Advanced Workshop: Linux from Scratch

pcy

ULYSSIS

Sponsored by emweb

10 May 2023

Prerequisites

ULYSSIS

Knowledge about these topics assumed:

- Basic shell usage: **cd**, **ls**, **tar**, **wget**, …
  Ideally, also **mount**, **fdisk**, …
- Know how to clone a git repository
- Must have compiled at least one C program before: **make**,
  `gcc -o outfile stuff.c -lm`
- Optional: have used QEMU before

Prerequisites

ULYSSIS

Some packages required:

Arch Linux: `qemu wget grub base-devel dosfstools bc git rsync libelf musl`

Ubuntu, Debian: `qemu-system-x86 wget grub-pc-bin grub2-common build-essential libncurses-dev bc git rsync libelf-dev musl{,-dev,-tools} bison flex`

Fedora: `wget qemu grub2-pc make gcc kernel-devel ncurses-devel bc musl-gcc bison flex`

Void Linux: `qemu wget grub base-devel ncurses-devel bc git rsync libelf musl-bootstrap`

Prerequisites

ULYSSIS

2 GB of disk space required.

Arch users: reboot if you updated your kernel after last powering on your computer (need loopback module, may cause issues when not loaded yet after a kernel update)

Introduction

We will dive right in: compilation takes a while, explanation will follow

Repository

Clone this repository:

`https://gitlab.ulyssis.org/pcy/workshop-linux`

Kernel and Busybox config files from it will be used

Also a useful reference for when you fall behind

If at any point you fall behind too much: grab binaries from `https://pcy.be/tmp/miscbin/lfs23-binaries.tar.xz`

Downloading the kernel

ULYSSIS

Available at `https://www.kernel.org/`

```
wget linux-6.2.14.tar.{sign,xz}
tar xf linux-6.2.14.tar.xz
```

Downloading the kernel

ULYSSIS

Available at `https://www.kernel.org/`

```
wget linux-6.2.14.tar.{sign,xz}
tar xf linux-6.2.14.tar.xz
```

Signature check (optional):

```
xz -cd linux-6.2.14.tar.xz \
    | gpg --verify linux-6.2.14.tar.sign -
```

Configuring the kernel

ULYSSIS

The kernel has lots of configuration options:

- Target hardware
- Modes of operation (eg. fast vs slower task switching)
- Enable/disable support for certain hardware
- Configure feature as builtin/module/disabled (*/M/)
- ...

Configuring the kernel

ULYSSIS

What are kernel modules?

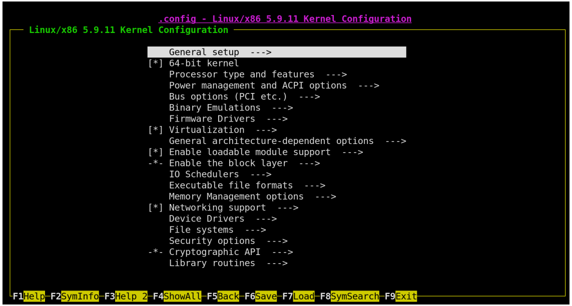
- Usually, most kernel code is in a single binary (the kernel image)
- Modules are separate files that can be loaded into the kernel at runtime
- Only load drivers when the hardware is attached
- Smaller kernel image $\Rightarrow$ faster booting
- Distros: one kernel image for multiple scenarios, different modules
- `lsmod`, `insmod`, `rmmod`, `modprobe`

Configuring the kernel

ULYSSIS

- Generate a default configuration: `make defconfig`
- Edit configuration: `make nconfig`

A quick demo:



Configuring the kernel

ULYSSIS

- Generate a default configuration: `make defconfig`
- Edit configuration: `make nconfig`
- Enable *current* modules only: `make localyesconfig`
  `lsmod > mods && LSMOD=mods make localyesconfig`
- Above, but as modules: `make localmodconfig`
- Enable everything: `make allyesconfig`
- Enable everything as modules: `make allmodconfig`
- Random config: `make randconfig`

Configuring the kernel

ULYSSIS

For cross-compiling:
(not in this workshop)

`ARCH=arch CROSS_COMPILE=target-triple-`
In *every* `make` invocation!
Example: **`ARCH=sparc64 CROSS_COMPILE=sparc-linux-gnu-`**

Boot medium formats differ *wildly* from platform to platform!

Configuring the kernel

ULYSSIS

For now, use a minimal `.config`:

- Very small, minimum requirement to boot
- Useless for real hardware: no USB, no network, no …
- Still works in QEMU
- Faster to compile!

```
cp path/to/repo/linux-workshop-minimal.config .config
```

Compiling the kernel

ULYSSIS

`make -j<N>`

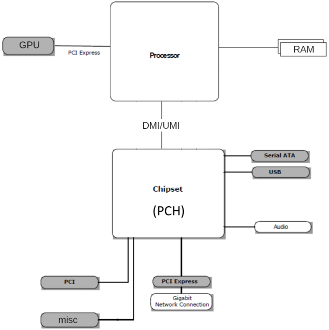
If dialog questions appear: enter the default

usually won't hurt

- `N`: number of CPU cores to use
- optimal: `N` equal to number of cores available

Intermezzo 1: PC boot process

What is a PC?



Source: "Advanced x86: Introduction to BIOS & SMM"
`https://opensecuritytraining.info/IntroBIOS.html`

Intermezzo 1: PC boot process

ULYSSIS

Extremely simplified:

1. Hardware powers up CPU
2. CPU boots into firmware (BIOS or UEFI)
3. Firmware inits other hardware (disks, GPU, ...)
4. Firmware loads OS or bootloader from disk

Intermezzo 1: PC boot process

ULYSSIS

- BIOS: boots into disk MBR (first 512b)
  → Too small to contain a kernel!
- UEFI: boots EFI executable from EFI System Partition
  → Kernel image is in a different format!

⇒ We need a bootloader!

Note on embedded

ULYSSIS

- This was PC
- Other things: much different, lots of variety
- Ask me for details if you want

While our kernels are compiling...

ULYSSIS

It's time for questions!

Ask me (almost) anything!

Running the kernel

Kernel image in `arch/x86/boot/bzImage`, let's try it out!

```
qemu-system-x86_64 -kernel arch/x86/boot/bzImage
```

To make QEMU stop grabbing your cursor: `ctrl+alt+g`

Running the kernel

Kernel image in `arch/x86/boot/bzImage`, let's try it out!

```
qemu-system-x86_64 -kernel arch/x86/boot/bzImage
```

To make QEMU stop grabbing your cursor: `ctrl+alt+g`

```
[    2.467374] Kernel panic - not syncing: VFS: Unable to mount root fs on unkno
wn-block(0,0)
[    2.467684] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.9.11 #1
[    2.467742] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1
.14.0-0-g155821a1990b-prebuilt.qemu.org 04/01/2014
[    2.467915] Call Trace:
[    2.468739]  dump_stack+0x57/0x6a
[    2.468841]  panic+0xf6/0x2b7
[    2.468900]  mount_block_root+0x196/0x21a
[    2.469022]  mount_root+0xec/0x10a
[    2.469064]  prepare_namespace+0x136/0x165
[    2.469120]  kernel_init_freeable+0x1c8/0x1d3
[    2.469161]  ? rest_init+0x9a/0x9a
[    2.469197]  kernel_init+0x5/0x106
[    2.469235]  ret_from_fork+0x22/0x30
[    2.469711] Kernel Offset: 0x1ba00000 from 0xffffffff81000000 (relocation ran
ge: 0xffffffff80000000-0xffffffffbfffffff)
[    2.470051] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs
 on unknown-block(0,0) ]---
```

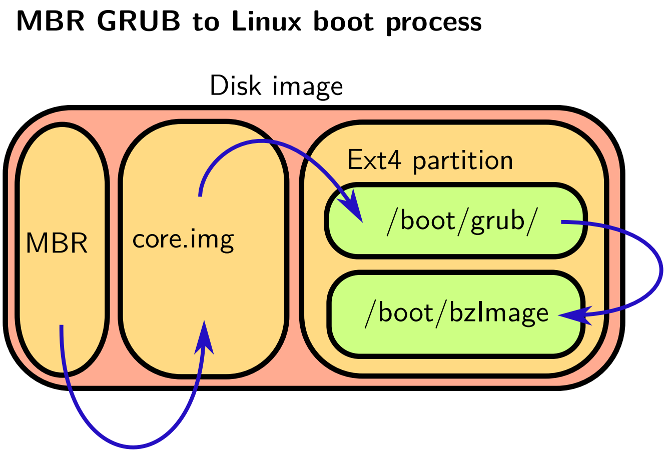
What went wrong?

ULYSSIS

- "Kernel panic: unable to mount root fs"
- The kernel has no root filesystem, but it needs one to run user code!
- Other than that, everything went fine!
- Let's create one

rootfs overview (BIOS)

ULYSSIS

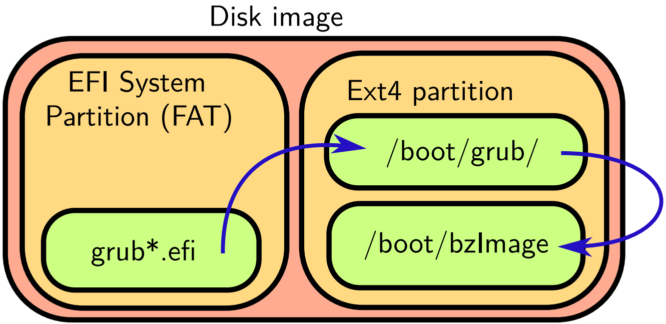


rootfs overview (UEFI)

ULYSSIS

For informational purposes, this workshop is BIOS-only.

EFI GRUB to Linux boot process



Generating the rootfs

ULYSSIS

`fallocate -l 32M root.img`	Create a large file
`fdisk root.img`	Format it
`n p 1 [empty] [empty] t 83 w`	

Now we need to put files on the partition *inside this file*

losetup

ULYSSIS

Most commands here might be ‘obvious’, except for `losetup`. It turns regular files into “loopback disk devices”, so you can mount them like other disks and partitions.

```
$ lsblk
sda 8:0 0 465,8G 0 disk

$ sudo losetup -P -f disk.img && lsblk
loop0 7:0 0 32M 0 loop
‘-loop0p1 259:0 0 31M 0 part
sda 8:0 0 465,8G 0 disk
```

Generating the rootfs

ULYSSIS

Now we use `losetup` to populate the root filesystem:

`sudo losetup -P -f --show root.img`	Make a device out of it
`sudo mkfs.ext4 /dev/loop0p1`	Format the partition

```
mkdir -p rootfs && sudo mount /dev/loop0p1 ./rootfs
```

Supplying a kernel

ULYSSIS

Copy over the kernel we just compiled into `/boot`:

```
cd ./rootfs && sudo mkdir boot
sudo cp ../linux-6.2.14/arch/x86/boot/bzImage boot/
```

Supplying a bootloader

ULYSSIS

Install GRUB:

```
sudo grub-install --modules=part_msdos \
   --target=i386-pc --boot-directory="$PWD/boot" \
   /dev/loop0
```

Fedora users: `grub2-install`

Configuring the bootloader

ULYSSIS

We still need to tell GRUB how to find our kernel image

```
$ fdisk -l ../root.img
```

⇒

```
Disk root.img:  32 MiB, 33554432 bytes, 65536 sectors
...
Disk identifier:  0x7b258ad4
...
```

Configuring the bootloader

ULYSSIS

We still need to tell GRUB how to find our kernel image

```
$ sudo -e boot/grub/grub.cfg
⇐
linux /boot/bzImage root=PARTUUID=7b258ad4-01
boot
```

Testing the rootfs

Ok, will it work this time?

```
cd ..  && sudo umount rootfs && sudo losetup -d \
    /dev/loop0
```

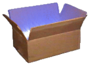
```
qemu-system-x86_64 root.img
```

```
[    1.230259] devtmpfs: mounted
[    1.266552] Freeing unused kernel image (initmem) memory: 728K
[    1.266858] Write protecting the kernel read-only data: 12288k
[    1.269080] Freeing unused kernel image (text/rodata gap) memory: 2044K
[    1.269656] Freeing unused kernel image (rodata/data gap) memory: 484K
[    1.269907] Run /sbin/init as init process
[    1.277385] Run /etc/init as init process
[    1.278578] Run /bin/init as init process
[    1.279246] Run /bin/sh as init process
[    1.279576] Kernel panic - not syncing: No working init found.  Try passing i
nit= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance
.
[    1.279811] CPU: 0 PID: 1 Comm: swapper Not tainted 5.9.11 #1
[    1.279867] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1
.14.0-0-g155821a1990b-prebuilt.qemu.org 04/01/2014
[    1.280028] Call Trace:
[    1.280773]  panic+0xde/0x273
[    1.281186]  ? rest_init+0x7a/0x7a
[    1.281231]  kernel_init+0xe8/0xf6
[    1.281300]  ret_from_fork+0x1f/0x30
[    1.281494] Kernel Offset: disabled
[    1.281768] ---[ end Kernel panic - not syncing: No working init found.  Try
passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for
 guidance. ]---
```

Enter Busybox

ULYSSIS

Implementation of a full userspace, small binary size



We will use it only for `/bin/sh`, even though it provides lots of utilities, as well as an init system.

Generating kernel headers

ULYSSIS

Before compiling, Busybox needs to know about our kernel:

```
mkdir linux-headers && cd linux-6.2.14

make CC=false V=1 INSTALL_HDR_PATH=../linux-headers \
   headers_install
```

Downloading Busybox

ULYSSIS

Available at https://busybox.net/downloads/

```
wget busybox-1.36.0.tar.bz2{.sig,.sha256,}
tar xf busybox-1.36.0.tar.bz2
```

Signature check (optional):

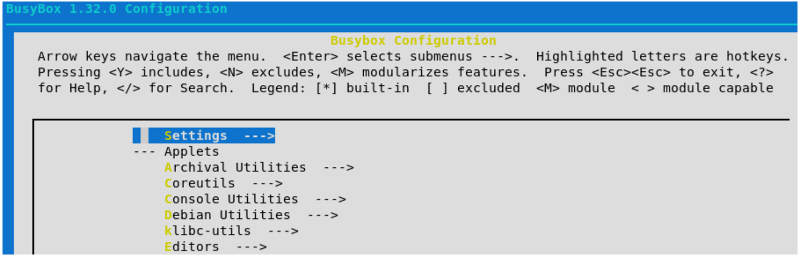
```
wget https://busybox.net/~vda/vda_pubkey.gpg && \
    gpg --import vda_pubkey.gpg && \
    sha256sum -c busybox-1.36.0.tar.bz2.sha256 &&\
    gpg --verify busybox-1.36.0.tar.bz2{.sig,}
```

Configuring Busybox

Similar config system as Linux:

```
make defconfig
make menuconfig
```

A quick demo:



Configuring Busybox

Similar config system as Linux:

```
make defconfig
make menuconfig

cp path/to/repo/busybox-workshop-minimal.config \
   .config
```

Configuring Busybox

Busybox needs our kernel headers:

$EDITOR .config

⇐

```
CONFIG_EXTRA_CFLAGS="-I../linux-headers/include/"
```

Compiling Busybox

ULYSSIS

```
make CC=musl-gcc -j<N>
```

- N: number of CPU cores to use
- optimal: N equal to number of cores available
- musl needed here: Busybox is picky (static glibc build breaks, passing musl as cross-compile prefix breaks on some distros, so using CC like this is the only option)

Intermezzo 2: userspace boot process

ULYSSIS

1. Kernel spawns PID 1 (‘init’)
   - If PID 1 dies, the kernel panics
   - PID 1 is responsible for garbage-collecting finished processes
   - Needs to start everything else

Intermezzo 2: userspace boot process

ULYSSIS

1. Kernel spawns PID 1 (‘init’)
   - If PID 1 dies, the kernel panics
   - PID 1 is responsible for garbage-collecting finished processes
   - Needs to start everything else
2. `init` forks, child carries out initialization (“early userspace”)
3. Init interfaces to kernel, filesystem, devices, ...
4. Start and manage daemons
5. Present login screen

While our Busyboxes are compiling...

ULYSSIS

It's time for questions again!

Ask me (almost) anything!

Installing busybox

ULYSSIS

Copy ./busybox to rootfs/bin/busybox:

```
cd ..  && sudo losetup -P -f --show root.img
sudo mount /dev/loop0p1 ./rootfs
sudo mkdir -p rootfs/bin
sudo cp busybox-1.36.0/busybox rootfs/bin/busybox
```

Installing busybox (2)

Basic shell utilities (from Busybox):

```
sudo ln -s bin rootfs/sbin
for x in $(rootfs/bin/busybox --list-full); do
  sudo ln -s /bin/busybox $x
done
```

Reconfigure bootloader

ULYSSIS

Tell GRUB to boot into `/bin/sh`:

```
sudo -e rootfs/boot/grub/grub.cfg
⇐
linux /boot/bzImage root=...  init=/bin/sh

sudo umount rootfs
sudo losetup -d /dev/loop0
```

Testing init

ULYSSIS

Third time is the charm:

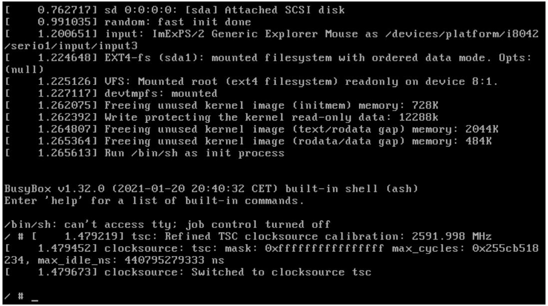
```
qemu-system-x86_64 root.img
```

Testing init

ULYSSIS

Third time is the charm:

`qemu-system-x86_64 root.img`



```
[    0.762717] sd 0:0:0:0: [sda] Attached SCSI disk
[    0.991035] random: fast init done
[    1.200651] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042
/serio1/input/input3
[    1.224648] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts:
(null)
[    1.225126] VFS: Mounted root (ext4 filesystem) readonly on device 8:1.
[    1.227117] devtmpfs: mounted
[    1.262075] Freeing unused kernel image (initmem) memory: 728K
[    1.262392] Write protecting the kernel read-only data: 12288k
[    1.264807] Freeing unused kernel image (text/rodata gap) memory: 2044K
[    1.265364] Freeing unused kernel image (rodata/data gap) memory: 484K
[    1.265613] Run /bin/sh as init process


BusyBox v1.32.0 (2021-01-20 20:40:32 CET) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/bin/sh: can't access tty; job control turned off
/ # [    1.479219] tsc: Refined TSC clocksource calibration: 2591.998 MHz
[    1.479452] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x255cb518
234, max_idle_ns: 440795279333 ns
[    1.479673] clocksource: Switched to clocksource tsc

/ # _
```

Final notes

For a real usable system, you need a few more things:

- More standard directories: `/bin /dev /etc /home /lib /proc /sys /var/lib /var/lock /var/log /var/run /root /tmp /usr`
- Standard files: `/etc/hostname /etc/passwd /etc/shadow /etc/group /etc/hosts /etc/fstab`
- Set up init and daemon supervision system
- $\Rightarrow$ Buildroot!

Final notes

If you want to learn more:

- Documentation in the kernel repository, LWN
- Arch, Gentoo wikis, LFS book
- Search engine!
- Some IRC, Matrix, ... channels

That's all folks!

Thanks to our sponsor:

Please fill in the feedback form!

https://ulyssis.org/
https://gitlab.ulyssis.org/pcy/workshop-linux/