Building a Kernel

CS 411

We're kernel hackers now. Building a kernel isn't for the faint of heart. **Pay attention to everything below and take copious notes when we discuss this material**, or you **will** find yourself starting over from the beginning with a new virtual machine image. Starting from a shell/terminal:

```
# The following "cd" simply ensures you're in your home directory.

cd

# Let's start off by configuring your global git tooling.  You'll only need
# to run the following commands once, unless your blow-up your VM and have
# to start over, or start working from another account.  Use identity
# values that make sense for you _and_ that others will understand.

git config --global user.name "Your Name"
git config --global user.email Your.Name@mail.goucher.edu
git config --global push.default simple

# Rather than using geany, you could use emacs, gedit, or vi.

git config --global core.editor geany

# Okay, it can be pretty annoying to have to enter your GitHub credentials
# every single time you interact with your remote, so let's get some
# help.  Run one of these two commands:

#  1) Cache credentials in memory for one hour (units are seconds).  Adjust
#  the timeout as you see fit.

git config --global credential.helper "cache --timeout=3600"

#  2) Store credentials unencrypted on disk permanently
#     Default storage file is ~/.git-credentials

git config --global credential.helper store

# If you're paranoid, to delete the credential cache before the timeout
# expires run

git credential-cache exit
```

```
# If you haven't already done so, go into Canvas and follow the link
# you'll find there to create your GitHub repository.
# Replace <your-repository> below with the URL of your repo, which will look
# something like:
#
#     https://github.com/GoucherCollegeCS411Spring2020/
#     project1syscall-tpkelliher.git
#
# Except, well, it won't be split across two lines.
#
# Okay, now we're ready to setup our Git repo and push the Linux kernel
# source code in linux-4.14.166 to remote on GitHub.

# Run these commands:

# Descend to the root of the kernel source tree.

cd linux-4.14.166

# Create an empty Git repo.

git init

# "." means the current directory, i.e., linux-4.14.166.  Add it to the set
# of staged files for the commit, coming next.  Add works recursively on
# directories, so this stages the entire kernel tree.
#
# This will take a bit to complete.  Hang on.

git add .

# Commit the staged files to the local repo.

git commit -m "First commit."

# Add a remote named origin for our GitHub repo.

git remote add origin <your-repository>

# Update the master branch at origin.  Sets this branch as the upstream, so
# that subsequent pushes can be run via just 'git push'.

git push -u origin master

# You can now check your GitHub account to view your repo there.

# Before building a kernel, let's wire-up the connections for the syscalls
```

```
# we'll be writing.  For now, we'll just put in stub functions for the
# syscall entry points.  Doing this now will shorten future kernel compile
# times.

# The following file paths are relative to the root of the kernel source
# tree.

# Open arch/x86/entry/syscalls/syscall_64.tbl for editing.  This file is
# the syscall table.  Find the line
#
#     332     common  statx                   sys_statx
#
# and add the following lines after it:

333     common  labsysc1                sys_labsysc1
334     common  labsysc2                sys_labsysc2
335     common  labsysc3                sys_labsysc3
336     common  labsysc4                sys_labsysc4

# Save and close the file.

# Open kernel/Makefile for editing.  This is the file that contains the
# "instructions" for compiling the kernel.  Find the line
#
#     async.o range.o smpboot.o ucount.o
#
# and change it to

async.o range.o smpboot.o ucount.o labsyscalls.o

# Our syscalls will go into labsyscalls.c.  This change will cause our
# syscall code to be compiled into the kernel.

# Save and close the file.

# Time to create the stub syscall entry points.  From the root of the
# kernel source tree, run these commands:

cd kernel
cp sys.c labsyscalls.c
cd ..

# Open kernel/labsyscalls.c for editing.  Find the line
#
#     #include <asm/unistd.h>
#
# and delete everything below it.  After this line, add the following
# lines:
```

```
#include <linux/semaphore.h>

/* The following are just stub system calls, present to allow the kernel to
 * compile, given that the names and entry points have been added to the
 * syscall table.  They will have to be changed before actually using them.
 */

/* syscall 333 */
SYSCALL_DEFINE1(labsysc1, const int, val)
{
  return -1;
}

/* syscall 334 */
SYSCALL_DEFINE1(labsysc2, const int, val)
{
  return -1;
}

/* syscall 335 */
SYSCALL_DEFINE1(labsysc3, const int, val)
{
  return -1;
}

/* syscall 336 */
SYSCALL_DEFINE1(labsysc4, const int, val)
{
  return -1;
}

# Save and close the file.

# Now, from the root of the kernel source tree run the command

git status

# Several file paths will be listed as not yet staged for commit, either
# because they've been modified or aren't currently being tracked.  Use
# 'git add' to stage all of them.  For example:

git add kernel/Makefile

# After adding all the files run

git status
```

```
again to make sure that all modified/untracked files are staged.  Then, run

git commit

# and add a meaningful commit message.  Then, run

git push

# This will push the changes to master at remote origin.
```

When it comes to git, commit early, commit often, and write meaningful commit messages. A single commit should encompass a single change, or a small set of related changes, with the commit message explaining why the change was made. See https://chris.beams.io/posts/git-commit/ for the philosophy and see https://github.com/torvalds/linux/commits/master for an example of the philosophy at work. You will be graded on this for this project and our other projects where we're using GitHub.

```
# Okay, now we're ready to compile our kernel.

# The character preceding the "=" below is the letter O, NOT the
# numeral 0.  This will keep the kernel object files out of the kernel
# source tree.  Why mix the two?  And, it's considered bad form to have
# binary code in a repo.
#
# In the menuconfig tool, run below in the next step, select 'General
# setup' and then select 'Local
# version'.  Change the local version string to something like -tpk00 .
# You can use the two digit number as a version number, incrementing it
# each time you add a new kernel feature.
# Exit the tool, saving the new .config file.

make O=/home/kdev/build menuconfig

# Build the kernel.  This will take a while the first time.  Subsequent
# builds will be faster.

make -j 3 O=/home/kdev/build

# The following command will install the kernel modules and the kernel.

sudo make O=/home/kdev/build modules_install install

# Reboot.  Your shiny new kernel should be available as one of the kernel
# choices in GRUB.  Are your brave enough to boot your shiny new kernel?

KABOOM!

# This isn't really necessary, since you can use sudo, but here's how you
# gain the full root environment from the kdev account:
```

```
su -

# Exit the root account, returning to the kdev account:

exit
```