Building non-native binary packages with Gentoo

The problem

- I want Gentoo
- Raspberry Pi A is arm6 with 512Mb of RAM
  - Very long builds (days for some packages)
  - Very long failed builds (days until OOM)
  - Kills SD cards
  - Binary updates preferred.
- Pi 4 – aarch64 build

The solution

- Install binary packages
- Build binary packages
- Build non-native binary packages
  - crossdev, distcc, chroot, qemu
- Backup / install OS - mkstage4
- Use Gentoo’s binary package server!!

What? A Gentoo binary server?

- Gentoo’s stage3 (minimal Gentoo environment)
  - https://www.gentoo.org/downloads/
  - Boot off USB (ventoy) / remove disk to running host
  - Format disk
  - Chroot
  - Download and untar a stage 3
  - Configure
  - Build kernel
  - Boot

A Gentoo binary ***package*** server?

- https://www.gentoo.org/news/2023/12/29/Gentoo-binary.html
  - “… we’re now also offering binary packages for download and direct installation!”
  - Most architectures: “limited to the core system and weekly updates”
  - For amd64 and arm64:
    - “>20 GByte of packages on our mirrors”
    - “Gentoo stable, updated daily. Enjoy!”

A Gentoo binary package server?

- What about the USE flags?
  - If USE flags don’t match, binaries ignored.
  - Sources downloaded, package compiled locally.
- What about fine tuning to CPU?
  - If CPU flags don’t match, binaries ignored.
  - Sources downloaded, package compiled locally.

DIY binary packages on Gentoo

- PKGDIR=/mnt/binpkg/ARMv6/
- --usepkg=y (Emerge flag)
  - Search PKGDIR for existing binary package.
  - Search will fail if USE flags differ!
  - If the binary package doesn’t exist, build from source.
- --usepkgonly=y (Emerge flag)
  - Search PKGDIR for existing binary package
  - Abort if doesn’t exist (with correct USE flags)
- buildpkg (FEATURE variable)
  - After building, copy binary package to PKGDIR

DIY binary packages on Gentoo

- quickpkg <package>
  - Build a binary package from already installed package
  - Beware – config files
  - Useful before breaking something critical
    - Can’t build because building is broken.

Arm6 installing gcc

```
These are the packages that would be merged, in order:

Calculating dependencies... done!
Dependency resolution took 1093.58 s (backtrack: 0/20).

[binary     U  ] sys-devel/gcc-13.2.1_p20240113-r1 [13.2.1_p20230826]
[ebuild     U  ] media-video/ffmpeg-6.0.1-r2 [6.0-r9]

Would you like to merge these packages? [Yes/No]
```

- Installing binary took 25 minutes.
- Local build took 4 days 43 minutes

mkstage4

- “stage 4 loosely defined term that just means a stage 3 with 'extra bits'.”
- https://github.com/TheChymera/mkstage4
- Bash script using tar to back up a working system

COMMAND LINE PREVIEW:
tar -cpP --ignore-failed-read --xattrs-include=*.* --numeric-owner --use-compress-prog=/bin/bzip2 --exclude=/dev/* --exclude=/var/tmp/* --exclude=/media/* --exclude=/mnt/*/* --exclude=/proc/* --exclude=/run/* --exclude=/sys/* --exclude=/tmp/* --exclude=/var/lock/* --exclude=/var/log/* --exclude=/var/run/* --exclude=/var/lib/docker/* --exclude=/home --exclude=/bootbackup --exclude=/export --exclude=/mnt/binpkg --exclude=/mnt/distfiles --exclude=/mnt/binpkg/stage4/losinj_2024-01-31.tar.bz2 --exclude=/usr/portage/* --exclude=/mnt/distfiles/* -f /mnt/binpkg/stage4/losinj_2024-01-31.tar.bz2 /

chroot

- Change root
- https://wiki.gentoo.org/wiki/Chroot
- Shares OS of the host machine
- Lightweight alternative to a VM
- Allows a system to find things where it expects.
  - Gentoo builds, installs, and looks for files in /usr/bin
  - Physical location is /mnt/gentoo/usr/bin
  - chroot runs a process with /mnt/gentoo “changed” to /
  - It also forbids access outside of “chroot jail”, so host safe.
  - Sym links won’t work. /mnt/gentoo/etc/resolve.conf ==> /etc/resolve.conf

chroot

```
cp /etc/resolve.conf <desired root>/etc/resolve.conf

mount --bind /export/binpkg/ <desired root>/mnt/binpkg
mount --bind /home           <desired root>/home

mount --bind /proc <desired root>/proc
mount --bind /sys <desired root>/sys
mount --bind /dev <desired root>/dev

chroot <desired root> /bin/bash
```

A persistent chroot environment

- Create a persistent chroot environment
  - Create chroot, and connect.
  - Start tmux (persistent terminal)
  - Detach from tmux (not exit)
  - Exit from chroot (persists inside tmux)
- Connect to persistent chroot
  - Attach to chroot
  - Attach to running tmux

Non-native chroot

- QEMU (Quick EMUlator)
- Will run arm6 binaries on an AMD64 box
- QEMU supports:

```
aarch64       aarch64_be  alpha      arm
armeb         cris        hexagon    hppa
i386          loongarch64 m68k       microblaze
microblazeel  mips        mips64     mips64el
mipsel        mipsn32     mipsn32el  nios2
or1k          ppc         ppc64      ppc64le
riscv32       riscv64     s390x      sh4
sh4eb         sparc       sparc64    sparc32plus
x86_64        xtensa      xtensaeb
```

QEMU

- https://wiki.gentoo.org/wiki/QEMU
- Needs KVM or AMD-V
- Kernel needs
  - CONFIG_HIGH_RES_TIMERS
  - CONFIG_KVM
  - CONFIG_KVM_AMD (... or _INTEL)

QEMU

- arm6
  - arch is armv7l
  - CHOST=armv6j-unknown-linux-gnueabihf
- aarch64
  - aarch64 kernel build on AMD64 Qemu took 2 days 12 hours to build.

Emulation

- ... is slow
  - Sees a CPU instruction, instead of feeding it to a CPU, run a piece of code that will do the same.
- QEMU
  - Dynamic Binary Translation
  - Cache sequences of instructions (sentences) for reuse, as opposed to individual instructions (words).
  - QEMU claims can “run virtual machines at near-native speed” ??
    - VM not same as emulation.
    - Running AMD64 VM on AMD64, not emulating anything.
    - QEMU = Quick, or EMUlate, not both ??
    - Running emulation with 6 cores and 20Gb ram may be faster than arm6 pi

chroot + qemu

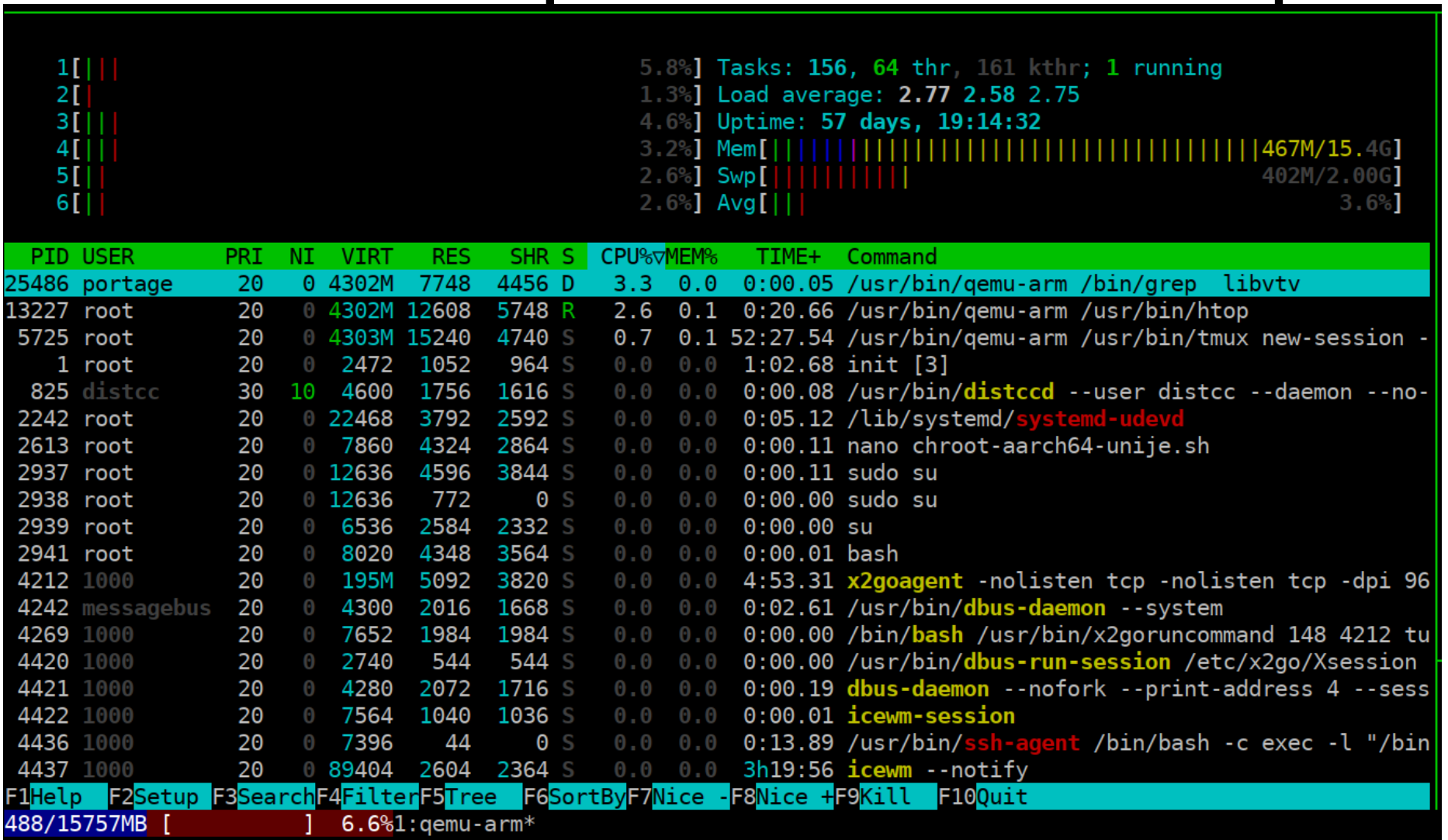
```
cd <desired root>
mount --bind /export/binpkg/ mnt/binpkg
mount --bind /home home

mount --bind /proc proc
mount --bind /sys sys
mount --bind /dev dev
mount --bind /dev/pts dev/pts
mount --bind /dev/shm dev/shm

if [ ! -f "./usr/bin/qemu-arm" ] ; then
  cp /usr/bin/qemu-arm usr/bin
fi

chroot . /bin/bash --login
```

chroot + qemu + tmux + htop



Building non-native binaries

- Copied arm6 stage4 onto host (amd64)
- Create an emulated environment (qemu/chroot)
- Build packages for binary distribution
  - distcc
  - crossdev

distcc

- Distributed C/C++ compiler
  - Client performs pre-compiling (include headers, perform macro expansion)
  - Client passes pre-compiled source to server
  - Server compiles object
  - Server returns binary
  - If it fails, client does compilation itself.
- Faster to compile a single source
- Parallelism possible.

C++ vs **Rust** vs **latest new thing**

<tug rant>

- A crap idea expressed in English is a crap idea.
- A crap idea expressed in French is still a crap idea.
- A new idea is better than a new language.
- distcc works if your source code is C/C++
- dist*rust*c not invented yet, so rust packages don’t benefit

</tug rant>

cross-compiler

- A compiler
  - takes source code
  - generates a sequence of machine code.
  - Having compiled it, you can run the binary.
- A cross-compiler
  - takes source code
  - generates a sequence of machine code, for an arbitrary CPU.
  - Having compiled it, you cannot run the binary.
- A cross-compiler is as fast as a native compiler
  - It generates a different output.
  - It doesn’t understand what it generates.

crossdev

- “a cross-compiler environment generator for Gentoo”
- https://wiki.gentoo.org/wiki/Project:Crossdev
- Build a C/C++ compiler for non-native binaries
- Faster than emulation
- Install crossdev on server
- Build required compilers
  - # crossdev -S -t aarch64-unknown-linux-gnu

gcc-config -l

```
[1] aarch64-unknown-linux-gnu-13 *

[2] armv6j-unknown-linux-gnueabihf-10
[3] armv6j-unknown-linux-gnueabihf-13 *

[4] armv6l-unknown-linux-gnueabihf-13 *

[5] armv7a-unknown-linux-gnueabihf-10
[6] armv7a-unknown-linux-gnueabihf-13 *

[7] avr-13 *

[8] x86_64-pc-linux-gnu-13 *
```

CFLAGS

- **-march=native**
  - distcc sees “native” and assumes “native to server”
  - Must pass real architecture
- Relationship between **mcpu**, **march**, and **mtune**… is complicated
  - X86 -**march** deprecated. Synonym for -**mtune**
  - Arm: -**mcpu** is -**march** + -**mtune**

CFLAGS

- run **resolve-march-native**
  - **arm6:** -march=armv6kz+fp
  - **aarch64:** -mcpu=cortex-a72 (tbd)
  - **x86 fitpc:** -march=bonnell -mno-cx16 --param=l1-cache-line-size=64 --param=l1-cache-size=24 –param=l2-cache-size=512
  - **amd64 laptop:** -march=btver2 --param=l1-cache-line-size=64 --param=l1-cache-size=32 --param=l2-cache-size=2048
  - **amd64 server:** -march=amdfam10 --param=l1-cache-line-size=64 --param=l1-cache-size=64 --param=l2-cache-size=512

Not everything runs perfectly...

- With Gentoo regular updates are important.
- Some package will not build
  - in a chroot.
  - with parallel compiles.
  - with distcc.

TODO

- Full QEMU VM for pi?
- Goal is to use dsh (distributed shell) to sequentially run processes across all systems.
  - Sync shared database
  - Start updates on all machines
  - Report failures.

The End – Try Gentoo. It’s fun!