University of Washington – Computer Science & Engineering

CSE 333 Autumn 2019 Final Exam: Solution

Last Name:	Programmer	
First Name:	Systems	
Student ID Number:		
Name of person to your Left | Right		
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE333 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**	Systems Programmer	

Do not turn the page until 2:30.

Instructions

- This exam contains **12** pages, including this cover page and a blank sheet at the end. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- Write your student ID number at the top of every page. This helps us reassemble your exam when the pages are inevitably separated.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed two pages (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- You have 110 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Breathe. You've got this. You belong here.

Question	1	2	3	4	5	6	7	8	9	Total
Possible Points	12	16	14	24	3	6	15	10	1	101

Student ID Number: ________________________

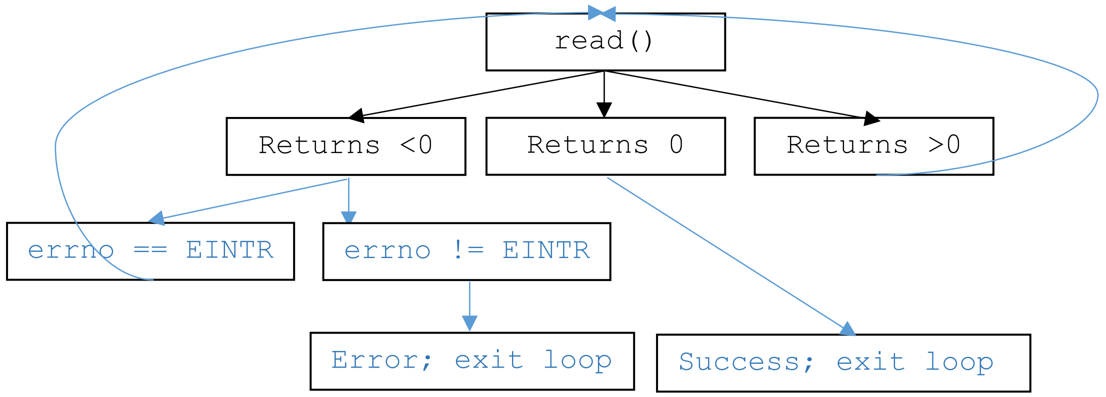
Question 1:

[12 pts] Recall that the `read()` syscall has the following signature:

```
ssize_t read(int fd, void *buf, size_t count);
```

and furthermore may set the global variable `errno` to values such as `EINTR` (the read was interrupted) or `EBADF` (the passed-in file descriptor was bad).

You are trying to read the entire contents of a file by invoking `read()` in a loop. Draw the flow chart for your loop.



Question 2:

[16 pts] Below, we list several items. Check the box if it has space reserved within a process's virtual memory; if it does exist *and* a process may have more than one instance of it in its memory, also check the second box.

	Exists	More than One
🦄 Unicorns 🦄	😢 *I wish!*	
Program counter / Instruction pointer	X	X
Recursion		
Heap	X	
Stack	X	X
Stack pointer	X	X
Static data segments (eg, `.data`, `.text`)	X	
Current value of general-purpose registers (eg, `eax`, `ebx`, `ecx`)	X	X

Student ID Number: ________________________

Question 3:

We've created a `SmartMutex` class that locks an (externally-instantiated) `pthread_mutex_t` and unlocks it when the instance goes out of scope. A client might write the following example code to use our class; assume all relevant headers have been `#include`'d.

```
char *g_buffer;           // protected by g_lock
pthread_mutex_t g_lock;   // protects g_buffer

void InitializeBuffer(int64_t length, char init) {
  SmartMutex m(&g_lock);  // implicitly locks g_lock

  if (g_buffer != nullptr || length <= 0) { return; }

  g_buffer = new char[length];  // 'new' may throw an exception
  for (int i = 0; i < length; ++i) {
    g_buffer[i] = init;
  }
}
```

(A) [4 pts] Why might this `SmartMutex` class be useful?

Even though it's quite simple, the example code still has 3 exit points. As it grows in complexity, the likelihood of forgetting to unlock `g_lock` will also grow. Using RAII to manage the lock will ensure it's unlocked regardless of how we exit the function.

(B) [6 pts] Based on our example code above, define and implement the `SmartMutex` class.

```
class SmartMutex {
 public:
   SmartMutex(pthread_mutex_t *m) : m_(m) {
     pthread_mutex_lock(m_);
   }
   ~SmartMutex() { pthread_mutex_unlock(m_); }

 private:
  pthread_mutex_t *m_;
};
```

(C) [1 pt] Our client noticed they didn't unlock `g_lock` when they exited `InitializeBuffer()` and has asked us to add an explicit `Unlock()` method to `SmartMutex`. Is this method necessary in their example code?

☐ Yes, necessary [✓ No, unnecessary]

(D) [3 pts] If this method is necessary, enumerate all the places where the client needs to call `Unlock()` in their sample code. If this method is not necessary, please explain your reasoning.

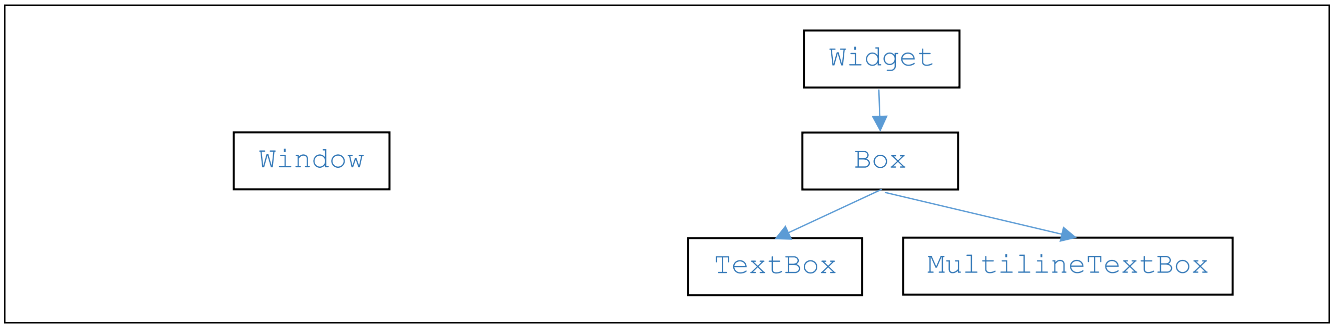
They do not need to manually unlock `g_lock` because it is automatically unlocked when the `Mutex m` stack variable goes out of scope.

Question 4:

Examine the following classes. Assume all relevant `#include` and `using` statements have been made.

```
class Window {
 public:
  virtual void DrawString(const string &s) { /* ... */ }
  virtual void Add(Widget *w) {
    // Take ownership of the argument, then set their owner pointer to us.
    ownedWidgets_.push_back(shared_ptr<Widget>(w));
    w->SetOwner(this);
  }
 private:
   vector<shared_ptr<Widget>> ownedWidgets_;
};

class Widget {
 public:
  virtual void Draw(Window *w) {  /* nothing to do */  }
  void SetOwner(Window *w) {  owner_.reset(w); }
 private:
  shared_ptr<Window> owner_;
};

class Box : public Widget {
 public:
  Box(int w, int h) : w_(w), h_(h) { }
  virtual void Draw(Window *w) {  /* nothing to do; boxes are invisible */  }
  int Area() { return w_ * h_; }
 protected:
  int w_, h_;
};

class TextBox : public Box {
public:
  TextBox(const string &msg, int w, int h) : Box(w, h), msg_(msg) { }
  virtual void Draw(Window *w) { w->DrawString(msg_); }
 private:
  string msg_;
};

class MultilineTextBox : public Box {
public:
  MultilineTextBox(const vector<string> &msgs, int w, int h)
    : Box(w, h), msgs_(msgs) { }
  virtual void Draw(Window *w) {
    for (const auto &m : msgs_) { w->DrawString(m); }
  }
  int Area() { return w_ * h_ * msgs_.size(); }
 private:
  vector<string> msgs_;
};
```

(A) [5 pts] Draw a diagram of the inheritance relationship between these classes (ie, the "inheritance hierarchy"). You do not need to add the methods; just the class names.



(B) [7 pts] Suppose we define the following variables:

```
Box *b = new Box(1, 1);
TextBox *t = new TextBox("hello winter break", 10, 20);
MultilineTextBox *m = new MultilineTextBox({"bye", "autumn", "quarter"}, 20, 20);

Widget *w_b = b;
Widget *w_t = t;
Box *b_m = m;

Window *win;  // used for Draw() calls, below.  You can assume it's initialized
```

Which class's method is invoked by each of the following calls? If the call would not compile, please write "Compiler Error" instead.

Method Call	Invoked Method's Class
`t->Draw(win)`	TextBox
`w_b->Draw(win);`	Box
`w_t->Draw(win);`	TextBox
`(*t).Draw(win);`	TextBox
`(*w_t).Draw(win);`	Widget
`(*w_b)->Area();`	Compiler Error; `Widget` does not have an `Area()` method
`b_m->Area();`	Box
`m->Area();`	MultilineTextBox

(C) [6 pts] Using the same variables from part (B), fill in the blank with the C++-style cast that would cause the statement to compile. If the statement compiles without a cast, check the "Unnecessary" box. Recall that you may choose from `static_cast<>`, `dynamic_cast<>`, `const_cast<>`, and `reinterpret_cast<>`.

	Unnecessary
`w_b =                                  (b);`	x
`b =     dynamic_cast<Box*>      (w_b);`	

`double x = 3.14159;`
`int pi =  static_cast<int>       (x);`	

`t =  reinterpret_cast<TextBox*> (1234);`	
`const Window *cp =                    (win);`	x

`const Window cw;`
`win =  const_cast<Window *>      (&cw);`	

Student ID Number: ______________________

Note: the final exam, as given, had a typo which made the question harder than intended. I've preserved the exam version as "D.i" because of its educational value. The question, as intended, is presented as "D.ii".

(D.i) [6 pts] Does this code snippet have any memory allocation errors?

```
Box *b = new Box(1, 1);
TextBox *t = new TextBox(10, 20);
MultilineTextBox *m = new MultilineTextBox(20, 20);
Window win;  // takes ownership of widgets added to it
win.Add(b);
win.Add(t);
win.Add(m);
```

☐ No errors [✓ Memory leak (fix described below)] [✓ Double delete (fix described below)]

Because the `Window` is stack-allocated, it is destroyed at the end of this snippet. This drops the `Widgets`' refcount to 0, so they are also destroyed. However, when their destructors are called, the refcount for `Window` also drops to 0 so it is destroyed a second time. To fix this, we should allocate `Window` on the heap, which triggers the memory leak described in *D.ii*

(D.ii) [6 pts] Does this code snippet have any memory allocation errors?

```
Box *b = new Box(1, 1);
TextBox *t = new TextBox(10, 20);
MultilineTextBox *m = new MultilineTextBox(20, 20);
shared_ptr<Window> win(new Window);  // takes ownership of widgets added to it
win->Add(b);
win->Add(t);
win->Add(m);
```

☐ No errors [✓ Memory leak (fix described below)] ☐ Double delete (fix described below)

There is a `shared_ptr` cycle between `Window` and the various `Widgets`. As discussed in lecture, we can break this cycle by changing the type of `Widget`'s `owner_` pointer to `weak_ptr` (we also accepted a raw pointer as a valid response).

Question 5:

[3 pts] Let's map the latency of common computer operations to the human-scale operations required for studying for the 333 final. You may use the following:

A. Reading a sticky note on your monitor (0.5 secs)
B. Finding the right page/paragraph in the textbook kept next to your monitor (2 mins)
C. Asking on Piazza (36 mins)
D. Texting another 333 student for the answer (1 hour)
E. Requesting a scanned article from UW Libraries (2 days)
F. Buying the physical textbook *without Amazon Prime* (1 week)
G. Re-taking CSE 351 and then re-taking CSE 333 (20 weeks)
H. Buying the physical textbook *currently on Jupiter* (6 years)
I. Buying the physical textbook *currently in the Alpha Centauri system* (78,000 years)

Computer Operation	Human Analogue
L1 cache reference	A
Main memory reference	B
Packet round trip within same datacenter	F
Disk seek	G
Packet round trip across a submarine cable	H

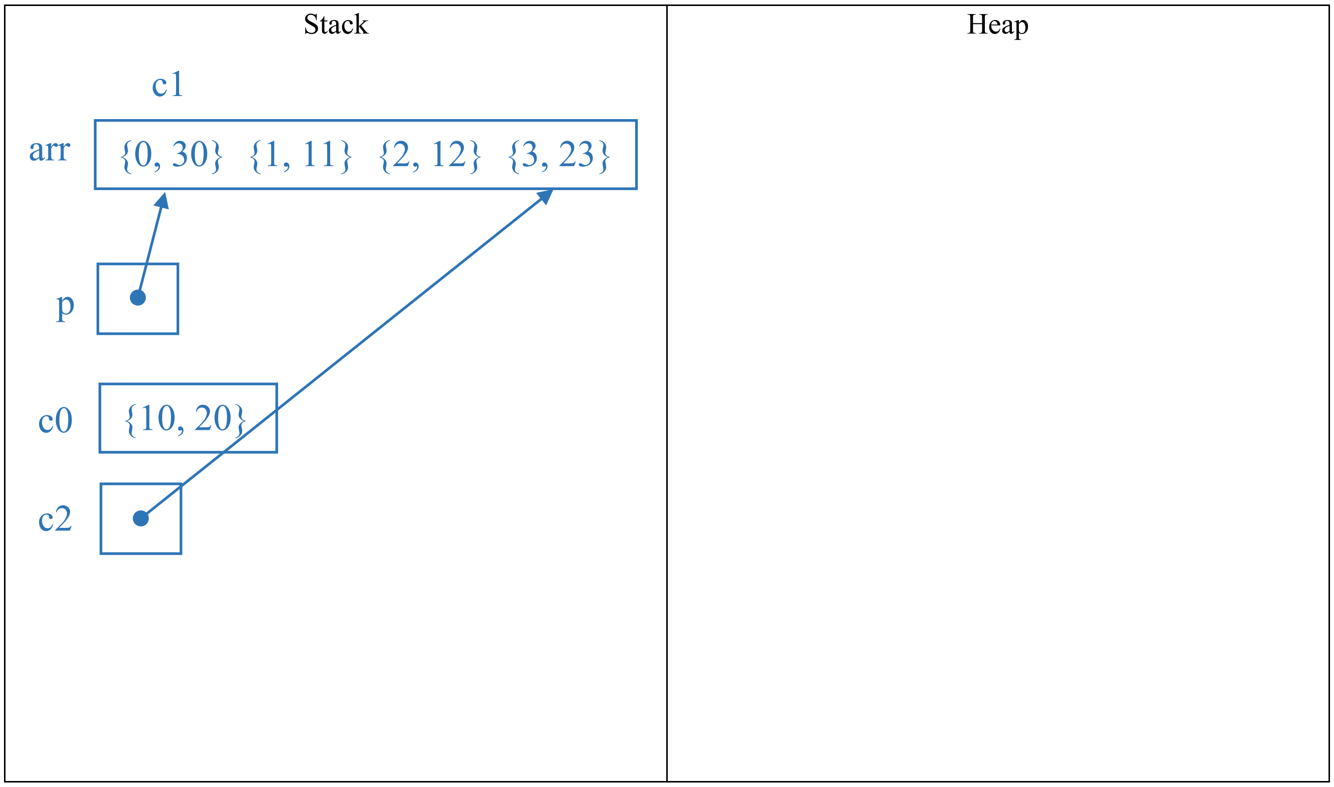
Question 6:

Consider the following code. Assume all relevant `#include` statements have been made.

```
struct Coordinate {
  int x, y;
};

void Modify(Coordinate &c) {
 c.y = c.y + 10;
}

int main(int argc, char *argv[]) {
  Coordinate arr[] = { {0, 10}, {1, 11}, {2, 12}, {3, 13} };
  Coordinate *p = &arr[0];

  Coordinate c0 = *p;
  c0.x = 10;
  Modify(c0);
  Modify(*p);

  Coordinate &c1 = arr[0];
  Modify(c1);
  Coordinate *c2 = arr + 3;
  Modify(*c2);

  // ** HERE **
  return EXIT_SUCCESS;
}
```

Student ID Number: ________________________

[6 pts] Draw a memory diagram showing the state of the program when we exit (ie, at "`*** HERE ***`").



Student ID Number: ______________________

Question 7:

(A) [7 pts] Recall that the 7 steps of server-side network programming are:

1. Get local IP address and port
2. Create socket
3. Bind socket to local IP address and port
4. Listen on socket
5. Accept connection from client
6. Read and write data on that connection
7. Close socket

The `HttpServer::Run()` method from HW4 implements the "server read/write loop"; a simplified version of it is below. Note that `SocketServer`'s methods have been renamed to make this question harder (sorry!).

In the right column, write the step number alongside the line in which it occurs. Note that every step occurs in a helper function, so a single line may have multiple step numbers .

```
void HttpServer::Run() {
  SocketServer ss;
  int listen_fd;                                          1, 2, 3, 4
  if (!ss.Method1(AF_INET6, &listen_fd)) {
    return;
  }

  ThreadPool tp(kNumThreads);
  while (1) {  // read/write loop
    HttpServerTask *hst = new HttpServerTask(
        &HttpServer_ThrFn);
    InitializeHST(hst);
                                                              5
    ss.Method2(&hst->client_fd,
               &hst->caddr,
               &hst->cport,
               &hst->cdns,
               &hst->saddr,
               &hst->sdns);                                   6
    tp.Dispatch(hst);  // Dispatch() invokes the
                       // function whose pointer was
                       // passed to hst's constructor
  }                                                           7
}
```

```
void HttpServer_ThrFn(HttpServerTask *hst) {
  HttpConnection conn(hst->client_fd);
  while (1) {
    HttpRequest req = conn.GetNextRequest();

    HttpResponse resp = ProcessRequest(req,
        hst->basedir, hst->indices);
    conn.WriteResponse(resp);

    if (req.GetHeaderValue("connection") == "close") {
      break;
    }
  }
}
```

For reference only.
Do not mark in this box.

(B) [4 pts] Oftentimes, when maintaining a server you want to answer questions such as “how many requests has it served?” or “what percentage of those requests are HTTP2?”. A common solution for these questions involve adding *global statistics* to your server. You reimplement `HttpServer_ThrFn()` as follows:

```
int g_request_count = 0;
int g_http2_request_count = 0;

void HttpServer_ThrFn(HttpServerTask *hst) {
  HttpConnection conn(hst->client_fd);
  while (1) {
    HttpRequest req = conn.GetNextRequest();
    g_request_count++;

    if (req.isHttp2()) {
      g_http2_request_count++;
    }

    HttpResponse resp = ProcessRequest(req,
        hst->basedir, hst->indices);
    conn.WriteResponse(resp);

    if (req.GetHeaderValue("connection") == "close") {
      break;
    }
  }
}
```

Does this code demonstrate a data race?

☐ No race (reason below) [✓ Yes, there is a race (fix described below)]

`HttpServer_ThrFn` is called by multiple threads concurrently. Therefore, we need to ensure access to our two global variables is synchronized. It is simplest to lock both of them with the same lock (though we accepted answers with two independent locks).

(C) [2 pts] Assume your server’s threadpool has 3 threads and it has processed 100 HTTP requests.

- If you answered “no race” in part (B), what is the value of `g_request_count` as viewed from the “main thread”?
- If you answered “yes, there is a race”, what set of values might `g_request_count` have, as viewed from the “main thread”? Assume the bug has not been fixed yet.

[1, 100]

Student ID Number: ______________________

(D) [2 pts] HW4 used threads to implement concurrency. If, instead, we had run `HttpServer_ThrFn()` in forked processes (using the double-fork trick), what would be the value or set of values for `g_request_count` as viewed from the "main process"? Justify your answer.

0. `g_request_count` is incremented by the forked process, which has its own virtual memory and therefore its own copy of `g_request_count`. The main process would never see the incremented value.

Question 8:

(A) [2 pts] Recall that each layer in the OSI network model (eg, physical, data link, network) has its own packet format, consisting of a packet header and a packet body. How is each layer's packet represented in the next layer down? For example, how is an IP packet (data link) represented as an ethernet packet (physical)?

Each packet's header and body is stored in the body of the layer below.

(B) [8 pts] Are each of the following statements true or false?

DNS is a network-level protocol	True / [✓ False]
SSH is an application-level protocol	[✓ True] / False
TCP is a client/server protocol (one server to one client)	[✓ True] / False
If a TCP packet needs to be split across multiple IP packets, it will add a sequence number to each IP packet to detect their intended ordering and to detect if any IP packets were lost	[✓ True] / False
If a host detects that an IP packet was lost, it will re-request the entire TCP packet.	True / [✓ False]
UDP is a client/server protocol (one server to one client)	True / [✓ False]
UDP is a good transport protocol for HTTP	True / [✓ False]
HTTP headers can contain arbitrary key/value pairs	[✓ True] / False

Question 9:

[1 pt; all non-empty answers receive this point] Draw or describe a friend for your unicorn, below.

Thanks for a great quarter! Have a wonderful winter break and come say "hi" in the new year!