Q1 Directions/assurance

0 Points

1. Please read the entire exam before starting to write. This should help you avoid getting bogged down on one problem.
2. We believe this is approximately two "exam hours" of content. However, you will have four hours to work on the exam, starting from when you begin it. The extra time is intended to correct for potential logistical issues and also to reduce stress during a time when we all likely have more than we need.
3. The exam will be open-book/open-notes in the following sense:
   A. You may use any edition of either textbook.
   B. You may refer to materials we provided you with, including the Intel PDFs on the course web site, the lecture slides we provided you with, the project handouts (the kernel specification and the thread-library handout may be particularly useful), and the lecture videos from this semester---in other words, materials on this semester's course web site.
   C. You may use notes you have taken.
   D. You may refer to your P0, P1, and P2 submissions.
4. But you are **not** authorized to get help from other people, or to use online resources that are not part of this semester's course web site.
5. You are **not** authorized to use compilers, assemblers, linkers, loaders, simulation engines, virtualization systems, proof checkers, etc.
6. The exam will not be proctored, i.e., you will be operating on the honor system in terms of resources we have said you can consult.
7. If you have a question while taking the exam, please check the published Zoom schedule. You may also send mail to the staff mailing list.
8. The weight of each question is indicated on the exam. Weights of question **parts** are estimates which may be revised during the grading process and are for your guidance only.
9. Please be concise in your answers. You will receive partial credit for partially correct answers, but truly extraneous remarks may count

against your grade.

I certify that my exam submission is my own work,
in compliance with the rules stated above.

Name:

Name

Date:

Date

Q2 Design

6 Points

When designing a body of code, at times one finds oneself thinking, "I wonder if I should use Approach A or Approach B?" According to the 15-410 design orthodoxy, you should follow a specific process to resolve your question.

Please document a design decision you made while working on your Project 2 thread library. For the purposes of this question we will be scoring based on your **description** of the decision, not whether we agree or disagree with what you chose.

Begin with a brief description of the problem (one to three sentences) and then show us your design decision---answers that correctly use the 15-410 approved data structure will receive higher scores.

Brief description:

Design decision (we will accept answers in reasonable form, text or image, e.g., PDF, PNG; please avoid color combinations that are difficult

to read, such as purple text on a black background).

No files uploaded

Q3 Register dump

4 Points

Below is a register dump produced by the "Pathos" P2 reference kernel when it decided to kill a user-space thread. Your job is to carefully consider the register dump and:

1. Determine which "wrong register value(s)" caused the thread to run an instruction which resulted in a fatal exception. You should say why/how the wrong value led to an exception, i.e., merely claiming a register has a "wrong" value will not receive full credit.
2. Briefly state the most plausible way you think that register could have taken on that value (i.e., try to describe a bug which could have this effect).
3. Then write a **small** piece of code which would plausibly cause the thread to die in the fashion indicated by the register dump. **This code does not need to implement exactly the set of steps that you identified as "most plausible" above, or result in the same register values; you should aim to achieve "basically the same effect."** Most answers will probably be in assembly language, but C is acceptable as well. Your code should assume execution begins in `main()`, which has been passed the typical two parameters in the typical fashion.

Please be sure that your description of the fatality and the code, taken together, clearly support your diagnosis.

```
Registers:
eax: 0x00000001, ebx: 0x000f2020, ecx: 0x0000004c,
edx: 0x00800000, edi: 0x01002004, esi: 0x00008086,
ebp: 0xffffffff, esp: 0xfffffffe, eip: 0x01000023,
 ss:     0x002b,  cs:     0x0023,  ds:     0x002b,
 es:     0x002b,  fs:     0x002b,  gs:     0x002b,
eflags: 0x00000282
```

Wrong register value(s) and exception/fault reason:

How the register(s) got the wrong value(s):

Code: if possible, please upload a text file containing your code, ideally with spaces rather than tabs. But we will also accept PDF or PNG -- if possible, please provide us with black text on a white background, and please avoid white text on a black background or stranger things (purple text on a black background, etc.).

No files uploaded

Q4 Semaphore problem

20 Points

Having heard that one of the 410 course staff very much enjoys semaphores, your partner has implemented a "small extension" to the standard semaphore design which increases the similarity between semaphores and condition variables. In particular, your partner's code tracks how many threads are waiting on a semaphore and provides a primitive, `sem_broadcast()`, designed to increment the semaphore's count by the number of waiting threads, and to wake up those threads.

```
typedef struct {
    int count;
    int waiters;
    mutex_t count_lock;
    mutex_t waiters_lock;
    cond_t cv;
} sem_t;

/* code omitted for exam purposes */
int sem_init(sem_t *sem, int count);
int sem_destroy(sem_t *sem);
```

```
/* code omitted for exam purposes */

int sem_wait(sem_t *sem) {
    mutex_lock(&sem->count_lock);
    if (sem->count > 0) {
        sem->count--;
        mutex_unlock(&sem->count_lock);
        return 0;
    } else {
        mutex_lock(&sem->waiters_lock);
        sem->waiters++;
        mutex_unlock(&sem->waiters_lock);
    }
    cond_wait(&sem->cv, &sem->count_lock);
    sem->count--;
    mutex_unlock(&sem->count_lock);
    mutex_lock(&sem->waiters_lock);
    sem->waiters--;
    mutex_unlock(&sem->waiters_lock);
    return 0;
}

int sem_signal(sem_t *sem) {
    mutex_lock(&sem->count_lock);
    sem->count++;
    cond_signal(&sem->cv);
    mutex_unlock(&sem->count_lock);
    return 0;
}

int sem_broadcast(sem_t *sem) {
    int i, w;
    // lock: we want only *existing* waiters
    mutex_lock(&sem->waiters_lock);
    w = sem->waiters;
    for (i = 0; i < w; i++)
        sem_signal(sem);
    mutex_unlock(&sem->waiters_lock);
    return 0;
}
```

There are (at least) two synchronization problems found in the code presented above. To receive full credit, identify two **different** problems (rather than two instances of the same mistake). Do not present more than two. The problems are found in the code presented to you (in other words, we are not looking for claims that `sem_init()` might be implemented incorrectly). You should assume that invocations of thread-

library primitives (e.g., `mutex_lock()`) succeed rather than detecting inconsistency or otherwise failing.

If you have correctly identified a sychronization problem, you will be able to **briefly and clearly** summarize it and show a **clear and compelling** execution trace using the tabular execution format from the lectures and homework assignment. Confusing descriptions and unclear execution traces will be read as evidence of incomplete understanding and will be graded as such. This means that it is to your advantage to **think your answers through before beginning to write**.

Q4.1 First problem

12 Points

Briefly and clearly describe the first synchronization problem you have identified with the code above.

Show a **clear and compelling** execution trace. We will accept a picture (PDF, PNG), or a text file containing one line per event (each event should consist of a thread i.d. and an action, e.g., `T0: x=3`).

No files uploaded

Q4.2 Second problem

8 Points

Briefly and clearly describe a second synchronization problem with the code above, which should be "a different kind of problem."

Show a **clear and compelling** execution trace. We will accept a picture (PDF, PNG), or a text file containing one line per event (each event should

consist of a thread i.d. and an action, e.g., `T0: x=3`).

No files uploaded

Q5 Condition variables

20 Points

The "condition variable" is an important concurrency primitive that encapsulates the notion of "stop running, so that other threads may use the processor, until the world changes in a way which probably enables me to continue working." A necessary part of implementing condition variables is solving the "atomic unlock-and-deschedule" problem: a waiting thread must release a lock and ask the kernel to block it; if another thread is trying to awaken the waiting thread, the "awaken" operation must not be lost just because its execution is interleaved in a troublesome way with the "release, then block" sequence.

The Pebbles kernel specification provides user code with two system calls which can be combined to solve the atomic-blocking problem, `deschedule()` and `make_runnable()`. Other systems provide other primitives, some of which are quite different. Linux provides "futexes" and signals; Plan 9 provides a primitive called "rendezvous."

`void *rendezvous(void *tag, void *value)}` -
Synchronize, and exchange values between, two threads in the same task. Two threads wishing to synchronize invoke `rendezvous()` specifying the same *tag* parameter and arbitrary *value* parameters. The first thread specifying a particular `tag` value will suspend execution until a second thread invokes `rendezvous()` with the same `tag` parameter. The two threads that rendezvous each obtain the `value` parameter specified by the other one. After the exchange, both threads are runnable. The return value of the system call is the `value` parameter specified by the other thread. The kernel places no interpretation on the values of the `tag` and `value` parameters except when it performs equality testing on the `tag` parameters.

Assumptions:

1. To implement condition variables you may use mutexes and the `rendezvous()` system call described above.
2. You may **not** use other atomic or thread-synchronization synchronization operations, such as, but not limited to: semaphores, reader/writer locks, `deschedule()`/`make_runnable()`, or any atomic instructions (`XCHG`, `LL/SC`).
3. You may assume that callers of your routines will obey the rules. **But you must be careful that you obey the rules as well!**
4. You must comply with the published interfaces of synchronization primitives, i.e., you cannot inspect or modify the internals of any thread-library data objects.
5. You may not use assembly code, inline or otherwise.
6. **For the purposes of the exam, you may assume that library routines and system calls don't "fail"** (unless you indicate in your comments that you have arranged, and are expecting, a particular failure).
7. You may **not** rely on any data-structure libraries such as splay trees, red-black trees, queues, stacks, or skip lists, lock-free or otherwise, that you do not implement as part of your solution.
8. You may use non-synchronization-related thread-library routines in the "`thr_xxx()` family," e.g., `thr_getid()` (note that the P2 handout documents the thread-library interface). If you wish, you may assume that `thr_getid()` is "very efficient" (for example, it invokes no system calls).

Please upload a single text file containing your declaration for a `struct cond` and your code for `cond_init()`, `cond_wait()`, and `cond_signal()` (you do not need to implement either `cond_broadcast()` or `cond_destroy()`). If you wish, you may also declare an auxiliary structure, `struct aux`, **but this is strictly optional**. Ideally the text file will use spaces rather than tabs. But we will also accept PDF or PNG -- if possible, please provide us with black text on a white background, and please avoid white text on a black background or stranger things (purple text on a black background, etc.).

```
typedef struct cond {
```

```
} cond_t;

typedef struct aux {



} aux_t; /* optional */
```

Code file:

No files uploaded