Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2012

Quiz I

There are 13 questions and 8 pages in this quiz booklet. Answer each question according to the instructions given. You have **50 minutes** to answer the questions.

Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

For true/false and yes/no questions, you will receive 0 points for no answer, and negative points for an incorrect answer. We will round up the score for every *numbered* question to 0 if it's otherwise negative (i.e., you cannot get less than 0 on a numbered question).

If you find a question ambiguous, be sure to write down any assumptions you make. **Be neat and legible.** If we can't understand your answer, we can't give you credit!

Write your name in the space below. Write your initials at the bottom of each page.

THIS IS AN OPEN BOOK, OPEN NOTES, OPEN LAPTOP QUIZ, BUT DON'T USE YOUR LAPTOP FOR COMMUNICATION WITH OTHERS.

CIRCLE your recitation section number:

10:00	1. Rudolph/Grusecki		
11:00	2. Rudolph/Grusecki	3. Abelson/Gokce	4. Katabi/Joshi
12:00		5. Abelson/Gokce	6. Katabi/Joshi
1:00	7. Shavit/Moll	8. Szolovits/Fang	
2:00	9. Shavit/Moll	10. Szolovits/Fang	

Do not write in the boxes below

1-5 (xx/27)	6-7 (xx/15)	8-10 (xx/32)	11-13 (xx/26)	Total (xx/100)

Name:

I Reading Questions

1. [4 points]: Simon, in "The architecture of complexity," claims that hierarchical systems evolve more quickly than non-hierarchical systems of comparable size. Which of the following arguments does he say support this claim.

(Circle True or False for each choice.)

A. True / False The observation that complex systems are more comprehensible if they are neatly decomposable.

B. True / False The parable of the watchmakers.

C. True / False Thermodynamic considerations about entropy.

D. True / False The distinction between state descriptions and process descriptions.

2. [5 points]: Answer the following questions based on the X Window System paper.

(Circle True or False for each choice.)

A. True / False When a Web browser runs on a Unix workstation and displays its pages on that workstation using X, the browser is the client of both the web server and the X server.

B. True / False The window manager must be built into the core of the X server because it can operate on the windows of multiple clients.

C. True / False The X server of the 1980's has a complex management infrastructure for color map management because display controllers of that era could not provide enough memory to store the color of every pixel in each image.

D. True / False The X server informs its clients when a region of one of their windows becomes obscured so that the clients can stop sending update requests for that region.

E. True / False Synchronization errors could happen between the X server and its clients when network latencies delayed client responses.

3. [6 points]: Answer the following questions based on the Unix paper.
(Circle True or False for each choice.)

A. True / False Checking the return value of the `fork()` function enables a child process to execute different instructions from its parent.

B. True / False Since a child process can write to all files that are open by its parent at the time of the fork, these writes can create a race condition with writes from the parent.

C. True / False One of the advantages of multitasking is that it makes the system more responsive to user inputs.

4. [6 points]: This question is in the context of the Eraser paper. Assume a multi-threaded program has three locks: `mu`, `mu0`, and `mu1`, as well as an array `a` with two locations, `a[0]` and `a[1]`. Whenever a thread is about to modify the whole array (i.e., both `a[0]` and `a[1]`) it acquires the lock `mu`, but whenever it is about to modify `a[0]` alone it acquires `mu0`, and whenever it is about to modify `a[1]` alone it acquires `mu1`.

A. Yes / No Could Eraser's lockset algorithm detect a race condition with respect to accesses to either `a[0]` or `a[1]`?

5. [6 points]: This question continues the Eraser question. Assume a multi-threaded program has one lock `mu2` and an array `a` with two locations, `a[0]` and `a[1]`. Whenever a thread is about to modify either `a[0]` or `a[1]` it acquires the same lock `mu2`.

A. Yes / No Could Eraser's lockset algorithm detect a race condition with respect to accesses to either `a[0]` or `a[1]`?

Initials:

II Complexity

6. [10 points]: Chapter 1 of the text describes several techniques for coping with complexity: Modularity (M), Abstraction (A), Layering (L), Hierarchy (H), Design for iteration (D), and Indirection (I). For each of the following advantages, mark the appropriate letter (M, A, L, H, D, I) or N for "none of these" to say which technique *best* provides that advantage. For each question, there is only one best answer, but a given technique might be the best answer to more than one question.

A. **M A L H D I N** Helps the designers incorporate feedback in system implementations.

B. **M A L H D I N** Helps simplify the task of debugging a complex system by letting implementers deal with smaller components

C. **M A L H D I N** Makes it easier for designers to take advantage of delayed binding in system implementations.

D. **M A L H D I N** Ensures that the implementation will obey the robustness principle.

E. **M A L H D I N** If this is done correctly, it can help reduce the number of inter-module interactions in large systems.

III Names

7. [5 points]: One of the examples in the first hands-on exercise asked you to notice that even though your home directory might be `/mit/YOU`, the sequence `cd /mit/6.033; cd ../YOU` when executed in the `tcsh` shell does not get you to your home directory. However, if you perform the same experiment in `bash`, it works. From the viewpoint of our name resolution discussion, which statement is correct in `bash`?

(Circle the BEST answer)

A. When executing the `cd ..` command, bash determines if any shorter symbolic links exist to the resulting directory and displays the shortest one.

B. The bash shell uses not only the current directory as its name mapping context but also some additional history of how the current directory was reached.

C. Bash uses only Unix pathnames, not inodes.

D. None of the above.

Initials:

IV Concurrency

In this question, you can assume that loads and stores to variables are atomic, and neither the compiler nor hardware will ever reorder instructions. `continue` transfers control flow to the beginning of the `while` loop.

Consider the following lock implementation using a variable `x` for threads numbered 1 through `n` where initially `x = 0`. Each thread's number is stored in variable `i`.

```
acquire():
  while True:
    if x != 0:
      continue  ## retry
    x = i
    if x != i:
      continue  ## retry
    return

release():
  x = 0
```

8. [10 points]: Which of the following is true for the above algorithm:
(Circle the BEST answer)

A. It does not guarantee mutual exclusion.

B. it guarantees mutual exclusion but not deadlock freedom.

C. it guarantees both mutual exclusion and deadlock freedom.

Consider the following lock implementation using variables `x` and `y` for threads numbered 1 through `n`, where each thread's number is stored in variable `i`, and where initially `y = 0`:

```
acquire():
  while True:
    x = i
    if y != 0:
      continue  ## retry
    y = 1
    if x != i:
      continue  ## retry
    return

release():
  y = 0
```

9. [10 points]: Which of the following is true for the above algorithm:
(Circle the BEST answer)

A. It does not guarantee mutual exclusion.

B. it guarantees mutual exclusion but not deadlock freedom.

C. it guarantees both mutual exclusion and deadlock freedom.

Initials:

V Client/Server and bounded buffers

Consider the following bounded buffer code (`send` and `receive`), assuming the variables `bb.in` and `bb.out` are 64 bits, never overflow, can be read and written atomically, and neither the compiler nor hardware will ever reorder instructions:

```
send(bb, m):
  // each invocation of send has
  // its own local variables:
  //   my_send_index (64-bit int)
  while True:
    acquire(bb.lock)
    if bb.in - bb.out < N:
      my_send_index = bb.in
      bb.in = bb.in + 1
      release(bb.lock)
      bb.buf[my_send_index mod N] = m
      return
    release(bb.lock)
```

```
receive(bb):
  // each invocation of receive
  // has its own local variables:
  //   my_rec_index (64-bit int)
  //   m (message)
  while True:
    acquire(bb.lock)
    if bb.in > bb.out:
      my_rec_index = bb.out
      bb.out = bb.out + 1
      release(bb.lock)

      acquire(bb.rec_lock)
      m = bb.buf[my_rec_index mod N]
      release(bb.rec_lock)

      return m
    release(bb.lock)
```

10. [12 points]: Which of the following is true for the above implementation:
(Circle True or False for each choice.)

A. True / False The code is correct if there is one sender and one receiver executing at same time.

B. True / False The code is correct if there is one sender and many receivers executing at same time.

C. True / False The code is correct if there are many senders and one receiver executing at same time.

D. True / False The code is correct it there are many senders and many receivers executing at same time.

Initials:

VI Operating systems

11. [8 points]: Circle all of the function calls that directly correspond to system calls in a Unix system (based on the Unix paper) in the following implementation of the `cp` program:

```
void cp(char *srcpath, char *dstpath) {
  int src = open(srcpath, O_RDONLY);
  int dst = open(dstpath, O_WRONLY);

  if (src < 0 || dst < 0)
    exit(-1);

  while (1) {
    char buf[1024];
    ssize_t cc = read(src, buf, sizeof(buf));
    if (cc <= 0)
      break;
    ssize_t n = write(dst, buf, cc);
    assert(cc == n);
  }

  close(dst);
  close(src);
}

int main(int argc, char **argv) {
  cp(argv[1], argv[2]);
  exit(0);
}
```

Initials:

12. [10 points]: Suppose we run two user-mode programs, A and B, which are independent (e.g., which do not access any common files), on a Unix OS, and we run the Unix OS in a virtual machine (VM). What can fail if a bug (such as a divide-by-zero or a random memory write) appears in different components of the system? Assume there are no other bugs. Draw an `X` in the appropriate locations in the table below:

	Program A fails	Program B fails	Kernel fails	VM monitor fails
Bug in program A				
Bug in program B				
Bug in the kernel				
Bug in the VM monitor				

13. [8 points]: Suppose that you discover a bug in the implementation of `read()` that is invoked by `cp`, and you want to fix this bug.

A. True / False Fixing this bug will require modifying the `cp` program.

B. True / False Fixing this bug will require modifying the OS kernel.

End of Quiz I

Please double check that you wrote your name on the front of the quiz,
and circled your recitation section number.

Initials: