Navid Malek
Project Phase 1

Operating Systems
Spring 2017

Adding simple system call to kernel from scratch

First create a directory named linuxSRC to download linux source, and Change directory to the folder:

console:

```
mkdir /linuxSRC

cd linuxSRC
```

after that you should first download the linux source, inorder to download every source on the package repository of ubuntu you should go to your package list ,You need to uncomment the deb-src lines in the main file /etc/apt/sources.list not the extra files in/etc/apt/sources.list.d:

open software & updates
go to ubuntu software tab

check source code

save and exit

after that you should get your current version linux kernel and build dependencies:

console:

```
mkdir newKernel

cd newKernel
```

now download the linux source and decompress it:

----

console:

sudo apt-get install linux source

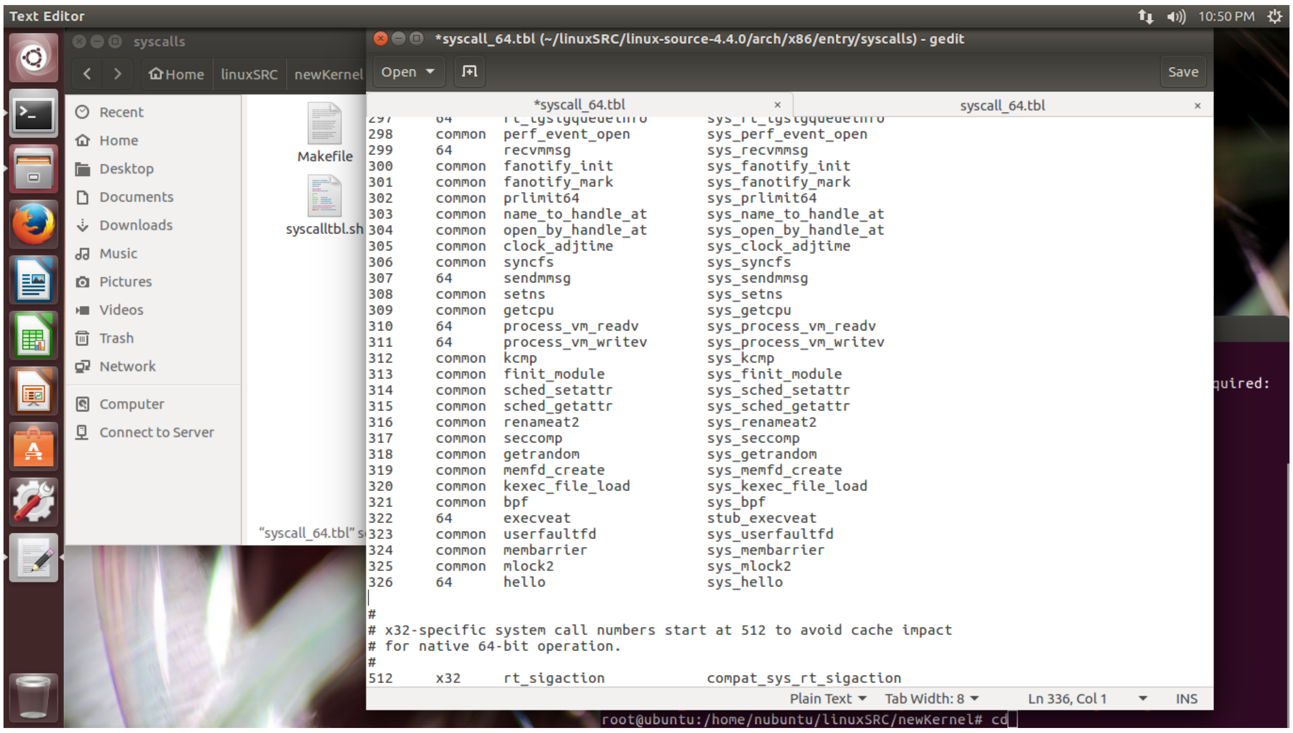
tar xjvf /usr/src/linux-source-<your version>tar.bz2

now go to the specified folder bellow :

/home/nubuntu/linuxSRC/linux-source-4.4.0/arch/x86/entry/syscalls

now open syscall_64.tbl or syscall_32.tbl (based on your system)(i use 64)

at the end add your hello syscall after the last number (here 333), with the following format, save and exit.



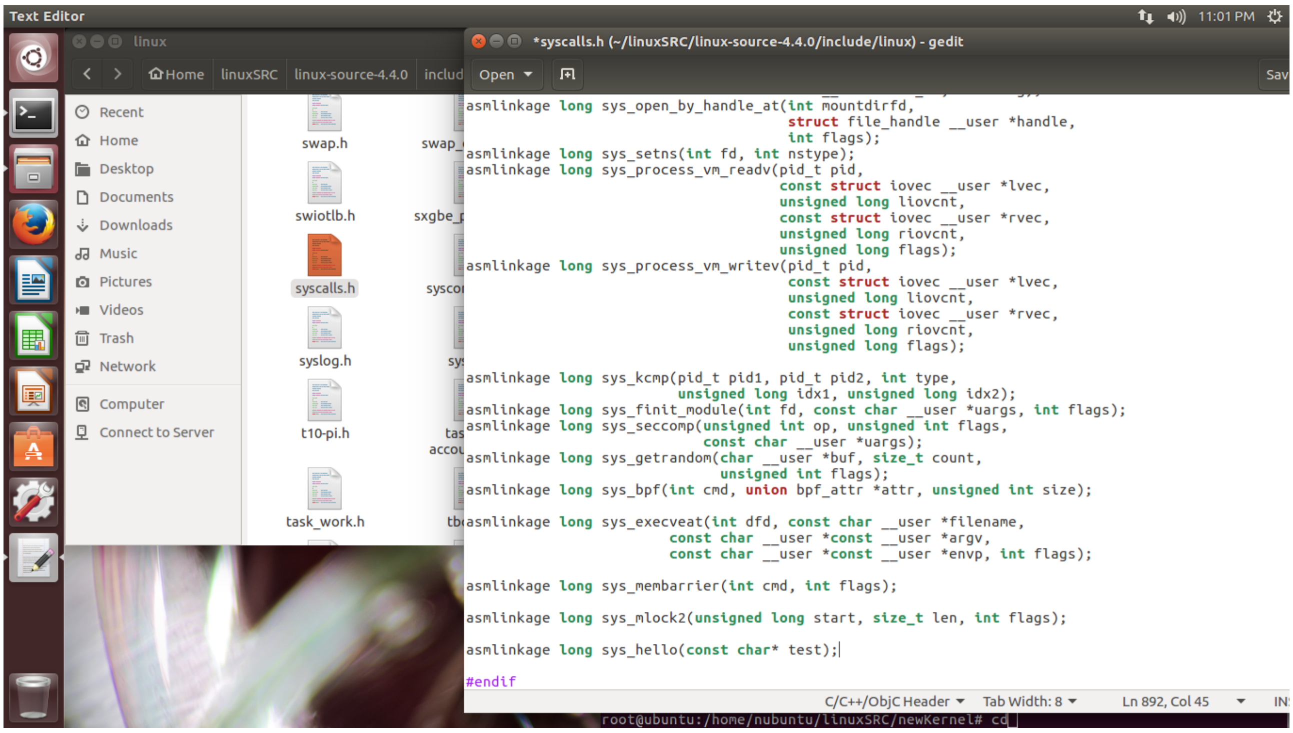
```
297     64      rt_tgsigqueueinfo       sys_rt_tgsigqueueinfo
298     common  perf_event_open         sys_perf_event_open
299     64      recvmmsg                sys_recvmmsg
300     common  fanotify_init           sys_fanotify_init
301     common  fanotify_mark           sys_fanotify_mark
302     common  prlimit64               sys_prlimit64
303     common  name_to_handle_at       sys_name_to_handle_at
304     common  open_by_handle_at       sys_open_by_handle_at
305     common  clock_adjtime           sys_clock_adjtime
306     common  syncfs                  sys_syncfs
307     64      sendmmsg                sys_sendmmsg
308     common  setns                   sys_setns
309     common  getcpu                  sys_getcpu
310     64      process_vm_readv        sys_process_vm_readv
311     64      process_vm_writev       sys_process_vm_writev
312     common  kcmp                    sys_kcmp
313     common  finit_module            sys_finit_module
314     common  sched_setattr           sys_sched_setattr
315     common  sched_getattr           sys_sched_getattr
316     common  renameat2               sys_renameat2
317     common  seccomp                 sys_seccomp
318     common  getrandom               sys_getrandom
319     common  memfd_create            sys_memfd_create
320     common  kexec_file_load         sys_kexec_file_load
321     common  bpf                     sys_bpf
322     64      execveat                stub_execveat
323     common  userfaultfd             sys_userfaultfd
324     common  membarrier              sys_membarrier
325     common  mlock2                  sys_mlock2
326     64      hello                   sys_hello

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
```

```
quired:
root@ubuntu:/home/nubuntu/linuxSRC/newKernel# cd
```

After that you should go to this directory :

/home/nubuntu/linuxSRC/linux-source-4.4.0/include/linux

add your asmlinkage for your hello system call before #endif at the bottom of file.



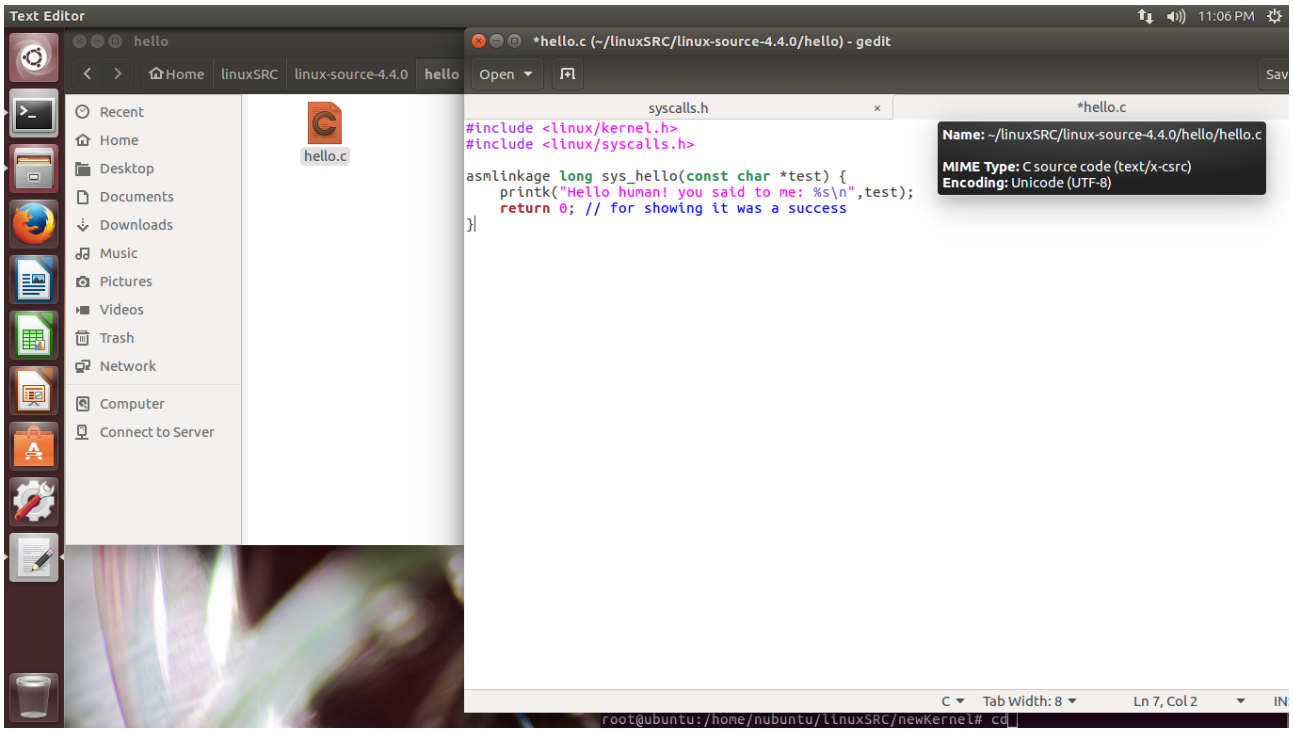
*syscalls.h (~/linuxSRC/linux-source-4.4.0/include/linux) - gedit

```
asmlinkage long sys_open_by_handle_at(int mountdirfd,
                                      struct file_handle __user *handle,
                                      int flags);
asmlinkage long sys_setns(int fd, int nstype);
asmlinkage long sys_process_vm_readv(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);
asmlinkage long sys_process_vm_writev(pid_t pid,
                                      const struct iovec __user *lvec,
                                      unsigned long liovcnt,
                                      const struct iovec __user *rvec,
                                      unsigned long riovcnt,
                                      unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_hello(const char* test);

#endif
```

C/C++/ObjC Header Tab Width: 8 Ln 892, Col 45

```
root@ubuntu:/home/nubuntu/linuxSRC/newKernel# cd
```

Next,you should create a new directory in the root of a clean copy of the kernel sources created for this tutorial. Call the new directory "hello". In this directory, we need to create two files. The first of which is the implementation of the system call itself:



*hello.c (~/linuxSRC/linux-source-4.4.0/hello) - gedit

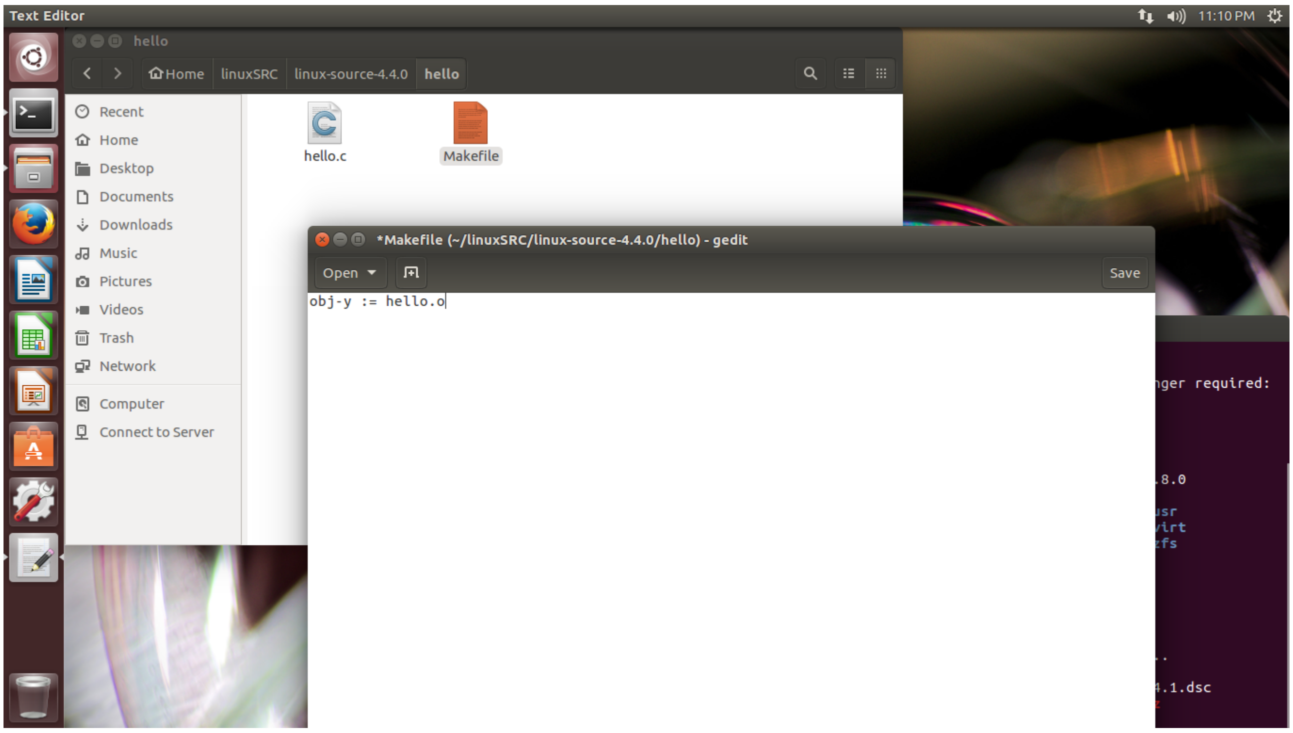
```
#include <linux/kernel.h>
#include <linux/syscalls.h>

asmlinkage long sys_hello(const char *test) {
    printk("Hello human! you said to me: %s\n",test);
    return 0; // for showing it was a success
}
```

Name: ~/linuxSRC/linux-source-4.4.0/hello/hello.c
MIME Type: C source code (text/x-csrc)
Encoding: Unicode (UTF-8)

C Tab Width: 8 Ln 7, Col 2

```
root@ubuntu:/home/nubuntu/linuxSRC/newKernel# cd
```

and beside your kernel module you should create a config file(called make file) for it in order to compile.



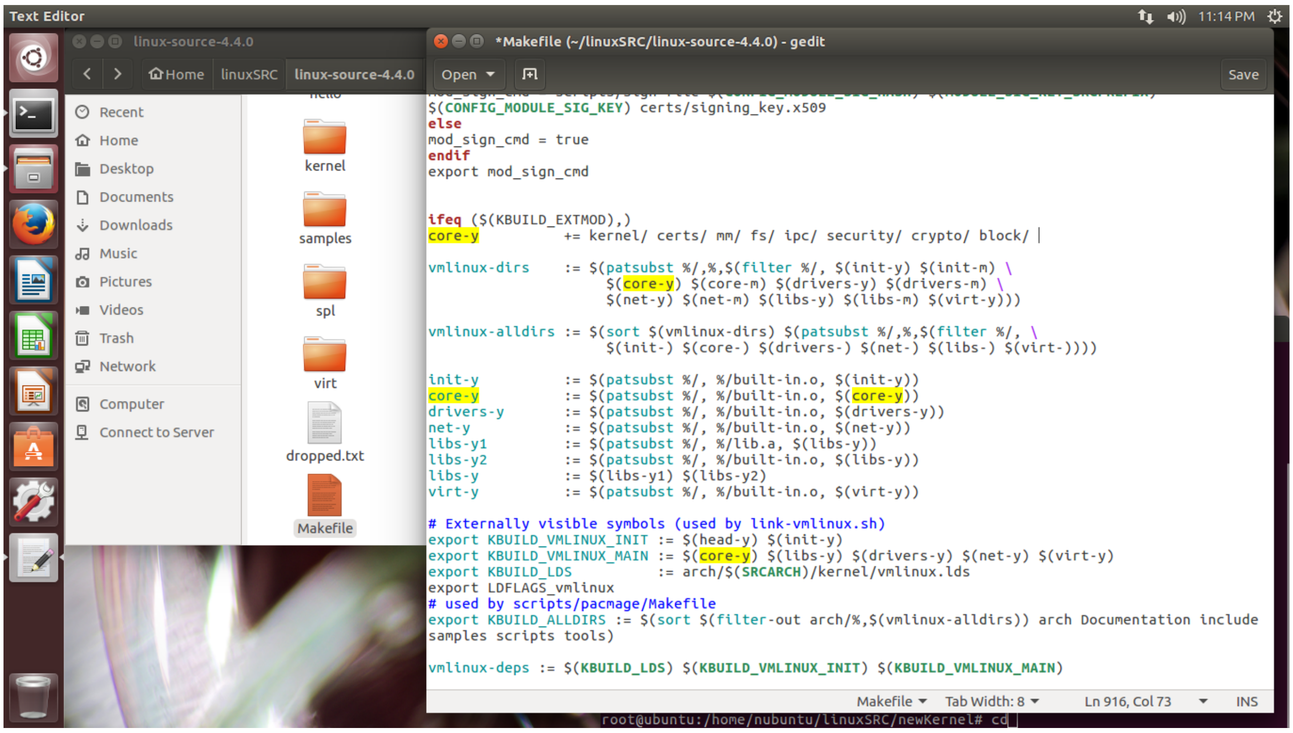
```
obj-y := hello.o
```

Now you should notify the kernel of your directory of syscall in order to compile the directory when kernel is compiled.

To do this, open up the root level Makefile. Look for the following section of the Makefile.

Navid Malek
Project Phase 1

Operating Systems
Spring 2017

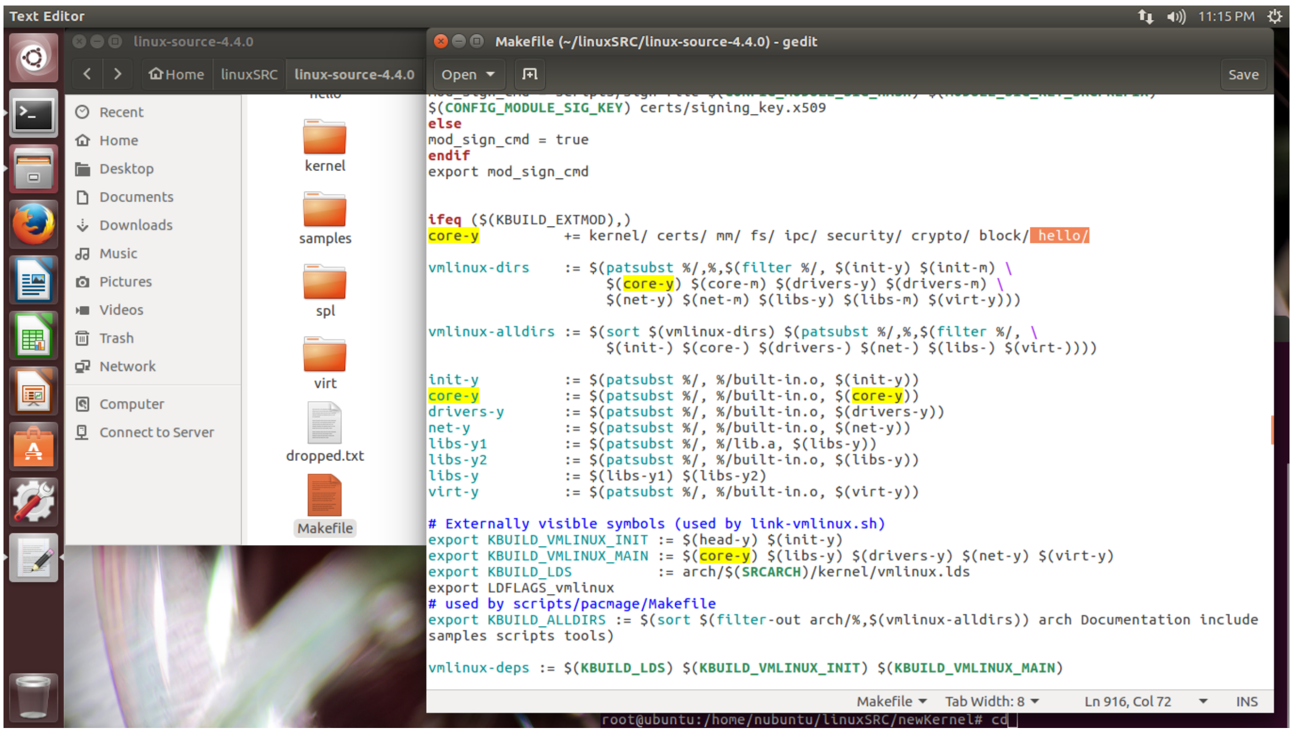


```
$(CONFIG_MODULE_SIG_KEY) certs/signing_key.x509
else
mod_sign_cmd = true
endif
export mod_sign_cmd


ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                     $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                     $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                     $(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))

init-y          := $(patsubst %/, %/built-in.o, $(init-y))
core-y          := $(patsubst %/, %/built-in.o, $(core-y))
drivers-y       := $(patsubst %/, %/built-in.o, $(drivers-y))
net-y           := $(patsubst %/, %/built-in.o, $(net-y))
libs-y1         := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2         := $(patsubst %/, %/built-in.o, $(libs-y))
libs-y          := $(libs-y1) $(libs-y2)
virt-y          := $(patsubst %/, %/built-in.o, $(virt-y))

# Externally visible symbols (used by link-vmlinux.sh)
export KBUILD_VMLINUX_INIT := $(head-y) $(init-y)
export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y) $(drivers-y) $(net-y) $(virt-y)
export KBUILD_LDS          := arch/$(SRCARCH)/kernel/vmlinux.lds
export LDFLAGS_vmlinux
# used by scripts/pacmage/Makefile
export KBUILD_ALLDIRS := $(sort $(filter-out arch/%,$(vmlinux-alldirs)) arch Documentation include
samples scripts tools)

vmlinux-deps := $(KBUILD_LDS) $(KBUILD_VMLINUX_INIT) $(KBUILD_VMLINUX_MAIN)
```

Makefile ▾   Tab Width: 8 ▾   Ln 916, Col 73 ▾   INS

```
root@ubuntu:/home/nubuntu/linuxSRC/newKernel# cd
```

You must modify the core-y line to add our new directory. Modify it to look like this:



```
$(CONFIG_MODULE_SIG_KEY) certs/signing_key.x509
else
mod_sign_cmd = true
endif
export mod_sign_cmd


ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                     $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                     $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                     $(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))

init-y          := $(patsubst %/, %/built-in.o, $(init-y))
core-y          := $(patsubst %/, %/built-in.o, $(core-y))
drivers-y       := $(patsubst %/, %/built-in.o, $(drivers-y))
net-y           := $(patsubst %/, %/built-in.o, $(net-y))
libs-y1         := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2         := $(patsubst %/, %/built-in.o, $(libs-y))
libs-y          := $(libs-y1) $(libs-y2)
virt-y          := $(patsubst %/, %/built-in.o, $(virt-y))

# Externally visible symbols (used by link-vmlinux.sh)
export KBUILD_VMLINUX_INIT := $(head-y) $(init-y)
export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y) $(drivers-y) $(net-y) $(virt-y)
export KBUILD_LDS          := arch/$(SRCARCH)/kernel/vmlinux.lds
export LDFLAGS_vmlinux
# used by scripts/pacmage/Makefile
export KBUILD_ALLDIRS := $(sort $(filter-out arch/%,$(vmlinux-alldirs)) arch Documentation include
samples scripts tools)

vmlinux-deps := $(KBUILD_LDS) $(KBUILD_VMLINUX_INIT) $(KBUILD_VMLINUX_MAIN)
```

Makefile ▾   Tab Width: 8 ▾   Ln 916, Col 72 ▾   INS

```
root@ubuntu:/home/nubuntu/linuxSRC/newKernel# cd
```

save and exit.

Now you should compile and install the kernel.

consider checking the above steps one more time!

Before building the kernel, you must configure it.

console:

```
cp -vi /boot/config-`uname -r` .config          //not suggested
sudo apt-get install libncurses5 libncurses5-dev //not suggested
make oldconfig
make localmodconfig
make menuconfig
```

after configuration you will finally start making the kernel(compiling kernel).
Commands(from wiki.ubuntu.com):

Now you can compile the kernel and create the packages:

```
make clean # only needed if you want to do a "clean" build

make deb-pkg
```

You can enable parallel make use `make -j`). Try 1+number of processor cores, e.g. 3 if you have a dual core processor:

```
make -j3 deb-pkg
```

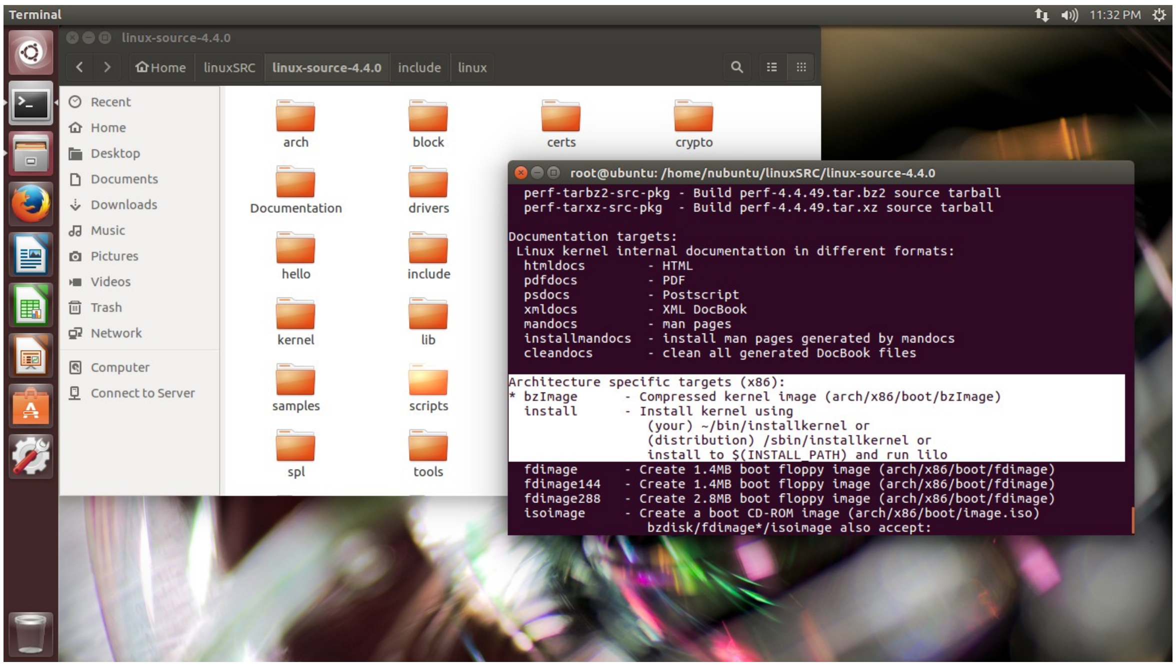
On a newer kernel, if you only need binary packages and want several builds (while editing the source) to not cause everything to be rebuilt, use:

```
make -j3 bindeb-pkg
```

if we use make help command you will see:

Navid Malek
Project Phase 1

Operating Systems
Spring 2017

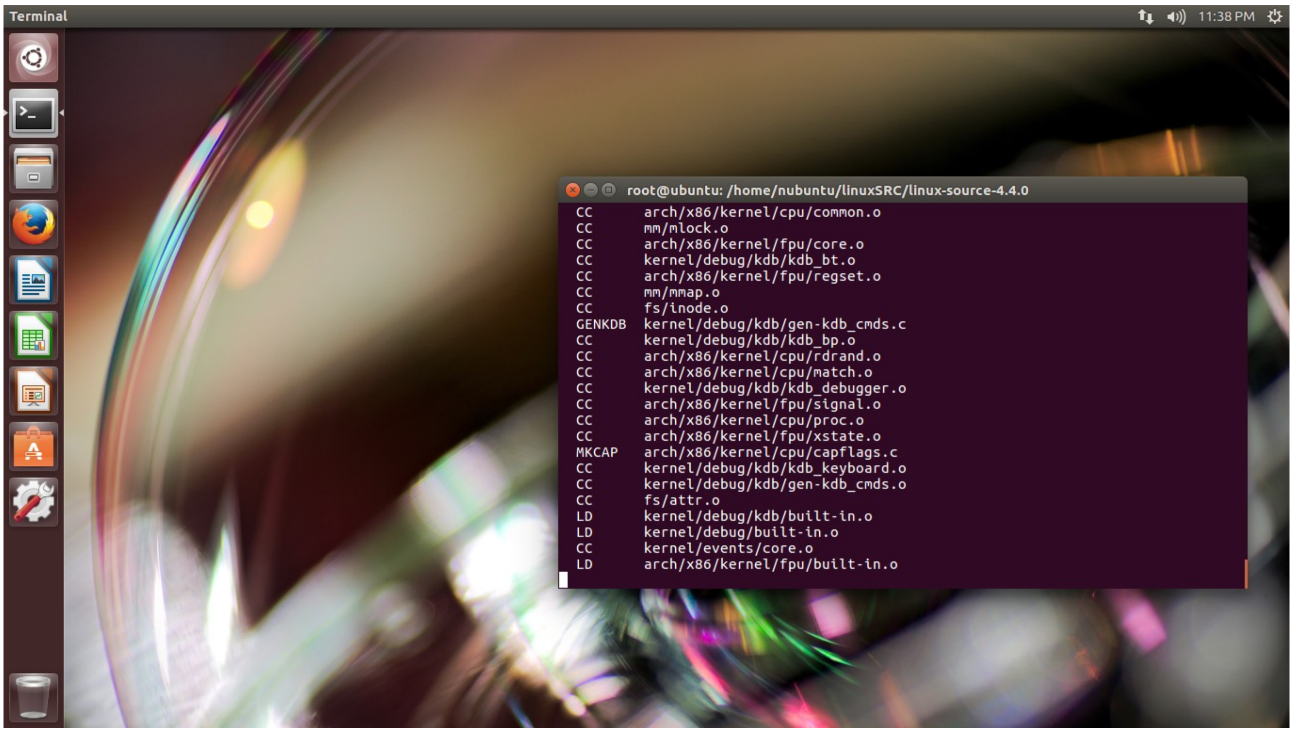


root@ubuntu: /home/nubuntu/linuxSRC/linux-source-4.4.0

```
  perf-tarbz2-src-pkg - Build perf-4.4.49.tar.bz2 source tarball
  perf-tarxz-src-pkg  - Build perf-4.4.49.tar.xz source tarball

Documentation targets:
 Linux kernel internal documentation in different formats:
  htmldocs        - HTML
  pdfdocs         - PDF
  psdocs          - Postscript
  xmldocs         - XML DocBook
  mandocs         - man pages
  installmandocs  - install man pages generated by mandocs
  cleandocs       - clean all generated DocBook files

Architecture specific targets (x86):
* bzImage      - Compressed kernel image (arch/x86/boot/bzImage)
  install      - Install kernel using
                  (your) ~/bin/installkernel or
                  (distribution) /sbin/installkernel or
                  install to $(INSTALL_PATH) and run lilo
  fdimage      - Create 1.4MB boot floppy image (arch/x86/boot/fdimage)
  fdimage144   - Create 1.4MB boot floppy image (arch/x86/boot/fdimage)
  fdimage288   - Create 2.8MB boot floppy image (arch/x86/boot/fdimage)
  isoimage     - Create a boot CD-ROM image (arch/x86/boot/image.iso)
                  bzdisk/fdimage*/isoimage also accept:
```

what is really important bzImage and installing it.
So we simply use only make or make all

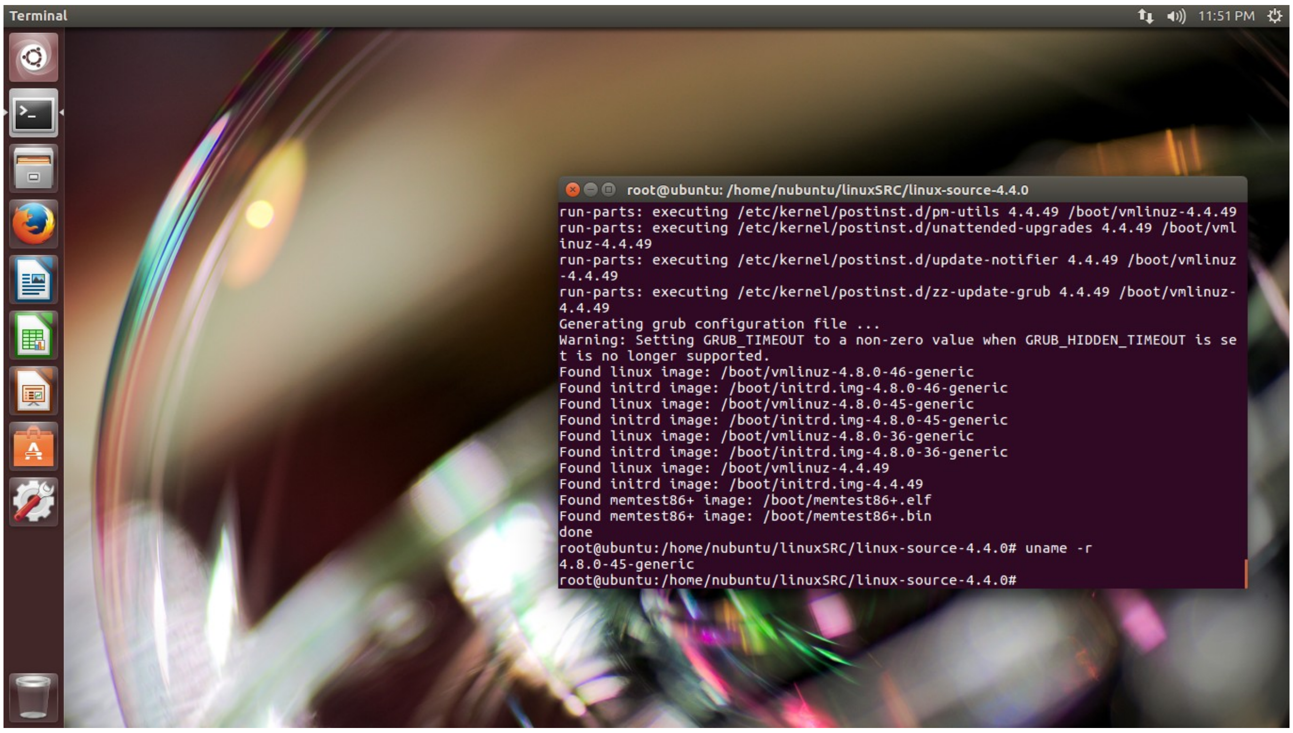
console:
make clean #only for the first time or every time the compilation gets interrupted
make -j5 all



```
  CC      arch/x86/kernel/cpu/common.o
  CC      mm/mlock.o
  CC      arch/x86/kernel/fpu/core.o
  CC      kernel/debug/kdb/kdb_bt.o
  CC      arch/x86/kernel/fpu/regset.o
  CC      mm/mmap.o
  CC      fs/inode.o
  GENKDB  kernel/debug/kdb/gen-kdb_cmds.c
  CC      kernel/debug/kdb/kdb_bp.o
  CC      arch/x86/kernel/cpu/rdrand.o
  CC      arch/x86/kernel/cpu/match.o
  CC      kernel/debug/kdb/kdb_debugger.o
  CC      arch/x86/kernel/fpu/signal.o
  CC      arch/x86/kernel/cpu/proc.o
  CC      arch/x86/kernel/fpu/xstate.o
  MKCAP   arch/x86/kernel/cpu/capflags.c
  CC      kernel/debug/kdb/kdb_keyboard.o
  CC      kernel/debug/kdb/gen-kdb_cmds.o
  CC      fs/attr.o
  LD      kernel/debug/kdb/built-in.o
  LD      kernel/debug/built-in.o
  CC      kernel/events/core.o
  LD      arch/x86/kernel/fpu/built-in.o
```

this gonna take a while…. :D

after make is done



```
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.4.49 /boot/vmlinuz-4.4.49
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.4.49 /boot/vml
inuz-4.4.49
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.4.49 /boot/vmlinuz
-4.4.49
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.4.49 /boot/vmlinuz-
4.4.49
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.8.0-46-generic
Found initrd image: /boot/initrd.img-4.8.0-46-generic
Found linux image: /boot/vmlinuz-4.8.0-45-generic
Found initrd image: /boot/initrd.img-4.8.0-45-generic
Found linux image: /boot/vmlinuz-4.8.0-36-generic
Found initrd image: /boot/initrd.img-4.8.0-36-generic
Found linux image: /boot/vmlinuz-4.4.49
Found initrd image: /boot/initrd.img-4.4.49
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@ubuntu:/home/nubuntu/linuxSRC/linux-source-4.4.0# uname -r
4.8.0-45-generic
root@ubuntu:/home/nubuntu/linuxSRC/linux-source-4.4.0#
```

current kernel version is shown(4.8.0-45), now install kernel and reboot.

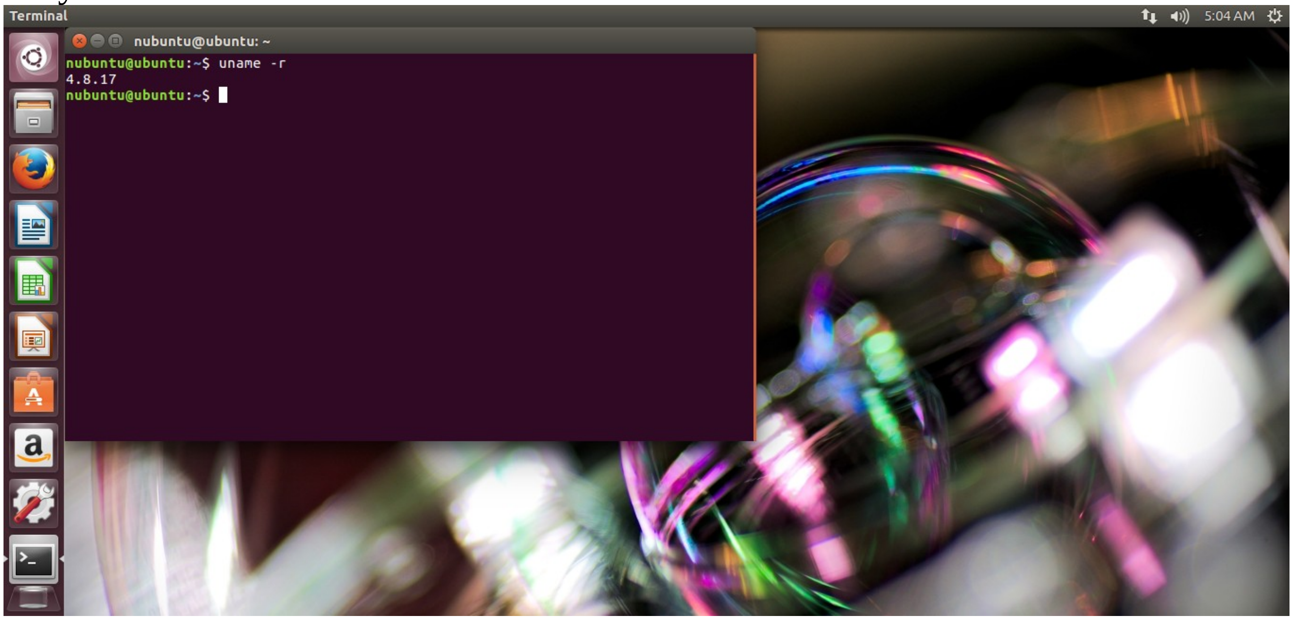
Navid Malek
Project Phase 1

Operating Systems
Spring 2017

Console:

make modules
make modules_install
make install
reboot #in order to boot new kernel

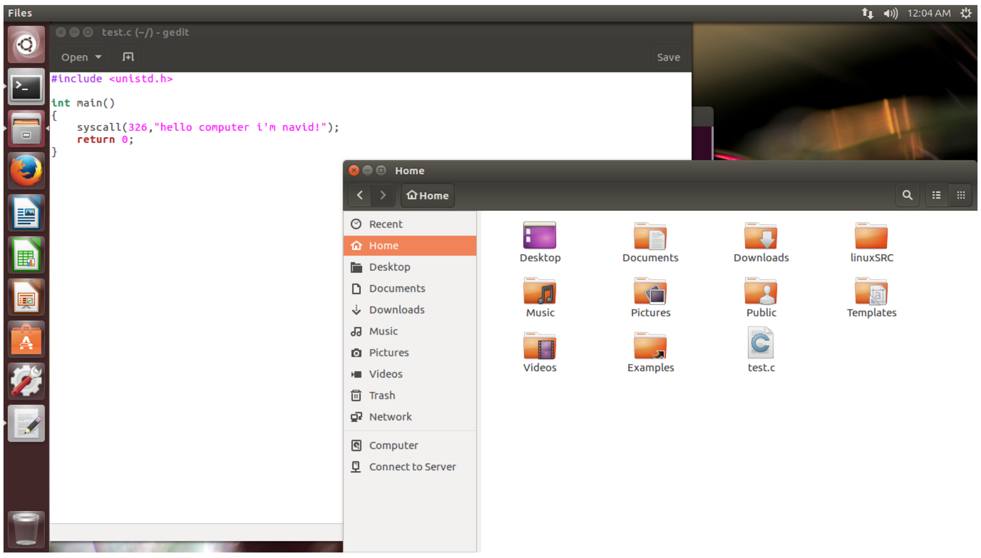
okay now the new kernel:



```
nubuntu@ubuntu:~$ uname -r
4.8.17
nubuntu@ubuntu:~$ 
```

welcome to the new kernel! It is 4.8.17

okay now we have to Implement the user space program to see our systemcall
create c file like this:



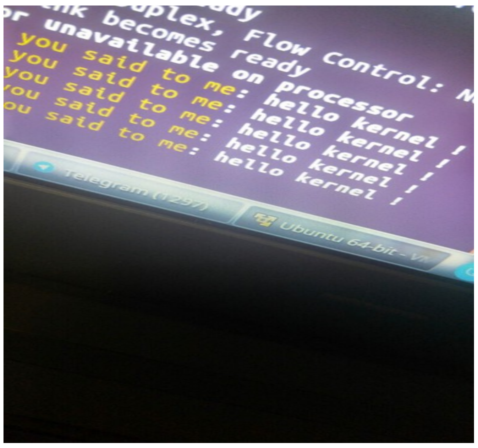
```c
#include <unistd.h>

int main()
{
    syscall(326,"hello computer i'm navid!");
    return 0;
}
```

Navid Malek
Project Phase 1

Operating Systems
Spring 2017
----

okay after creating c , we compile it with gcc and run it.
Console:
gcc -o test test.c
./test
dmesge

output:
this is the picture of my screen ,first time I’ve added hello world :D nostalgic!



```
uplex, Flow Control: N
nk becomes ready
r unavailable on processor
you said to me: hello kernel !
you said to me: hello kernel !
you said to me: hello kernel !
ou said to me: hello kernel !
```

!!note!!
this is my implemention , on different systems it can be a little more different.

Important noteee !!!
if you have done all parts and you can’t get a resuault try downloading the source with this command:
```
sudo apt-get source linux-image-`uname -r`
```

then if it’s locked (can’t write into it) use:
```
sudo chmod 777 -R /ADRESSOFFOLDER
```

now its editable!
then do the rest of parts :D