CS 3214 Final Exam Solutions

Solutions are shown in this style.

1. Are you Unix literate? (16 pts)

10 years ago, the CS department offered CS 2204, Introduction to Unix, which focused on using Unix for simple processing and system programming tasks. When revamping our curriculum into its current form in 2009, we decided to remove that course in favor of our current sequence of courses, which delve much deeper into these topics. But does training to be a car mechanic really makes you a proficient driver along the way? In this question, you have a chance to show us your skills by answering the following hypothetical situations:

a) If someone said: “Run ls minus l and grep for ‘abc’ and save the result in a file ‘output’” then you would type:

```
ls -l | grep abc > output
```

b) If someone said: “Put ~cs3214/bin in your path” then you would type:

```
export PATH=~cs3214/bin:$PATH
```

A common mistake was to replace the PATH – this means you won’t be able to run command such as ls.

c) If someone said: “my read() call on this file descriptor returns -1 and I don’t know why” then you would advise them to:

… check `errno` (or use `perror()`)

d) If someone said: “calling read() on this socket returns 0 bytes read” then you explain that as:

… the connection having been closed by the other party.

Note you can do better than stating “EOF” since the question mentioned a socket.

e) If someone asked you to “check which rlogin machine you’re on” then you would type:

```
hostname
(we also accepted: “look at the shell prompt” if you have a
suitably set up prompt)
```

f) If someone asked you to “kill your ‘outfoxed’ process on hornbeam” then you would:

`ssh hornbeam killall outfoxed`

`(or ssh hornbeam; ps ax | grep outfoxed; kill <pid>)`

g) If someone claimed that “they don’t have gdb on their EC2 instance” you would advise them to:

… install it.

h) If someone suggested to “test their server by connecting with nc to 275.3.60.28 port 8080” you would reply:

.. that 275.3.60.28 is not a valid IP address.

2. Automatic Memory Management (14 pts)

a) (6 pts) Explicit memory management using `malloc()` and `free()` is known to be error prone. Consider the following alternative design:

```
#include <stdlib.h>

struct block {
  int  cnt;
  char data[0];
};

void *smalloc(size_t sz)
{
  struct block * b
       = malloc(sz + sizeof(int));
  b->cnt = 1;
  return b->data;
}

static void dec(void *p) {
  struct block * bp = p - sizeof(int);
  if (--bp->cnt == 0)
    free(bp);
}

static void inc(void *q) {
  struct block * bq = q - sizeof(int);
  bq->cnt++;
}
```

```
#define ASSIGN(p, q) do { \
  if (p != q) {           \
     if (p)               \
        dec(p);           \
     inc(q);              \
     p = q;               \
   }                      \
} while(0)

#define CLEAR(p) do {     \
   dec(p);                \
} while(0)


// Sample use
int
main()
{
  int *p = smalloc(4 * sizeof(int));
  void *q = smalloc(2 * sizeof(int));
  ASSIGN(q, p);    // q = p
  CLEAR(q);  // q goes out of scope
  CLEAR(p);  // p goes out of scope
}
```

In this design, programmers call `smalloc()` instead of `malloc()`. Furthermore, they never call `free()`. However, they have to replace all pointer assignments (e.g. `p = q` with the `ASSIGN(p, q)` macro. In addition, whenever a pointer goes out of scope, they have use the `CLEAR()` macro. Assume that a programmer applies these rules consistently throughout their code and assume that no client-side pointer arithmetic is being used.

i. (3 pts) Does this design guarantee the absence of "use-after-free" errors? Justify your answer!

Yes it would. The code given performs reference counting, a form of automatic memory management.

ii. (3 pts) Does this design guarantee that all memory which a program is unable to access will eventually be freed? Justify your answer!

It does not – if there is a cycle in the object graph, those objects' reference counts will never reach zero.

b) (2 pts) The `java.util.ArrayList` JDK class provides an implementation of an array-backed linear list. Shown below are relevant excerpts of its `remove()` method:

```
/**
 * Removes the element at the specified position in this list.
 * Shifts any subsequent elements to the left (subtracts one from their
 * indices).
 *
 * @param index the index of the element to be removed
 * @return the element that was removed from the list
 * @throws IndexOutOfBoundsException
 */
public E remove(int index) {
    // rangeCheck and concurrent modification check elided
    E oldValue = elementData(index);

    int numMoved = size - index - 1;
    if (numMoved > 0)
        System.arraycopy(elementData, index+1, elementData, index,
                         numMoved);
    elementData[--size] = null;   // ← X

    return oldValue;
}
```

Here, `size` refers to the number of elements in the list (which may be less than the size of the `elementData` array backing it.)

Explain the significance of the statement in bold (marked with an X)! What would happen if it were absent?

It removes a reference from the list to its last object, which if not done could cause a memory leak once no other references to this object exist. It is not acceptable for an object to be unreclaimable just because it was at some point in an ArrayList.

c) (6 pts) Suppose you are hired as a Java engineer for a company that provides web services. Every so often, one of their servers runs out of memory. The operations engineer has started to run the JVM with the `-XX:-HeapDumpOnOutOfMemoryError` flag, which causes the JVM to write a heap snapshot when it runs out of memory. You are given the heap dump from the last time it crashed. Describe the process you would use to investigate and determine the root cause of the crash! In your discussion, address at least 2 reasons for why a Java program may run out of memory!

Although debugging memory-related failures can be difficult to debug in general, a first approach to post-mortem analysis should be to examine the heap dump with an analysis tool such as the Eclipse memory analyzer and identify which objects have the largest retained heap size. Then, try to identify whether their memory consumption is due to a) a leak (memory kept alive that isn't accessed in the future), or b) a large live heap size (objects that are kept alive because the program may access them in the future, such as a cache), or c) runtime bloat (objects taking unexpectedly a lot more memory than is expected from their function.

3. Virtual Memory (12 pts)

a) (6 pts) The `mmap()` system call can be used to map a file so that a program can use pointer operations to access its content.

i. (3 pts) On a 32-bit Linux machine with 16 GB of RAM and a 1TB disk, can you mmap() a 6 GB file? Justify your answer!

No you can't. On a 32-bit Linux machine, the user address space is less than 4GB, but mmap() requires the availability of a contiguous chunk of virtual address space.

ii. (3 pts) Could you mmap() the same file on a 64-bit Linux machine with only 4 GB of RAM? Justify your answer!

Yes you can. There is enough virtual address space. The availability of RAM is not checked when deciding whether to allow a process to mmap().

b) (6 pts) You are writing a program that processes "big data" in memory. After all data has been read into memory, you observe that the program appears to make much slower progress than you had expected, although you do not

receive any out of memory errors. You are running on a multiprocessor with 4 cores, but the system usage meter shows that CPU consumption is near zero. A friend suggests to increase the swap space size to better accommodate the program's virtual memory requirements.
Will this approach help with the phenomenon you observe or not? Justify your answer!

The phenomenon described (large memory consumption, low CPU utilization, slow progress) is typical of thrashing – a state where processes spend most of their time paging data in and out from/to disk. The root cause is a shortage of physical memory. Therefore, adding swap space will not solve the problem. In fact, if a shortage of swap space had been an issue, the program would have seen out of memory errors (malloc() failing, or in Linux the OOM killer killing the program.)

4. Explicit Memory Management (13 pts)

a) (4 pts) *Optimizing Boundary Tags.* In the technique for dynamic memory allocators discussed in lecture, 2 identical boundary tags are placed in the header and footer of each block. These boundary tags contain the size of the block as well as 1 bit describing whether it is allocated or free. Your project partner proposes the following idea: include a 2nd bit in the boundary tag header that denotes if the previous block (i.e., the one immediately located to the "left") is allocated. Then, omit the boundary tag footer for allocated blocks, which will decrease internal fragmentation and increase the utilization score.

Discuss the merits of this idea! Will it work or not? Justify your answer!

It will work since the size field in a boundary tag footer is not examined unless the block to which the footer belongs is free. In fact, it is a technique commonly exploited. Note that the boundary tag footer is retained for free blocks.

b) (9 pts) *Region-based management.* Some programming languages support a memory management paradigm that is based on regions. Each allocated object belongs to one region. A program may create as many regions as it wishes. All objects that belong to a region are freed when a region is destroyed. Consider the following example, taken from WikiPedia:[1]

```
Region *r = createRegion();
ListNode *head = NULL;
for (int i = 1; i <= 1000; i++) {
    ListNode* newNode = allocateFromRegion(r, sizeof(ListNode));
    newNode->next = head;
```

[1] https://en.wikipedia.org/wiki/Region-based_memory_management

```
    head = newNode;
}
// ...
// (use list here)
// ...
destroyRegion(r);
```

Describe how would efficiently implement `createRegion()`, `allocateFromRegion()`, and `destroyRegion()`!

i. `createRegion()` [discuss any data structures here]

ii. `allocateFromRegion()`

iii. `destroyRegion()`

An example implementation is given below.

```
// region.h
#include <stdlib.h>
#include "list.h"

struct Chunk {
    struct list_elem elem;
    size_t size;   // counted in bytes starting from data
    char data[0];
};

typedef struct _Region {
    struct list chunks;     // list of chunks
    struct Chunk *current;  // current chunk
    void *next; // next available bytes in current chunk
} Region;

// region.c
#include "region.h"

#define CHUNK_SIZE  4096
#define max(a, b) ((a) > (b) ? (a) : (b))

static void
allocateChunk(Region *r, size_t minsz) {
    struct Chunk * c = malloc(max(CHUNK_SIZE,
                              minsz + sizeof(struct Chunk)));
    c->size = CHUNK_SIZE - sizeof(struct Chunk);
    list_push_back(&r->chunks, &c->elem);
    r->current = c;
    r->next = c->data;
}

Region *
createRegion()
{
    Region * r = malloc(sizeof (Region));
    list_init(&r->chunks);
    allocateChunk(r, 0);
    return r;
}

void *
allocateFromRegion(Region *r, size_t sz)
{
    // allocate new chunk if needed (uncommon case)
    if (r->next + sz > (void *)r->current->data + r->current->size)
        allocateChunk(r, sz);

    void * m = r->next;
    r->next += sz;
    return m;
}

void
destroyRegion(Region *r)
{
    for (struct list_elem *e = list_begin(&r->chunks);
            e != list_end(&r->chunks); ) {
        struct Chunk *c = list_entry(e, struct Chunk, elem);
        e = list_remove(e);
        free (c);
    }
    free (r);
}
```

Your solution needed to explain only a basic outline of a working implementation – it should include an efficient, “bump-a-pointer” allocator and some way to link multiple chunks of continuous memory.
Partial credit was given for a solution advocates allocating the entire region contiguously (which is as efficient in terms of allocation time, but would suffer too many practical drawbacks, such as either low limits or large fragmentation.)

5. Networking and HTTP (20 pts)

a) (4 pts) *Mismatched senders and receivers.* It is well-known that TCP provides reliable data transmission between a sender and a receiver. Suppose a receiver is able to receive only small amounts of data per time unit (perhaps because it performs a considerable amount of processing for each piece of data it receives.) Suppose a sender sends data at a high rate, outpacing the receiver’s ability to receive and process the data.
Is TCP designed for this scenario? If not, how will it fail? If so, how will it handle it?

TCP's flow control mechanism is designed for this scenario. First, the excess data will be buffered, then the sender will be blocked from sending more data until the buffers have been drained by the receiver.

b) (4 pts) *The bane that is SIGPIPE*. As many of you learned the hard way, SIGPIPE is sent to processes attempting to write into sockets that are closed by the other party. The man page for write(2) states:

```
   EPIPE  fd is connected to a pipe or socket whose reading end
is closed.  When this happens the writing process will also
receive a SIGPIPE signal.  (Thus, the write return value is seen
only if the program catches, blocks or ignores this signal.)
```

Explain a possible rationale for this design, specifically explain why Unix's designers chose to send a signal rather than only flagging an error! (Hint: note that the provision applies to both pipes and sockets!)

Consider running `a | b` on the command prompt. If '`b`' crashed or exited for whatever reason, it is clear that 'a' should terminate as well. However, we don't expect 'a' to implement any special handling for the case that it is run as part of a pipeline. Thus the kernel sends a SIGPIPE signal to '`a`' in this case, which by default leads to its termination. This works equally nicely in a one-process-per-client approach to network servers – if a client disconnects, the server process serving it will terminate. In a multi-threaded approach, more care is needed.

c) (2 pts) In HTTP, the `Content-Type:` response header describes the type of the content being returned; why is it necessary?

It is necessary because the content type cannot in general be inferred from anything else – in particular, it cannot be inferred from the URL (www.google.com serves text/html but does not end in .html). Without it, the client does not know how to process the served content.

d) (4 pts) When you use 'curl –v https://www.google.com/', Google's HTTP response contains the following header line (note that line breaks are not significant):

Set-Cookie: NID=**79=UKisvQ_0xMV1-YlFkUfPNTHTSLGAP2Fj0Xnf0uG6CaJe-bCWv4R8o6SBhYjK2Q11dR-eB3eZleJBKxRKcqDZW64SrVQxQRpZL26iptPhlFTQhtAFLx6EIp_oGx8DcpdRJe9a3CDYMip4bQ**; expires=Tue, 08-Nov-2016 02:04:31 GMT; path=/; domain=.google.com; HttpOnly

How did Google's server compute the part that is highlighted in bold (79=...) and what purpose does it have?

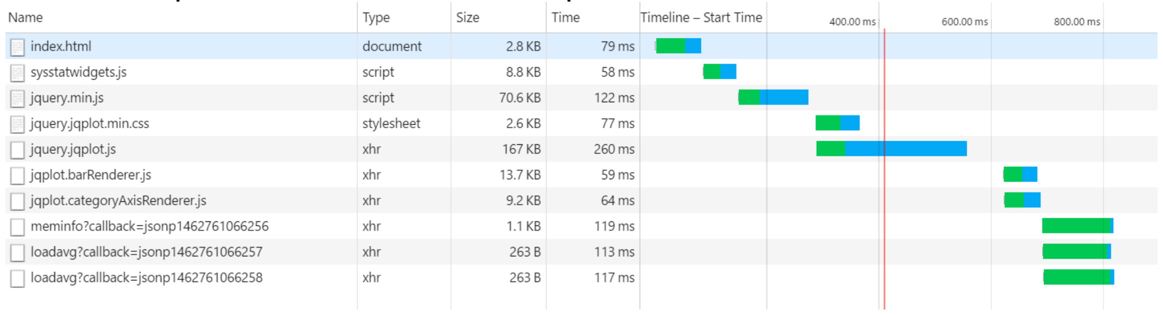
i. How computed:

It is computed using a cryptographically secure hash function so as to not be forgeable.

ii. Used for:

Identifying future HTTP requests made by the client as belonging to the same session.

e) (6 pts) Consider the following screenshot, which shows a timeline of loaded resources when a user visits a web page; this timeline was obtained from the network panel in the Chrome developer console.



Name	Type	Size	Time	Timeline – Start Time
index.html	document	2.8 KB	79 ms	
sysstatwidgets.js	script	8.8 KB	58 ms	
jquery.min.js	script	70.6 KB	122 ms	
jquery.jqplot.min.css	stylesheet	2.6 KB	77 ms	
jquery.jqplot.js	xhr	167 KB	260 ms	
jqplot.barRenderer.js	xhr	13.7 KB	59 ms	
jqplot.categoryAxisRenderer.js	xhr	9.2 KB	64 ms	
meminfo?callback=jsonp1462761066256	xhr	1.1 KB	119 ms	
loadavg?callback=jsonp1462761066257	xhr	263 B	113 ms	
loadavg?callback=jsonp1462761066258	xhr	263 B	117 ms	

i. Why are some resources loaded sequentially?

Because some resources are referenced in earlier resources (e.g., index.html includes a `<script>` element referring to sysstatwidget.js; thus the browser can't start fetching them until it has received and parsed the document referencing them.

ii. Why are not all resources loaded sequentially?

In HTTP/1.1, the browser will use multiple connections to fetch resources in order to reduce latency and exploit parallel processing on the server's part.

iii. Suggest one idea for how to shorten the time until the user can see the page render in the browser!

Many ideas are possible and are used in practice; examples include:

- Bundling and/or inlining of multiple .js files into one
- Bundlings of multiple CSS stylesheets into one
- Asynchronous loaders where only essential JavaScript is loaded initially, without blocking the browser's ability to render the page
- (in HTTP/2.0) server push where the server anticipates that the client will need those documents.

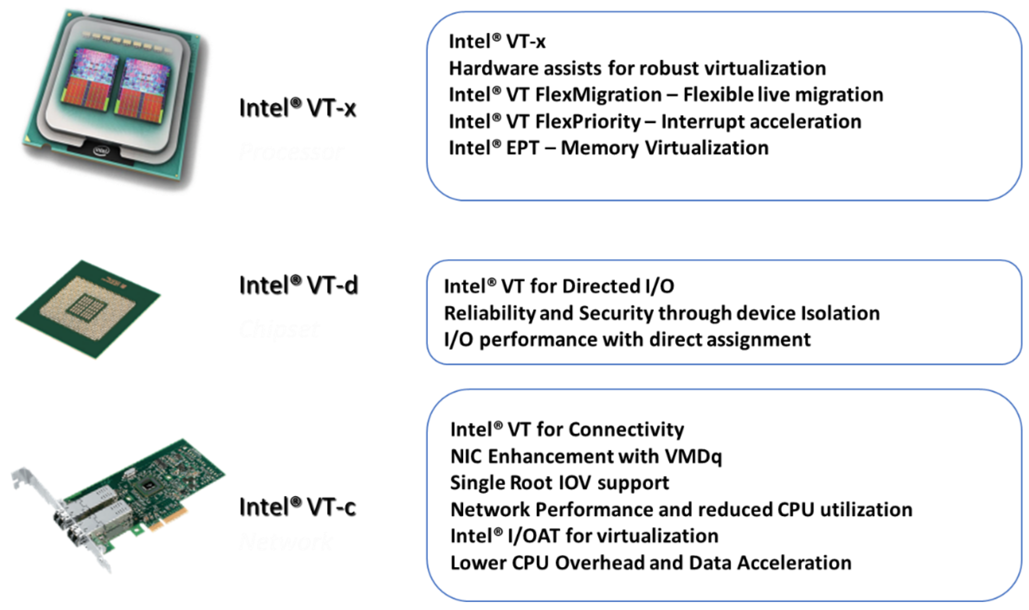
6. Virtualization (9 pts)

a) (3 pts) What is the key difference between the Java (or C#) virtual machine and a virtual machine running on Amazon EC2?

The instruction set of the Java/C# virtual machines is artificial, whereas the instruction of an Amazon EC2 VM is a subset of an existing, real ISA (usually x86).

(This is not withstanding attempts in the 1990s to construct processors that can directly execute Java bytecode).

b) (6 pts) Consider the following marketing slide from Intel's Software and Services Group, which describes features introduced into Intel's recent product lines related to virtualization:



Intel® VT-x
Processor

Intel® VT-x
Hardware assists for robust virtualization
Intel® VT FlexMigration – Flexible live migration
Intel® VT FlexPriority – Interrupt acceleration
Intel® EPT – Memory Virtualization

Intel® VT-d
Chipset

Intel® VT for Directed I/O
Reliability and Security through device Isolation
I/O performance with direct assignment

Intel® VT-c
Network

Intel® VT for Connectivity
NIC Enhancement with VMDq
Single Root IOV support
Network Performance and reduced CPU utilization
Intel® I/OAT for virtualization
Lower CPU Overhead and Data Acceleration

Mention 2 issues related to the x86's architecture original design that impeded virtualization and which are addressed by these features!

i. (Issue 1)

The x86 architecture was not virtualizable because it contained sensitive instructions that behaved differently in user than in kernel mode, but did not cause a trap. This made traditional deprivileging impossible. See [Bugnion et al, 2012, Section 2] for more details. VT-x hardware assists corrects that.

ii. (Issue 2)

The MMU was also difficult to virtualize since it uses a hardware-reloaded TLB. This required that the hardware page table be maintained by the hypervisor via expensive shadow page tables. Intel “EPT” – extended page tables allows the nesting of page tables so that guest OS and hypervisor can independently compute their part of the virtual to machine address mappings.

Another possible answer includes the difficulty of virtualizing I/O devices, which required either device drivers in the hypervisor or a split driver architecture. Intel’s directed I/O architecture allows the hypervisor to directly assign devices to virtual machines so that the device drivers can run inside those VMs.

7. Essay Question: Shell or no shell? (16 pts)

Today, Linux is the operating system with the largest installed base. Yet, most of these installations are via its Android variety, which (by default) does not even offer its users access to an interactive command line shell. In light of this fact, is learning how to write a shell for a Unix-like OS outdated in an educational context such as this course?

Briefly discuss your view on this subject, considering pros and cons!

Note: This question will be graded both for content/soundness of your technical arguments (10 pts) and for your ability to communicate effectively in writing (6 pts). Your answer should be well-written, organized, and clear.

No solution provided.