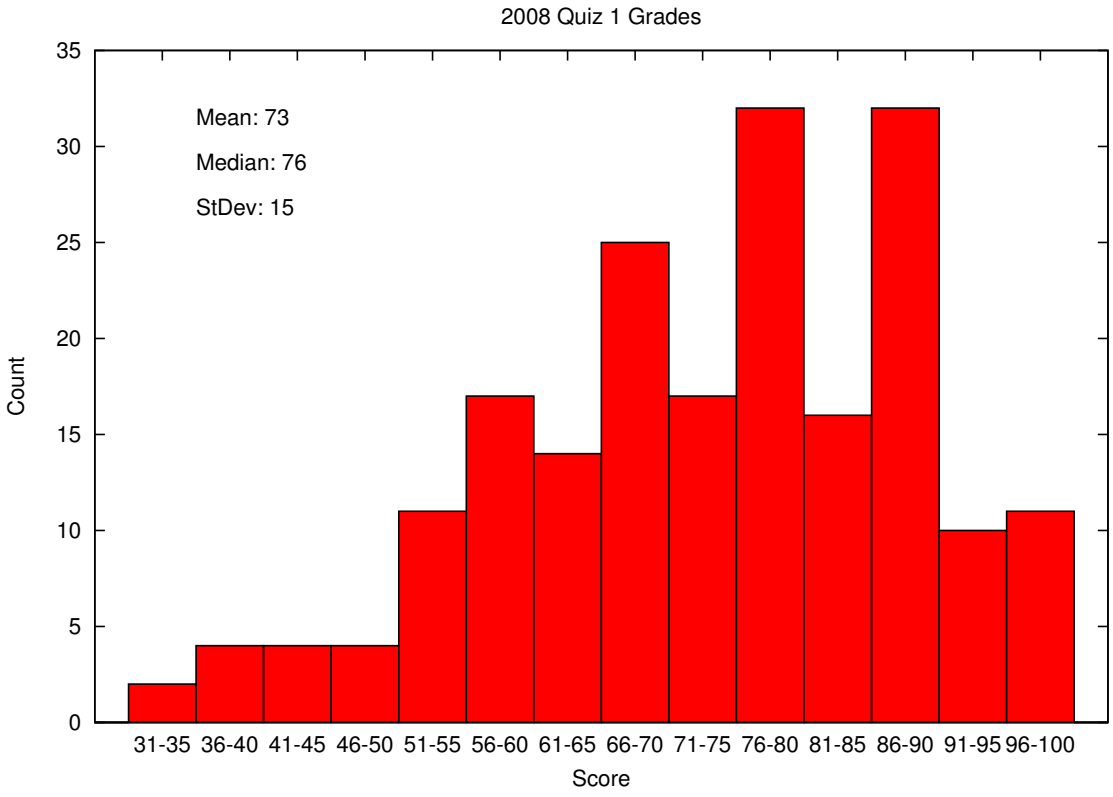
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2008

Quiz I - Solutions



I Reading Questions

1. [6 points]: Which of the following statements are true of the X Window System (as described in the X Windows paper, reading #5)?

(Circle True or False for each choice.)

A. True / False The only use for windows in X is to give each application its own virtual display.

False. *They are also used for menus, buttons, etc. This is the reason for the window hierarchy. It is only "top-level" windows that are meant to abstract the display.*

B. True / False X version 10 supports running an X client on the same computer as the X server.

True.

C. True / False When using an X server in a typical way, with several applications running in separate top-level windows, it's only the server that is responsible for determining the placement of top-level windows.

False. *The window manager also plays an important role, and it is not part of the server.*

2. [8 points]: Which of the following statements about Eraser (described in reading #7) is true?

(Circle True or False for each choice.)

A. True / False By extending the basic Lockset algorithm to reduce false positives for initialization and read sharing (Section 2.2), the number of false negatives is also reduced.

False. *The extended algorithm ignores unprotected access during initialization by adding the exclusive and shared states. However, there is a possibility of an undetected race condition when starting a thread. A modification could be made by the parent thread while the variable is still in the exclusive state, concurrent with starting a child thread that will access that variable. Depending on how quickly the child thread starts running, Eraser might or might not flag this as a violation.*

B. True / False For a given program, Eraser detects the same race conditions for every execution of this program.

False. *Eraser only flags errors in code that executes. Some code branches (e.g. error handling code) will only execute rarely, and may not execute in any given run of the program.*

C. True / False For each *shared-modified* variable, the Lockset algorithm determines a set of candidate locks. The Lockset algorithm reports a race condition if this set is empty.

True. *By the definition of the Lockset algorithm.*

D. True / False A use of locks that eliminates race conditions also eliminates deadlocks.

False. `acquire(lock); acquire(lock);` *has no races but will cause deadlock.*

Name:

3. [8 points]: Based on the description of UNIX in the 1974 Ritchie and Thompson paper (reading #6), which of the following statements are true?

(Circle True or False for each choice.)

A. True / False The `execute()` system call creates a new process, loads the requested image, loads arguments onto the stack, and jumps to the appropriate entry point.

False. `execute()` *does not create a new process; it replaces the currently running image.*

B. True / False A file's *i-node* is stored in the first 64 bytes of the file.

False. *I-nodes are located in an i-node table, in a part of the disk distinct from file contents.*

C. True / False Suppose a process allocates a variable `x` on the stack, then `fork`s a child process. The child may change its value of variable `x` without changing the value of `x` seen by the parent process.

True. *The child's memory is distinct from the parent's.*

D. True / False Any two running processes can communicate by creating a shared pipe.

False. *Two processes can only communicate through a pipe if a common ancestor created the pipe, and let the processes inherit the pipe file descriptors.*

4. [8 points]: Based on the the investigation of the Therac-25 accidents (reading #4), which of the following statements about the Therac-25 are true?

(Circle True or False for each choice.)

A. True / False The race conditions that caused some of the accidents could have been avoided by the use of locks and condition variables.

True. *Proper use of locks and condition variables would have eliminated at least one of the bugs. For example, locking the MEOS two-byte variable would have prevented the prescription from being changed after Datent has read the index from the high-order byte of MEOS.*

B. True / False The manufacturer proved that faulty switches caused the first accidents.

False. *The manufacturer believed that this was the cause of the accident, but were not able to show their theory was correct.*

C. True / False The authors of the paper believe that, in practice, hardware interlocks are necessary for safety.

True. *The authors discuss the need for hardware interlocks in critical systems in the "System Engineering" section of "Lessons Learned", on p. 38 of the paper.*

D. True / False The fact that the Therac-25 was a multi-function machine, supporting two types of radiation, contributed to the accidents.

True. *Some of the accidents occurred when one part of the machine was set for electron radiation and another part was set for a X-rays.*

Name:

II UNIX File System

For his many past sins on previous 6.033 quizzes, Ben Bitdiddle is assigned to spend eternity maintaining a PDP-11 running the version of UNIX described in the 1974 Ritchie and Thompson paper. The file system implementation is described in Section IV of the paper.

Recently, one of his user's database applications failed after reaching the file size limit of 1,082,201,088 bytes (approx 1 GB). In an effort to solve the problem, he upgraded the computer with an old 4 GB (2^{32} byte) drive – the disk controller hardware supports 32 bit sector addresses, and can address disks up to 2 TB in size. Unfortunately, Ben is disappointed to find the file size limit unchanged after installing the new disk.

In this question, the term *block address* refers to the block pointers stored in i-nodes. In Ben's system (described in the paper) each i-node contains 13 block addresses of 4 bytes each; the first 10 block addresses point to the first 10 blocks of the file, with the remaining 3 addressing the rest of the file. The 11th block address points to an indirect block, containing 128 block addresses, the 12th block address points to a double-indirect block, containing 128 indirect block addresses, and the 13th block address points to a triple-indirect block, containing 128 double-indirect addresses.

5. [12 points]: Which of the following adjustments, applied *individually*, will allow files larger than the current 1 GB limit to be stored?

(Circle ALL that apply)

A. Increasing the file size field in the i-node from a 32 bit to 64 bit value.

False. *The maximum file size is bounded to approx 1 GB by the number of block addresses in an i-node and its indirect blocks. Increasing the size field alone won't help.*

B. Increasing the number of bytes per block from 512 to 2048 bytes.

True. *If the block size is b, the triple-indirect block can refer to up to $b * (b/4)^3$ bytes. This is 256 GB for 2048-byte blocks. Now the maximum file size would be 4 GB, limited by the 32-bit size field in the i-node.*

C. Reformatting the disk to increase the number of i-nodes allocated in the i-node table.

False. *Increasing the number of i-nodes increases the number of files the file system can contain, but does not increase the maximum size of a file.*

D. Replacing one of the direct block addresses in each i-node with an additional triple-indirect block address.

True. *This will almost double the maximum file size.*

Name:

Ben observes that there are 52 bytes allocated to block addresses in each i-node (13 block addresses at 4 bytes each), and 512 bytes allocated to block addresses in each indirect block (128 block addresses at 4 bytes each). He figures that he can keep the total space allocated to block addresses the same, but change the size of each block address, to increase the maximum supported file size. While the number of block addresses in i-nodes and indirect blocks will change, Ben keeps exactly one indirect, one double-indirect and one triple-indirect block address in each i-node.

6. [12 points]: Which of the following adjustments, applied *individually* (without any of the modifications in the previous question), will allow files larger than the current 1 GB limit to be stored?
(Circle ALL that apply)

A. Increasing the size of a block address from 4 bytes to 5 bytes.

False. *This will reduce the maximum file size, because indirect blocks will store fewer block addresses.*

B. Decreasing the size of a block address from 4 bytes to 3 bytes.

True. *This will increase the maximum file size by increasing the number of addresses that each indirect block can contain. 24-bit addresses are big enough to address more than 1 GB of blocks.*

C. Decreasing the size of a block address from 4 bytes to 2 bytes.

False. *Although this will increase the number of blocks that can be referenced by an indirect block, the total number of blocks addressable is only 64K and thus the total size of the file system is limited to 24MB. This is smaller than the original maximum file size of 1GB.*

Name:

III Course Swap

The Subliminal Sciences department, in order to reduce the department head's workload, has installed a web server to help assign lecturers to classes for Fall 2008. There happen to be exactly as many courses as lecturers, and department policy is that every lecturer teach exactly one course, and every course have exactly one lecturer. For each lecturer in the department, the server stores the name of the course currently assigned to that lecturer. The server's web interface supports one request: to swap the courses assigned to a pair of lecturers.

Version One of the server's code looks like this:

```
// CODE VERSION 1

String assignments[]; // assignments[lecturer] -> a course name

void server():
  while true:
    m = wait for a request message
    value = m.function(m.arguments, ...) // execute function in request message
    send value to m.sender

String exchange(lecturer1, lecturer2):
  temp = assignments[lecturer1]
  assignments[lecturer1] = assignments[lecturer2]
  assignments[lecturer2] = temp
  return "OK"
```

Note that there is only this one application thread on the server; the server handles only one request at a time. Requests contain a function and arguments (in this case `exchange(lecturer1, lecturer2)`) which is executed by the `m.function(m.arguments, ...)` call in the `server()` procedure.

Name:

For all following questions, assume that there are no lost messages and no crashes. The operating system buffers incoming messages. When the server program asks for a message of a particular type (e.g. a request), the operating system gives it the oldest buffered message of that type.

Assume that network transmission times never exceed a fraction of a second, and that computation also takes a fraction of a second. There are no other concurrent operations other than what is mentioned in the question (and implied by the code), and no other activity on the server computers.

Suppose the server starts out with the following assignments:

```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Numerology"
```

7. [8 points]: Lecturers Katabi and Morris decide they wish to swap lectures, so that Katabi teaches Numerology and Morris teaches Steganography. They each send an `exchange("Katabi", "Morris")` request to the server at the same time. If you look a minute later at the server, which states are possible?

(Circle ALL that apply)

A.
```
assignments["Katabi"] = "Numerology"
assignments["Morris"] = "Steganography"
```

No.

B.
```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Numerology"
```

Yes. *The server is single threaded and there is no possibility of races, hence the server will process both requests one at a time, in the order in which they arrive. The result will be two swaps.*

C.
```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Steganography"
```

No.

D.
```
assignments["Katabi"] = "Numerology"
assignments["Morris"] = "Numerology"
```

No.

Name:

The Department of Dialectic decides it wants its own lecturer assignment server. Initially it installs a completely independent server from that of Subliminal, with the same rules (an equal number of lecturers and courses, with a one-to-one matching). The two departments later decide that they wish to allow their lecturers to teach courses in either department, so they extend the server software in the following way. Lecturers can send either server a `crossexchange` request, asking to swap courses between a lecturer in that server's department and a lecturer in the other server's department. In order to implement `crossexchange`, the servers can send each other `set-and-get` requests, which set a lecturer's course and return the lecturer's previous course. Here's Version Two of the server code, for both departments:

```
// CODE VERSION 2
// server() and exchange() are the same as in Version One

crossexchange(local-lecturer, remote-lecturer):
  temp1 = assignments[local-lecturer]
  send "set-and-get", remote-lecturer, temp1 to other server
  temp2 = wait for response to "set-and-get"
  assignments[local-lecturer] = temp2
  return "OK"

set-and-get(lecturer, course):
  old = assignments[lecturer]
  assignments[lecturer] = course
  return old
```

Name:

Suppose the starting state on the Subliminal server is:

```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Numerology"
```

And on the Dialectic server:

```
assignments["Jerison"] = "Epistemology"
assignments["Goemans"] = "Reductionism"
```

8. [12 points]: At the same time, lecturer Katabi sends a `crossexchange("Katabi", "Jerison")` request to the Subliminal server, and lecturer Goemans sends a `crossexchange("Goemans", "Morris")` request to the Dialectic server. If you look a minute later at the Subliminal server, what states are possible?

(Circle ALL that apply)

A.
```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Numerology"
```

Yes. *If each server starts processing its request at about the same time, so that their set-and-get messages cross, then neither will be able to respond to the other's set-and-get. Both of them will be waiting for a response in* `crossexchange()`*, not waiting for a request in* `server()`*. The result will be deadlock, and thus no changes to the lecturer assignments.*

B.
```
assignments["Katabi"] = "Epistemology"
assignments["Morris"] = "Reductionism"
```

Yes. *If the servers finish one request before they start the other, then both swaps will happen.*

C.
```
assignments["Katabi"] = "Epistemology"
assignments["Morris"] = "Numerology"
```

No. *If one swap completes, there is nothing to stop the other swap from also completing.*

Name:

In a quest to increase performance, the two departments make their servers threaded: each server serves each request in a separate thread. Thus if multiple requests arrive at roughly the same time, the server may process them in parallel. Each server has multiple CPUs. Here's the threaded server code, Version Three:

```
// CODE VERSION 3
// exchange(), crossexchange(), and set-and-get() are the same as in Version Two

server():
  while true:
    m = wait for a request message
    allocate_thread(doit, m) // create a new thread that runs doit(m)

doit(m):
  value = m.function(m.arguments, ...)
  send value to m.sender
  exit() // terminate this thread
```

9. [12 points]: With the same starting state as the previous question, but with the new version of the code, lecturer Katabi sends a `crossexchange("Katabi", "Jerison")` request to the Subliminal server, and lecturer Goemans sends a `crossexchange("Goemans", "Morris")` request to the Dialectic server, at the same time. If you look a minute later at the Subliminal server, what states are possible?

(Circle ALL that apply)

A.
```
assignments["Katabi"] = "Steganography"
assignments["Morris"] = "Numerology"
```

No.

B.
```
assignments["Katabi"] = "Epistemology"
assignments["Morris"] = "Reductionism"
```

Yes. *The threads allow each server to simultaneously wait for the other server's reply to set-and-get, and process the other server's set-and-get request. This solves the deadlock from the previous question, so both exchanges will finish.*

C.
```
assignments["Katabi"] = "Epistemology"
assignments["Morris"] = "Numerology"
```

No. *Version 3 of the code is susceptible to race conditions, owing to its use of threads and concurrent modification of shared variables. The particular requests in this question access disjoint variables however, and hence no race condition is exposed.*

Name:

An alert 6.033 student notes that Version Three may be subject to race conditions. He changes the code to have one lock per lecturer, stored in an array called `locks[]`. He changes `exchange()`, `crossexchange()`, and `set-and-get()` to acquire locks on the lecturer(s) they affect. Here is the result, Version Four:

```
// CODE VERSION 4
// server() and doit() are the same as in Version Three

exchange(lecturer1, lecturer2):
  acquire(locks[lecturer1])
  acquire(locks[lecturer2])
  temp = assignments[lecturer1]
  assignments[lecturer1] = assignments[lecturer2]
  assignments[lecturer2] = temp
  release(locks[lecturer1])
  release(locks[lecturer2])
  return "OK"

crossexchange(local-lecturer, remote-lecturer):
  acquire(locks[local-lecturer])
  temp1 = assignments[local-lecturer]
  send "set-and-get", remote-lecturer, temp1 to other server
  temp2 = wait for response to "set-and-get"
  assignments[local-lecturer] = temp2
  release(locks[local-lecturer])
  return "OK"

set-and-get(lecturer, course):
  acquire(locks[lecturer])
  old = assignments[lecturer]
  assignments[lecturer] = course
  release(locks[lecturer])
  return old
```

Name:

10. [14 points]: This code is subject to deadlock. For each situation below, indicate whether deadlock can occur. In each situation, there is no activity other than that mentioned.

(Circle Yes or No for each choice.)

A. Yes / No Client A sends `exchange("Katabi", "Morris")` at the same time that client B sends `exchange("Katabi", "Morris")`, both to the Subliminal server.

No.

B. Yes / No Client A sends `exchange("Katabi", "Morris")` at the same time that client B sends `exchange("Morris", "Katabi")`, both to the Subliminal server.

Yes. *Each exchange could acquire one of the locks, and then both will wait forever waiting for the other lock. Neither can continue (and release its lock) until the other finishes.*

C. Yes / No Client A sends `crossexchange("Morris", "Jerison")` to the Subliminal server at the same time that client B sends `crossexchange("Goemans", "Katabi")` to the Dialectic server.

No.

D. Yes / No Client A sends `crossexchange("Morris", "Jerison")` to the Subliminal server at the same time that client B sends `crossexchange("Jerison", "Morris")` to the Dialectic server.

Yes. *Each crossexchange could acquire its local lock at about the same time, thus blocking the other's set-and-get from proceeding. Neither can continue (and release its lock) until the other finishes.*

E. Yes / No Client A sends `crossexchange("Morris", "Jerison")` to the Subliminal server at the same time that client B sends `crossexchange("Goemans", "Morris")` to the Dialectic server.

No. *While one of these may need to wait for the other to release a lock, there is no cyclic dependency, and thus no deadlock.*

End of Quiz I

Name: