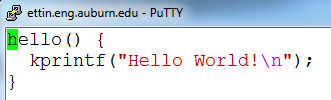
# COMP 3500 – Lecture 8: Project 2 Modify the Kernel

**Xiao Qin @ Auburn University**

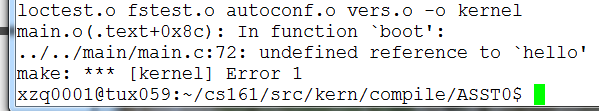
**Note:** I rebuilt the kernel by adding "hello world!" into the boot message. In what follows, I summarize my process of rebuilding the OS161 kernel. You may also found the three common mistakes at the end of this document.

Your hello.c should be as simple as the following code:



When we compile OS161 after adding hello.c, we may encounter the following compilation error.





1. edit kern/conf/conf.kern

cd ~/cs161/src/kern/conf

vi conf.kern

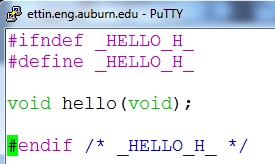
/main.c #search main.c in conf.kern see also line 374

Add the following line:

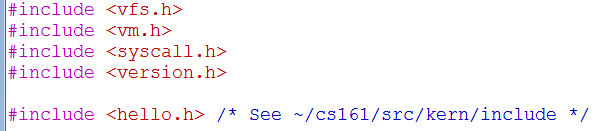
file main/hello.c

2. In directory ~/cs161/src/kern/include

Create a header file called hello.h



3. Add #include <hello.h> into main.c



4. Important! Configure your tree for the machine on which you are working.

cd ~/cs161/src

./configure

5. Configure a kernel named ASST0.

cd ~/cs161/src/kern/conf

./config ASST0

6. Build the ASST0 kernel. (cs161/src/compile). Use ‘echo $PATH’ to check your $PATH.

cd ../compile/ASST0

make depend

make

**make install** Important!

7. Now also build the user level utilties.

cd ~/cs161/src

make

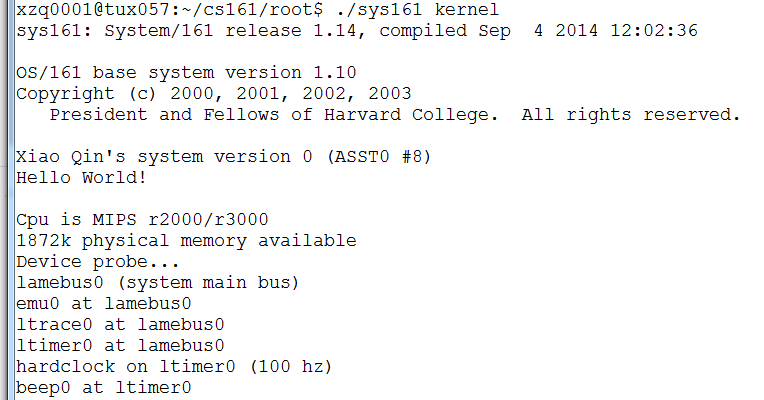
8. Run OS161: Change into your root directory.

%cd ~/cs161/root

Run the machine simulator on your operating system.

%./sys161 kernel

Now you should see the following outcome:



**Common Mistakes**

1. You must rebuild your kernel by following the three steps. If you forget to install the kernel using "make install", your kprintf() will not print “hello world”.

cd ../compile/ASST0

make depend

make

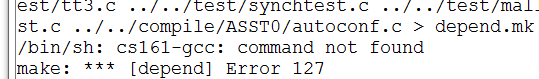
make install

2. In your hello.c, you must use kprintf() rather than printf().

3. Remember to setup PATH using the following command.

export PATH=~/cs161/bin:$PATH

If you forget to configure PATH, you will encounter the error message below.



# COMP3500: Project 2 How to use the DEBUG macro?

**Xiao Qin @ Auburn University**

**Note:** The instructions below show you (1) how to use the DEBUG() macro, and (2) how to configure the debug flags in the OS/161 kernel. Please also refer to the lecture slides for details.

Important export PATH=~/cs161/bin:$PATH

1. Find the implementation of the DEBUG() macro in kern/include/lib.h

cd ~/cs161/src/kern/include

vi lib.h



The above macro is equivalent to the following code:

if (dbflag & (d) != 0)

kprintf(fmt, ##args);

/\* else do nothing \*/

2. In lib.h you also can find a list of DEBUG bit flags below.



Each flag is used as the ‘d’ parameter in DEBUG(d, fmt, args, …). For example, we have:



The above macro is equivalent to the following code:

if (dbflag & (DB\_VM) != 0)

kprintf(fmt, ##args);

/\* else do nothing \*/

3. You can configure DEBUG messages using the dbflag variable defined in kern/lib/kprintf.c

cd ~/cs161/src/kern/lib

vi kprintf.c



If dbflags = 0, all the debug messages are disabled. You are allowed to enable debug message of any module (i.e., VM) or the combination of a list of modules. For example, to enable debug messages of the virtual memory module, you simply set dbflag in kprintf.c as:

u\_int32\_t dbflags = DB\_VM;

Please note that DM\_VM is a debug bit flag defined in lib.h (see also item 2 on page 1). For another example, to enable debug messages of the virtual memory module and the syscall module, you simply set dbflag in kprintf.c as:

u\_int32\_t dbflags = DB\_VM | DB\_SYSCALL; /\* Don’t use || here.\*/

Here you must use the bitwise operator | rather than ||.

4. Now, let me show you how the debug-message configuration works. We insert a debug message in the boot() function of main.c.

vi ~/cs161/src/kern/main/main.c

Go to line 76, add the following code. You may add the debug messages anywhere in boot().



5. Let me configure the dbflag variable in kprintf.c to determine which debug message should be displayed (see also item 3 above regarding dbflag configuration).

vi ~/cs161/src/kern/lib/kprintf.c

Go to line 14, modify the following code.

u\_int32\_t dbflags = DB\_VM | DB\_SYSCALL;

6. Rebuild the ASST0 kernel. (cs161/src/kern/compile). Use ‘echo $PATH’ to check your $PATH.

cd ~/cs161/src/kern/compile/ASST0

make depend

make

make install

cd ~/cs161/src

make

8. Run OS161: Change into your root directory.

%cd ~/cs161/root

Run the machine simulator on your operating system.

%./sys161 kernel

Now you should see the following outcome:



**Modify dbflag to Configure Debug Messages**

1. Let’s update the dbflag variable in kprintf.c to configure debug messages. In this example, we will disable the SYSCALL message while leaving the VM message untouched (see also item 3 on page 2 regarding dbflag configuration).

vi ~/cs161/src/kern/lib/kprintf.c

Go to line 14, modify the following code.

u\_int32\_t dbflags = DB\_VM; /\* Remove DB\_SYSCALL \*/

2. Rebuild the ASST0 kernel. (cs161/src/kern/compile). Use ‘echo $PATH’ to check your $PATH.

cd ~/cs161/src/kern/compile/ASST0

make depend

make

make install

cd ~/cs161/src

make

3. Run OS161: Change into your root directory.

%cd ~/cs161/root

Run the machine simulator on your operating system.

%./sys161 kernel

Now you should see the following outcome:

