The University of Texas at Austin
CS 372H Introduction to Operating Systems: Honors: Spring 2010

FINAL EXAM

- This exam is **3 hours**. Stop writing when "time" is called. *You must turn in your exam; we will not collect them.* 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 **28** questions 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, PDAs, calculators, etc.** You may ref ... Times New Roman font, 1 inch or larger r ...
- If you find a que ... s you make.
- Follow the instru ... reasoning and any important assumpti ... **ps will not work and will waste time.** ... . Be neat. If we can't understand your ...
- To discourage gu ... 25%-33% of the credit for any problem ... t at zero points for the problem. Note t ... oint total is listed. *Subproblems* with n ... you attempt any subproblem, you ma ...
- The exception is ... True/False item: correct items earn p ... s earn negative points. However, the mi ... alse items—is 0. Don't overthink all of t ... e answer, then answer the problem. If y ...
- 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/12)	II (xx/13)	III (xx/15)	IV (xx/19)	V (xx/16)	VI (xx/17)	VII (xx/8)	Total (xx/100)

Name: Solutions **UT EID:**

I Interrupts, concurrency, scheduling (12 points total)

1. [2 points] What are the three ways that a CPU executing in user mode (that is, executing instructions from a user-level process) can switch to supervisor mode (and presumably execute kernel instructions)? (Hint: we have discussed this question in class, using the phrase, "What are the three ways that the kernel gets invoked?")

A. Interrupts from a device.

B. Traps, often used for invoking system calls. Confusingly, on the x86, the `int` instruction generates traps.

C. Exceptions, such as divide by zero, illegal memory access, or execution of an illegal opcode.

2. [2 points] P [illegible] ng an operating system.
Pat is told by th [illegible] o be correct only on a
machine with a si [illegible] is unacceptable to write
a kernel in which [illegible] ly dislikes the approach
that JOS takes). C [illegible] *ll* the time. Pat's kernel
has data shared b [illegible] t runs when an external
I/O event happen [illegible] protect this shared data
are needed becau [illegible]

State whether P [illegible] **elow:**

Pat's reasoning is [illegible] rrupt handler might run
and invalidate an [illegible] s fixing up a linked list,
and the interrupt [illegible] his, Pat needs to protect
data that is shared [illegible] so, Pat probably needs
to disable interrup [illegible] difies. Doing so ensures
the invariant that [illegible] kernel code is not (and
vice-versa).

3. [8 points] In this problem, you will implement a monitor to help a set of *drivers* (modeled as threads) synchronize access to a set of *five keys* and a set of *ten cars*. Here is the problem setup:

- The relationship between the keys and the cars is that key 0 operates cars 0 and 1, key 1 operates cars 2 and 3, etc. That is, key i works for cars $2i$ and $2i + 1$.
- If a key is being used to operate one car, it cannot be used to operate the other.
- A driver requests a particular car (which implies that the driver needs a particular key). However, there may be many more drivers than cars. If a driver wants to go driving but cannot get its desired car or that car's key, it waits until the car and key become available. When a driver finishes driving, it returns its key and notifies any drivers waiting for that key that it is now free.

- You must allow multiple drivers to be out driving at once, and you must not have busy waiting or spin loops.
- We repeat: there could be many, many instances of `driver()` running, each of which you can assume is in its own thread, and all of which use the same monitor, `mon`.

On the next page, fill in the monitor's remaining variable(s) and implement the monitor's `take_key()` and `return_key()` methods. *Follow the coding standards given in class.*

```
typedef enum {FREE, IN_USE} key_status;

class Monitor {
    public:
        Monitor() { memset(&keys, FREE, sizeof(key_status)*5); }
        ~Monitor() {}
        void take_key(int desired_car);
        void return_key(int desired_car);
    private:
        Mutex mutex;
        key_status keys[5];
        /* YOU MUST ADD MATERIAL BELOW THIS LINE */



};

void driver(thr
     /* you sho
     mon->take_
     drive();
     mon->retur
}

void Monitor::t
     /* YOU MUS                                                 efers
        to the









}

void Monitor::return_key(int desired_car) {
     /* YOU MUST FILL IN THIS FUNCTION. Note that the argument refers
        to the desired car. */









}
```

Name: Solutions   UT EID:

```
/* easiest way to extend monitor is with just a single condition
variable (could also have one condition variable for each key) */

class Monitor {

   .....

   Cond cond;
};

void Monitor::take_key(int desired_car)
{
    int which

    acquire(&m

    while (ke
        wait(
    }

    keys[whic
    release(&m
}


void Monitor:
{
    int which
    acquire(&m

    keys[whic
    cond_broadcast(&mutex, &cond);
    release(&mutex);
}
```

II Virtual memory and paging (13 points total)

4. [2 points] Circle True or False:

True / False A virtual memory system that uses paging is vulnerable to external fragmentation.

False. Paging provides fine-grained allocation of memory, so the only fragmentation that can happen is wastage within a page, which is internal fragmentation.

5. [9 points] Consider a 32-bit computer with the following funky virtual memory architecture:

- Each page is 2KB (2^{11} bytes).
- Physical me [illegible] ot mean 2^{32} bytes.
- Associated [illegible] ng control and reserved bits. The ex [illegible] completeness, they are: three bits co [illegible] ntrolled by the hardware (Accessed a [illegible] TE_COW).
- The machin [illegible] ed in JOS: each process has a page d [illegible] try in the page directory has the sam [illegible]

Below, state the [illegible] **ry, and explain *briefly*:**

31 bits. There ar [illegible] ere are 2^{24} pages in the machine. That m [illegible] bits (to be able to select the physical page [illegible] eed at least $24 + 7 = 31$ bits per entry.

Below, state the [illegible] **xplain *briefly*:**

31 bits, again. Th [illegible] ble to "resolve" 2^{24} bits since a page table [illegible] e given that the number of control and res [illegible]

Now assume that the entry size is rounded up to the nearest multiple of 4 bytes. Further assume that a page directory or page table must fit on a page. **Programs on this machine use 32-bit quantities as instruction operands, but when the operand is an address, not all of these 32 bits are examined by the processor. How many address bits are actually used in this architecture?**

29 bits. A page table entry is 4 bytes (per the padding), and a page table is 2KB (because it needs to fit on a page). Thus, a page table can have no more than 512 (2^9) entries, which means that no more than 9 bits are used to index into the page table. Same for the page directory. Finally, since a page is 2KB, 11 bits of the address determine the offset into the page. Altogether, this is a maximum of $9 + 9 + 11 = 29$ bits of address in use.

How large is the *per-process* virtual memory space?

512 MB only. That is because a process can use only 29 bits of addresses, and $2^{29} = 512$ MB. Note that on modern 32-bit x86s, something similar happens: each process gets 2^{32} bytes of virtual address space, but the machine can actually have more physical memory than that.