every time:

```
malloc(16)[1] -> 0x10018                              malloc(80)[1] -> 0x60038
malloc(16)[2] -> 0x10028                              malloc(80)[2] -> 0x60088
malloc(32)[1] -> 0x30048                              malloc(96)[1] -> 0x70048
malloc(32)[2] -> 0x30068                              malloc(96)[2] -> 0x700a8
malloc(48)[1] -> 0x40078                              malloc(112)[1] -> 0x80058
malloc(48)[2] -> 0x400a8                              malloc(112)[2] -> 0x800c8
malloc(64)[1] -> 0x500a8                              malloc(128)[1] -> 0x90068
malloc(64)[2] -> 0x500e8                              malloc(128)[2] -> 0x900e8
```

Note that rather printing the raw addresses, the program prints the pointer difference to the very first allocation, which is the distance counted in bytes. We arranged the output in columns to save space.

From this output, what can you conclude about mimalloc's allocation policy? Which storage allocation technique(s) does `mimalloc` appear to be using?

It uses size classes and moreover, it uses separate memory areas (sometimes called blocks or "pages") for each size class. The lecture slides called this style "simple segregated storage."

(c) (2 pts) How much memory is lost due to internal fragmentation in the block labeled `malloc(80)[1]`?

Zero bytes since `malloc(80)[2]` starts right after it.

(d) (2 pts) Now consider a sequence of allocations and deallocations of blocks of size 256:

```
int main()
{
    void *ref = malloc(8);
    size_t sz = 256;
    int n;
    for (n = 1; n <= 4; n++) {
        void *p = malloc(sz);
        printf("malloc(%d)[%d] -> 0x%x\n", sz, n, p-ref);
        free(p);
        printf("freed(0x%x)\n", p-ref);
    }
}
```

The observed output is:

```
malloc(256)[1] -> 0x100e8
freed(0x100e8)
malloc(256)[2] -> 0x101e8
freed(0x101e8)
malloc(256)[3] -> 0x102e8
freed(0x102e8)
malloc(256)[4] -> 0x103e8
freed(0x103e8)
```

What can you conclude about mimalloc's management of freed blocks?

It does not immediate reuse freed blocks.

(e) (2 pts) How would your p3 allocator's policy be different from mimalloc's for the last example?

Many groups' implementations would have reused a freed block immediately for an allocation of the same size.

4 Automatic Memory Management (10 pts)

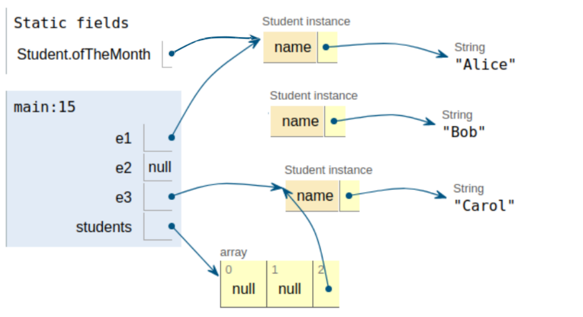
4.1 Understanding Reachability (4 pts)

Consider the following Java program:

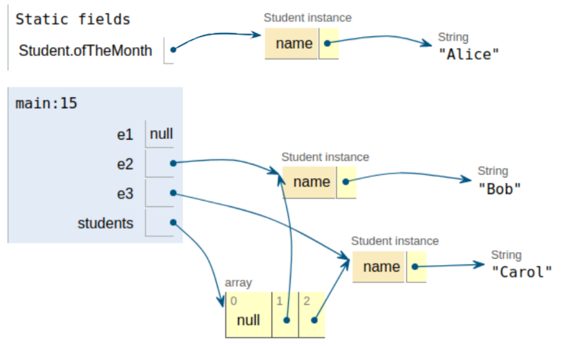
```
public class Student {
  static Student ofTheMonth;
  private String name;
  public Student(String theName) {
    this.name = theName;
  }

  public static void main(String[] args) {
    Student e1 = new Student("Alice");
    Student e2 = new Student("Bob");
    Student e3 = new Student("Carol");
    Student.ofTheMonth = e1;
    e2 = null;
    Student students[] = {e1, e2, e3};
  }
}
```

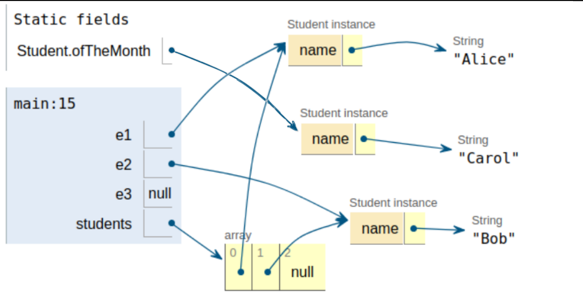
Which of the following graphs depicts this program's heap after executing the statements up until (and including) line 14? Assume that no garbage collection has taken place. (Check the corresponding box.)



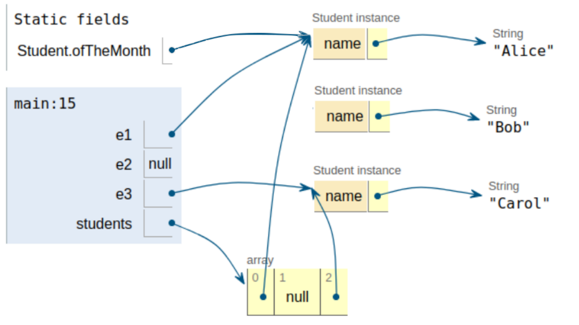
☐ Graph 1



☐ Graph 2



☐ Graph 3



☑ Graph 4

4.2 Out Of Memory Situations (6 pts)

(a) (3 pts) During exercise 4, you were asked to write a Java program that would run out of memory. One student made this attempt:

```
import java.util.TreeSet;

public class OOM {
    public static void main(String []av) {
        for (long i = 0; ; i++) {
            TreeSet<Long> list = new TreeSet<>();
            list.add(new Long(i));
        }
    }
}
```

Would this program run out of memory or not? Justify your answer.

It would not run out of memory. In each iteration of the loop, a new `TreeSet` object is created, but the local reference variable `list` pointing to it is overwritten such that the object from the previous iteration because garbage, which the collector can free. Thus, only the last `TreeSet` object is alive, which means that the live heap size does not increase.

(b) (3 pts) You are working on a large server-side enterprise application written in Java. Your DevOps team informs you that they have to restart your application server at least once a day in order to prevent it from running out of memory. You investigate a heap dump that they shared with you when it did run out of memory, but even after close examination you cannot find any reachable objects you could identify as leaks. What are your options now? Name and explain one.

If leaks are ruled out, the only options left are to

- Reduce bloat by using more space-efficient data structures (e.g., `int` instead of `Integer`)
- Reduce the overall amount of memory used, for instance, by introducing caches and bounding on their size (or reducing the amount of memory allocated to such caches). Weak references could be used here.
- Asking for a memory upgrade

Non-answers include

- running the garbage collector more often would not help since its an increase in the live heap size that is causing the running out of memory
- reducing churn (the allocation rate) for the same reason

5 Virtualization (12 pts)

5.1 Container Namespaces (6 pts)

In exercise 5, all students could demo their webservers by using a container. Here is the command run inside the container (as specified in the provided Dockerfile):

```
CMD ./server -p 9999 -R ../svelte-app/build -a
```

1. (2 pts) What does the `-p 9999` indicate to the server?

   It indicates that the server should bind to and listen on port 9999.

2. (2 pts) What would happen if, instead of using containers, all students demoed their webservers by directly issuing that command (`./server -p 9999 -R ../svelte-app/build -a`) on an rlogin node?

   These servers would encounter port conflicts ("port already in use")

3. (2 pts) Containers take advantage of kernel support to provide namespacing for what resources that are otherwise shared by processes on the system? (list 2)

   i) network namespace (e.g., port numbers)
   ii) pid namespace (separate process ids)
   iii) users (separate user ids)
   iv) file system/mount points
   v) machine/domain name (uts)

5.2 Virtual Machine or Container? (6 pts)

Virtual machines and containers are two ways to package applications for the cloud.

1. (3 pts) Suppose Alice has a custom kernel module that is to be installed alongside her application for acceleration and monitoring. Which should she choose? Why?

   A VM because it has its own private OS kernel to install a module into.

2. (3 pts) Bob cares about how quickly his application can be started in response to an event. Which should he choose? Why?

   A container to save the overhead of starting a VM and booting the guest kernel in it.