Assignment 2
Non-preemptive scheduling

COS 318 - Operating System
Fall 2006

Main things you need to deal with

- Process Control Block (PCB)
- Context Switch procedure
- System Call mechanism
- Stacks
- Synchronization
- Inline assembly
- Design review requirements

Process Control Block (PCB)

- File: kernel.h
- What should be in the PCB?
  - pid, in_kernel, stack?
  - next, previous
- What else should goes in PCB?
  - Design review.

Context Switch Procedure

- How to switch between processes and threads?
  - They must call yield() explicitly.
  - Time slice expires. (Preemptive, next assignment)
- Where to save it?
  - Stack?
  - PCB.
How does a process get services from the kernel?
- This assignment: special function call - a "jump table"
- Real stuff: Interrupt/trap mechanism (later assignments)

At runtime, load the address of kernel_entry() into memory location 0xf00.

How to do that?
- Define this
  ```c
  #define ENTRY_POINT (void(**)(int))0xf00
  ```
- Declare the following in syslib.c
  ```c
  void (**entry_point)(int)=ENTRY_POINT
  ```
- To load
  ```c
  *entry_point = kernel_entry
  ```

The following diagram shows the kernel_entry in the memory:

```
0xf00
0x1000
```

How many stacks?
- 2 per process, 1 per thread. Why?

Where to put them in memory?
- Upper limit: 640K (= 0xa0000)
- Suggestion: between 0x10000 and 0x20000
- See memory layout on the next slide

Size of each stack:
- 4KB should be fine.
Memory Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01000</td>
<td>BIOS</td>
</tr>
<tr>
<td>0x07C00</td>
<td>Bootblock</td>
</tr>
<tr>
<td>0xA0000</td>
<td>Stacks</td>
</tr>
<tr>
<td>0xB8000</td>
<td>Video RAM</td>
</tr>
</tbody>
</table>

Synchronization

- Locks are used by threads
- Many threads can try to acquire a lock
  - Need to maintain queue of threads waiting for a lock. (where?)
- Lock_acquire()
  - 1. Check lock
  - 2. Get lock? Great!
  - 3. If not, block itself
- Lock_init(), Lock_release()

Inline Assembly

- See guide on course page
  - http://linuxassembly.org/articles/rmiyagi-inline-asm.txt (Extended Asm)
  - http://linuxassembly.org/resources.html
- Google.
- Ask us.
- To access a C variable in inline assembly
  - `asm volatile("statements":output_regs:input_regs:used_regs);`
- Examples
  - `asm volatile("movl %0,%esp":="q"(cur_running->stack));`
  - `asm volatile("movl %0,%0":="q"(cur_running->stack));`

Design Review Requirements

Please list your plan of actions for the following:

- Process Control Block (PCB)
  - What's in it?
- Context Switch procedure
- System Call mechanism
- Stacks
- Synchronization
  - How are locks implemented?
More hints: 1. Flat Address Space

- The bootblock code switches to protected mode. It also sets up the CS, DS, and other segment registers so that you can use the entire memory using just registers like eax.
- Do NOT modify the segment registers.
- You have access to the first 1MB of memory which includes the video-memory area(0xB8000).
- To be safe make sure your code, data, and stacks reside within the first 640KB of memory.

More hints: 2. Synchronization

- The synchronization primitives can only be used by threads within the kernel.
- You will notice that there are two functions block() and unblock() in the kernel that are to be used by the synchronization code.
- This has been done since blocking a process is not specific to locks, but is a general purpose service that the kernel should support.