Speedy Distro Porting via the `cpu` Command

Daniel Maslowski
Introduction
Thank you, it’s good to be back!

Hi, I’m Daniel!

▶ professional app and web developer
▶ been to openSUSE Conference many times
▶ hacking on firmware and operating systems
▶ started to write code in Go and Rust
Thank you, it’s good to be back!

Hi, I’m Daniel!

- professional app and web developer
- been to openSUSE Conference many times
- hacking on firmware and operating systems
- started to write code in Go and Rust

Remember? In 2019, we talked about open source firmware!
Thank you, it’s good to be back!

Hi, I’m Daniel!

- professional app and web developer
- been to openSUSE Conference many times
- hacking on firmware and operating systems
- started to write code in Go and Rust

Remember? In 2019, we talked about open source firmware!
Agenda

- Distributing an OS
- Porting Firmware
- Speeding Things up
Distributing an OS
Building Software

Software needs to be built, by the distro or the end user. Building software requires toolchains, meeting assumptions, and patching.
Building Software

For distribution, software needs to be built, by the distro or the end user.
Building Software

For distribution, software needs to be built, by the distro or the end user.

Building software requires toolchains, meeting assumptions, patching.
Welcome to openSUSE Build Service

The openSUSE Build Service is the public instance of the Open Build Service (OBS) used for development of the openSUSE distribution and to offer packages from same source for Fedora, Debian, Ubuntu, SUSE Linux Enterprise and other distributions.

Please find further details of this service on our wiki pages

Users can find a special package search interface here. Users of any distribution can search there for built packages for their distribution. For developers it is an efficient place to build up groups and work together through its project model.

This instance receives weekly deployments during the openSUSE Maintenance Downtime (Thursday 08:00 - 10:00 CET) and smaller deployments on request at any time.
Hello OBS!

Building as a Service - BaaS

Welcome to openSUSE Build Service

The openSUSE Build Service is the public instance of the Open Build Service (OBS) used for development of the openSUSE distribution and to offer packages from same source for Fedora, Debian, Ubuntu, SUSE Linux Enterprise and other distributions.

Please find further details of this service on our wiki pages

Users can find a special package search interface here. Users of any distribution can search there for built packages for their distribution. For developers it is an efficient place to build up groups and work together through its project model.

This instance receives weekly deployments during the openSUSE Maintenance Downtime (Thursday 08:00 - 10:00 CET) and smaller deployments on request at any time.
Processors and Architectures

Software on lower levels involves platform specifics. The kernel is commonly the lowest level part of an operating system.

One kernel, one distro image per architecture.

1 https://azrael.digipen.edu/~mmead/www/Courses/CS180/OSOverview.html
Processors and Architectures

Software on lower levels involves platform specifics.

¹https://azrael.digipen.edu/~mmead/www/Courses/CS180/OSOverview.html
Software on lower levels involves platform specifics.

The kernel is commonly the lowest level part of an operating system\(^1\).

\(^1\)https://azrael.digipen.edu/~mmead/www/Courses/CS180/OSOverview.html
Software on lower levels involves platform specifics.

The kernel is commonly the lowest level part of an operating system\(^1\).

One kernel, one distro image per architecture.

\(^1\)https://azrael.digipen.edu/~mmead/www/Courses/CS180/OSOverview.html
Wait, what happened?

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>openSUSE-Tumbleweed-RISC-V-E20-efi.riscv64-2022.04.29-Build1.34.rawxz</td>
<td>31-May-2022 18:25</td>
<td>1.6G</td>
</tr>
<tr>
<td>openSUSE-Tumbleweed-RISC-V-E20-efi.riscv64-2022.04.29-Build1.34.raw.xz</td>
<td>31-May-2022 18:25</td>
<td>481</td>
</tr>
<tr>
<td>openSUSE-Tumbleweed-RISC-V-E20.riscv64-rootfs.riscv64-2022.04.29-Build1.34.tar.xz</td>
<td>31-May-2022 18:11</td>
<td>1.5G</td>
</tr>
<tr>
<td>openSUSE-Tumbleweed-RISC-V-E20.riscv64-rootfs.riscv64-2022.04.29-Build1.34.tar.xz.sha256.asc</td>
<td>31-May-2022 18:11</td>
<td>481</td>
</tr>
</tbody>
</table>
Revisting Assumptions
Many chip vendors and multiple architectures imply fragmentation.
Revisting Assumptions

Many chip vendors and multiple architectures imply fragmentation.

What may an OS safely assume? (our core question)
Porting Firmware
Hello RISC-V!
Hello RISC-V!

```
tools/mkimage -T sunxi_toc1 -d toc1.cfg u-boot.toc1
```
Hello RISC-V!

```
tools/mkimage -T sunxi_toc1 -d toc1.cfg u-boot.toc1
```
Hello, RISC-V?

https://fedoraproject.org/wiki/Architectures/RISC-V/Allwinner
Why, RISC-V?

Why, RISC-V?

We are working on a simpler oreboot port for the JH7100 / VisionFive. :)

Speeding Things up
Offering LinuxBoot

Take firmware for granted. Focus on the OS itself!

https://www.linuxboot.org/
Offering LinuxBoot

Take firmware for granted. Focus on the OS itself!

https://www.linuxboot.org/
Offering LinuxBoot

Take firmware for granted. Focus on the OS itself!

https://www.linuxboot.org/
oreboot and LinuxBoot

oreboot initializes your hardware and executes a payload. LinuxBoot provides you with a Linux environment, including bootloaders.
oreboot and LinuxBoot

oreboot

oreboot initializes your hardware and executes a payload.
oreboot and LinuxBoot

oreboot

oreboot initializes your hardware and executes a payload.

LinuxBoot

LinuxBoot provides you with a Linux environment, including boot loaders.
LinuxBoot and cpu

Now do what you want and have a lot of fun!
Include `cpud`, the `cpud` daemon, in your LinuxBoot environment
LinuxBoot and cpud

Include cpud, the cpud daemon, in your LinuxBoot environment
Now do what you want and have a lot of fun!
Okay, \texttt{cpu} - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.

Simple command

Use \texttt{cpu} instead of moving an SD card between host and test device!

\texttt{cpu target-device ./program-to-test --with-some-args}

Advanced (just as simple)

You can even \texttt{kexec} over \texttt{cpu}. :-(

\texttt{cpu target-device ./kexec ./your-next-kernel}
Okay, cpu - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.
Okay, `cpu` - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.

**Simple command**

Use `cpu` instead of moving an SD card between host and test device!
Okay, cpu - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.

**Simple command**

Use `cpu` instead of moving an SD card between host and test device!

```
cpu target-device ./program-to-test --with-some-args
```
Okay, cpu - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.

**Simple command**

Use cpu instead of moving an SD card between host and test device!

cpu target-device ./program-to-test --with-some-args

**Advanced (just as simple)**

You can even kexec over cpu. :-)

Okay, cpu - but what is it?

Think of a little stub which lets you access and run anything on a remote that you bring from your host system.

Simple command
Use cpu instead of moving an SD card between host and test device!
cpu target-device ./program-to-test --with-some-args

Advanced (just as simple)
You can even kexec over cpu. :-)  
cpu target-device ./kexec ./your-next-kernel
cpu DEMO
Leveraging `cpu` for Distro Testing

Use Linux boot to define a well-known system state. Attach to network. Power on the board, and check for readiness on the console. Deploy Linux with `cpud` as the init (`cpukernel`).

Now, build your distro. Use NFS root and `kexec` into your new kernel. Does it boot? Yes -> yay! Nope -> you found a bug!
Leveraging `cpu` for Distro Testing

Use LinuxBoot to define a well-known system state. Attach to network.
Leveraging cpu for Distro Testing

Use LinuxBoot to define a well-known system state. Attach to network. Power on the board, and check for readiness on the console.
Leveraging `cpu` for Distro Testing

Use LinuxBoot to define a well-known system state. Attach to network. Power on the board, and check for readiness on the console.

Deploy Linux with `cpud` as the init (`cpukernel`).
Leveraging cpu for Distro Testing

Use LinuxBoot to define a well-known system state. Attach to network. Power on the board, and check for readiness on the console.

Deploy Linux with cpu as the init (cpukernel).
Now, build your distro. Use NFS root and kexec into your new kernel.
Leveraging cpu for Distro Testing

Use LinuxBoot to define a well-known system state. Attach to network. Power on the board, and check for readiness on the console.

Deploy Linux with cpu as the init (cpukernel).
Now, build your distro. Use NFS root and kexec into your new kernel. Does it boot? Yes -> yay! Nope -> you found a bug!
Hey openQA!

Life is too short for manual testing!

Welcome to openQA
Life is too short for manual testing!

openSUSE Tumbleweed

- Build20220601 (about 7 hours ago)
  - 227 passed
  - 23 uncommitted
  - 51 soft failed
  - 15 failed

- Build20220531 (a day ago)
  - 243 passed
  - 62 soft failed
  - 12 failed

- Build20220530 (2 days ago)
  - 242 passed
  - 62 soft failed
  - 14 failed

openSUSE Tumbleweed AArch64

- Build20220530 (2 days ago)
  - 142 passed
  - 23 soft failed
  - 37 failed

- Build20220528 (3 days ago)
  - 141 passed
  - 24 soft failed
  - 37 failed

- Build20220523 (9 days ago)
  - 144 passed
  - 37 soft failed
  - 34 failed

Life is too short for manual testing!
Hey openQA!

Life is too short for manual testing!

Where is RISC-V? Let’s make it happen! :-)

openSUSE Tumbleweed
- Build20220601 (about 7 hours ago)
  - 227 passed
  - 23 unknown
  - 51 soft failed
  - 15 failed

- Build20220531 (a day ago)
  - 243 passed
  - 62 soft failed
  - 12 failed

- Build20220530 (2 days ago)
  - 242 passed
  - 62 soft failed
  - 14 failed

openSUSE Tumbleweed AArch64
- Build20220530 (2 days ago)
  - 142 passed
  - 23 soft failed
  - 37 failed

- Build20220528 (3 days ago)
  - 141 passed
  - 24 soft failed
  - 37 failed

- Build20220523 (9 days ago)
  - 144 passed
  - 37 soft failed
  - 34 failed
Testing Strategies and Setup

- Test different build setup variations (e.g., cmdline args)
- Assert on serial console and video output
- Reset when done with each case (hard reset)

Requirements
- Second piece of hardware for monitoring, reset, instrumentation
- HDMI capture, USB-HDMI, VNC etc
- Connect to UART
- Hook up GPIO to reset
- e.g., Raspberry Pi, 3m deb RTE, DIY…
- Some glue logic in build service (CI)
Testing Strategies and Setup

- test different build setup variations (e.g., cmdline args)
- assert on serial console and video output
- reset when done with each case (hard reset)
Testing Strategies and Setup

- test different build setup variations (e.g., cmdline args)
- assert on serial console and video output
- reset when done with each case (hard reset)

Requirements

- second piece of hardware for monitoring, reset, instrumentation
  - HDMI capture, USB-HDMI, VNC etc
  - connect to UART
  - hook up GPIO to reset
  - e.g., Raspberry Pi, 3mdeb RTE, DIY…
- some glue logic in build service (CI)
Try it out!

Join our workshop at 17:00 in Seminarraum 2

Preparation: Install either Docker or QEMU plus (optionally) Go 1.18.

```
dama@orangelemp ~/P/L/l/m/l/generic (main)> ~/Firmware/ipc/arm-cpu/bin/cpu -sp 23 -key ~/.ssh/cpu_rsa -timeout 5p 3s 192.168.11.245 cat /proc/cpuinfo
```

```plaintext
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 122
model name : Intel(R) Celeron(R) J4125 CPU @ 2.00GHz
stepping : 8
microcode : 0x1a
cpu MHz : 1996.800
cache size : 4096 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 4
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuuid level : 24
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bt s rep_good nopl xtopology nonstop_tsc aperfmperf tsck knownfq pni pclmulqdq dtes64 monitor ds cpl vmx est tm2 ssse3 sdbg xmmv txrret pdcm sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave xaxr lahf_lm3dnowprefetch cpuid_fault cat_l2 cdp_l2 ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept ad fsgsbase tsc_adjust sgx smep erms mpx rdt a rsread eptflsopt intel_pt sha_ni xsaveopt xsaves xgetbv1 xsavec xsaveopt avx1 arch_capabilities
vmx flags : vmx_nested_root_xsaveopt vmx_nested_root_ept vmx_nested_root_ept xapic
```