Working With OS/161

This document provides information about working with OS/161 and sys/161. It assumes that you've already installed OS/161 by following the instructions in the installation guide. In the remainder of this document, the symbol `$OS161TOP` refers to the top-level OS/161 directory that was created when you installed OS/161 into your account. If you followed the installation instructions, `$OS161TOP` refers to the directory `cs350-os161` in your course account.

Here is a table of contents:

Building an OS/161 Kernel

If you followed the instructions in the installation guide, you have already configured, built, and installed an OS/161 kernel. However, as you modify the OS/161 source code during your assignments, you will want to prepare new kernels so that you can test your changes.

Preparing a kernel involves several steps: configuration, building, and installation:

1. **configure:**
   You performed an initial kernel configuration when you followed the instructions for installing OS/161 for the current assignment. Normally, it is not necessary to re-configure each time you prepare a new kernel. One exception to this when you have added new kernel source files or removed kernel source files. In that case, you must edit the kernel configuration file and reconfigure the kernel before you attempt to build it. The other exception is when you are first starting a new assignment. Kernel configuration is assignment-specific, so you when you start a new assignment you should re-configure before building the kernel for the first time.

   The main kernel configuration file is located in the directory `$OS161TOP/os161-1.99/kern/conf` in a file called `conf.kern`. This file defines various kernel options and devices. Towards the end of `conf.kern` you will find file declarations like this

   ```
   file      thread/spinlock.c
   ```

   for each section of the kernel code, e.g., the thread section or the file system section. If you add a new file to the kernel code, you must add a corresponding `file` declaration in `kern.conf` and then reconfigure the kernel. Once you have changed `kern.conf`, you re-configure the kernel by running

   ```
   ./config ASSTX
   ```

   in the same directory. Here, the *ASSTX* parameter indicates which assignment you are working on. For the first assignment, use *ASST1*, for the second assignment, use *ASST2*, and so on.
2. **build:**
   Kernels should always be built in the kernel build directory, which is `$OS161TOP/os161-1.99/kern/compile/ASSTX` for the *X*th assignment. (This directory is created when you configure the kernel.) To build a new kernel, issue the following commands in the kernel build directory:

   ```
   bmake depend
   bmake
   ```

   If your kernel builds successfully, you should see a line similar to this near the end of the output from `bmake`:

```
*** This is ASST1 build #5 ***
```

   Each new version of your kernel gets assigned a new "build number" when it gets made. Take note of this number - you'll see why when we talk about running the kernel.

3. **install:**
   If everything goes smoothly with the kernel build, you may then install the new kernel. Do this by running the command

   ```
   bmake install
   ```

   in the kernel build directory. This will put a copy of your newly-built kernel in the directory `$OS161TOP/root` in a file called `kernel-`*`ASSTX`*. It will also set the symbolic link `kernel` (also in the `$OS161TOP/root` directory) to refer to the newly-installed kernel.

Building the OS/161 User-Level Programs

OS/161 comes with a variety of user-level programs that can run on top of the OS/161 kernel. These include standard UNIX-style utility programs, like `ls` and `cat`, and variety of test programs. The source files for the utility programs are located in `$OS161TOP/os161-1.99/user/{bin,sbin}`. The source files for the test programs are located in `$OS161TOP/os161-1.99/user/{uw-testbin,testbin}`.

Note that many user programs will **not** run with OS/161 as it is intially distributed, since many system calls are not implemented in this version of the operating system.

User-level programs are built and installed from the directory `$OS161TOP/os161-1.99/`. To build and install all of the user-level programs, simply run

```
bmake
bmake install
```

User-level programs are installed under `$OS161TOP/root/` in the `bin`, `sbin`, `testbin` and `uw-testbin` directories.

Running OS/161

The OS/161 kernel, as well as OS/161 user-level programs, run on the sys/161 machine simulator. To run your OS/161 kernel, you start up the simulator and you pass it the name of the file that contains your kernel binary.

When you run OS/161, you should be in the runtime root directory, which is `$OS161TOP/root`. From the runtime root directory, use the command

```
sys161 kernel-ASSTX
```

or simply:

```
sys161 kernel
```

Both do the same thing, because `bmake install` makes `kernel` a symbolic link to `kernel-`*`ASSTX`*.

If the system doesn't recognize the `sys161` command, this probably means that you've not properly set up your environment variables as described in the installation guide. Also, the sys/161 simulator expects to find a configuration file called `sys161.conf` in the directory in the runtime root directory. Again, if you followed the installation instructions, that file should be there.

The kernel will print various messages as it boots. When it has finished booting, it will present you with a command prompt. The output should look something like this:

```
linux028.student[182]% sys161 kernel
```

```
sys161: System/161 release 1.99.06, compiled Aug 23 2013 10:23:34

OS/161 base system version 1.99.05
Copyright (c) 2000, 2001, 2002, 2003, 2004, 2005, 2008, 2009
   President and Fellows of Harvard College.  All rights reserved.

Put-your-group-name-here's system version 0 (ASST0 #1)

312k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0

cpu0: MIPS r3000
OS/161 kernel [? for menu]:
```

You can type `q` at the command prompt to cause the kernel to shut down. Typing `?` will show you other commands. Note that some menu commands will not work properly at the beginning, because they depend on kernel functionality that you have not yet implemented. One particular thing that you can do from the kernel menu is to launch an initial user-level program, such as a command shell or some test program. (The `p` and `s` commands do this. For more information, see the on-line help for the operations menu.) Again, most of these programs will not work initially because they depend on unimplemented kernel functionality.

One thing you should notice, near the beginning of the kernel's boot messages, is the line:

```
Put-your-group-name-here's system version 0 (ASST1 #5)
```

What the kernel is doing here is printing its "build number" - the same number that was displayed in the `bmake` output when the kernel was built. This is a great way for you to ensure that the kernel that you are running is actually the kernel that you built, and not some earlier version of your kernel.

Debugging an OS/161 Kernel using GDB

To debug OS/161, you should use the CS350 version of GDB, which is called `cs350-gdb`. This version of GDB has been configured for the MIPS architecture and has been patched to be able to talk to System/161. The difference between debugging a regular program and debugging an OS/161 kernel is that the kernel is running in a machine simulator. You want to debug the kernel; running the debugger on the machine simulator is not very illuminating and we hope it will not be necessary. If you were to issue the command:

```
cs350-gdb sys161
```

from the runtime root directory you would be attempting to debug the simulator (`sys161`). This will not work, because the simulator is not compiled for MIPS. If you were instead to issue the command

```
cs350-gdb kernel
```

to try to debug the kernel, you would find that this also does not work, because the kernel has to be run on System/161.

Instead, what you need to do is start your kernel running in sys/161 and then run cs350-gdb and tell it to attach to sys161's debugger port. This is easiest to do using two windows, one to run sys/161 in and one to run GDB in. Note that you must be logged in to the **same machine** in both windows. If you are logged in to `mef-linux018` in one window and `mef-linux002` in another, this will not work. You can use the `hostname` command in a window to check which machine that window's commands are running on.

To debug, first start OS/161 on sys/161 in one of the windows.

```
sys161 -w kernel
```

Make sure that you are in the runtime root directory when you do this. The `-w` option tells sys/161 to wait for a debugger connection. You should see output that looks something like this:

```
@mef-linux018.student[107]% sys161 -w kernel
sys161: System/161 release 1.99.05, compiled Feb  4 2011 15:14:48
sys161: Waiting for debugger connection...
```

At this point, `sys161` is paused, waiting for a connection from the debugger.

Next, in the other window (your debug window), move into the runtime root directory - the same directory in which you are running `sys161` - and run `cs350-gdb` on the kernel and tell GDB to connect to sys/161:

```
cs350-gdb kernel
```

You should see some output like this from the debugger:

```
@mef-linux018.student[104]% cs350-gdb kernel
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=mips-harvard-os161"...
(gdb)
```

The `(gdb)` is the debugger's command prompt. The debugger is now stopped, waiting for you to enter a command. You want to give it the following commands:

```
(gdb> dir ../os161-1.99/kern/compile/ASSTX
(gdb) target remote unix:.sockets/gdb
```

The first command just tells `gdb` where to find the `.o` files for your kernel. The second command tells `gdb` to connect to the waiting `sys161` program that is running in your other window. (Don't forget the "`.`" before `sockets/gdb`.) You should see output something like this in the debugger window:

```
(gdb) dir ../os161-1.99/kern/compile/ASST0
Source directories searched: /u5/kmsalem/cs350-os161/root/../os161-1.99/kern/compile/ASST0:$cdir:$cwd
(gdb) target remote unix:.sockets/gdb
Remote debugging using unix:.sockets/gdb
__start () at ../../arch/mips/mips/start.S:24
24          addiu sp, sp, -20
Current language:  auto; currently asm
(gdb)
```

and you should see a message like `sys161: New debugger connection` in the `sys161` window.

At this point, your kernel is running under control of the debugger. Your kernel is waiting to start its execution (in the assembly language function `start.S`) as soon as the debugger allows it to proceed. You may now issue any `gdb` commands that you would like to, e.g., you may set breakpoints to force kernel execution to halt wherever you want. When you are ready to go, you can use `gdb`'s `continue` command to allow the kernel's execution to proceed:

```
(gdb) c
```

When you do this, the kernel should start running in the other window. Unless you've set breakpoints, the kernel should print its usual boot messages and eventually provide you with the kernel command prompt.

For more information about using GDB, see Debugging with GDB.

System Documentation

There are UNIX-style manual (man) pages for all of the OS/161 system calls, the OS/161 device drivers, the OS/161 user-level utility programs, the OS/161 test programs, and standard C library, which contains basic functions that can be used by user-level applications. All of these manual pages are in HTML format and are included as part of the OS/161 distribution.

Once OS/161 has been installed, you can view the OS/161 man pages by starting a WWW browser and using it to open the file `$OS161TOP/root/man/index.html`. This is a master index file that contains links to groups of individual man pages. These man pages are also available on-line in the course web space.

There is also documentation for the sys/161 simulator, available on line