CS3210: Clone and spinlock

Tutorial 7

Agenda

- Kernel threads via. clone system call
- In class exercise:
  - Completing the implementation of clone system call and spinlock

Fork and thread

- Q: Difference between fork and thread?
- Q: What do threads share with parent process?

Fork and thread

- Fork creates a separate address space for each process
- Threads share
  - Address space including text, data, and BSS segments
  - File descriptors, signals, current working directory, user and group ID
- Do not share
  - Thread ID, saved registers, stack pointer, instruction pointer
  - Stack, signal mask, priority (scheduling information)

Clone system call

- Clone system call creates and sets up kernel stack (allocproc() in xv6)
- It also sets up thread user stack using the parent stack
  - Sets the base and stack pointers, and then copies parents stack
- Shares other references (files, pagedir)

Stack basics

```
void MyFunction3(int x, int y, int z)
{
  int a=x, int b=y, int c=z;
  ...
  return;
}
```

```
|  2  | [ebp + 16] (3rd function argument)
|  5  | [ebp + 12] (2nd argument)
| 10 | [ebp + 8]  (1st argument)
| RA| [ebp + 4]  (return address)
| FP | [ebp]     (old ebp value)
|     | [ebp - 4]  (1st local variable)
:     :
:     :
|     | [ebp - X]  (esp - the current stack pointer.
The use of push / pop is valid now)
```

Lab excercise goal today

- Our patch has thread library to create thread
- Thread library allocates stack page and passes it to clone
- You will complete a stack setup first
- Next, you will complete spinlock code in thread library (threadlib.c)
  - lock_init(lock_t * lock)
  - lock_acquire(lock_t * lock)
  - lock_release(lock_t * lock)

Tutorial

- Get a clean xv6 code and then get the patch from cs3210-pub

```
$ git clone https://github.com/mit-pdos/xv6-public.git
$ git clone git://tc.gtisc.gatech.edu/cs3210-pub
```

- Copy the patch to xv6 dir and apply like this

```
$ patch -p1 < tut07_students.patch
```