COS 318 Project 2
Non-Preemptive Scheduling

Precept 2

Agenda

- Questions from design review / emails
- Miscellaneous
- Grading criteria
- GDB on bochs (CJ Bell)

Blocking Semantics

- When a thread required a LOCKED lock, it gets blocked, not coming back to ready queue
- when a thread releases a lock, it unlocks the first waiting thread
- lock_release() does not imply do_yield()
- When a thread is unblocked, it is not executed until all unblocked tasks at the time have yielded

Example Code

Thread 1:

```
lock_init(&lock);
lock_acquire(&lock);
do_yield();
lock_release(&lock);
do_exit()
```

Thread 2:

```
while(TRUE) {
        do_yield();
}
```

Thread 3:

```
do_yield();
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```

Thread 4:

```
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```

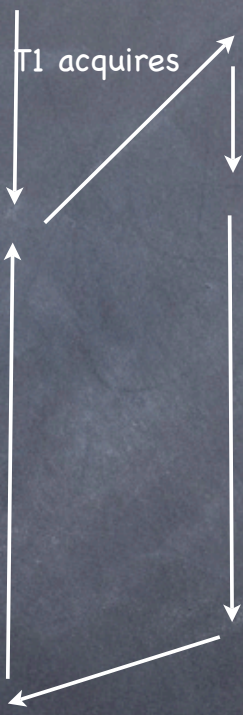
Example Code

Thread 1:

```
lock_init(&lock);
lock_acquire(&lock);
do_yield();
lock_release(&lock);
do_exit()
```

Thread 2:

```
while(TRUE) {
    do_yield();
}
```



Thread 3:

```
do_yield();
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```

Thread 4:

```
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```

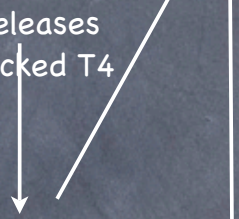
Example Code

Thread 1:

```
lock_init(&lock);
lock_acquire(&lock);
do_yield();
lock_release(&lock);
do_exit()
```

Thread 2:

```
while(TRUE) {
    do_yield();
}
```

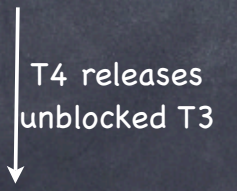


Thread 3:

```
do_yield();
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```

Thread 4:

```
lock_acquire(&lock);
lock_release(&lock);
do_exit();
```



Lock Implementation

- Simple lock implementation: LOCKED, UNLOCKED
- Do not have to handle deadlock
- Think what you should do when:
  - two threads are blocked waiting for the same lock? Follow FIFO rule
  - Some other process tries to acquire the lock before the unblocked process starts running

PCB

- PCBs are statically allocated in memory for this project
- No recycling of memory space of any kind
  - stack, pcb, locks.....
- You may add whatever you feel necessary
  - start address of a program
  - kernel thread or user process

Context Switching

- kernel_entry()
  - Used to switch between user process and the kernel for system calls
  - saves and restores user registers
- scheduler_entry()
  - used to switch between kernel threads and user processes that are in kernel
  - saves and restores system registers
- yield() system call (in syslib)
  - goes through kernel_entry() to switch to kernel mode
  - goes through scheduler_entry() to switch to another process/thread

Example

- Process P -> Thread T
  - yield() system call -> kernel_entry
    - save registers, load kernel stack (working on kernel stack now)
    - do_yield()
    - load user stack, restore user registers
  - do_yield()
    - enqueue P to ready queue
    - scheduler_entry()
      - save P's kernel registers
      - scheduler()
      - load T's kernel registers
      - ret

First time to switch to a task

- There is no return address on stack
- How do you find where to return to?
  - ask scheduler to jump to the entry point of the program if it is the first time to run
  - what else?

Inline Assembly

- General format:

```
_ _asm_ _ (
"instruction"
"instruction"
....
"instruction"
:"=flags"
:"=flags"
:"=flags"
) ;
```

- volatile : _ _asm_ _ volatile
- flags: refer to the resource page provided on project website

Grading Criteria

- Total: 10 points + 1 extra credit point
- Kernel threads and scheduling: 3 points
- Processes and system calls: 3 points
- Mutual exclusion: 2 points
- Timing a context switch: 1 point
- Coding style, comments, and README: 1 point
- If your program runs on bochs, but does not run on fishbowl, 1 point penalty at most

GDB for bochs

- Thanks, CJ