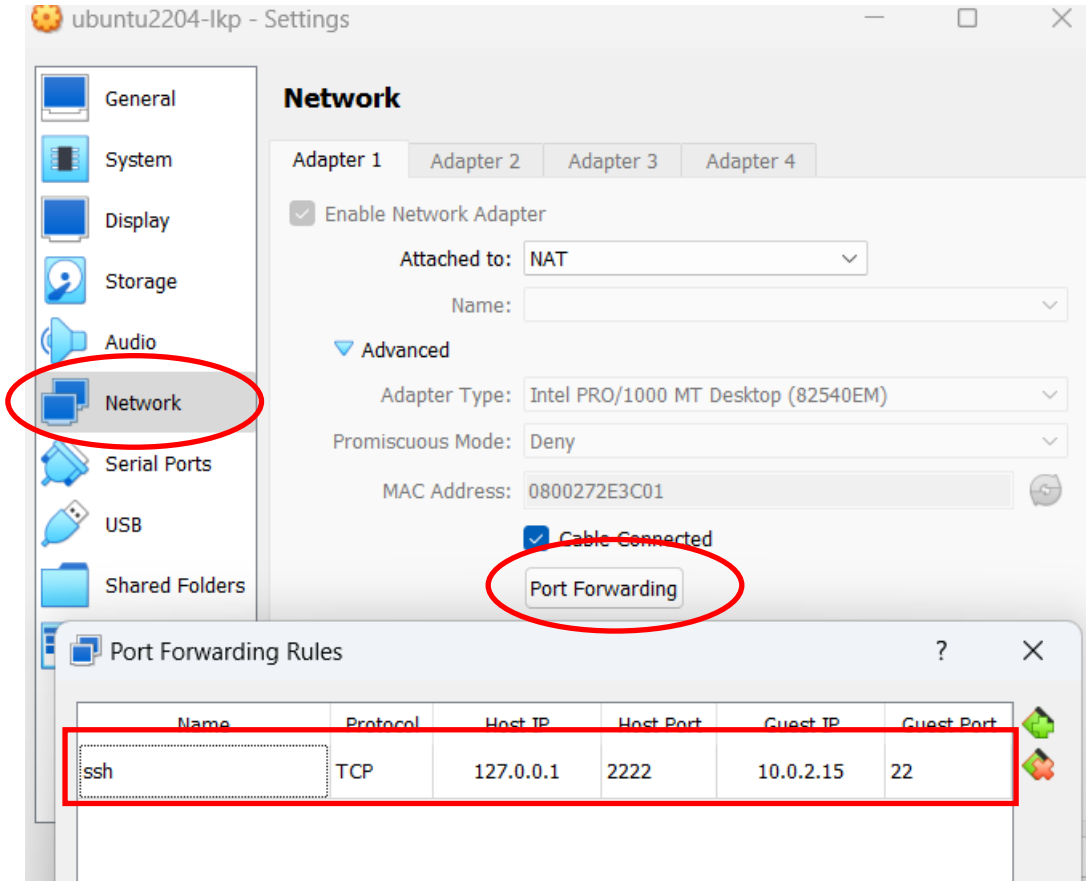
Building and exploring Linux kernel

Xiaoguang Wang

UIC COMPUTER SCIENCE

Have you successfully installed Linux?



Have successfully installed Linux?

```
PS C:\Users\xgwan> ssh xiaoguang@127.0.0.1 -p 2222
xiaoguang@127.0.0.1's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-91-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

```
xiaoguang@lkp:~$ sudo usermod -aG sudo xiaoguang
[sudo] password for xiaoguang:
xiaoguang@lkp:~$ sudo visudo
xiaoguang@lkp:~$ sudo apt update
Hit:1 http://us.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
```

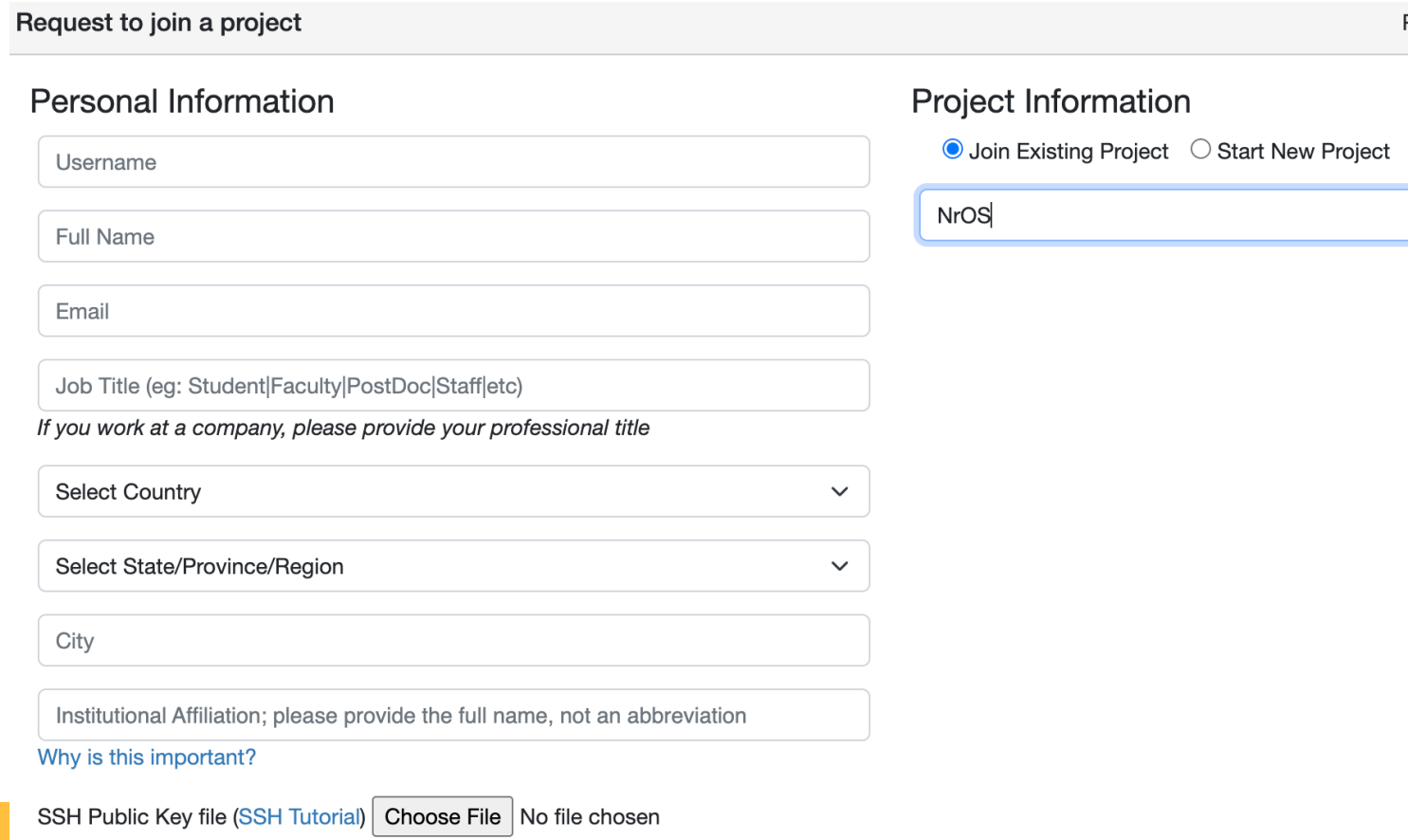
```
ssh [your_username]@localhost -p 2222
```

```
scp -P 2222 src_file [your_username]@localhost:[target_path]
```

CloudLab

If you don't have a well-qualified machine (e.g., ARM CPUs or limited disk space), sign up CloudLab:

- https://cloudlab.us/signup.php
- Join Existing Project: **NrOS**



Today's lecture

Tools

- Version control: `git`, `tig`
- Configure, build, and install the kernel: `make`
- Explore the code: `cscope`, `ctags`
- Editor: `vim`, `emacs`
- Screen: `tmux`

Kernel v.s. user programming

Why software tools are important?

Linux source code is huge and evolves very fast

- 27 million lines of code (LoC) ← 1,600 developers / release

```
✔ ~/workspace/research/linux [v5.19]
$ tree .
├── arch
│   ├── alpha
│   │   ├── boot
│   │   │   ├── bootloader.lds
│   │   │   ├── bootp.c
│   ...
├── lib
│   ├── irqbypass.c
│   ├── Kconfig
│   └── Makefile
└── Makefile

5003 directories, 76650 files
```

Obtaining the kernel source code

Tar ball

- https://kernel.org/

Linus's git repository

- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

GitHub mirror of Linus's git repository

- https://github.com/torvalds/linux

Let's explore above web sites!

Version control: git

Git is a version control software

- tracking changes in computer files

Initially developed by Linus Torvalds for development of the Linux kernel

- Extensively using in many other software development
- GitHub https://github.com/ is a git service provider

Distributed revision control system

- Every git directory on every computer is a full-fledged repository with complete history

Essential git commands

```
$ # 1. install and configure
$ sudo apt-get install git
$ git config --global user.name "John Doe"       # set your name/email for history
$ git config --global user.email johndoe@example.com

$ # 2. create a repository
$ git init                    # create a new local repo
$ git clone https://github.com/torvalds/linux.git      # clone an existing repo

$ # 3. tags
$ git tag                     # list all existing tags
$ git checkout v6.1           # checkout the tagged version
```

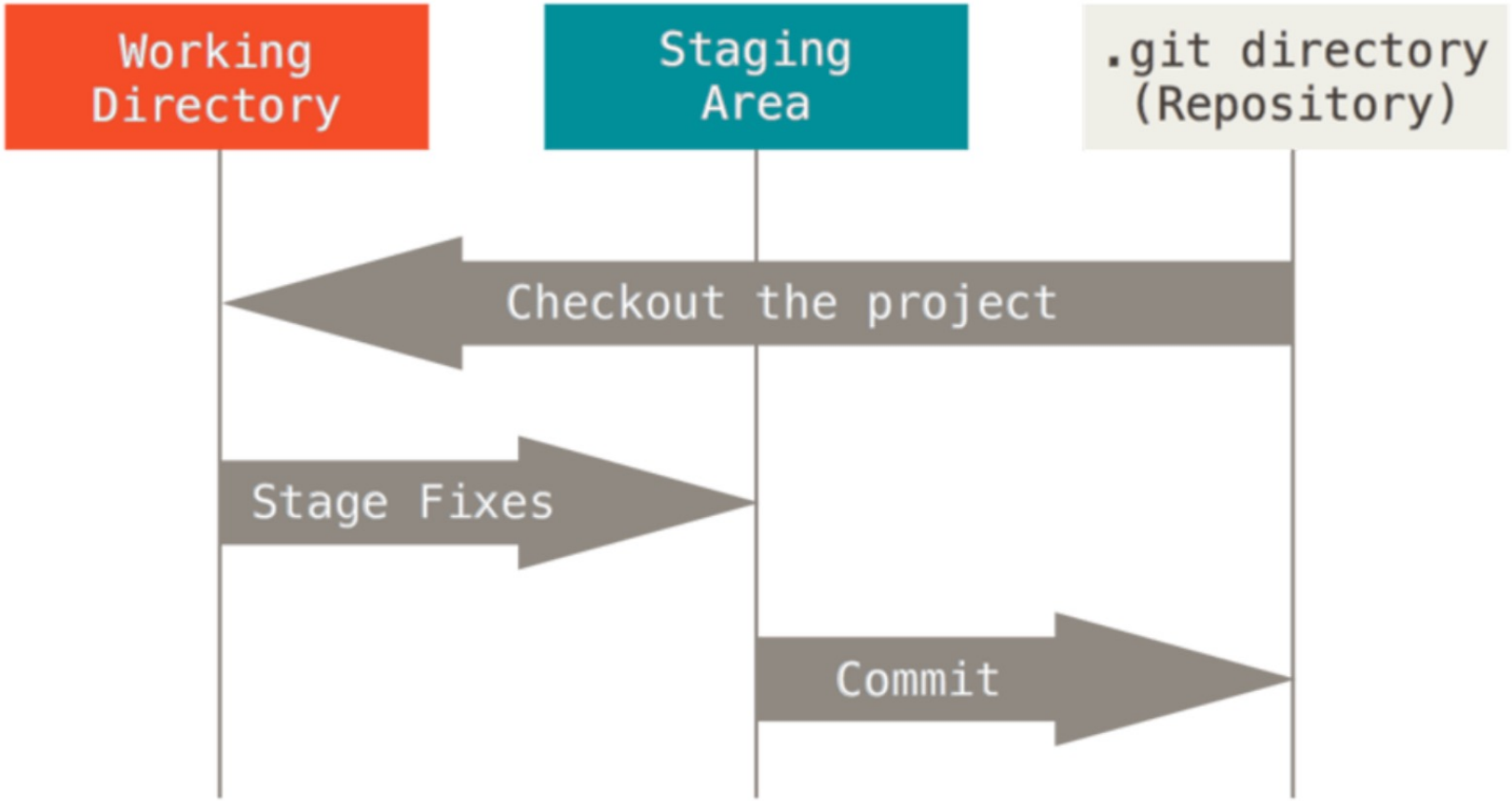
Essential git commands

```
$ # 4. commit history (or use tig for prettier output)
$ git log           # show all commit history
$ git log <file>    # show changes over time for a file
$ git blame <file>  # who changed what and when in <file>

$ # 5. local changes
$ git status        # show changed files
$ git diff          # show changed lines
$ git add <file>    # add <file> to the next commit
$ git commit        # commit previously staged files to my local repo
```

Essential git commands

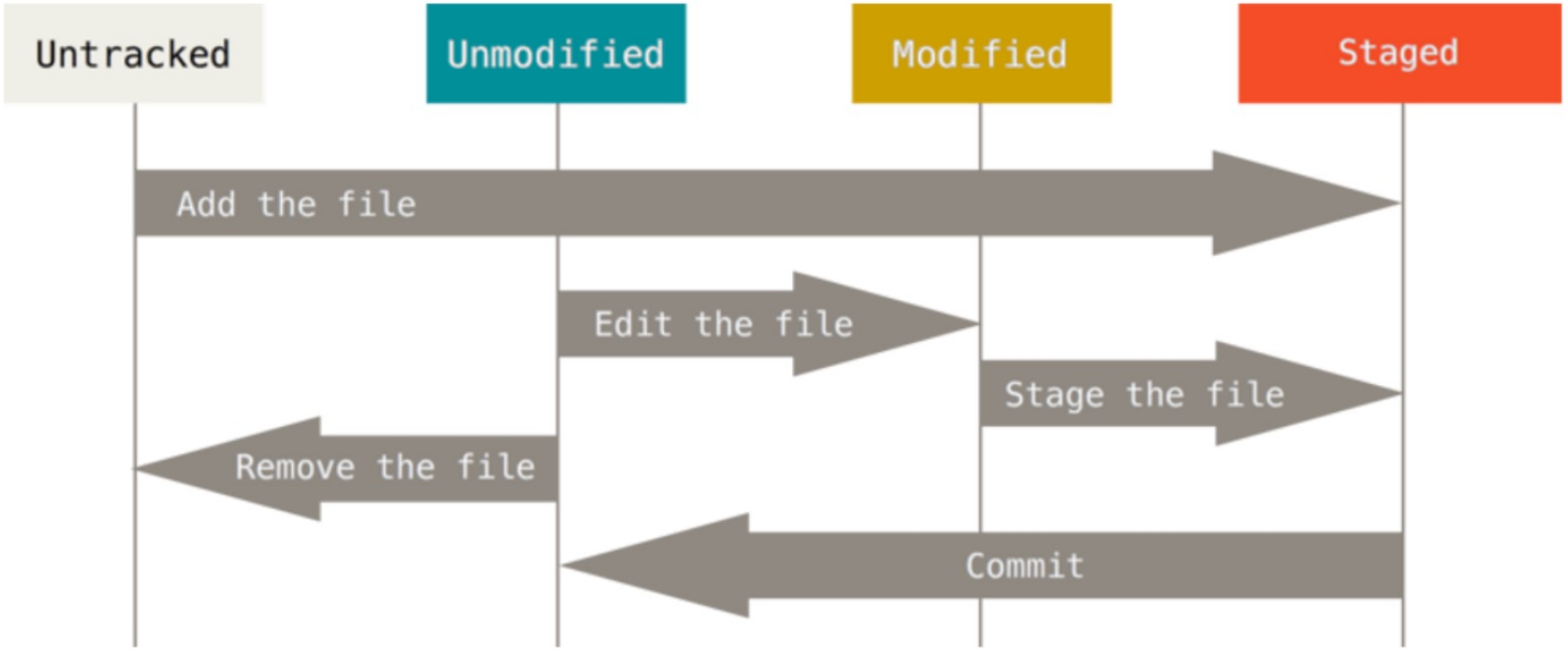
```
$ # 6. publish and update
$ git push          # publish a committed local changes to a remote repo
$ git pull          # update a local repo

$ # 7. other git tricks
$ cat ~/.gitconfig
[alias]
      br = branch
      co = checkout
      st = status
      lg = log --graph
      lp = log --graph --pretty=oneline
```

git workflow



Pro Git: https://git-scm.com/book/en/v2

git workflow



Pro Git: https://git-scm.com/book/en/v2

The kernel source tree

```
$ git clone https://github.com/torvalds/linux.git # clone the kernel repo
$ cd linux; git checkout v6.1 # checkout v6.1
$
$ tree -d -L 2 # list top two-level directories
├── arch # * architecture dependent code
│   ├── arm # - ARM architecture
│   └── x86 # - Intel/AMD x86 architecture
├── block
├── Documentation
├── drivers
│   ├── accessibility
│   ├── acpi
… …
```

The kernel source tree

```
… …
├── fs
│   ├── 9p
├── include
├── init
├── kernel
│   ├── bpf
├── mm
├── net
└── virt
    ├── kvm
    └── lib
633 directories
```

Build the kernel

Step 1. Configuring the kernel

- Configuration file defining compilation options (~ 3700 for x86)

Step 2. Compiling the kernel

- Compile and link the kernel source code

Step 3. Installing the new kernel

- Install compiled new kernel image to a system

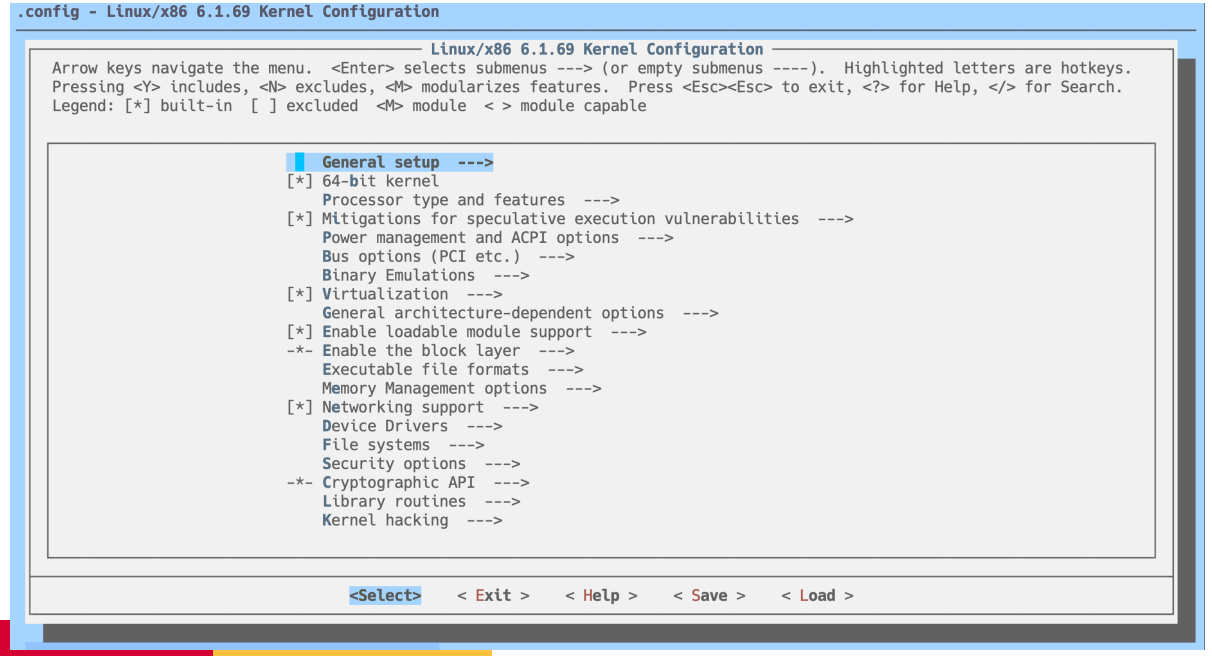
`make help` to see other make options

Ref: Documentation/admin-guide/README.rst

Configure the kernel

`make menuconfig`

- Need `libncurses`, `flex` and `bison`
  - `sudo apt install -y flex bison libncurses5-dev` # Debian/Ubuntu



Configure the kernel

`make defconfig`

- Generate the default .config of the running platform
- `linux/arch/x86/configs/x86_64_defconfig`

`make oldconfig`

- Use the configuration file of the running kernel
- Will ask about new configurations
  - If you are not sure, choose default options

`make localmodconfig`

- Update current .config disabling modules not loaded

Kernel configuration file: .config

.config file is at the root of the kernel source

preprocessor flags in the source code

```
$ head .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 6.1.0 Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=110400
CONFIG_CLANG_VERSION=0
```

Compile the kernel

Step 1. Compile the kernel: `make`

- Compile the kernel source code
- Compile kernel image: `linux/arch/x86/boot/bzImage`

Step 2. Compile modules: `make modules`

Parallel make

- make <target> -j<number of CPUs>
- e.g., `make -j16`

Install the new kernel

```
$ sudo make modules_install        # install the new kernel modules
$ ls /lib/modules

$ sudo make install                # install the new kernel image
$ sudo reboot

$ uname -a                          # new kernel version
$ dmesg                             # kernel log
```

Test your new kernel in the VM first!

Alternative way to build/install the kernel

A more portable way to build a Linux kernel

Generate .deb or .rpm files

- `make deb-pkg` # for Debian/ubuntu
- `make rpm-pkg` # for Redhat

```
sudo dpkg -i linux-image-6.1_amd64.deb linux-headers-
6.1_amd64.deb
```

Alternative way to build/install the kernel

More details:

- Build Debian kernel: https://wiki.debian.org/BuildADebianKernelPackage
- Build Ubuntu kernel: https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel

Building the kernel

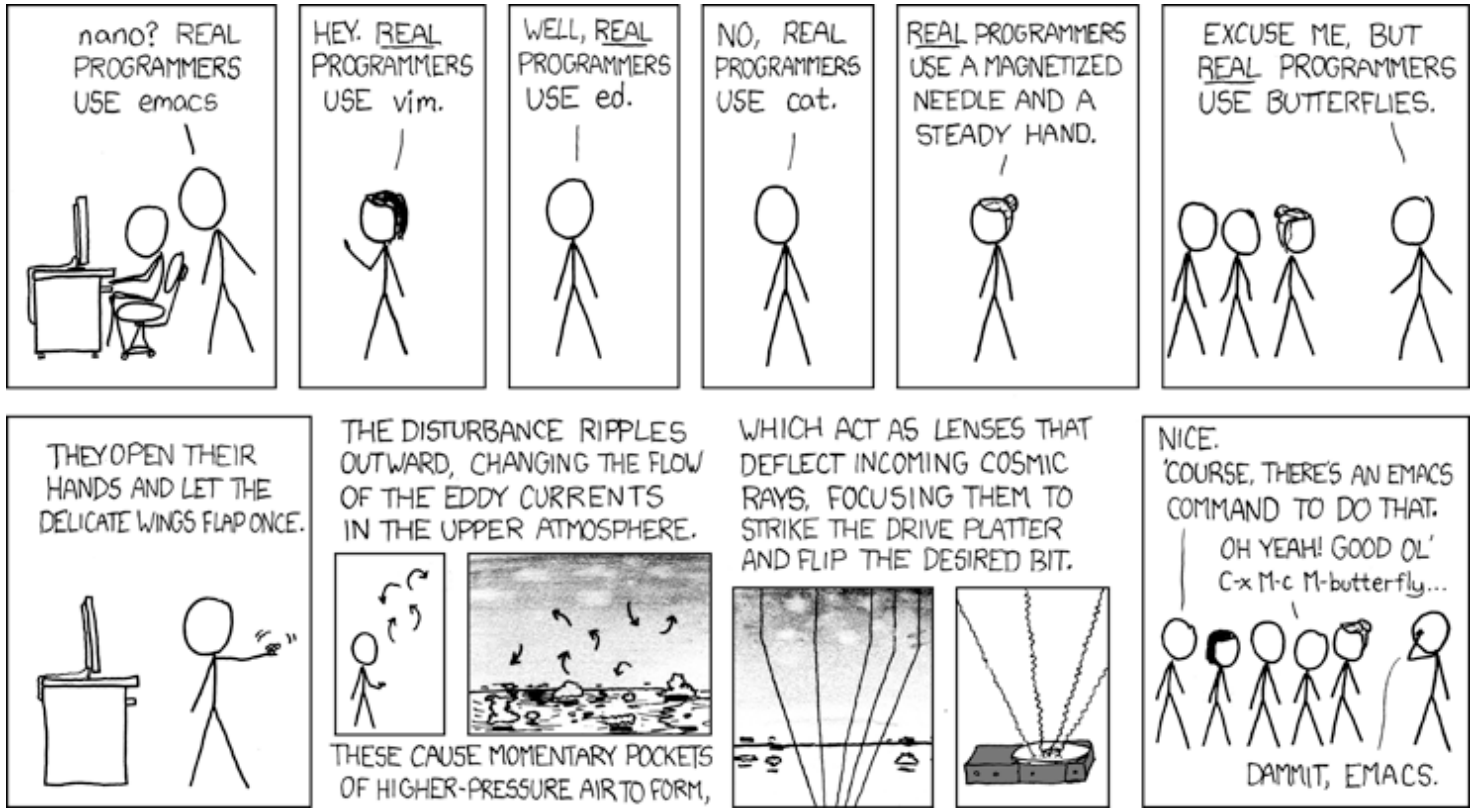
Building the kernel is quite easy. Change your working directory to the root of the kernel source tree and the

```
LANG=C fakeroot debian/rules clean
# quicker build:
LANG=C fakeroot debian/rules binary-headers binary-generic binary-perarch
# if you need linux-tools or lowlatency kernel, run instead:
LANG=C fakeroot debian/rules binary
```

Editor

There are many good editors

- `vim`, `emacs`, `nano`
- choose your preference



https://xkcd.com/378/

Exploring the code

- Linux Cross Reference (LXR)
- `cscope`
- `vim` with `cscope` or `ctags`
- `emacs` with `cscope`
- …

Linux Cross Reference (LXR)

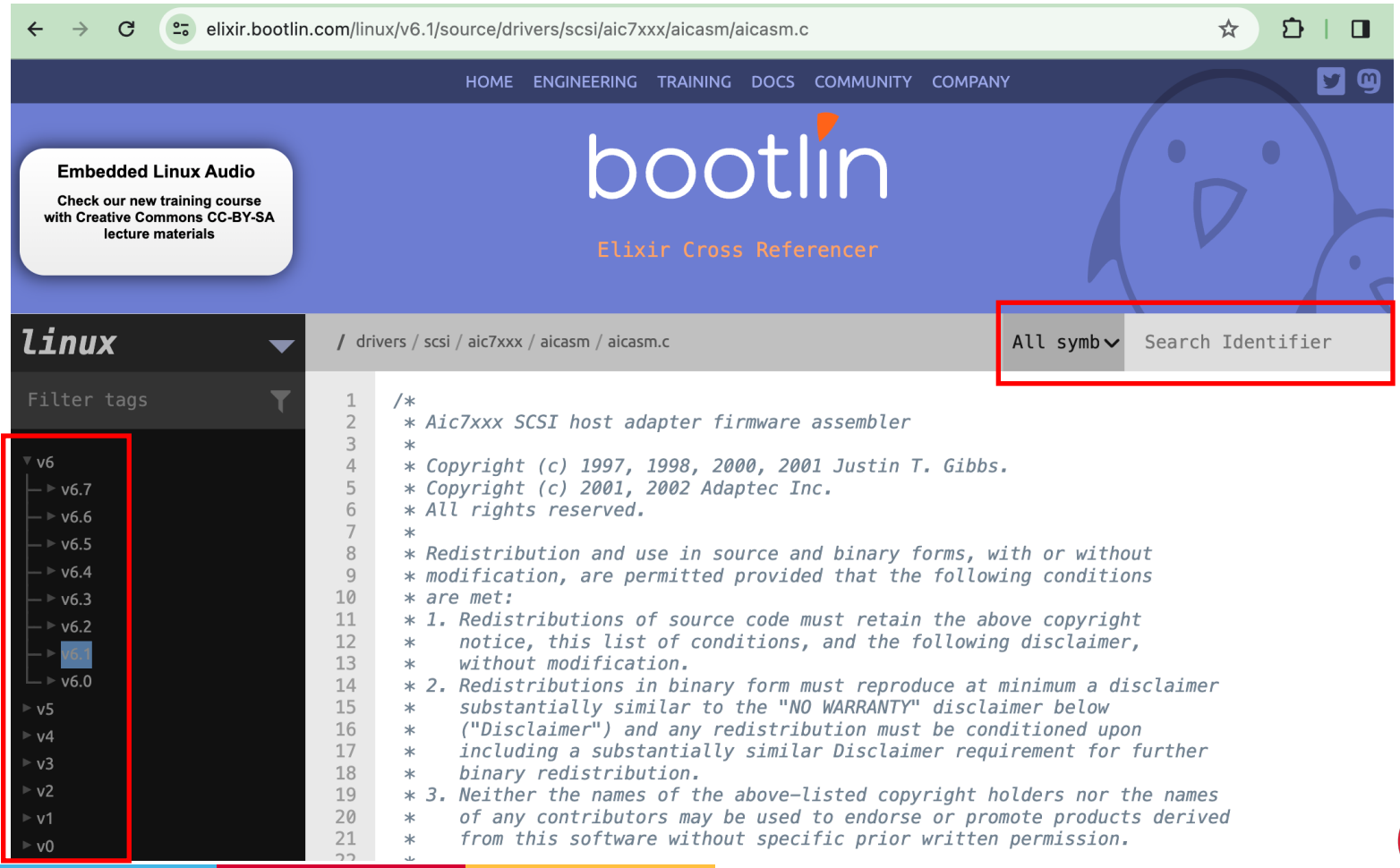
Code indexing tool with a web interface

- No need to install anything
- https://elixir.bootlin.com/linux/latest/source

Allow to:

- Browse the code of different Linux versions
- Search for identifiers (functions, variables, etc.)
- Cross-ref definitions/references for identifiers

Linux Cross Reference (LXR)



Choose a kernel version

Search an identifier

Linux Cross Reference (LXR)

Defined in 1 files as a prototype:

include/linux/syscalls.h, line 499 *(as a prototype)*

Defined in 3 files as a function:

fs/read_write.c, line 621 *(as a function)*
tools/include/nolibc/sys.h, line 871 *(as a function)*
tools/testing/selftests/proc/proc-self-syscall.c, line 25 *(as a function)*

Show the definition

Referenced in 5 files:

arch/arm64/include/asm/unistd32.h, line 20
include/uapi/asm-generic/unistd.h, line 206
tools/include/nolibc/sys.h, line 879
tools/include/uapi/asm-generic/unistd.h, line 206
tools/testing/selftests/proc/proc-self-syscall.c, line 49

Show the references

Linux Cross Reference (LXR)

```
620
621  SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
622  {
623          return ksys_read(fd, buf, count);
624  }
625
626  ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t count)
627  {
628          struct fd f = fdget_pos(fd);
629          ssize_t ret = -EBADF;
630
631          if (f.file) {
632                  loff_t pos, *ppos = file_ppos(f.file);
633                  if (ppos) {
634                          pos = *ppos;
635                          ppos = &pos;
636                  }
637                  ret = vfs_write(f.file, buf, count, ppos);
638                  if (ret >= 0 && ppos)
639                          f.file->f_pos = pos;
640                  fdput_pos(f);
641          }
642
643          return ret;
644  }
```

Click on it!

Linux Cross Reference (LXR)

```
621  SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
622  {
623          return ksys_read(fd, buf, count);
624  }
625
626  ssize_t                                                    e_t count)
627  {
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644  }
```

Defined in 1 files as a prototype:

include/linux/syscalls.h, line 1289 *(as a prototype)*

Defined in 1 files as a function:

fs/read_write.c, line 602 *(as a function)*

Referenced in 2 files:

arch/s390/kernel/compat_linux.c, line 230
fs/read_write.c, line 623

cscope: browse C code

Installation: *sudo apt install cscope*

Build `cscope` database

- `cscope -R` # most common way for C code
- `cd linux; make cscope` # for all architectures
- `cd linux; ARCH=x86 make cscope` # only for x86
- Need to rebuild after code changes

Although `cscope -R` is the common way to build `cscope` database, `make cscope` is optimized for the kernel source code

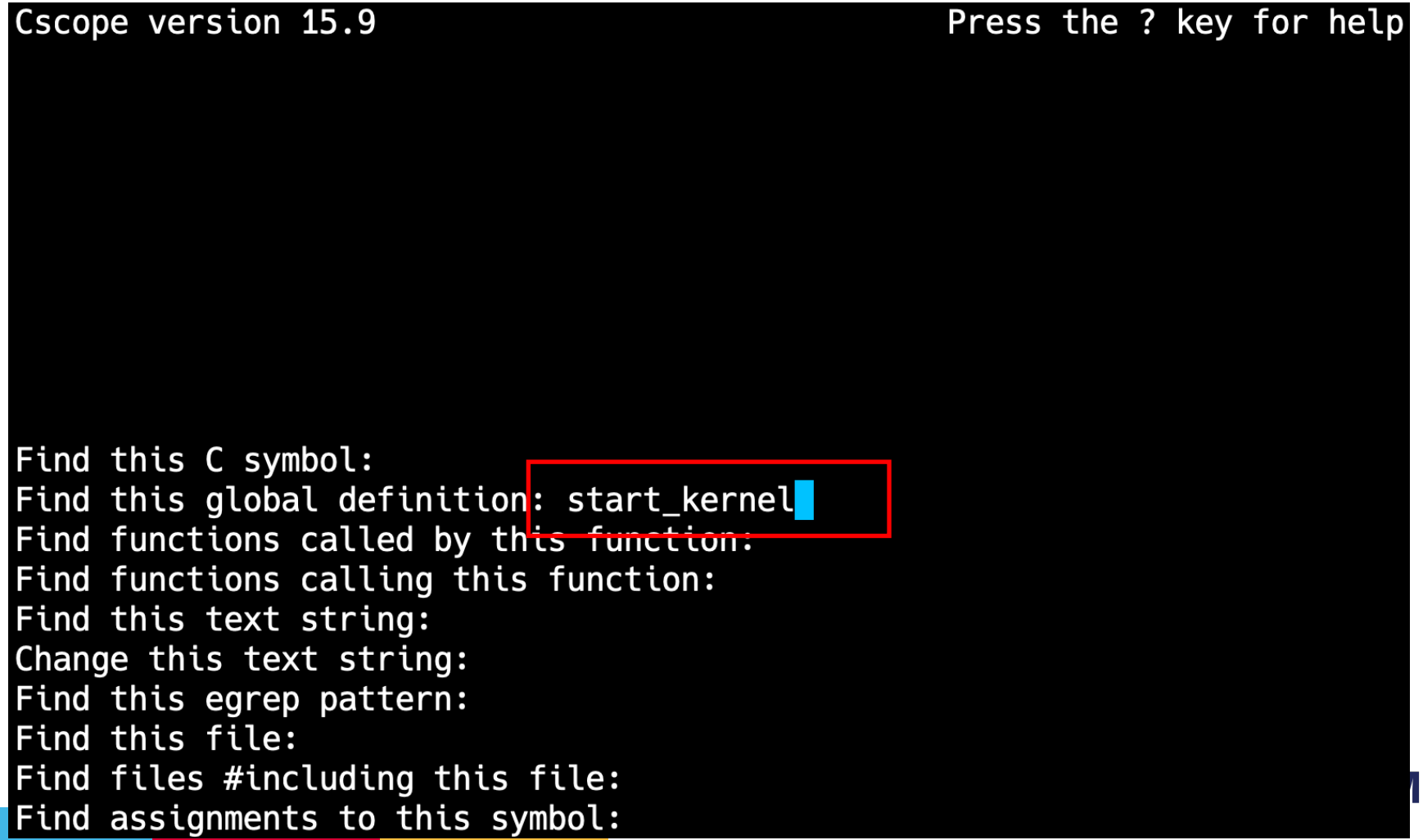
cscope

Search for:

- C identifiers
- Function/variables definitions
- Functions called by/calling function f
- Text strings

Terminating cscope: ctrl-d

cscope


Cscope version 15.9
Press the ? key for help
Find this C symbol:
Find this global definition: start_kernel
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:

cscope

```
933         memblock_free(unknown_options, len);
934 }
935
936 asmlinkage __visible void __init __no_sanitize_address start_kernel(void)
937 {
938         char *command_line;
939         char *after_dashes;
940
941         set_task_stack_end_magic(&init_task);
942         smp_setup_processor_id();
943         debug_objects_early_init();
944         init_vmlinux_build_id();
```

Cursor on the symbol and press `Ctrl-]` or `Ctrl-t` to navigate

To navigate back and forth between files: `:bp` or `:bn`

vim with cscope or ctags

`vim` can use tag database of `cscope`, and `ctags`

- `sudo apt install cscope exuberant-ctags`

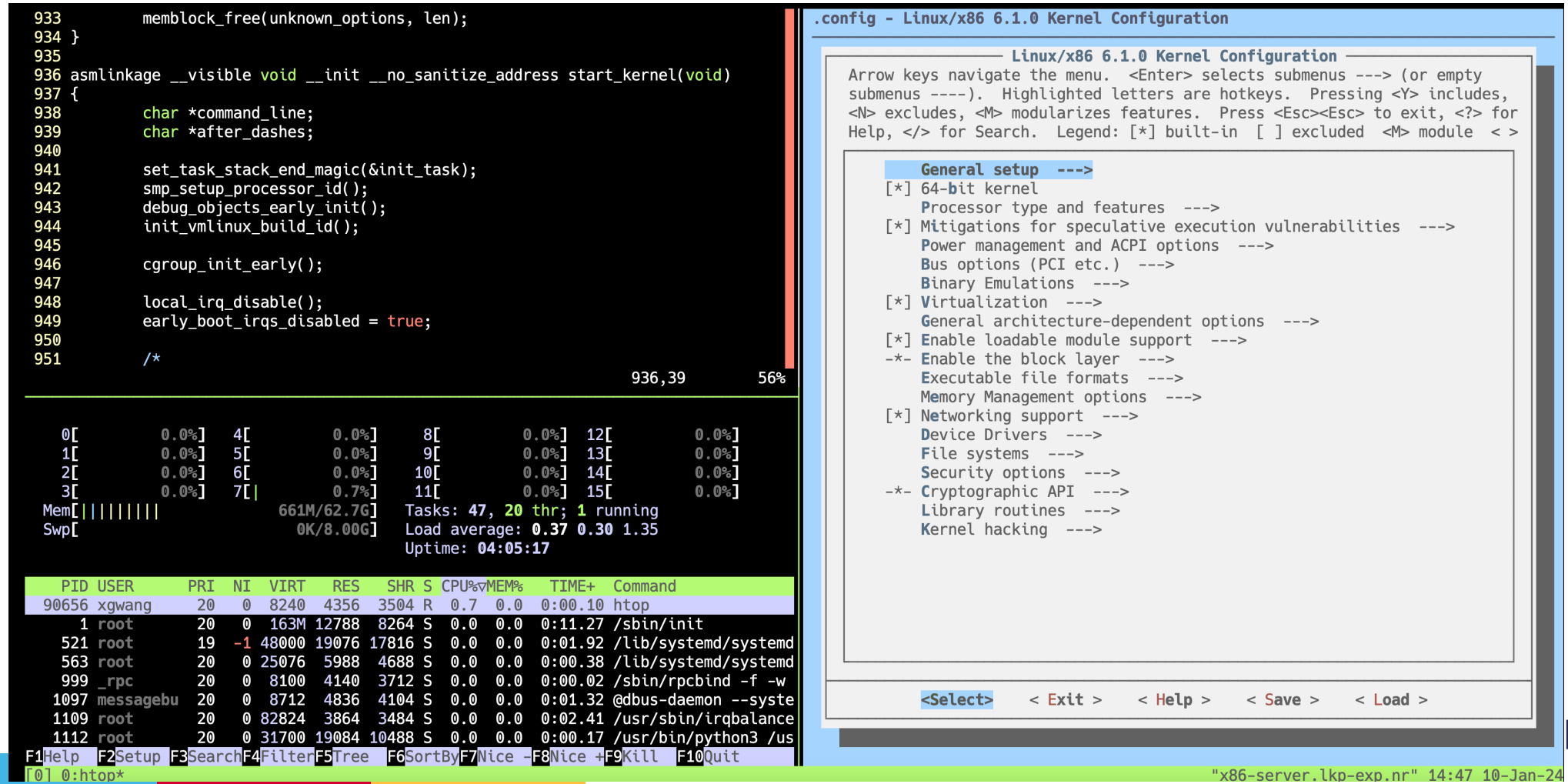
Generate the database

- `cd linux; ARCH=x86 make cscope tags -j2`

Search a function declaration

- `:tag start_kernel` or `:cs find global start_kernel`

Screen: tmux



ENCE

Essential tmux commands

`tmux`: start a new tmux session

`Ctrl-b %`: split a pane vertically

`Ctrl-b ”`: split a pan horizontally

`Ctrl-b z`: zoom (or unzoom) a pane

`Ctrl-b c`: create a new window

`Ctrl-b d`: detach from a session

`tmux a`: attach to an existing session

tmux: https://www.youtube.com/watch?v=nTqu6w2wc68

Kernel v.s. user programming

No libc or standard headers

- Instead, the kernel implements lots of libc-like functions

Examples

- `#include <string.h>` → `#include <linux/string.h>`
- `printf("Hello!")` → `printk(KERN_INFO "Hello!")`
- `malloc(64)` → `kmalloc(64, GFP_KERNEL)`

Kernel v.s. user programming

Use GCC extensions

Inline functions

- `static inline void func()`

Inline assembly: less than 2%

- `asm volatile("rdtsc" : "=a" (l), "=d" (h));`

Branch annotation: hint for better optimization

- `if (unlikely(error)) {...}`
- `if (likely(success)) {...}`

Kernel v.s. user programming

No (easy) use of floating-point numbers

Small, fixed-size stack: 8 KB (2 pages) in x86 (kernel stack)

No memory protection

- `SIGSEGV` → kernel panic (oops)

An example of kernel oops

```
[12710.153112] oops init (level = 1)
[12710.153115] triggering oops via BUG()
[12710.153127] ------------[ cut here ]------------
[12710.153128] kernel BUG at /home/duck/Articles/linuxoops/oops.c:17!
[12710.153132] invalid opcode: 0000 [#1] PREEMPT SMP PTI
[12710.153748] CPU: 0 PID: 5531 Comm: insmod
[12710.156191] RSP: 0018:ffffb41340e6fdd8 EFLAGS: 00010246
[12710.156849] RAX: 0000000000000019 RBX: ffffffffc1015040 RCX: 0000000000000000
[12710.157513] RDX: 0000000000000000 RSI: ffffffff83bc9d39 RDI: 00000000ffffffff
[12710.158171] RBP: ffff8d6101bd1d50 R08: 0000000000000000 R09: ffffb41340e6fc90
[12710.158826] R10: 0000000000000003 R11: ffffffff83f3d1e8 R12: ffffb41340e6fde0
[12710.159483] R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
[12710.160143] FS:  00007f6c290b31c0(0000) GS:ffff8d6411a00000(0000) knlGS:00000
[12710.160820] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[12710.161478] CR2: 00000000004134f0 CR3: 000000018be34005 CR4: 0000000003706f0
[12710.162156] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[12710.162824] DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
[12710.163474] Call Trace:
[12710.164129]  <TASK>
[12710.164779]  do_one_initcall+0x56/0x230
[12710.165424]  do_init_module+0x4a/0x210
[12710.166050]  __do_sys_finit_module+0x9e/0xf0
[12710.166711]  do_syscall_64+0x37/0x90
```

Kernel v.s. user programming

Synchronization and concurrency

- Muti-core processor → synchronization among tasks
  - A piece of kernel code can execute on two more processors
- Preemptive multitasking → synchronization among tasks
  - A task can be scheduled and re-scheduled at any time
- Interrupt → synchronization with interrupt handlers
  - Can occur during execution (e.g., accessing resource)
  - Need to synchronize with interrupt handler

Linux kernel coding style

- Indentation: 1 tab → 8-character width (not 8 spaces)
- No CamelCase, use underscores: `SpinLock` → `spin_lock`
- Use C-style comments: `/* use this style */` // not this
- Line length: 80 column
- Write code in a **similar style with other kernel code**

Ref: Documentation/process/coding-style.rst

Linux kernel coding style

```
/*
 * a multi-lines comment
 * (no C++ '//' !)
 */
struct foo {
        int member1;
        double member2;
}; /* no typedef ! */
#ifdef CONFIG_COOL_OPTION
int cool_function(void) {
        return 42;
}
#else
int cool_function(void) { }
#endif /* CONFIG_COOL_OPTION */
```

Linux kernel coding style

```
void my_function(int the_param, char *string, int a_long_parameter,
        int another_long_parameter)
{
        int x = the_param % 42;
        if (!the_param)
                do_stuff();
        switch (x % 3) {
        case 0:
                cool_function();
                break;
        case 1:
        /* Fall through */
        default:
                do_other_stuff();
        }
}
```

Summary of tools

- Version control: `git, tig`
- Configure the kernel: `make oldconfig`
- Build the kernel: `make -j8; make modules -j8`
- Install the kernel: `make install; make modules_install`
- Explore the code: `make cscope tags -j2; cscope, ctags`
- Editor: `vim, emacs`
- Screen: `tmux`

Other useful sources

The Linux Kernel Document: the extensive documents extracted from kernel source

Linux Weekly News: easy explanation of recently added kernel features

Linux Inside: textbook-style description on kernel subsystems

Kernel newbies: useful information for new kernel developers

Next steps

Master the essential tools, seriously

- editor: vim, emacs
- code navigation: cscope, ctags
- version control: git, tig
- terminal: ssh, tmux

Useful lecture videos: Vim, tmux,ssh, Git

- https://missing.csail.mit.edu/

hw1 deadline is due Friday

hw2 is released (due next Friday)

Next lecture

Isolation and system call

Explore how following system calls are implemented in the kernel

```
fd = open("out", 1);
write(fd, "hello\n", 6);
pid = fork();
```

Feedback