

The University of Texas at Austin
CS 439 Principles of Computer Systems: Spring 2013

Midterm Exam I

- This exam is **120 minutes**. Stop writing when “time” is called. *You must turn in your exam; we will not collect it.* Do not get up or pack up between 110 and 120 minutes. The instructor will leave the room 123 minutes after the exam begins and will not accept exams outside the room.
- There are **21** problems in this booklet. Many can be answered quickly. Some may be harder than others, and some earn more points than others. You may want to skim all questions before starting.
- **This exam is closed book and notes. You may not use electronics: phones, calculators, laptops, etc.** You may refer to ONE 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. **Please leave your UT IDs out.**
- 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 that you can answer crisply. Be neat. If we can’t understand your answer, we can’t give you credit!
- There is no credit for leaving questions blank. However, to discourage unfocused responses, we will be grading the clarity of your answer. Moreover, some questions impose sentence limits.
- 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.**
- **Circle your TA’s name and section meeting day below.**

Do not write in the boxes below.

I (xx/27)	II (xx/30)	III (xx/23)	IV (xx/20)	Total (xx/100)

TA (circle one): Sebastian (Tues.) Parth (Wed.) Navid (Thurs.)

Name:

UT EID:

I Processes and so forth (27 points total)

1. [6 points] Recall that a process can be in a number of states: NEW, READY, RUNNING, WAITING, or TERMINATED. When a process is running, it is in the state RUNNING.

Immediately after a process is preempted, which state is it in? You do not need to justify your answer.

Now, assume that a process is in the WAITING state.

Is this process guaranteed to transition to the READY state eventually? Explain either way, but be brief (no more than two sentences).

2. [4 points] **Circle True or False for each item below:**

True / False User-space threads can be preemptively scheduled.

True / False If a priority inversion has happened, then the system has experienced deadlock.

True / False To avoid deadlock, one must avoid all four of its preconditions.

True / False Mike Dahlin's commandments do not mention deadlock.

3. [3 points] A file descriptor is an example of an abstraction that is ...

Circle the BEST answer:

- A ... provided by modern disks
- B ... provided by the standard C library
- C ... provided by the hardware's privileged mode
- D ... provided by Unix
- E ... provided by the linker

Name:

UT EID:

4. [3 points] Recall that when the shell creates a new process, it first calls `fork()`, to create a copy of itself, and then, in the child, the shell calls `exec()`. This call to `exec()` replaces what state in the process?

Circle ALL that apply:

- A the file descriptor state
- B process memory
- C device drivers
- D the kernel
- E the PID

5. [5 points] Many operating systems give the appearance of executing many applications at once, even on single-CPU machines. Giving this appearance requires the operating system to perform many *context switches*.

What is a context switch? (Answer briefly.) What steps does the operating system perform to execute a context switch? (Write the steps in a bulleted list of four or five brief bullets.)

6. [3 points] Which of the following components is responsible for loading the initial value in the program counter (also known as the instruction pointer) for a user-level process before it starts running?

- A The boot loader (aka boot ROM)
- B The semaphore `up()` operation
- C The loader
- D The linker
- E The compiler

7. [3 points] Consider the following command, executed at the command prompt (also known as the shell) on the UTCS Linux machines:

```
$ man 2 write
```

What does the above command do? (Do not use more than one sentence for your answer.)

Name:

UT EID:

II Concurrency and synchronization (30 points total)

8. [4 points] A thread within a process has its own:

Circle ALL that apply:

- A stack
- B `main()` function
- C registers
- D global variables
- E program code
- F heap

9. [3 points] Assume a four-CPU machine. Assume that the system provides a user-level threading package and no kernel-level threading.

How many CPUs can a single process use, if the process has four user-level threads? You do not need to justify your answer.

10. [6 points] Recall that the UTCS department requires that you follow the six commandments of thread programming.

Why does UTCS impose these commandments? Do not write more than one or two sentences.

Name three of the commandments.

Name:

UT EID:

11. [4 points] Which of the following statements about semaphores is true?

Circle ALL that apply:

- A Semaphores were a key cause of the Therac-25's errors.
- B Semaphores are a program construct that we told you not to use in application code.
- C Semaphores are the chief vehicle for explaining deadlock in the Bryant & O'Halloran book
- D Semaphores are a way of abstracting network sockets.
- E Semaphores are a way of abstracting OS signals.
- F Semaphores atomically increment and decrement a count.

12. [4 points] Let `cv` be a condition variable, and let `mutex` be a mutex. Consider the following pattern:

```
if (!predicate()) {  
    wait(&mutex, &cv);  
}  
// it is an error if code gets to here with the predicate false
```

When is the above pattern correct? Do not write more than one or two sentences.

13. [3 points] You decide that you want to implement a concurrency primitive, and you want to do it correctly.

You will need to consult the manuals for which system components?

Name:

UT EID:

14. [6 points] Consider the following implementation of a spinlock:

```

struct Lock {
    int locked;
}

int exchange_value(int* ptr, int val) {
    int was;
    was = *ptr;
    *ptr = val;
    return was;
}

void acquire (Lock *lock) {
    pushcli();    /* this disable interrupts */
    while (1) {
        if (exchange_value(&lock->locked, 1) == 0)
            break;
    }
}

void release(Lock *lock){
    exchange_value(&lock->locked, 0);
    popcli();    /* this restores interrupts to the state they were in */
}

```

This implementation differs from the one that we saw in class. Specifically, the one that we saw in class implemented `exchange_value()` using an assembly instruction, `xchg`, that performs the exchange atomically.

This question asks whether the code above is correct. *Assume that the machine provides sequential consistency.* If the lines `pushcli()` and `popcli()` are confusing or distracting, then you can ignore them; they are there for completeness.

Is the code correct? If the code is correct, explain what invariant is maintained. If the code is not correct, give a problematic interleaving and explain why it is problematic.

Name:

UT EID:

III Scheduling, readings, and lab (23 points total)

15. [4 points] These questions concern various scheduling disciplines.

Circle True or False for each item below.

True / False We showed in class how an implemented scheduler can achieve the optimal average waiting time (also known as response time).

True / False FCFS (first come first served) optimizes throughput.

True / False FCFS (first come first served) optimizes CPU utilization.

True / False Round-robin scheduling ensures no starvation.

16. [3 points] In describing scheduling, Hailperin gives several examples of scheduling decisions that you, the reader, might need to make in real life.

Below, state one of these real-world examples.

17. [4 points] Describe the *symptoms* of one of the radiation injuries inflicted by the Therac-25 (that is, we are asking for more than “the patient ultimately died” or “massive radiation burns”). If you have not read the paper, please skip the question; we do not want to read creative writing here. If you have read the paper, feel free to supply graphic details from the paper.

Describe one of the injuries. Be brief; we will read only the first two sentences. Skip the question to receive half of a point.

Name:

UT EID:

18. [4 points] This question concerns your pair programming experience.

Write your partner's name:

Describe the worst bug encountered by your team in Lab 2.

19. [5 points] This question is about the boot process in JOS. In lab 3, you were asked to use gdb to step through the process of booting. (Here, the word “process” has its non-technical meaning.) You would have stepped through `boot/boot.S`, then into `bootmain()` (in `boot/main.c`), and then `readsect()` (also in `boot/main.c`). In particular, the lab says, “Trace through the rest of `readsect()` and back out into `bootmain()`, and identify the begin and end of the for loop that reads the remaining sectors of the kernel from the disk.”

How does the boot loader decide how many sectors it must read in order to fetch the entire kernel from disk? Where does it find this information?

20. [3 points] This is to gather feedback. Any answer, except a blank one, will get full credit.

Please state the topic or topics in this class that have been least clear to you.

Please state the topic or topics in this class that have been most clear to you.

Name:

UT EID:

IV Multithreaded programming (20 points total)

21. [20 points] Your three-year old sibling, Kelly, has recently become obsessed with finding pennies. Kelly looks for them everywhere—on the sidewalk, in the parking lot, under the table at the restaurant. These pennies are very dirty! As the loving older sibling, you decide that you will clean them for Kelly using ordinary household liquids. Unfortunately, you don't know which household liquids will be the most effective. Your plan, then, is to use five different household liquids to clean five pennies and see which pennies are the cleanest. Your sibling will (obviously) provide the pennies, and, since you are at your parents' house, a parent will provide the liquids. Once the pennies are finished soaking (you learn they are finished by receiving a signal from an omniscient being), you will remove them and analyze the results. When you have finished your experiment—*which involves five pennies and five liquids*—return the pennies to the penny pile and the liquids to your parent. Kelly is very busy and not waiting for the pennies (so no need to notify Kelly!), but your parents, who are anxious to clean the kitchen, are waiting for you to finish.

In thinking about your plan, you realize that this is really a case of synchronization (you need resources!) and mutual exclusion (no putting away the liquids while you are still using them!), so you decide to model you, your family members, and the omniscient being as threads. Each of you is a thread. However, note that there can be multiple threads of the same type: multiple parents and multiple little siblings (each of whom acts like Kelly).

Each thread executes one of the functions below:

```
void big_sib();           // the function executed by the big sibling thread
void little_sibling()    // the function executed by a little sibling thread
void parent();          // the function executed by a parent thread
void omniscient_being(); // the function executed by the omniscient being
```

and each of *those* functions invokes a corresponding method on a monitor, called Expt, defined on the next page. (You can imagine that each of the functions above invokes the corresponding monitor method once.)

Implement the Expt monitor: write down the state variables, the synchronization objects, and implement the four methods. Be certain to follow the 439 coding conventions. You may indicate activities other than synchronization and mutual exclusion using angle brackets (for example, <find penny> and <clean pennies>).

Before you get started, you may wish to follow the design approach outlined in class: write down the synchronization constraints (mutual exclusion, scheduling, etc), write down the shared state, etc. However, we will be grading only the implementation of Expt.

Name:

UT EID:

```
class Expt {

    public:

        // Acquires (and releases!) the appropriate resources and conducts
        // the experiment. This method should also notify the omniscient
        // being when the experiment has begun.
        void BigSibStuff();

        // Finds a penny and signals the big sibling
        void LittleSibStuff();

        // Gathers five liquids, notifies the big sibling that the liquids
        // are ready, and then waits until the sibling is ready for the
        // liquids to be put away.
        void ParentStuff();

        // This method rests until hearing that the experiments have begun.
        // It then watches the experiment and notifies the scientist (the
        // big sibling) when the results are ready
        void OmnsiccientBeingStuff();

    private:

        // FILL THIS IN

};

// Here and on the next page, give the implementations of
// Expt::Expt() [this should initialize all of the shared state],
// Expt::BigSibStuff(),
// Expt::LittleSibStuff(),
// Expt::ParentStuff(),
// Expt::OmnsiccientBeingStuff
```

Name:

UT EID:

Space for code and/or scratch paper

Name:

UT EID:

Space for code and/or scratch paper

End of Midterm

Name:

UT EID: