The Midterm Exam

Administrative

1. Midterm exam scores are on the course web. If you wish, stop by my office to pick up your exam.

2. You may contest the grading, but if you do, I reserve the option to reassess your score for any part of the exam.

3. You have the option of taking a concurrency hour exam to supplement/qualify your midterm grade.
   - three familiar synchronization problems, my choice
   - I pick two of the problems at random to grade
   - your score is averaged with your midterm score
   - if you choose this option, there is no going back

4. Midterm questions may come back to haunt you later!
Midterm Scores

mean == median == 117 out of 200
But I had a pretty decent vacation anyway.

Problem 1a: Ring Cycle with Semaphores

```c
void TakeTurns(int id) {
    while(1) {
        sem->P();
        printf("%d...", id);
        sem->V();
    }
}
```

Variations of this solution were worth less than half credit (5 or 6 out of 12).

Ignoring the bootstrap problem, do not assume that semaphores are FIFO (e.g., Nachos semaphores are not!).

```c
void TakeTurns(int id) {
    while(1) {
        sem[id]->P();
        printf("%d...", id);
        sem[(id+1) % N]->V();
    }
}
```

If you want to use a semaphore to signal a specific thread, create a semaphore that is private to that thread.

What about the bootstrapping problem?
Example: Ping-Pong

```c
void PingPong() {
    while(1) {
        mx->Acquire();
        cv->Signal();
        cv->Wait();
        mx->Release();
    }
}
```

Will this work using Birrell’s CV semantics?

How would you do this using semaphores?

Problem 1b: Ring Cycle with One CV

```c
void TakeTurns(int id) {
    mx->Acquire();
    while(1) {
        while (turn != id)
            cv->Wait();
        printf("%d...", id);
        turn = (turn+1) % N;
        cv->Signal();
    }
    mx->Release();
}
```

Variations of this solution were worth less than half credit (5 or 6 out of 12).

Do not assume that condition variables are FIFO!

What if `Signal` wakes the wrong thread?
Problem 1b-c: Ring Cycle with One CV

```c
void TakeTurns(int id)
{
    mx->Acquire();
    while(1) {
        while (turn != id)
            cv->Wait();
        printf("%d...", id);
        turn = (turn+1) % N;
        cv->Broadcast();
    }
    mx->Release();
}
```

How many context switches per turn?

Generally: N.

On a multiprocessor? N/P per processor perhaps, but still N total....

...or could be more due to lock contention (worth two points).

If you used signal instead of broadcast for (b), your bogus answer to part (c) was worth about 75%.

Problem 1d: Ring Cycle with Multiple CVs

```c
void TakeTurns(int id)
{
    mx->Acquire();
    while(1) {
        cv[id]->Wait();
        printf("%d...", id);
        cv[(id+1) % N]->Signal();
    }
    mx->Release();
}
```

How many context switches per turn?

Generally: one.

On a multiprocessor? 1 or 3

Note that is legal and required to use these N condition variables with a single mutex.
Problem 2: Impatient Wait

```c
void Condition::Wait(int timeout) {
    s = new Semaphore();
    w = new WaitRecord(s, timeout);
    list->Append(w);
    mx->Release();
    thread Fork(Waker, w);
    s->P();
    delete s; delete w;
    mx->Acquire();
}
void Condition::Waker(WaitRecord* w) {
    alarm->Pause(w->timeout);
    mx->Acquire();
    if (list->FindAndRemove(w))
        w->s->V();
    mx->Release();
}
```

*Wait* may also be implemented using *Sleep* directly, by disabling interrupts and doing the timed wakeup from the timer interrupt handler.

Details aren’t critical: hocus-pocus and handwaving cost ten points at most.

Grossly incorrect CV implementations earned 20/50 max.

Other alternatives:

Pause the waiting thread instead of blocking: up to 40 points.

Short pause and loop each tick: up to 45.

Problem 4a: Event with Semaphores

```c
void Event::Wait() {
    if (!signalled)
        sem->P();
}
void Event::Signal() {
    signalled = TRUE;
    sem->V();
}
```

What happens if multiple threads call *Wait()* before the *Signal()*?

This solution is worth 10 or 11 out of 15.
Problem 4a: Second Try

```c
void Event::Wait()
{
    if (!signalled)
    {
        waiters++;
        sem->P();
    }
}

void Event::Signal()
{
    signalled = TRUE;
    while(waiters)
    {
        waiters--;
        sem->V();
    }
}
```

**Problem:** how to do a Broadcast on a semaphore?

**Answer:** not this way. This solution is worth a generous half-credit.

*If you need to count, always let the semaphore do the counting for you!*

**Alternatives:**
- use another semaphore for locking
  ...but get it right
- use hocus-pocus
  e.g., a semaphore primitive to read the semaphore count

---

Problem 4a: Third Try

```c
void Event::Wait()
{
    sem->P();
    sem->V();
}

void Event::Signal()
{
    sem->V();
}

void Event::Reset()
{
    sem->P();
}
```

YES
Problem 4b: Event with CV

```c
void Event::Wait()
{
    mx->Acquire();
    if (!signalled)
        cv->Wait();
    mx->Release();
}

void Event::Signal()
{
    mx->Acquire();
    signalled = TRUE;
    cv->Broadcast();
    mx->Release();
}
```

Problem #3: Creating and Running a Process

(a) Initialize memory segments from executable file sections:
   - read text section into text segment
   - read (initialized) data section into data segment
   - zero the (uninitialized) BSS data and stack segments

(b) Initialize process user registers:
   - PC is where the program will start executing.
     start virtual address of text section (from exec file header)
   - SP is the base (highest) virtual address of the user stack.
     ...wherever the kernel decides to put it...
     ...but it better be aligned properly!
   - it’s good practice to zero the other registers
Problem #3: Mode and Context

(c) What could cause the machine to re-enter kernel mode?

- trap (syscall), fault, or interrupt: 7 points
- for faults and traps, set SP to a per-process kernel stack
- for interrupts, typically set SP to a kernel interrupt stack
- set PC to kernel handler for trap, fault, or interrupt (3 points)
- save old SP and PC on the kernel stack

(b) What could cause the machine to re-enter user mode?

- return from trap, fault, or interrupt
- restore saved PC and SP (perhaps increment PC)
- other examples: voluntary or involuntary context switch (block, fork, exit, or quantum expire), signal or APC upcall.