CMPS111 Winter 2018 : 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 17, 2018 (one week)

Setup

SSH in to the CMPS111 teaching server using your CruzID Blue password:

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

Set your SAMBA password: ***(only do this the first time you log in)***

```
$ smbpasswd
```

Your initial password ("Old SMB password") is your student id - change it to something memorable. Your CruzID Blue password is as good a choice as any.

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

```
$ mkdir -p CMPS111/Lab1
$ cd CMPS111/Lab1
```

Install the lab environment: ***(only do this the first time you log in)***

```
$ tar xvf /var/classes/CMPS111/Winter18/Lab1.tar.gz
$ ./setenv
$ . ~/.bashrc
```

Build Pintos:

```
$ cd ~/CMPS111/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 )
```

Accessing the teaching server file system from your personal computer

The teaching server has Samba (https://www.samba.org) installed, which lets you access its file system from another machine using a Microsoft standard protocol. To access the teaching server file system from your personal computer, follow these instructions:

macOS:	https://users.wfu.edu/yipcw/atg/apple/smb/
Windows:	https://help.lafayette.edu/samba/win7nondomain
Linux:	https://help.ubuntu.com/community/Samba/SambaClientGuide

In all cases, use:

Server Address:	smb://noggin.soe.ucsc.edu/<cruzid>
Registered User Name:	<cruzid>
Password:	<whatever you set it to> (see "Setup" section above)

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.

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.

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 the 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. You 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%)

   b. You submission is determined to be a copy of another past or current CMPS111 student's submission (-100%)

What to submit

In a command prompt:

```
$ cd ~/CMPS111/Lab1/pintos/src/threads
$ make submit
```

This creates a gzipped tar archive named `CMPS111-Lab1.tar.gz` in your home directory.

UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT.

Looking ahead

Subsequent labs will 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 (e.g. don't use magic numbers)

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

---------- 8< ----------

Appendix 1 - Running on your own computer

1) Download and install Oracle Virtual Box:

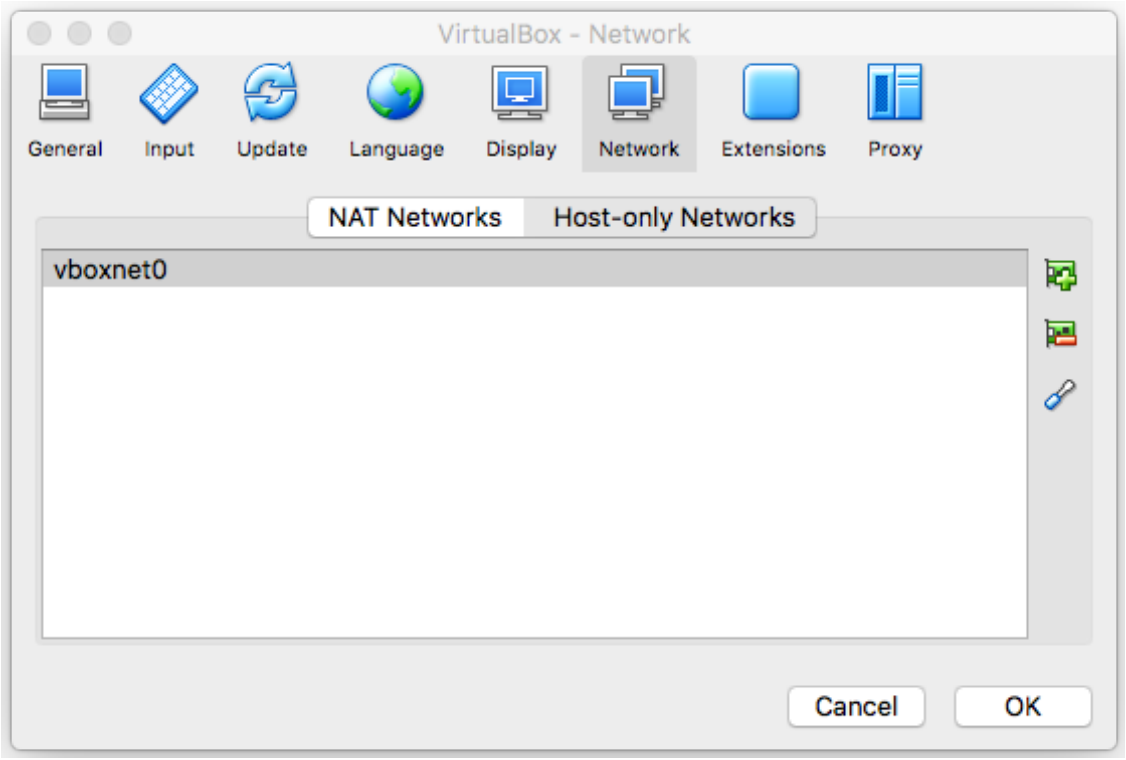
https://www.virtualbox.org/wiki/Downloads

Make sure to get the right one for your host operating system.

2) Download and Install Virtual Box Extension Pack, also from:

https://www.virtualbox.org/wiki/Downloads

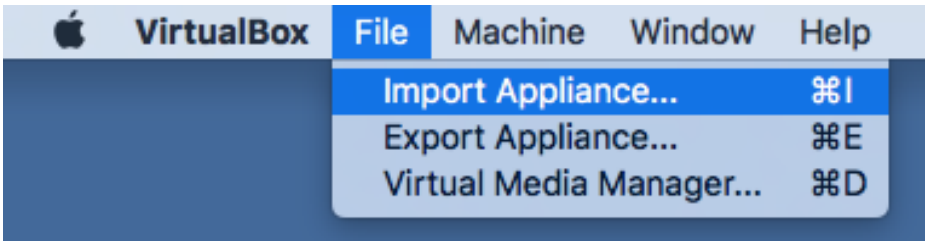
3) Create a Host-only Network by selecting the “Preferences” menu, then the “Network” tab, then the “Host-only Network” sub-tab, and then pressing the  button.



4) Download and install the CMPS111 virtual appliance:

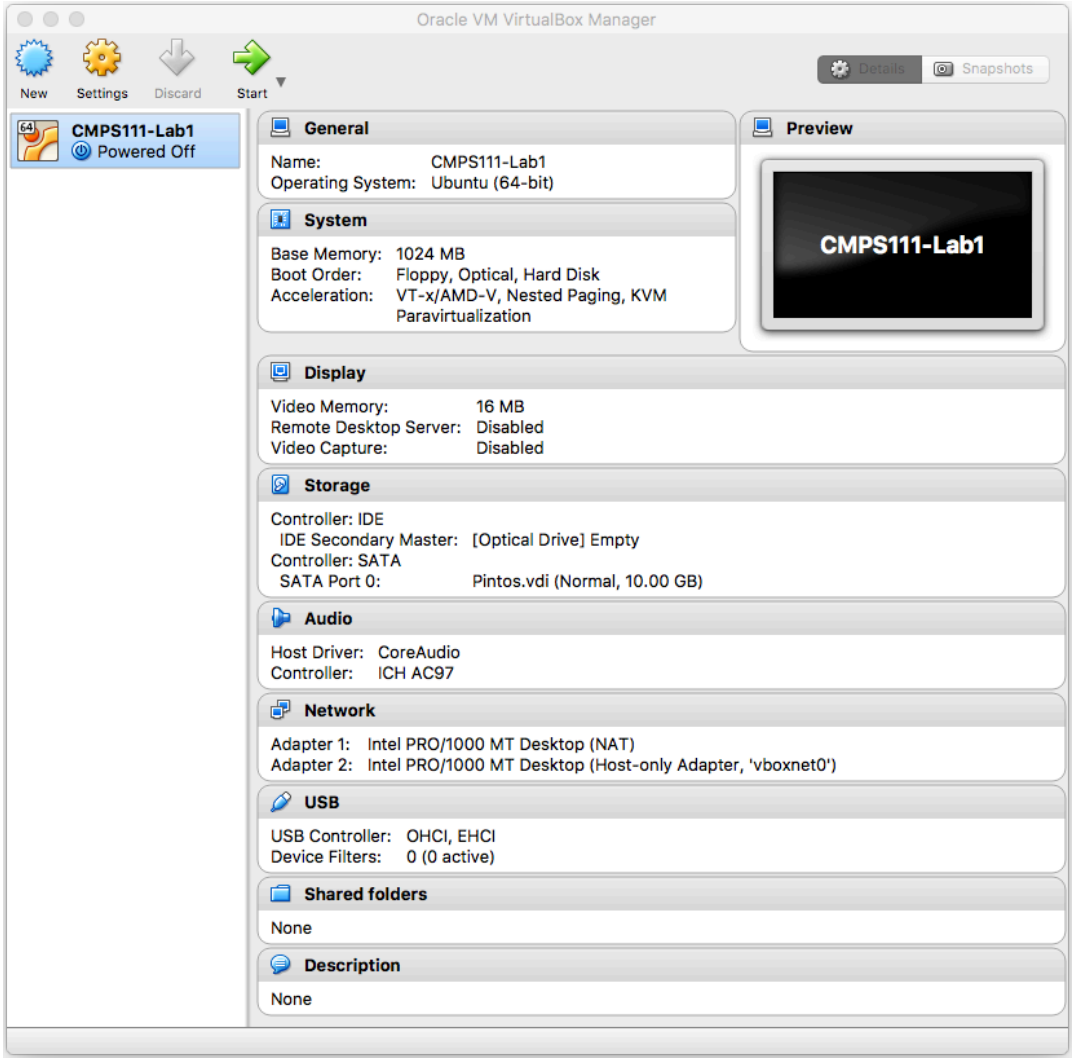
https://classes.soe.ucsc.edu/cmps111/Winter18/SECURE/CMPS111.ova

Import this appliance into Virtual Box using the “File->Import Appliance” menu:



Select the downloaded .ova and accept all defaults.

Once imported, the Virtual Box Manager should look something like this - note that this is an old screen shot, the Appliance name is now “CMPS111” as it is used for all labs:



Note that Network Adapter 2 is connected to the Host-only Adapter you created earlier.

Click the  button to start the lab appliance, an Ubuntu Desktop instance.

Appendix 2 - Log in to the virtual appliance from the host operating system

The supplied virtual appliance Ubuntu instance has an SSH daemon running. To log into the appliance from your host operating system, simply connect an SSH client using the following information:

Server Address:	192.168.56.104
User Name:	pintos
Password:	pintos

On Linux or macOS, this is as simple as typing:

```
$ ssh pintos@192.168.56.104
```

Accepting the ECDSA key fingerprint of the appliance (only needs to be done once) and entering the password.

On Windows, you'll have to download Putty (http://www.putty.org/) and configure a new session using the information above.

Once logged in, you'll need to follow the instructions in the "setup" section, the only difference being that when you need to get the lab environment, you'll fetch it via scp from the teaching server:

```
$ scp <cruzid>@noggin.soe.ucsc.edu:/var/classes/CMPS111/Winter18/Lab1.tar.gz .
```


Appendix 3 - Accessing the virtual appliance file system from host operating system

The supplied virtual appliance Ubuntu instance has Samba Server (https://www.samba.org) installed which lets you access it's file system from another machine using a Microsoft standard. To access the virtual appliances file system from you host, follow these instructions:

macOS:	https://users.wfu.edu/yipcw/atg/apple/smb/
Windows:	https://help.lafayette.edu/samba/win7nondomain
Linux:	https://help.ubuntu.com/community/Samba/SambaClientGuide

In all cases, use:

Server Address:	smb://192.168.56.104/pintos
Registered User Name:	pintos
Password:	pintos

Appendix 4 - Using an IDE to modify the Pintos code

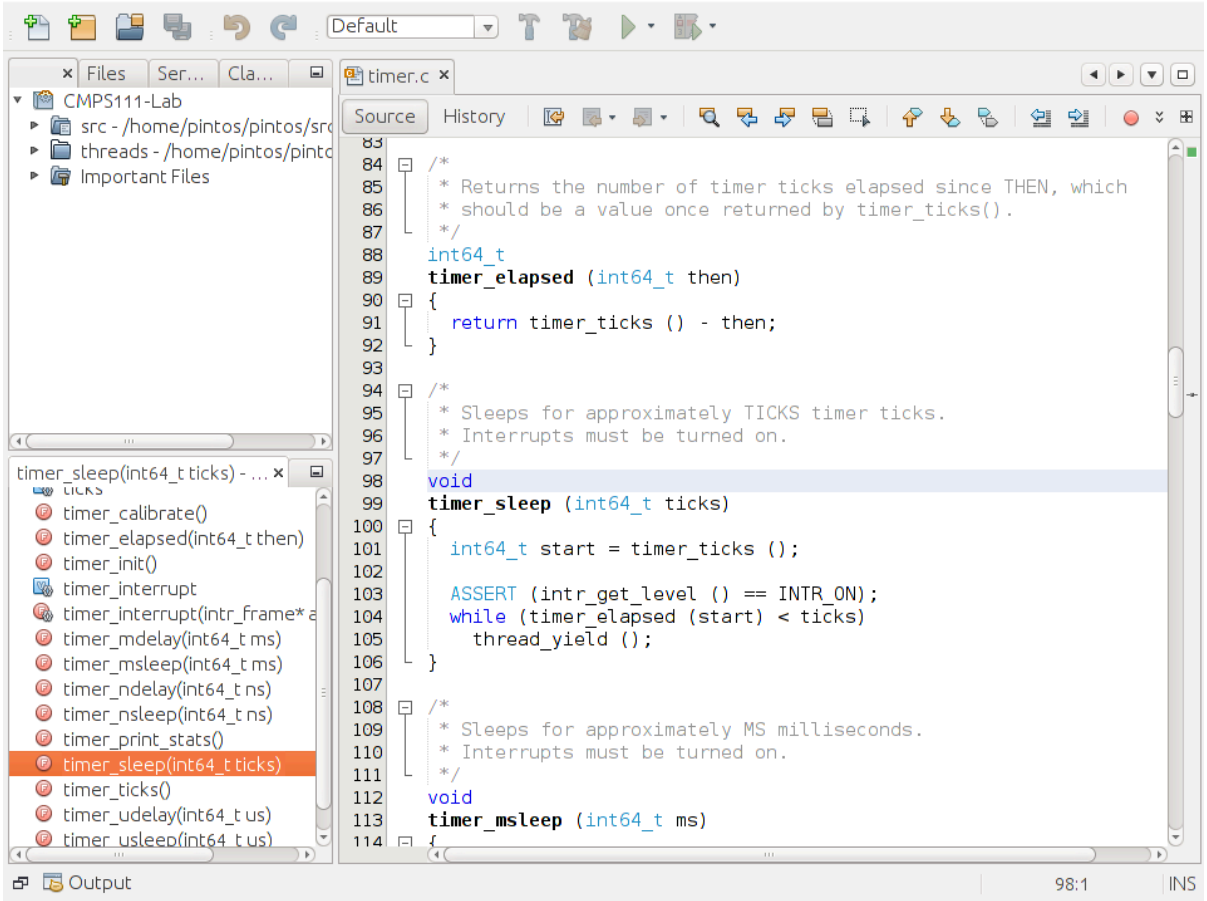
Whilst it is entirely possible to compete this lab and the following two using command line editors, most people prefer to use an IDE these days.

If you've followed the instructions in Appendix 2, you have access to the appliance file system from your host, so feel free to fire up the IDE of your choice and point it at the Pintos installation.

If you're using a "New Project with Existing Makefile" style of IDE (like Eclipse), use one found at `CMPS111/Lab1/pintos/src/threads/Makefile`.

If you'd rather use an IDE inside the virtual appliance, a copy of NetBeans for C/C++ is installed

Simple click the  button in the appliance to start NetBeans.



§

