The University of Texas at Austin
CS 372H Introduction to Operating Systems: Honors: Spring 2011

FINAL EXAM

- This exam is **3 hours**. Stop writing when "time" is called. *You must turn in your exam; we will not collect it.* Do not get up or pack up in the final ten minutes. The instructor will leave the room 3 hours and 3 minutes after the exam begins and will not accept exams outside the room.
- There are **27** questions in this booklet. Some may be harder than others, and some earn more points than others. You may want to skim all questions before starting. Note that you are going to need to move through the short ones quickly.
- **This exam is closed book and notes. You may not use electronics: phones, calculators, laptops, etc.** You may refer to TWO two-sided 8.5x11" sheet with 10 point or larger Times New Roman font, 1 inch or larger margins, and a maximum of 55 lines per side.
- If you find a question unclear or ambiguous, be sure to write any assumptions you make.
- Follow the instructions: if they ask you to justify something, explain your reasoning and any important assumptions. **Write brief, precise answers. Rambling brain dumps will not work and will waste time.** Think before you start writing so you can answer crisply. Be neat. If we can't understand your answer, we can't give you credit!
- Grading for True/False questions is as follows. We grade by individual True/False item: correct items earn positive points, blank items earn 0 points, and incorrect items earn negative points. However, the minimum score on any question—that is, any group of True/False items—is 0.
- *There is almost no credit for leaving questions blank.* The exception is as follows: if a problem is worth 6 or more points, then completely blank answers will get 15%-20% of the credit. Note that by *problem* we mean numbered questions for which a point total is listed. *Sub-problems* with no points listed are not eligible for this treatment. Thus, if you attempt any sub-problem, you may as well attempt the other sub-problems in the problem.
- Don't linger. If you know the answer, give it, and move on.
- **Write your name and UT EID on this cover sheet and on the bottom of every page of the exam.**

Do not write in the boxes below.

I (xx/25)	II (xx/20)	III (xx/19)	IV (xx/23)	V (xx/13)	Total (xx/100)

Name: Solutions **UT EID:**

I Concurrency and other pre-midterm material (25 points total)

1. [4 points]

Circle True for False for each item below:

True / False On the x86, if a given memory reference (load or store) causes a TLB miss, then that memory reference also causes a page fault.

False. The processor can often patch up a TLB miss using the page structures, without needing to ask the OS for help via a page fault.

True / False In JOS, if an environment issues a syscall, thereby causing the x86 to begin executing the kernel, the processor flushes the TLB before switching to supervisor mode and executing kernel code.

False.

True / False Under gcc's calling conventions, when a function `f()` calls a function `g()` that takes arguments, `f()` pushes arguments on the stack for `g()`.

True.

True / False Under gcc's calling conventions, when a function `f()` calls a function `g()` that takes arguments, `g()` can gain access to the arguments through the stack frame pointer (`%ebp` on the x86).

True.

2. [3 points] Most round-robin schedulers use a fixed size quantum. Give an argument in favor of and against a small quantum.

Below, *briefly* state two arguments, one for and and one against a small quantum:

A small quantum means finer-grained sharing (more responsiveness) but at the cost of higher overhead, since each context switch costs something.

3. [15 points] In this problem, you are the organizer of an expo that specializes in electronic entertainment. You want to allow attendees to play a new game demo. You model the attendees as threads, called *players*, and your job is to synchronize access to a single copy of the game, as follows:

- When a player arrives, he or she waits in a waiting area.
- Once there are 4 or more players waiting to play, you allow exactly 4 of them to leave the waiting area to begin playing. These four leave the waiting area and approach the game console.
- When a player reaches the console, the player waits until all four players are at the console, at which point all four players begin playing.
- A player may leave the console. However, you cannot allow *any* new players to begin playing until *all* four players have left.
- You need not let players out of the waiting area in the order in which they arrived.
- You cannot assume that a player will ever finish playing.

You decide to solve this synchronization problem using two types of barriers:

```
GameBarrier gb;
ConsoleBarrier cb;
```

The definition of these barriers is on the next page. The use of these barriers is given by the following sequence for a player. Note that each player thread has a pointer to two global barriers.

```
void player(thread_id tid, GameBarrier* gb, ConsoleBarrier* cb) {

    gb->waitToPlay();

    cb->waitAtConsole();

    play();

    gb->donePlaying();
}
```

The `GameBarrier` can be in one of three states:

- `GAME_NOTREADY`: There are fewer than four players waiting to play. When the barrier is in this state, no player can progress beyond `waitToPlay()`.
- `GAME_FILLING`: There are (or were) at least four players waiting to play, and either we are in the first turn or else all four players from the prior turn have departed (via `donePlaying()`). When the barrier is in this state, a player can progress beyond `waitToPlay()`, and in fact four players must progress beyond this function. When exactly four players have progressed beyond this function, the barrier enters the `GAME_FILLED` state.
- `GAME_FILLED`: Four players have been sent to the console, and the turn is not over (meaning that the departure of all four *from* the console via `donePlaying()` has not yet taken place). When the barrier is in this state, no waiting player can progress beyond `waitToPlay()`.

The `ConsoleBarrier` can be in one of two states:

- `CONSOLE_WAIT`: Four players have not yet arrived at the console in the current round. When the barrier is in this state, no player can progress beyond `waitAtConsole()`.
- `CONSOLE_ALLOW`: Four players have arrived in the current round. When the barrier is in this state, all four waiting players must progress beyond `waitAtConsole()`, after which the state reverts to `CONSOLE_WAIT`.

Note that part (but not all) of your work is to ensure that the barriers make the correct state transitions.

Below, where indicated, fill out the variables and methods for the `GameBarrier` and `ConsoleBarrier` objects. Follow the coding standards from class.

Name: Solutions **UT EID:**

```
class GameBarrier {
  public:
    GameBarrier();        /* You will partially implement this */
    ~GameBarrier() {}
    void waitToPlay();    /* You will implement this */
    void donePlaying();   /* You will implement this */
  private:
    /* this barrier can be in one of three states; note the 'state' variable */
    typedef enum {GAME_NOTREADY, GAME_FILLING, GAME_FILLED} state_t;

    Mutex mutex;
    state_t state;
    /* INSERT MORE BELOW */




};

GameBarrier::GameBarrier() {
    state = GAME_NOTREADY:
    /* INITIALIZE ANY OTHER VARIABLES. */



}

void
GameBarrier::waitToPlay() {
    /* YOU MUST FILL IN THIS FUNCTION */















}
```

```
void
GameBarrier::donePlaying() {
    /* YOU MUST FILL IN THIS FUNCTION */









}

class ConsoleBarrier {
  public:
    ConsoleBarrier();       /* You will partially implement this */
    ~ConsoleBarrier() {}
    void waitAtConsole();   /* You will implement this */

  private:
    /* this barrier can be in one of two states; note the 'state' variable */
    typedef enum {CONSOLE_WAIT, CONSOLE_ALLOW} state_t;

    Mutex mutex;
    state_t state;
    /* INSERT MORE BELOW */



};


ConsoleBarrier::ConsoleBarrier() {
    state = CONSOLE_WAIT;
    /* INITIALIZE ANY OTHER VARIABLES */



}
```

```
void
ConsoleBarrier::waitAtConsole() {
    /* YOU MUST FILL IN THIS FUNCTION */







}
```

```
class GameBarrier {

    ..............

    Cond cv;
    int num_waiters;
    int num_players;
};


GameBarrier::GameBarrier() {
    state = GAME_NOTREADY:

    num_waiters = 0;
    num_players = 0;
}

void
GameBarrier::waitToPlay() {
    mutex.acquire();

    if (++num_waiters >= 4 && state == GAME_NOTREADY) {
        state = GAME_FILLING;
        cv.broadcast(&mutex);
    }

    while (state != GAME_FILLING) {
        wait(&mutex, &cv);
    }

    --num_waiters;

    if (++num_players == 4)
        state = GAME_FILLED;

    mutex.release();
}

void
GameBarrier::donePlaying() {
    mutex.acquire();

    if (--num_players = 0) {
        state = GAME_NOT_READY;
        if (num_waiters >= 4) {
            state = GAME_FILLING;
            cv.broadcast(&mutex);
        }
    }
    mutex.release();
}


class ConsoleBarrier {

    ...............

    Cond cv;
    int num;

};


ConsoleBarrier::ConsoleBarrier() {
    state = CONSOLE_WAIT;
    num = 0;
}

void
ConsoleBarrier::waitAtConsole() {
    mutex.acquire();

    if (++num == 4) {
        state = CONSOLE_ALLOW;
        cv.broadcast(&mutex);
    }

    while (state == CONSOLE_WAIT) {
        cv.wait(&mutex);
    }

    if (--num == 0) {
        state = CONSOLE_WAIT;
    }

    mutex.release();
}
```

4. [3 points] This question is about the correctness of the pseudocode below. The programmer intends that `g()` not execute unless `f()` has executed; the two functions are called by different threads. Assume POSIX thread semantics (Hansen semantics); that is, the thread package provides the same guarantees that it did in lab T. *Read the code carefully.*

```
int f_ran = FALSE;
Mutex mutex;
Cond cv;

Monitor::t1() { // called by a thread
    mutex.acquire();
    if (f_ran == FALSE)
        cv.wait(&mutex);

    g(); /* <-- It is an error if g() executes before f() */
    mutex.release();
}

Monitor::t2() { // called by another thread
    mutex.acquire();
    f();
    f_ran = TRUE;

    cv.broadcast(&mutex);
    mutex.release();
}
```

Under which conditions is the above pseudocode correct? Circle the BEST answer and then briefly JUSTIFY your answer.

A The code executes on a single processor.

B The memory model is sequential consistency.

C The threads are user-level threads.

D The system contains only two threads, one that calls `t1()` and one that calls `t2()`.

E The `broadcast()` is replaced with a `signal()`.

F The code is correct if conditions **A**, **C**, and **D** all hold simultaneously.

G The code is correct if conditions **B** and **D** both hold simultaneously.

H The code is correct if conditions **A**, **C**, **D**, and **E** all hold simultaneously.

I None of the above.

Justify your answer briefly below:

I. This question was on the midterm too. This code is never correct; none of the proposed conditions in any combination makes it correct. As we mentioned a number of times in lecture, the `if (foo) cv.wait()` pattern is always wrong. Why? Several reasons. Beyond the fact that the coding standards ruled out this pattern, the threading package is allowed to return from `wait()` at any time, a point that we discussed in lecture. (In fact, the POSIX documentation says the same thing, and that waiters should thus spin in a while loop.) This is true regardless of whether we have one or two threads, user-level or kernel-threads, signal or broadcast, sequential consistency or not, etc. Since a thread can wake at any time, even when not signaled, the code *must* check any required barrier conditions after waking from `wait()` and before proceeding.

II I/O, Disks, file systems, transactions (20 points total)

5. [4 points] Consider a computer with a processor that operates at 1 GHz (10^9 cycles/second). When a network packet arrives, the network card interrupts the CPU, which then processes the packet. The cost of the following sum to 10,000 cycles: a context switch to the interrupt handler, handling a packet, and the context switch out of the interrupt handler. A lot of other computers want to talk to this computer, and for a time, it receives 100,000 packets per second. Assume one interrupt per packet.

Below, state the total percentage of the processor's cycles that are spent in interrupt-related code (meaning the context switching and packet handling). Explain your answer *briefly* (for example, by showing your work).

100%. 10,000 cycles/interrupt * 1 interrupt/packet * 100,000 packets/second = 10^9 cycles/second on interrupts, which is all that the processor has.

Fill in the blank: during this busy time, the device driver for the network card should not use interrupts but rather ________________.

polling.

6. [3 points] Consider a system that uses transactions to provide atomicity (specifically, all-or-nothing atomicity) with respect to stable storage. (Other names for *stable storage* are *cell storage* and *non-volatile storage*.) Recall that this kind of atomicity provides the following: after a system restart, each transaction will be such that either all of the updates in the transaction will appear to be "done" or none of the updates in the transaction will appear to be "done". Here, "done" means "applied to stable storage". To provide this type of atomicity, what rule must the transaction manager obey for each update, with respect to its write-ahead log?

Circle the BEST answer below:

A The transaction manager can apply an update to cell storage only after recording that update in its write-ahead log.

B The transaction manager can record an update in its write-ahead log only after applying the update to cell storage.

C The order doesn't matter: the transaction manager can record an update in the log and in cell storage in any order, as long as it respects the invariant that COMMIT records are atomic.

A. It's a write-ahead log. The whole concept of write-ahead logs is that one modifies cell storage only after logging the change.

7. [6 points] Below are three different disk scheduling algorithms and three desirable properties of a disk scheduling algorithm. Your job will be to mark up a table to indicate which algorithms have which properties. The three algorithms are:

– First Come First Served (**FCFS**). The algorithm processes requests in the order that they are received.

– Shortest Seek Time First (**SSTF**). The algorithm picks the request with the shortest seek time. Assume that seek times are predictable, which means that this algorithm can be implemented.

– Elevator Scheduling (**Elevator**). The algorithm maintains a direction flag and sweeps along its current direction, picking the request with the shortest seek time. The algorithm switches direction only if no requests are left in the current direction.

The three desirable properties are:

– Whether the algorithm exploits request locality (**exploit locality**)

– Whether the algorithm bounds waiting time for all requests (**bounded waiting**)

– Whether the algorithm provides fairness (**fairness**). Fairness means that each disk request is treated equally, regardless of what position on the disk the request is referencing.

Fill in the cells of the table below, using a check mark to indicate that the algorithm has the desired property. If the algorithm does not have the desired property, leave the table cell blank.

	exploit locality	bounded waiting	fairness
FCFS		x	x
SSTF	x		
Elevator scheduling	x	x	

8. [7 points] Consider a file system that has the following description:

- The disk is divided into 1024-byte blocks.
- The beginning of the disk contains an array of 2^{16} inodes, each of which can represent a file or be unallocated.
- A file has an indexed structure: an inode contains (a) 8 data block pointers, each of which is 4 bytes and each of which points to a disk block and (b) a pointer to ONE indirect block, which is a disk block that itself contains data block pointers.
- The inode also contains a userid (2 bytes), three time stamps (4 bytes each), protection bits (2 bytes), a reference count (3 bytes), and the size (4 bytes).
- A directory contains a list of `(file_name, inode_number)` pairs, where the `file_name` portion is always *exactly* 14 bytes, including the null terminator (if the `file_name` would otherwise be fewer than 14 bytes, it is padded to 14 bytes).

Below, state the maximum file size, and explain *briefly*, for example by showing your work. You may express your answer as a sum of powers-of-two.

The data pointed to by the direct block pointers can be as large as $1024 \cdot 8$ bytes. There is a single indirect block pointer, and it can contain as many as $\frac{1024}{4}$ pointers, each of which points to 1024 bytes of data. Thus, the total is:

$$1024 \cdot 8 + 1024\left(\frac{1024}{4}\right) = 2^{13} + 2^{18} = 270,336$$

Below, state the maximum number of files in a directory, and explain *briefly*, for example by showing your work. Again, you may express your answer as a sum of powers-of-two.

Directories are implemented as files. Also, each directory entry is 16 bytes (14 bytes for the file name and 2 bytes for the inode number, since there are 2^{16} inodes). Thus, we take our answer to the previous question and divide by 16 bytes:

$$\frac{2^{13} + 2^{18}}{2^4} = 2^9 + 2^{14} = 16,896$$

Since there are 2^{16} inodes on the disk, our answer is $\min\{2^{16}, 16896\} = 16896$.

If we wanted to move to LFS (the log-structured file system), which of the items in the description above would we *have* to modify, and why? Note that we are asking "which items" and "why"; we are not asking you to describe the exact changes required nor are we asking about *other* needed changes.

The only one that needs to change is the inode array, since, under LFS, inodes do not live in a fixed location on the disk.

Name: Solutions **UT EID:**

III Networks, RPC, distributed systems (19 points total)

9. [2 points] Some application-layer protocols include a destination field in the application-layer header. Why?

Circle the BEST answer:

A So the protocol can check that the network layer delivered the packet containing the application's message to the correct endpoint.

B Because it is the application layer that makes routing and forwarding decisions.

C Because the network layer uses the application-layer header to route and forward the packet.

D Because the sender's link layer needs this field to decide which network protocol to use.

A is right: the application layer cannot depend on the layers below it. B is not right because it is not the application layer that makes forwarding decisions. C is not right because the network layer has its own header. D is not right because the link layer doesn't decide on the network protocol.

*T*he above exercise is borrowed from J. H. Saltzer and M. F. Kaashoek, *Principles of Computer System Design: An Introduction*, Morgan Kaufmann, Burlington, MA, 2009. Chapter 7. Available online.

10. [2 points] Ethernet cards have unique addresses built into them. What role do these unique addresses play in the Internet?

Circle the BEST answer:

A None. They are there for Macintosh compatibility only.

B A portion of the Ethernet address is used as the domain name of the computer using the card.

C They provide routing information for packets destined to non-local subnets.

D They are used as private keys in the Security Layer of the ISO protocol.

E They provide addressing within each subnet for an Internet address resolution protocol.

F They provide secure identification for warranty service.

E.

The above exercise is borrowed from J. H. Saltzer and M. F. Kaashoek, *Principles of Computer System Design: An Introduction*, Morgan Kaufmann, Burlington, MA, 2009. Chapter 7. Available online.

11. [3 points] The NFS authors had a goal of *transparency*. They wanted applications to be unable to distinguish whether a file system was (a) a remote file system served from an NFS server; or (b) a typical, local Unix file system. They did not succeed. (In fact, their goal was impossible.)

Below, state *precisely* one way in which application code can experience different behavior when interacting with a remote NFS file system versus a local Unix file system. Your answer should

be in terms of what application code sees (rather than in terms of what a global observer sees).
close() can return an error (if the server runs out of space when the client flushes its write cache); operations can hang; reads and writes can suddenly fail because another client on another machine deleted the file; permissions different from in Unix; execute-only implies read, unlike in Unix; etc.

12. [6 points] In this question, two computers, *A* and *B*, are connected by a network link that runs at 1 gigabit per second ($1 \cdot 10^9$ bits/second). The propagation delay is 20 ms; that is, it takes 20 ms for a bit to travel from *A* to *B* or back. Assume that the link does not drop or duplicate packets. Further assume that processing time at the two endpoints is zero. Last, assume that *A* sends *B* 625-byte packets (the 625 includes all headers, framing, and inter-packet spacing).

What is the maximum number of 625-byte packets per second that *A* could in principle send into the wire? Explain your answer *briefly* (for example, by showing your work). You may make small approximations if needed.

10^9 bits/second * 1 byte/8 bits * 1 packet/625 bytes = $200,000$ packets/second.

Now, consider the above link and the following protocol, and assume that all packets are again 625 bytes. In the protocol, *A* sends 4000 packets into the network as quickly as it can and then waits for a one-byte ACK from *B*. Whenever *A* receives an ACK from *B*, *A* immediately sends another 4000 packets into the network. Meanwhile, *B* ACKs packet 1, packet 4001, packet 8001, etc. That is, *B* ACKs every 4000 packets starting with the *first* one in a burst by *A*.

What is the long-term throughput of this protocol, expressed as *both* (a) bits per second and (b) a percentage of the link's bandwidth? Explain your answer *briefly* (for example, by showing your work). You may make small approximations if needed.

The bandwidth-delay product is 10^9 bits/second * 20 ms = 2.5 megabytes = 4000 packets of size 625 bytes. If *A* sent 4000 packets every 20 ms, it would fully use the link. However, from the protocol description, we know that the protocol sends 4000 packets every *40* ms (because the round-trip time is 40 ms, and that's how long it takes for *A* to get *B*'s ACK). Thus, the throughput is one-half the link's bandwidth, or $500 \cdot 10^8$ bits/second.

13. [3 points] Consider the following statement: "If machines were guaranteed not to crash, we would not need two-phase commit: the coordinator could, in one phase, decide whether a distributed transaction would commit, and then instruct the workers to apply their piece of the transaction."

Is the above statement true or false? Justify your answer *briefly* below:

False. The point to the first phase of 2PC is to see whether all of the machines agree to the proposed transaction.

14. [3 points] Consider two generals, A and B, who are encamped with their armies as in the Two Generals Problem. The two generals communicate by messengers that have the following characteristics: a messenger sent by one general always reaches the other general, a messenger never mangles the messages that it is supposed to deliver, and a messenger delivers exactly one copy of the message that it is supposed to deliver. However, a messenger occasionally is delayed for up to 24 hours. Assume that A and B know all of the above but have no way to predict which messengers will be delayed or when. Assume that A decides the time of the attack. Can A and B successfully coordinate an attack, as in the Two Generals problem?

If they can successfully coordinate an attack, give a protocol that does so. If they cannot, then explain why not.

They can successfully coordinate. A sends a message at time X saying, "attack at $X + 25$ hours". B is guaranteed to get the message in time for the attack, and A knows this. Hence both attack at the same time.

IV Guest lectures, readings, and security (23 points total)

15. [3 points] During his lecture, Keith passed around an item of clothing that was decorated with code. This question asks what the item of clothing was and what the code was.

What was the item of clothing?

What code decorated this clothing item? Circle the BEST answer below:

A Code to root the PlayStation

B Code that would allow anyone to hack the magnetic tickets in Boston's subways

C Code to measure Ethernet voltage

D Code that would allow anyone to extract data on disks with quarter-track encoding

E Code to descramble DVDs

Keith passed around a tie with his qrpff program, which descrambles DVDs.

16. [2 points] Jon Howell's guest lecture was about binary rewriting. Which of the following tools did he use during his demonstration?

Circle ALL that apply:

A `gdb`

B `git`

C `readelf`

D `ld` (the linker)

E `strace`

A,C,D,E.

17. [2 points] Recall that Jon's demonstration cheated slightly. Instead of going directly from `hello` to the rewritten binary, called `hello_rewritten`, he had an intermediate step. This step was to create a binary, `hello_cheating_by_linking`, that contained `hello`'s `main()` function together with two functions: `writeizzle()` and `empty_space()`. The purpose of `writeizzle()` was to add "izzle" to the end of the buffers seen by the `write()` system call. This question asks about `empty_space()`: what code did Jon (via his rewriter, `rewrite`) write into `empty_space()`?

State the one-word name of that functionality (using the technical term for the general mechanism) or else describe the functionality.

Jon put the trampoline in `empty_space()`. The trampoline's purpose was to put back the overwritten `mov` instruction and to set up a stack frame for `writeizzle()`.

Name: Solutions **UT EID:**

18. [2 points] This question is about the assigned reading, "Keeping Secrets in Hardware: the Microsoft XBox^{TM} Case Study", by Andrew "bunnie" Huang. Huang successfully attacked the X-box by doing which of the following?

Circle the BEST answer below:

- **A** Using an electron microscope to read the secret key out of the CPU.
- **B** Replacing the DRAM chips with modified DRAM chips that stored a copy of the secret key in an off-chip NVRAM.
- **C** Tapping a high-speed bus between ROM and the CPU to extract a secret key and the code in a secret boot block.
- **D** Mounting a dictionary attack on the Xbox's password file.
- **E** Modifying the on-disk kernel image to cause a buffer overflow attack in the bootloader, overriding the hardware-based protection of the secret key.
- **F** Installing read-write entries in the x86-visible page tables, allowing him to overwrite key kernel data structures.

C. Huang used inexpensive custom hardware to extract the electrical signals on the bus between the southbridge and northbridge.

19. [2 points] One of our assigned readings was "An Access Control Hierarchy for Secure File Logging".

What was the central thesis of this paper? Circle the BEST answer below:

- **A** When developers don't think carefully about their threat model, they can be surprised by attacks that subvert their abstractions.
- **B** The current Unix approach to access control is incoherent.
- **C** The current Unix approach to access control is coherent, but the coarse-grained notion of privilege in Unix creates many vulnerabilities.
- **D** An attacker who gains access to the kernel's logging facility can subvert all of the access control in the file system.
- **E** None of the above.

E.

20. [4 points] These questions concern the hacks to the C compiler that Ken Thompson describes in "Reflections on Trusting Trust".

Circle True or False for each item below:

True / False After Thompson's hacks, the *source code* for the C compiler, if examined, would contain a hint that the login program had been bugged.

True / False After Thompson's hacks, the C compiler *binary*, if disassembled and examined, would contain a hint that the login program had been bugged.

The first is false; the second, true. Thompson's hack was such that no trace of the bugged C compiler or the bugged login was visible from the source code. What allowed Thompson to pull off this hack is that he bugged the binary—and the hint of that would certainly be clear if someone were to disassemble the binary (there would be compiled code that, on matching a pattern in an input file, output a copy of its own logic and some logic to bug the login program).

21. [3 points] Recall that the `ping` program sends ICMP packets using a raw socket and that the `passwd` program changes the user's password by writing to the `/etc/passwd` file. For the purpose of the first two items below, there is no conceptual difference between `ping` and `passwd`: what will be true or false for one will be true or false for the other.

Circle True or False for each item below:

True / False Assuming a normal and bug-free system, one needs to be logged in as the root user to use `ping` and `passwd` successfully.

True / False Assuming a normal and bug-free system, root needs to delegate its privileges (or a subset of those privileges) to ordinary users for them to use `ping` and `passwd` successfully.

The first is false; the second, true. Ordinary users can call `ping` and `passwd`. They must do so using root's privileges because, under Unix's security model, only a process with a real or effective ID of root can open a raw socket or write to the password file.

True / False To delegate its privileges to ordinary users as they run particular binaries, root sets the setuid bit on those binaries.

True.

22. [2 points] This question is about mandatory access control. You may recall that Tanenbaum defined *mandatory access controls* as follows: under mandatory access controls, the system "ensure[s] that the stated security policies are enforced . . . in addition to the standard discretionary access controls [in which individual users determine who may read and write their files]". Tanenbaum goes on to describe a phenomenon that undermines mandatory access controls. He prefaces the description with, "we discuss how information can still leak out even when it has been rigorously proven that such leakage is mathematically impossible".

What phenomenon undermines mandatory access controls? Circle the BEST answer:

- **A** Two-factor authentication
- **B** Weak passwords
- **C** A program that has access to privileged information and, as a result of a bug that is not modeled by the specification, writes it to a world-readable file
- **D** Covert channels
- **E** Buffer overflow attacks on user-level programs

Name: Solutions **UT EID:**

D. With respect to the others, A. isn't right because two-factor authentication doesn't weaken mandatory access control. B, C, and E aren't right because, while they weaken systems generically, they do not undermine mandatory access control. The point of mandatory access control is to enforce security policies independent of how buggy the users of the system are. In particular, the threat model in mandatory access control is usually that a process *is* buggy, and it's the system's job to prevent that process from, for example, leaking privileged information to a less privileged process.

23. [3 points] In his textbook, Tanenbaum says, "We will from time to time look at 'obsolete' concepts", by which he means concepts that are irrelevant under current technology.

Below, *briefly* and *precisely* state Tanenbaum's reason for covering obsolete concepts. Note that the answer is not "so we can learn from our mistakes".

The concepts can become newly relevant as newer technology emerges, particularly at the lower end of the performance scale.

V OS design, labs, and feedback (13 points total)

24. [7 points] This question asks about adding a feature to JOS: signals. As background, POSIX signals in Unix systems are a way for the kernel to notify a process (an environment) of some event of interest. For instance, the kernel delivers the `SIGTERM` signal to a process when a human user types Ctrl+C. There are also user-defined signals, such as `SIGUSR1` and `SIGUSR2`.

A process can register *signal handlers*. For example, if a process registers to handle `SIGUSR1`, the kernel invokes the `SIGUSR1` handler whenever that signal is generated for the process; this generation could happen either of the kernel's volition, or else at the request of another process. If no handler for `SIGUSR1` is registered, the signal has no effect on the environment. (As an aside, if no handler for `SIGTERM` is registered, the kernel kills the process.)

Given a signal, the handler may choose to handle it, to exit gracefully, or simply to do nothing (as an aside, if the signal in question is `SIGTERM`, doing nothing greatly annoys the human user, who would start to slam Ctrl+C with increasing force to no avail). If the signal handler chooses not to exit, then the environment resumes execution where it left off when it received the signal.

Note that a process does not wait to receive a signal (as it does with IPC messages); rather, signal delivery interrupts the environment. If the environment is not currently executing when the signal is generated, the kernel queues up the signal for delivery when the environment runs again.

Given the above information, describe how you would implement simple signal support in JOS. Address the items below *briefly*; you do not need to write a lot as long as you answer the question.

- What data structures would you need to create or modify, and how?
- Design an interface (perhaps a simple system call) to register functions as signal handlers.
- Discuss how control would flow over the life of a signal, from the time the signal is generated in the kernel to the time that the environment resumes regular execution.

Thanks to Ivo Popov for this question.

One approach is to handle signals similarly to the way that you handled user-level page faults in lab 4. A key thing to take away from this exercise is that signals are a software version of interrupts: signals are to processes as real interrupts are to the kernel.

25. [2 points] What is the precise mechanism by which JOS ensures that the kernel runs with interrupts disabled?

State your answer briefly below:

IDT (entry points into kernel) contains only interrupt gates, not trap gates.

26. [3 points] This question asks you to state one way in which JOS is an exokernel. Recall that an exokernel is an operating system design in which the kernel itself implements as *few* abstractions as possible. That is, the kernel abstracts as *little* of the hardware as possible, concerning itself only with the abstractions that it *must* implement for the purposes of allocating the resources of the machine.

There are several ways to answer this question. You could state something that the JOS kernel exposes that a normal kernel would not, or something that the JOS kernel intends to be implemented in user space that a normal kernel would implement itself. (Note that paging to disk, which JOS does not do, is not a good answer because JOS does not intend for there to be any paging.)

Below, state one way in which JOS is an exokernel (you do not need to justify your answer):

27. [1 points] This is to gather useful feedback while we still have your attention. For both of the questions below, any answer will receive full credit. *A blank answer will earn zero points.*

Please list the two topics in this course that you most enjoyed:

There is no right answer.

Please list the two topics in this course that you least enjoyed:

There is no right answer.

End of Final

Congratulations on being almost done with CS372H!

Enjoy your final projects and then the summer!!