CS 326: Operating Systems

Exploring the OS

Lecture 2

Today’s Schedule

- Development Environment Setup
- Exploring xv6

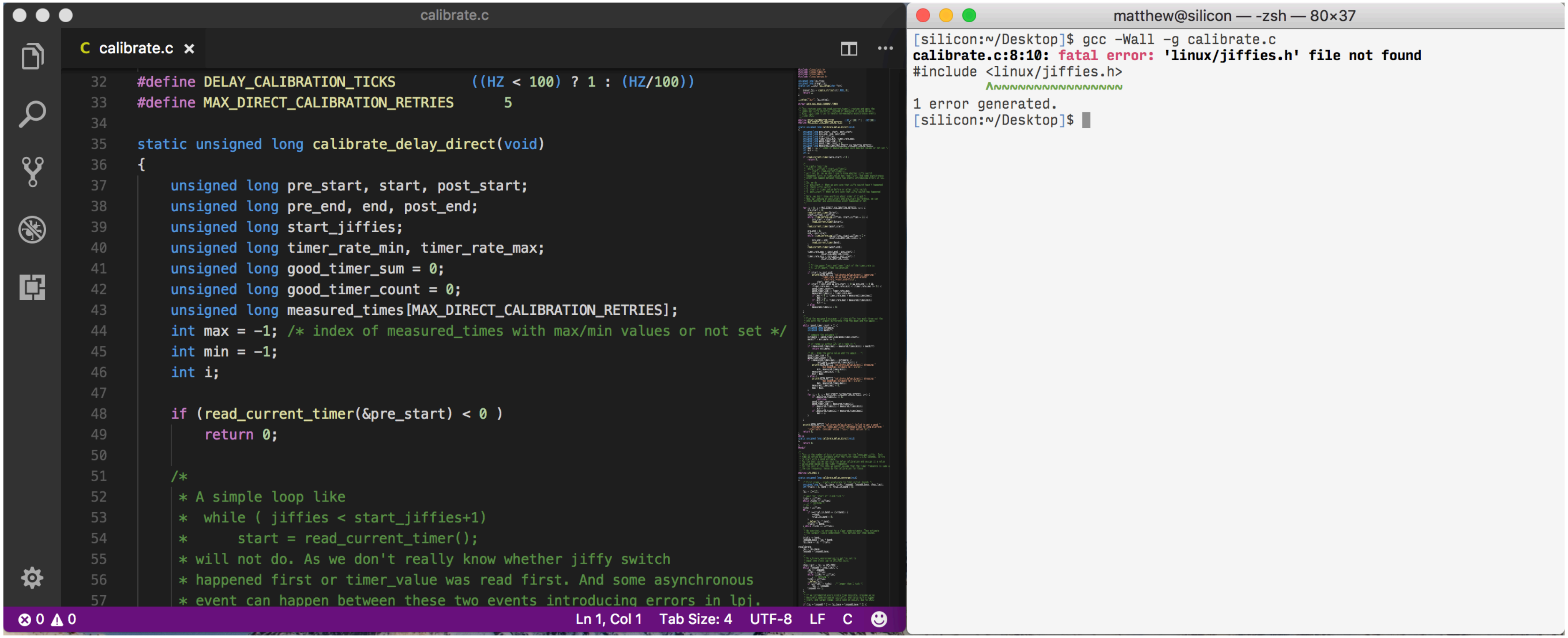
Today's Schedule

- **Development Environment Setup**
- Exploring xv6

Writing C Programs [1/3]

- Using an IDE (like Eclipse, IntelliJ, etc) is less common in the C world
- Many C developers prefer to use a text editor and a terminal to write their programs
  - Text editor: edit, save
  - Terminal: compile, run
- There’s information on the course schedule page for setting up your editor and C compiler

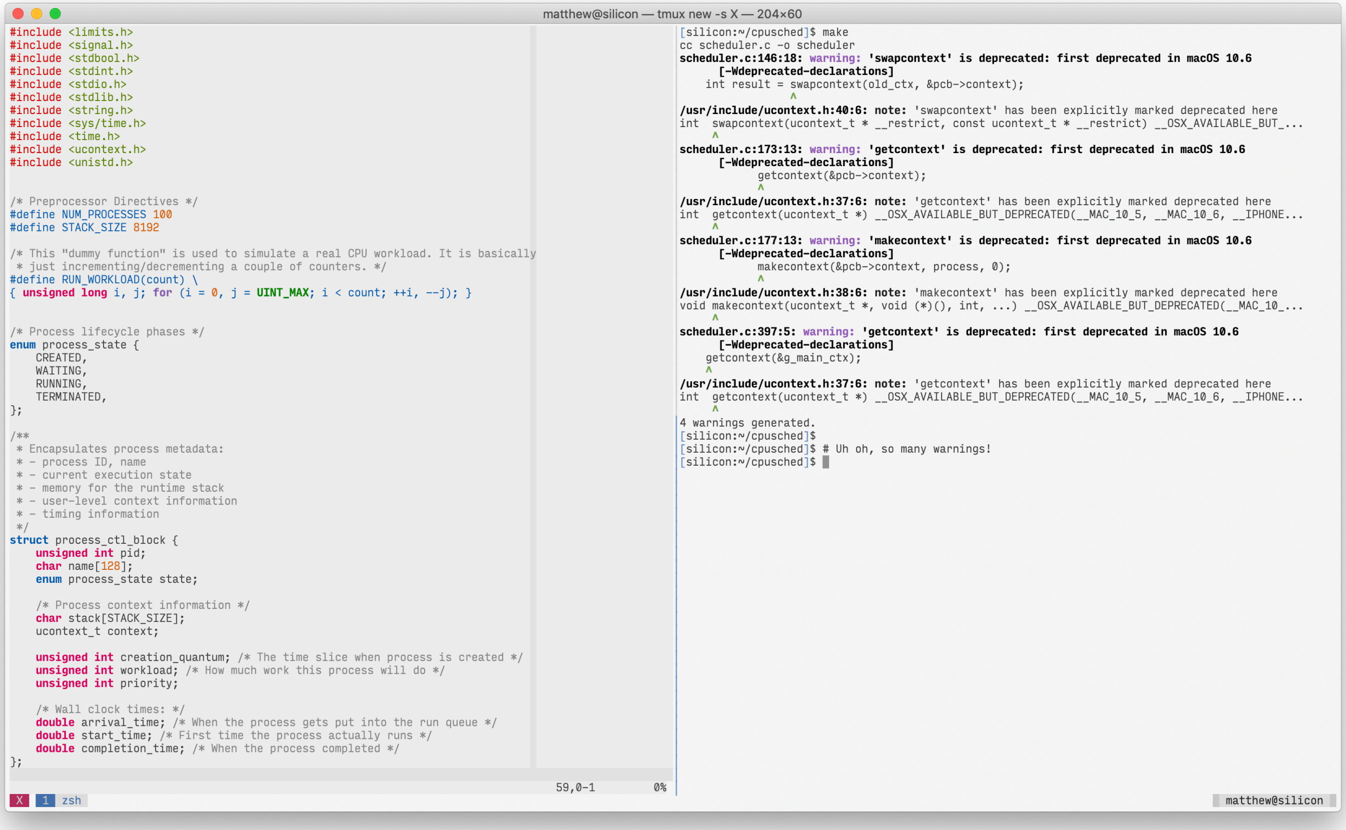
Writing C Programs [2/3]



```c
#define DELAY_CALIBRATION_TICKS         ((HZ < 100) ? 1 : (HZ/100))
#define MAX_DIRECT_CALIBRATION_RETRIES      5

static unsigned long calibrate_delay_direct(void)
{
    unsigned long pre_start, start, post_start;
    unsigned long pre_end, end, post_end;
    unsigned long start_jiffies;
    unsigned long timer_rate_min, timer_rate_max;
    unsigned long good_timer_sum = 0;
    unsigned long good_timer_count = 0;
    unsigned long measured_times[MAX_DIRECT_CALIBRATION_RETRIES];
    int max = -1; /* index of measured_times with max/min values or not set */
    int min = -1;
    int i;

    if (read_current_timer(&pre_start) < 0 )
        return 0;

    /*
     * A simple loop like
     *  while ( jiffies < start_jiffies+1)
     *      start = read_current_timer();
     * will not do. As we don't really know whether jiffy switch
     * happened first or timer_value was read first. And some asynchronous
     * event can happen between these two events introducing errors in lpj.
```

```
[silicon:~/Desktop]$ gcc -Wall -g calibrate.c
calibrate.c:8:10: fatal error: 'linux/jiffies.h' file not found
#include <linux/jiffies.h>
         ^~~~~~~~~~~~~~~~~

1 error generated.
[silicon:~/Desktop]$ 
```

Writing C Programs [3/3]



```c
#include <limits.h>
#include <signal.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <ucontext.h>
#include <unistd.h>


/* Preprocessor Directives */
#define NUM_PROCESSES 100
#define STACK_SIZE 8192

/* This "dummy function" is used to simulate a real CPU workload. It is basically
 * just incrementing/decrementing a couple of counters. */
#define RUN_WORKLOAD(count) \
{ unsigned long i, j; for (i = 0, j = UINT_MAX; i < count; ++i, --j); }


/* Process lifecycle phases */
enum process_state {
    CREATED,
    WAITING,
    RUNNING,
    TERMINATED,
};

/**
 * Encapsulates process metadata:
 * - process ID, name
 * - current execution state
 * - memory for the runtime stack
 * - user-level context information
 * - timing information
 */
struct process_ctl_block {
    unsigned int pid;
    char name[128];
    enum process_state state;

    /* Process context information */
    char stack[STACK_SIZE];
    ucontext_t context;

    unsigned int creation_quantum; /* The time slice when process is created */
    unsigned int workload; /* How much work this process will do */
    unsigned int priority;

    /* Wall clock times: */
    double arrival_time; /* When the process gets put into the run queue */
    double start_time; /* First time the process actually runs */
    double completion_time; /* When the process completed */
};
```

```
[silicon:~/cpusched]$ make
cc scheduler.c -o scheduler
scheduler.c:146:18: warning: 'swapcontext' is deprecated: first deprecated in macOS 10.6
      [-Wdeprecated-declarations]
    int result = swapcontext(old_ctx, &pcb->context);
                 ^
/usr/include/ucontext.h:40:6: note: 'swapcontext' has been explicitly marked deprecated here
int  swapcontext(ucontext_t * __restrict, const ucontext_t * __restrict) __OSX_AVAILABLE_BUT_...
     ^
scheduler.c:173:13: warning: 'getcontext' is deprecated: first deprecated in macOS 10.6
      [-Wdeprecated-declarations]
            getcontext(&pcb->context);
            ^
/usr/include/ucontext.h:37:6: note: 'getcontext' has been explicitly marked deprecated here
int  getcontext(ucontext_t *) __OSX_AVAILABLE_BUT_DEPRECATED(__MAC_10_5, __MAC_10_6, __IPHONE...
     ^
scheduler.c:177:13: warning: 'makecontext' is deprecated: first deprecated in macOS 10.6
      [-Wdeprecated-declarations]
            makecontext(&pcb->context, process, 0);
            ^
/usr/include/ucontext.h:38:6: note: 'makecontext' has been explicitly marked deprecated here
void makecontext(ucontext_t *, void (*)(), int, ...) __OSX_AVAILABLE_BUT_DEPRECATED(__MAC_10_...
     ^
scheduler.c:397:5: warning: 'getcontext' is deprecated: first deprecated in macOS 10.6
      [-Wdeprecated-declarations]
    getcontext(&g_main_ctx);
    ^
/usr/include/ucontext.h:37:6: note: 'getcontext' has been explicitly marked deprecated here
int  getcontext(ucontext_t *) __OSX_AVAILABLE_BUT_DEPRECATED(__MAC_10_5, __MAC_10_6, __IPHONE...
     ^
4 warnings generated.
[silicon:~/cpusched]$
[silicon:~/cpusched]$ # Uh oh, so many warnings!
[silicon:~/cpusched]$
```

My Recommendation [1/2]

- In 326, you are going to be using the terminal and command line interfaces with your VM **a lot**
- I recommend you to embrace it, learn it, and (maybe) love it
  - Sets you up for working in cloud computing, DevOps, system administration
  - The interface is all text: facilitates command **composition**
- If you're not super comfortable with Unix commands, don't worry! You'll get lots of practice

My Recommendation [2/2]

- Learn vim
  - …or emacs, nano, micro, etc.
- The point: know enough about using a terminal text editor to be able to get things done
- Maybe you won’t spend 100% of your time there, but it can come in handy in a pinch

Other Options

- Lots of IDEs have remote editing functionality
  - *rsub* (Remote Sublime) is a popular option
  - Visual Studio Code has very powerful remote editing, syntax highlighting, autocompletion
- FTP/SFTP clients like **Cyberduck**, **Termius**, **Forklift** can automatically sync your changes with a remote server
- Ultimately, use what you're comfortable with
  - Spending 60 hours learning vim is awesome, but not if it means you can't get your projects done

Getting Help

- When you're working from the terminal, the `man` (manual) pages are a great resource for help
- Many times your Google searches are just going to locate man pages that have been converted to HTML!
- There are a few sections in the man pages:
  1. User commands
  2. System calls
  3. C library functions
  4. …and more

Reading man pages

- `man whatever`
  - `man man`
- Specify the section like so:
  - `man 3 printf`
  - This is particularly important for our class: we need section **2** for system calls and section **3** for C functions
- Man pages also will often explain config files' syntax and options

Creating an ssh alias

- Please take a look at the Working Remotely page on the course schedule for more hints!
- The best bit of advice: creating an `ssh` alias for **gojira**
  - I also highly recommend setting up an ssh key so you won't need to type your password over and over

Today’s Schedule

- Development Environment Setup
- **Exploring xv6**

Exploring xv6

- By building your OS off xv6, you benefit from a lot of existing code
  - …but now you might have to read and understand that code
  - reading code is a good skill to learn, but it's not always fun
- Let's take a tour of xv6 in just a minute…
  - To start, how big is the codebase? `sloccount` can give us an idea…

Make

- You’ll be using `make` to build your code in this class. Lab 0 requires us to modify the *Makefile*
  - This tells the `make` utility what to do
  - Essentially just a recipe for building your program
- Hints:
  - `make` – compile kernel
  - `make qemu` – compile kernel, user space, create file system, and run the OS in QEMU
  - `make clean` – clean up all build artifacts

Exploring

- Take 5 minutes to look around the xv6 codebase
  - Find something that you think is interesting
- We'll regroup and go through its structure