Preemption

- Set timer interrupts
  - Switch threads on interrupt
  - Same idea as other switches
- But now, must synchronize
- Two tools:
  - Enable/disable interrupts
  - test_and_set(), clear()

Preemption (cont'd)

- Enable/disable interrupts to synchronize thread code
  - `int old = splx(HIGH) // disable`
    `splx(old) // re-enable`
- Use test_and_set for other stuff
  - Why not disable interrupts?

Multi-thread web server (1)

- web/sioux.c – singlethreaded web server
  - Read in command line args, run the web server loop
- web/sioux_run.c – the webserver loop
  - Open a socket to listen for connections (`listen`)
  - Wait for a connection (`accept`)
  - Handle it
    - Parse the HTTP request
    - Find and read the requested file (www root is ./docs)
    - Send the file back
    - Close the connection
- web/web_queue.c – an empty file for your use

Multi-thread web server (2)

- Make the web server multithreaded
  - Create a thread pool
    - A bunch of threads waiting for work
    - Number of threads = command-line arg
  - Wait for a connection
  - Find an available thread to handle connection
    - Current request waits if all threads busy
  - Once a thread grabs onto connection, it uses the same processing code as before

Multi-thread web server (3)

- Each connection is identified by a socket returned by `accept`
  - Which is just an int
  - Simple management of connections among threads
- Threads should sleep while waiting for a new connection
  - Condition variables are perfect for this
- Don't forget to protect any global variables
  - Use part 2 mutexes, CVs
- Develop + test with pthreads initially

- Mostly modify sioux_run.c and/or your own files
- Stick to the sthread.h interface!

Semaphores (1)

```
wait(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add to S->list;
        block()
    }
}
signal(semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove P from S->list;
        wakeup(P);
    }
}
```

- Where are critical sections?

Semaphores (2)

```
wait(semaphore *S) {
    while (TestAndSet(S->guard)) ;
    S->value--;
    if (S->value < 0) {
        add to S->list;
        block()
    }
    S->guard = false
}
```

- What's wrong with this?

Semaphores (3)

```
wait(semaphore *S) {
    while (TestAndSet(S->guard)) ;
    S->value--;
    if (S->value < 0) {
        add to S->list;
        S->guard = false
        block()
    }
}
```

- OK?

Semaphores (4)

```
wait(semaphore *S) {
    while (TestAndSet(S->guard)) ;
    S->value--;
    if (S->value < 0) {
        add to S->list;
        S->guard = false
        block()
    }
    S->guard = false;
}
```

- No, really, OK?

Semaphores (5)

```
wait(semaphore *S) {
    while (TestAndSet(S->guard)) ;
    S->value--;
    if (S->value < 0) {
        add to S->list;
        S->guard = false
        block()
    } else {
        S->guard = false;
    }
}
```

Alarm clock (1)

```
monitor alarm {
    condition alarm;
    wakeme(int num_ticks) {
        ...
        alarm.wait();
    }
    tick() {
        ...
        alarm.signal();
    }
}
```

Alarm clock (2)

```
monitor alarm {
    condition alarm;
    wakeme(int num_ticks) {
        ...
        alarm.wait();
    }
    tick() {
        ...
        alarm.broadcast();
    }
}
```

Alarm clock (3)

```
monitor alarm {
    sorted_list list;
    int time = 0;

    wake_me(int num_ticks) {
        c = new condition;
        list->insert (
            time + num_ticks, c);
        c->wait();
    }
```

Alarm clock(4)

```
monitor alarm {
    ...
    tick() {
        time++;
        for (head()->time == time)
            c->signal();
    }
```