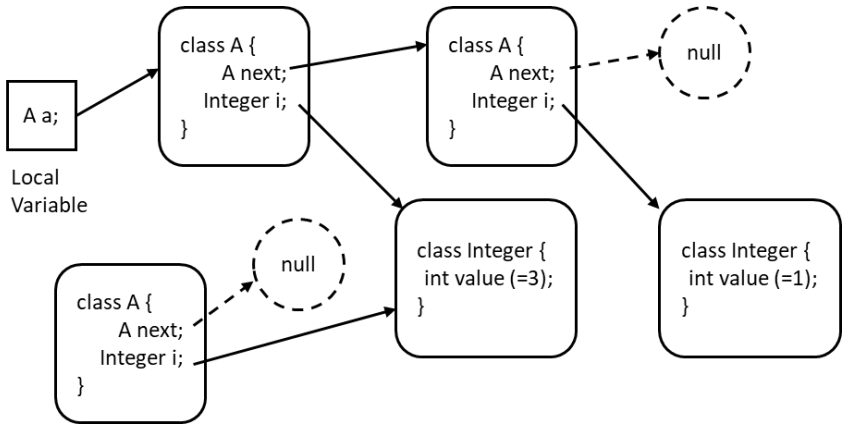
Sample Final Exam

Solutions are shown in this style. This exam was originally given in Spring 2018.

1. Automatic Memory Management (20 pts)

a) (8 pts) *Reachability.* When writing programs in languages that support automatic memory management, it is important to understand the impact of a program's code on the reachability graph as it is seen by the garbage collector. Below is shown a part of a reachability graph resulting from a snippet of Java code:

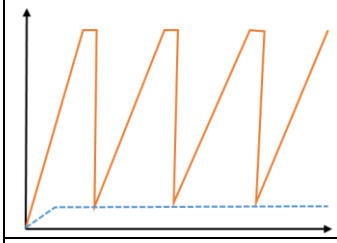
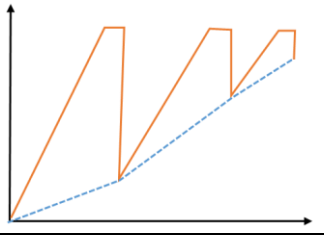
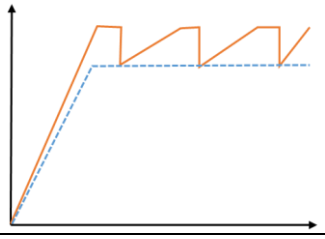


Complete the following Java program so that, at the end of main(), the reachability subgraph shown above could be observed:

```
public class A {
    A next;
    Integer i;

    A(A next, Integer i) {
        this.next = next;
        this.i = i;
    }

    public static void main(String [] av) {
        /* insert your code here, defining variables as necessary! */

        A a = new A(null, new Integer(1));
        a = new A(a, new Integer(3));
        new A(null, a.i);

        /* if the garbage collector walked the heap here, it should
           observe the reachability graph shown above! */
    }
}
```

Assume that the compiler will not perform optimizations that remove statements without side effects. You may use the `java.lang.Integer(int)` constructor. Do not introduce more local variables than necessary!

b) (6 pts) Understanding the difference between memory leaks, churn, and bloat is an important skill for practicing programmers in Java or other languages that employ garbage collection. Consider the following 3 allocation profiles showing the size of the live heap (dotted line) and the total memory allocated (solid line) vs. time.
Identify which graph denotes churn, bloat, or a leak!

App 1	App 2	App 3
		
Check one:	Check one:	Check one:
☑ Churn	❑ Churn	❑ Churn
❑ Bloat	❑ Bloat	☑ Bloat
❑ Leak	☑ Leak	❑ Leak

- Churn is characterized by high allocation rates of garbage collectable objects: the allocation curve is steep, but live heap size is not significant.
- Bloat is characterized by high data structure overhead – the amount of live heap memory to hold an application's live data is large.
- A Leak is characterized by a steady increase in live heap size.

c) (3 pts) Assuming that the cost of garbage collection is dominated by the cost of identifying unreachable objects, how does the time spent collecting garbage compare between the applications? Assume that the 3 charts use the same scale on their respective axes!

❑	App 1 spends more than App 2, and App 2 spends more than App 3
❑	App 3 spends more than App 1, and App 1 spends more than App 2
☑	App 3 spends more than App 2, and App 2 spends more than App 1
❑	App 2 spends more than App 3, and App 3 spends more than App 1

The cost of identifying unreachable objects is proportional to the size of the live heap at each garbage collection.

d) (3 pts) The 2016 blog post "Modern Garbage Collection" by Mike Hearn contains the following paragraph, discussing the HotSpot Java Virtual Machine (JVM): "*It's possible to switch between GC's just by restarting the program because compilation is done whilst the program runs, so the*

different barriers the different algorithms need can be compiled and optimised into the code as needed."

Why are GC "barriers" needed in a Java virtual machine?

A	❑	A barrier is inserted as a marker to know when it is safe to perform garbage collection, i.e., when there are no references to objects from machine registers.
B	❑	A barrier is inserted to stop threads when a garbage collection is pending to avoid further mutation, i.e., as a break point.
C	☑	A barrier is inserted to inform the garbage collector that objects may be alive when a reference to them is read or stored.
D	❑	A barrier ensures that other cores will see the latest value/references stored in a field for multi-threaded programs.

2. Explicit Memory Management (15 pts)

Which of the following statements regarding managing free space in memory allocators that implement a traditional explicit memory management model (e.g., `malloc()/free()`) are true?

a) (3 pts) User-level memory allocators typically employ best-fit schemes that exhibit an average case complexity of O(n) where *n* is the number of free blocks.

❑ True ☑ False

Typically employed schemes perform at O(log n) or O(1).

b) (3 pts) Link elements (e.g. next, prev pointers) used for free list management are stored within the unused memory of free blocks.

☑ True ❑ False

c) (3 pts) Free blocks (those not currently in use) require only header boundary tags, but no footer boundary tags.

❑ True ☑ False

They do. Otherwise, it would be impossible to coalesce in constant time if a to-be-freed block's left neighbor is free.

d) (3 pts) Coalescing free blocks of different sizes tends to increase fragmentation.

❑ True ☑ False

Segregated free list schemes coalesce across size classes.

There's no reason why such coalescing would increase fragmentation.

e) (3 pts) Some user-level memory allocators batch multiple `malloc()` requests for better placement decisions.

❑ True ☑ False

No. malloc()'s API does not allow batching – the allocator has to respond to requests synchronously.

3. Virtual Memory (20 pts)

a) (5 pts) The machines on our current rlogin cluster run a 64-bit version of Linux using 2 Intel Xeon processors with 48-bit wide virtual addresses and up to 46-bit wide physical addresses, and are each equipped with 96 GB of RAM and 4 GB of additional swap space backed by an SSD drive. Consider the following program, which attempts to allocate 1GB-sized chunks of memory until it runs out of memory:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#define GIGA_BYTE   (1024l*1024*1024)

int
main(int ac, char *av[])
{
    long i;
    for (i = 0; ; i++) {
        char *p = malloc(GIGA_BYTE);
        if (p == NULL)
            break;
    }

    printf("Allocated %ld * %ld = %lld bytes\n", i, GIGA_BYTE,
            (long long)i * GIGA_BYTE);
    return 0;
}
```

What number of bytes would be reported by this program, if any?

A	B	C	D	E
❑	❑	❑	☑	❑
~64 GB	~96 GB	~100 GB	~128 TB	Process is killed by OOM killer

Large malloc() calls are turned directly into mmap() calls that request a sufficiently large chunk of virtual address space. Linux does not take physical memory availability into account when granting requests for virtual address space as long as it is not touched; therefore, malloc() will fail due to address space exhaustion. A 64-bit Linux system devotes about 47-bits, or 128 TB, to a process's user address space. Only if the process accesses the memory will physical memory be allocated (which then leads to the OOM killer killing the program).
5 easy points for those who followed along when I demoed this in class.

b) (6 pts) Processes generally have full control of their virtual address space. For instance, the `mprotect()` system call can be used to change the virtual memory permissions associated with a region of a process's virtual address space. Consider the following C code:

```
static void
catch_signal(int signo, siginfo_t *info, void * _ctxt)
{
    /* Write your answer here:                                          */
    printf("Your program encountered a: stack overflow\n");
    exit(EXIT_FAILURE);
}

static void
install_detection(size_t max)
{
    struct sigaction act;

    int PGSIZE = getpagesize(); // returns system's page size
    void * guard = (void *)(((uintptr_t)&act - max) & ~(PGSIZE-1));

    mprotect(/* addr */ guard, /* len */ PGSIZE, /* prot */PROT_NONE);

    /* Standard code to set up SIGSEGV handler. */
    act.sa_sigaction = catch_signal;
    act.sa_flags = SA_SIGINFO | SA_ONSTACK;
    sigemptyset (&act.sa_mask);
    sigaction (SIGSEGV, &act, NULL);

    /* Set up alternate stack used during signal handling. */
    stack_t signalstack = {
        .ss_sp = malloc(SIGSTKSZ),
        .ss_size = SIGSTKSZ,
        .ss_flags = 0
    };
    sigaltstack(&signalstack, NULL);
}
```

Programs using this code could then call `install_detection()`, for instance, `install_detection(1048576)`.

What kind of error situation will this code detect?
Write your succinct answer in the code above!

The details of the `sigaction()/sigaltstack()` calls are shown for completeness, but they are not needed to answer this question. The relevant code is shown in bold. `uintptr_t` is a C99 integer type that is large enough to hold a pointer. Relevant excerpts from the man page for mprotect are shown below:

```
NAME
       mprotect - set protection on a region of memory

SYNOPSIS
       #include <sys/mman.h>

       int mprotect(void *addr, size_t len, int prot);

DESCRIPTION
      mprotect() changes protection for the calling process's memory
      page(s) containing any part of the address range in the interval
      [addr, addr+len-1].  addr must be aligned to a page boundary.

      If the calling process tries to access memory in a manner that
      violates the protection, then the kernel generates a SIGSEGV
      signal for the process.

      prot is either PROT_NONE or a bitwise-or of the other values in
      the following list:

       PROT_NONE  The memory cannot be accessed at all.
       PROT_READ  The memory can be read.
       PROT_WRITE The memory can be modified.
       PROT_EXEC  The memory can be executed.
```

Looking carefully, you see that `mprotect()` protects a page-sized region 'max' below the current stack pointer (approximated by the address of local variable `act`.) A segmentation fault on this page indicates stack overflow. Such "guard pages" are a common technique to detect stack overflows.

c) (4 pts) Processors with memory management units (MMUs) that support virtual memory typically provide a so-called "dirty" bit in each TLB and each page table entry. This bit is set when data inside the corresponding page was updated through an instruction.
Briefly describe when and how a virtual memory manager inside an OS would make use of this bit!

The dirty bit is examined by the OS's memory manager when it evicts a page from its frame to storage. If it is set, the page's data must be saved by writing it to a storage location (file system or swap storage).

For movie buffs: Mark Zuckerberg (played by Jesse Eisenberg in the movie "The Social Network") would have known the purpose of the dirty (or modified bit as it's sometimes called).
A number of you confused the dirty with the accessed (or referenced) bit. The latter is used to determine recency and approximate an LRU algorithm. Although some page replacements take the dirty bit into account as a secondary measure when judging the cost of eviction, its primary purpose is not to guide eviction decisions. Resetting the dirty bit, as some of you proposed.

d) (5 pts) Programmers generally expect that programming errors such as out-of-bounds accesses result in segmentation faults, but consider the following program, which does not cause a seg fault on our current rlogin machines when compiled with `-O0`:

```
int main() {
    char * b = malloc(16384);
    b[16384] = 1;
}
```

Despite clearly overflowing the bounds of the allocated memory block, why exactly does this program not cause a segmentation fault?

The program will not segfault because it likely accesses an address where the footer of the block is stored. If the footer of allocated blocks is optimized out (by storing the allocation status in the block neighboring to the right), then this address would contain the next block's header. It could also contain padding. In all cases, the address is either on the same page as &b[16383] (and thus mapped), or on the next page, which is also mapped. Segmentation faults are caused only by accesses to virtual addresses that are not currently mapped. The stipulation that the code was compiled without optimizations was to make clear that the reason for the lack of a segmentation fault is not the compiler optimizing away the dead assignment to b[16384].

Your answer should have conveyed that this out-of-bound access likely reached into padding or allocator data structures, which are part of already mapped memory.

4. Virtualization (15 pts)

Answer the following questions regarding virtualization:

a) (3 pts) Fill in the blank (This question is asking to complete an analogy):

A guest virtual machine is to a hypervisor as a process is to an OS kernel.

b) (3 pts) One question at this semester's midterm talked about "direct limited execution," a term that denotes the safe execution of user programs with reduced privileges directly on the CPU, while handling exceptions that can occur by switching to kernel mode. Is "direct limited execution" a

technique that can also be used in virtual machines (as defined by Popek/Goldberg)?

☑ Yes ❑ No

Yes – in fact, efficient, mostly native execution is one of their criteria.

c) (3 pts) Is it technically possible to execute an OS such as Linux or Windows inside a virtual machine without any support from the guest OS, such as special drivers or configurations?

☑ Yes ❑ No

Yes. Although OS typically run better with an awareness of running under a hypervisor, hypervisors such VMWare or VirtualBox can execute unchanged, off-the-shelf OS.

d) (3 pts) Operating systems that support virtual memory have the ability to move process data from pages in RAM to disk (and back if needed), without the executing program noticing.
Do virtual machine monitors have the same ability when it comes to the pages guest OS manage?

☑ Yes ❑ No

Yes. Hypervisors can page out their guest's memory because they can intercept memory accesses in the same way an OS kernel can intercept memory accesses for its processes, by controlling the virtual-to-physical address mapping. (In practice, however, it's difficult to do paging to disk efficiently due to conflicting policies and the risk of double-paging.)

e) (3 pts) Containers are a popular alternative to virtual machines. Containers are groups of processes running on an OS such as Linux that have their own namespaces (such as for process ids or files) as well as their own resource limits (CPU share, physical memory). Although these environments can provide increased efficiency, they sacrifice some degree of isolation when compared to virtual machines.
What specific potential failure event do containers not isolate against?

Containers cannot isolate against failure of the shared kernel.
If the kernel fails, all containers running on top of it will fail. If the kernel in a virtual machine fails, only the affected virtual machine will fail. Because such kernel failures are rare, containers are considered a suitable alternative for many scenarios.

5. HTTP & Web Applications (18 pts)

a) (6 pts) Debugging project 4. Project 4 involved upgrading a small HTTP server to support version 1.1 of the HTTP protocol. Shown below is debugging output from a project 4 server in development:

```
recvfrom(4, "POST /api/login HTTP/1.1\r\nUser-Agent: curl/7.29.0\r\nHost:
localhost:9999\r\nAccept: */*\r\nContent-Type: application/json\r\nContent-
Length: 45\r\n\r\n{\"username\":\"user0\",\"password\":\"thepassword\"}",
2048, MSG_NOSIGNAL, NULL, NULL) = 185
sendto(4, "HTTP/1.0 200 OK\n", 17, MSG_NOSIGNAL, NULL, 0) = 16
sendto(4, "Server: CS3214-Personal-Server\nSet-Cookie:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjU0OTAzMDEsImlhdCI6MTUyNTQ
wMzkwMSwic3ViIjoidXNlcjAifQ.y7iSO-
mH6OfDRrzlH0TasMRfyerla8rvEg4irzI4bgM;\nContent-Type:
application/json\nContent-Length: 2\n\n", 266, MSG_NOSIGNAL, NULL, 0) = 243
sendto(4, "{\"exp\":1525490301,\"iat\":1525403901,\"sub\":\"user0\"}", 49,
MSG_NOSIGNAL, NULL, 0) = 49
```

(Note that line breaks within quoted strings are due to word wrapping).
The response contains 5 mistakes that would have prevented you from passing the compliance tests. List three:

i. (2 pts) Mistake 1: HTTP/1.0 should be HTTP/1.1

ii. (2 pts) Mistake 2: Server should use CRLF \r\n instead of \n

iii. (2 pts) Mistake 3: The Content-Length should be the number of bytes in the body, which is larger than 2.

Mistake 4: The proper Cookie format needs to include a name, such as auth_token=

Mistake 5: The Cookie is lacking a Path= specifier.

Mistake 6: sendto(..., 17, ...) passes only 16 bytes; similarly, sendto(..., 266,) passes only 243 bytes.

b) (4 pts) Express is an HTTP server library written in JavaScript, which is commonly used with the node.js JavaScript virtual machine. In its default configuration, every HTTP response contains an Etag: header which looks like this:

```
ETag: W/"2a5-163281e6a3f"
```

The header value is computed using a hash digest function such as MD5 or SHA-256 from the body of the HTTP response.
Without the server having to share the exact method by which the digest is

computed with the client, how could client and server use the value contained in the Etag header to improve efficiency?

The client could cache the file, then on a future request send the value of the Etag header and ask the server to return this file again only if it has changed. The server checks the request and the Etag and returns the file only if the Etag is no longer current. This is in fact commonly done and is called a "conditional GET" request.

c) (4 pts) In 2015, the HTTP/2 protocol was finalized in RFC 7540 with the goal of improving the efficiency of HTTP transactions, while retaining the semantics of HTTP/1.1. One of its goals is to allow browsers to use a single connection per origin without loss of performance. According to its FAQ, HTTP/2 is fully multiplexed, unlike HTTP/1.1, which is "ordered and blocking."
Explain what is meant by HTTP/1.1 being "ordered and blocking!"

It is "ordered" because HTTP/1.1 responses must be sent in the same order as the requests to which they correspond. It is "blocking" because a response to an earlier request that takes a longer time to compute blocks the server from sending an already available response to a later request. HTTP/2 corrects both of these shortcomings.
Note that "blocking" here does not refer to blocking threads.

d) (4 pts) Single Page Applications (SPA). Project 4 provided a glimpse at how to provide a backend for a type of modern web application called SPA, or single page application. SPAs typically send almost no HTML from the server when invoked, bundle all assets (JavaScript, CSS, and even images) encoded in a few large objects, and after the start of the application interact with the server only through JSON-based APIs like the small API you have implemented.
List 2 advantages of this design, when compared to the traditional design in which every navigation action such as mouse clicks transfers a new HTML page from the server to the client!

- The server does not need to implement presentation logic; specifically, API requests carry only application data and no presentation data.
- This can result in lower latency.
- User interface templates and logic, as well as static assets, can be transferred efficiently from the server in large chunks.
- Static assets are long-lived and can be cached, reducing bandwidth requirements during repeated use.
- It makes it easy to use different servers for API and static assets.

Other answers were accepted as long as they didn't simply echo the facts stated in the question.

6. JWT & Security (12 pts)

Which of the following statements are true regarding the use of JWT (JSON Web Tokens), such as in project 4. Assume that the payload consists of a JSON object with claims and that HS256 or RS256 HMAC is used.

a) If an attacker intercepts a token, they can use it to gain access to the server as the user for whom the token was issued.

☑ True ❑ False

Yes, tokens represents claims, in this case, access rights, directly.

b) If an attacker intercepts a token, they can change the "sub" claim and replace it with a different username of equal length to gain access to the server as that user.

❑ True ☑ False

No, the signature prevents the token from being tampered with, and lacking knowledge of the secret key the server used to sign the token the attacker cannot produce a properly signed token with a different sub claim.

c) If the token's signature were based on a private/public key pair (e.g., RS256 instead of HS256), then an intercepting attacker would not be able to read the claims contained in the token.

❑ True ☑ False

No, RS256 can be used to verify the identity of the token's issuer; it is not used to encrypt the token's body. A separate encryption would need to be used.

d) If the server crashes, all tokens must be invalidated and reissued.

❑ True ☑ False

False. Tokens are self-validating. (Your p4 server's validation code did not depend on anything the server kept in its memory.) See jwt.io

e) If the server key changes, tokens will remain valid until their expiration time.

❑ True ☑ False

False. Tokens are tied to the server key. If it changes, all issued tokens will no longer validate. See jwt.io

f) Clients can examine the token's "issued-at" and "expiration" time claims.

☑ True ❑ False

True. The payload is not encrypted. The client can simply base64 decode it.