CSE130 Winter 2020 : Lab 1

In this lab you will make minor modifications and minor additions to the source code for the teaching operating system Pintos.

Specifically, you will implement a more efficient version of the Pintos system call `timer_sleep` found in `src/devices/timer.c` in the Pintos distribution supplied for this lab.

This lab is worth 5% of your final grade.

Submissions are due NO LATER than 23:59, Wednesday January 15 2020 (1.5 weeks)

Note the elongated time span for this lab is provided so you can become familiar with the CSE130 teaching server environment and the Pintos teaching operating system. Subsequent labs will be significantly more complex yet come with only marginally longer deadlines.

Setup

SSH in to one of the two CSE130 teaching servers using your CruzID Blue password:

```
$ ssh <cruzid>@noggin.soe.ucsc.edu   ( use Putty http://www.putty.org/ if on Windows )
Or  $ ssh <cruzid>@nogbad.soe.ucsc.edu
```

Authenticate with Kerberos: ***(do this every time you log in)***

```
$ kinit      ( you will be prompted for your Blue CruzID password )
```

Authenticate with AFS: ***(do this every time you log in)***

```
$ aklog
```

Create a suitable place to work: ***(only do this the first time you log in)***

```
$ mkdir -p CSE130/Lab1
$ cd CSE130/Lab1
```

Install the lab environment: ***(only do this once)***

```
$ tar xvf /var/classes/CSE130/Winter20/Lab1.tar.gz
```

Build Pintos:

```
$ cd ~/CSE130/Lab1/pintos/src/threads      ( always work in this directory )
$ make
```

Also try:

```
$ make check      ( runs the required functional tests - see below )
$ make grade      ( tells you what grade you will get - see below )
```


Additional Information

`void timer_sleep(int64_t ticks)` should suspend execution of the calling thread until time has advanced by at least the requested number of ticks. Unless the system is otherwise idle, the thread need not wake up after exactly `ticks`. It is entirely valid to just put it on the ready queue after they have waited for the right number of ticks.

Note that the `ticks` argument is expressed in timer ticks, not in milliseconds or any other unit. Do not change this data type unless you want lots of the tests to fail.

Separate functions `timer_msleep`, `timer_usleep`, and `timer_nsleep` exist for sleeping a specific number of milliseconds, microseconds, and nanoseconds respectively, but these will call `timer_sleep` as appropriate. You do not need to modify them.

Accessing the teaching servers' file systems from your personal computer

You home directories on the teaching servers are UCSC AFS, i.e. they are the same as if you logged into the UCSC Unix Timeshare. To access files in your home directory to edit them using an IDE or editor of choice on your laptop, you can do the following:

- Use an SSH synchronization client like Cyberduck: https://cyberduck.io
    - Configure Cyberduck to point at any of the teaching servers
    - Connect with your CruzID and Blue Password

Detailed instructions on how to do this are included in **Appendix 1** but you should follow the setup instructions above before attempting this.

Requirements

Basic:

- You have modified the implementation of `void timer_sleep(int64_t ticks)` function found in `src/devices/timer.c`

- Your modified implementation passes the following functional tests:
    - alarm-single
    - alarm-multiple
    - alarm-simultaneous
    - alarm-zero
    - alarm-negative

- Your implementation is demonstrably efficient (a non-functional test).

What steps should I take to tackle this?

Come to sections and ask.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 20 lines of executable code.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

    a. Tests pass [5% per test] (25%)
    b. Your implementation is demonstrably more efficient than the original (75%)

2. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

    b. Your submission is determined to be a copy of a past or present student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 33% copied code No deduction
        - 33% to 66% copied code (-50%)
        - > 66% copied code (-100%)

What to submit

In a command prompt:

```
$ cd ~/CSE130/Lab1/pintos/src/threads
$ make submit
```

This creates a gzipped tar archive named `CSE130-Lab1.tar.gz` in your home directory.

****** UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT ******

Looking ahead

Subsequent labs may require a short report and will have additional assessment criteria, including:

Is it well written?

Marks awarded for:

- Compilation free of errors and/or warnings.
- Clarity
- Modularity

Marks deducted for:

- Failing to do any of the above
- Not following C language good practice (don't use magic numbers, for example)

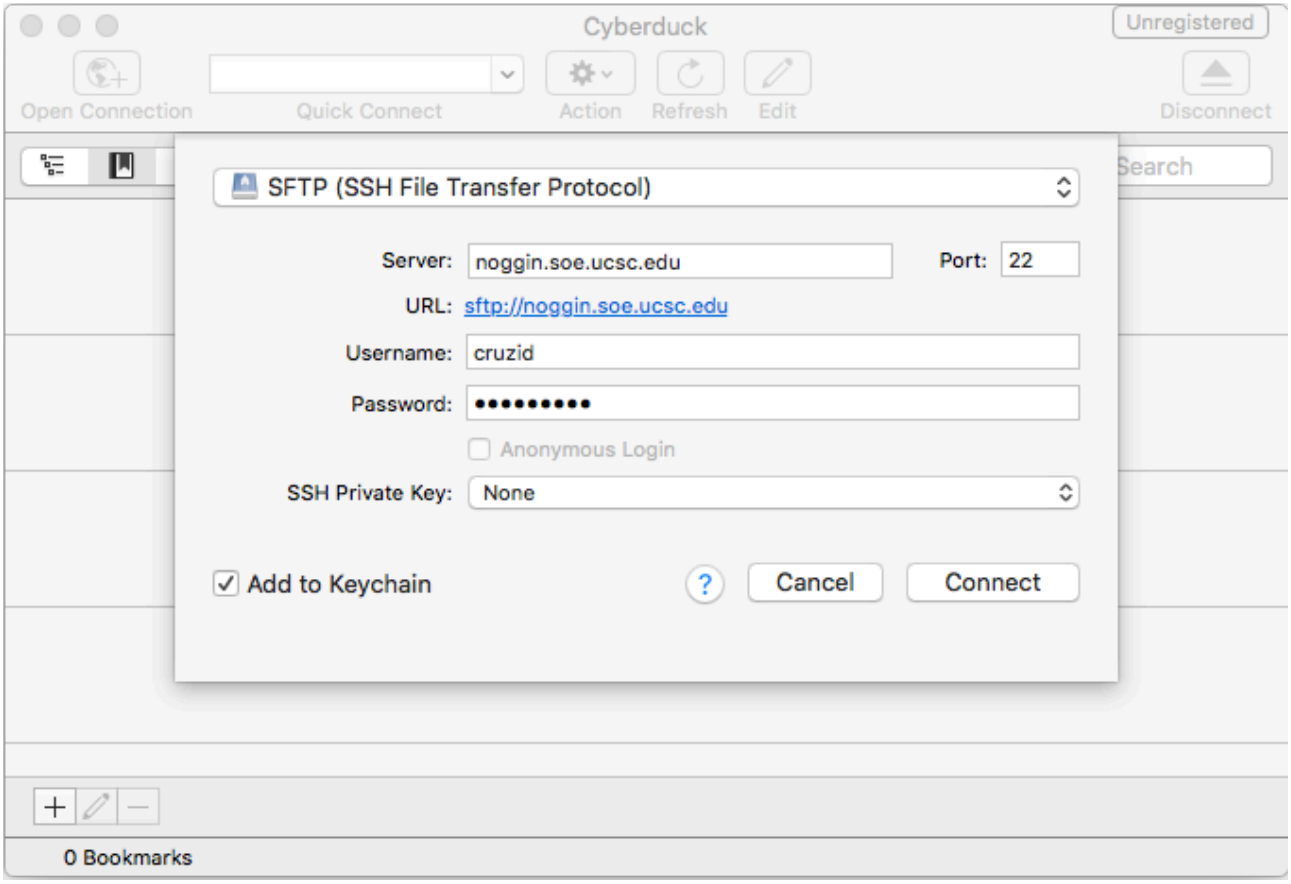
I strongly recommend you consider these issues in this lab to get some practice in.

Appendix 1 – Accessing the teaching servers' filesystems with Cyberduck

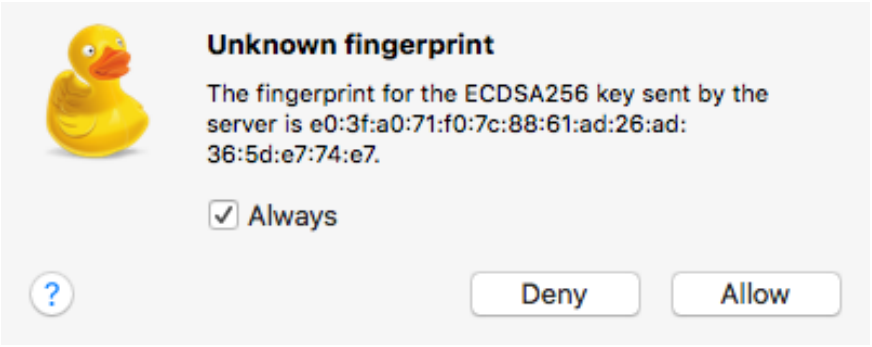
If you are on an BSOE Lab Workstation, Cyberduck is already installed, if using your own machine, Cyberduck can be downloaded from https://cyberduck.io.

Launch Cyberduck and click the "Open Connection" button.

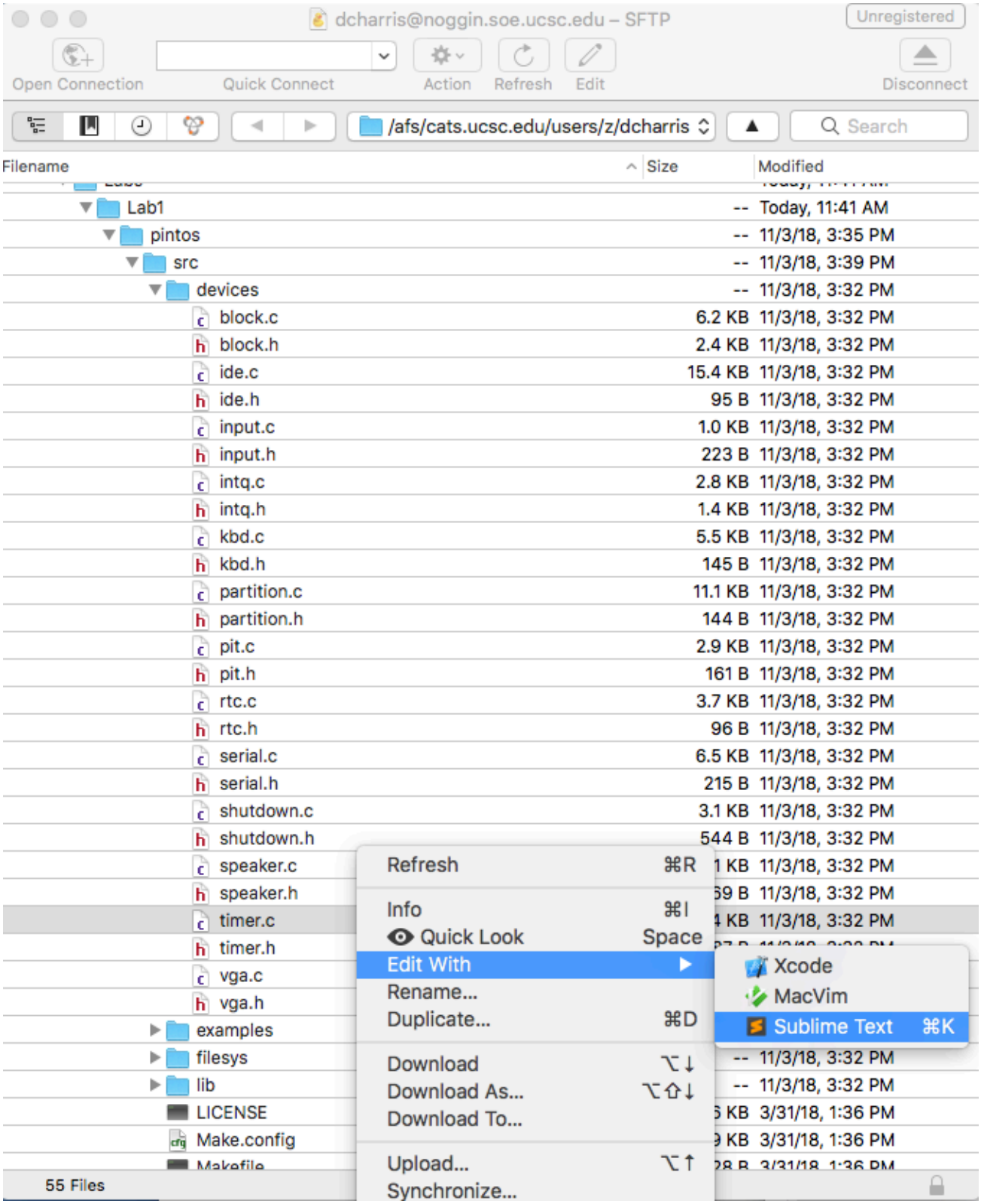
Select "SFTP (SSH File Transfer Protocol)" and enter the name of the teaching server and your Blue CruzID credentials:



Select "Always" and click "Allow" to accept the teaching server's fingerprint.

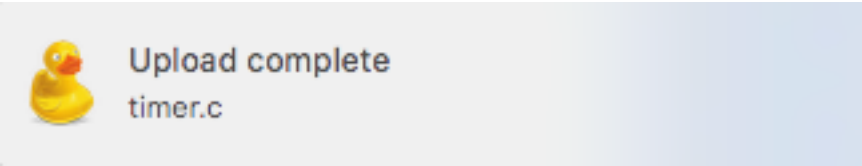


Navigate to your CMPS11 Lab 1 directory and open `timer.c` for editing via the context ("right mouse") menu then select "Edit With" then choose your editor of choice.



If your editor does not appear in the list, or there no editors in the list, speak with a TA and have them create a file association for you. This will only happen on personal Windows machines that have not been used for editing C source code before; Mac users should be OK and it will work fine on the BSOE Lab workstations.

When the file is open in your editor or choice, modify it slightly and save it. A small notification will appear on your screen letting you know the file has been successfully uploaded to the server.



In a terminal window, you can now build and test your modified code.

IMPORTANT: Before closing your laptop and/or disconnecting from the WiFi, close any editor windows you have open and shutdown Cyberduck.

§

