MINCS

- The Container in the Shell (script) -

Masami Hiramatsu <masami.hiramatsu@linaro.org>
Tech Lead, Linaro Ltd.

Open Source Summit Japan 2017

LEADING
COLLABORATION
IN THE ARM
ECOSYSTEM

Who am I...

Masami Hiramatsu

- Linux kernel kprobes maintainer
- Working for Linaro as a Tech Lead

Demo

```
# minc top
# minc -r /opt/debian/x86_64
# minc -r /opt/debian/arm64 --arch arm64
```

What Is MINCS?

My Personal Fun Project

to learn how linux containers work :-)

What Is MINCS?

Mini Container Shell Scripts (pronounced ‘minks’)

- Container engine implementation using POSIX shell scripts
  - It is small (~60KB, ~2KLOC) (~20KB in minimum)
  - It can run on busybox
  - No architecture dependency (* except for qemu/um mode)
  - No need for special binaries (* except for libcap, just for capsh --exec)
- Main Features
  - Namespaces (Mount, PID, User, UTS, Net*)
  - Cgroups (CPU, Memory)
  - Capabilities
  - Overlay filesystem
  - Qemu cross-arch/system emulation
  - User-mode-linux
  - Image importing from dockerhub

  And all are done by CLI commands :-)

Why Shell Script?

That is my favorite language :-)

- Easy to understand for *nix administrators
  - Just a bunch of commands
- Easy to modify
  - Good for prototyping
- Easy to deploy
  - No architecture dependencies
  - Very small
  - Able to run on busybox (+ libcap is perfect)

MINCS Use-Cases

For Learning

- Understand how containers work

For Development

- Prepare isolated (cross-)build environment

For Testing

- Test new applications in isolated environment
- Test new kernel features on qemu using local tools

For products?

- Maybe good for embedded devices which has small resources

What Is A Linux Container?

There are many linux container engines

- Docker, LXC, rkt, runc, ...

They are using similar/same technologies provided by Linux kernel

- Namespace
- Cgroups
- Capabilities and/or LSM

They also need other common techniques

- Bind mount
- Layered (snapshot) file-system
- chroot/pivot_root

MINCS Internal

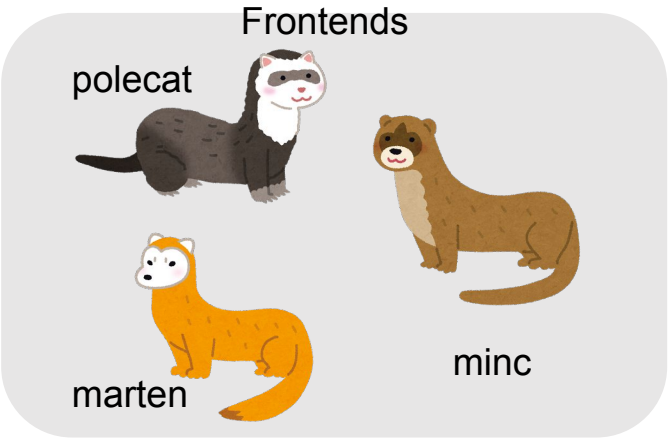
MINCS Design

Minc boot process step by step

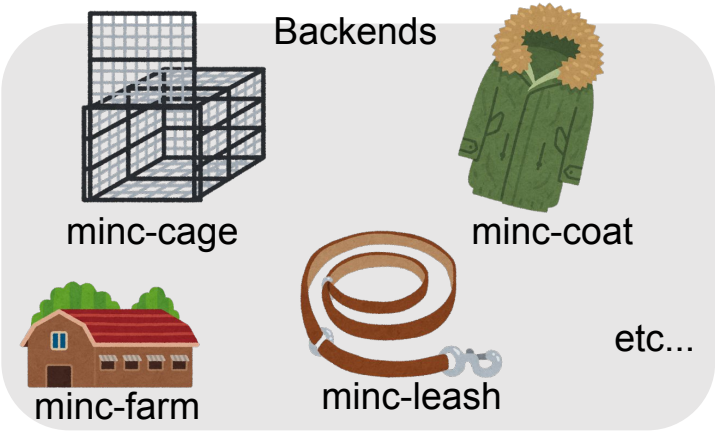
MINCS Design

MINCS has 2 layers

- Frontend Tools, parse options and run backend library scripts
  - **Minc**
  - Marten
  - Polecat
- Backend Library scripts, do actual work
  - Shell scripts start with minc-*, installed under **libexec/**



call



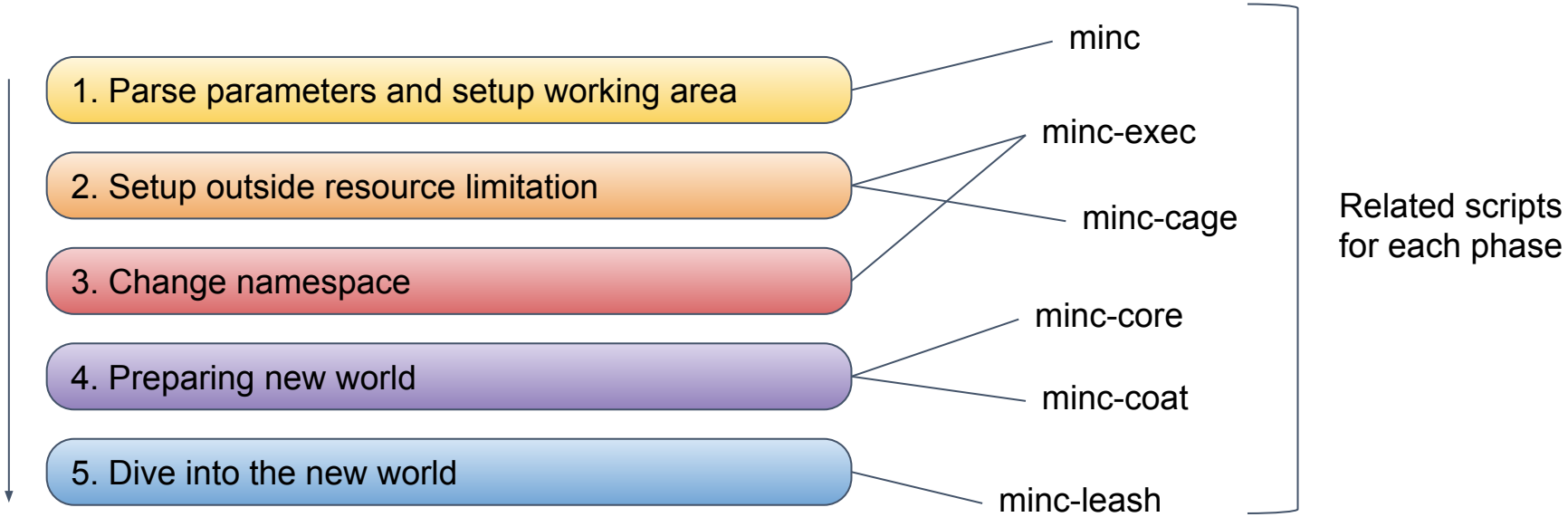
Overview of MINC boot process

Minc container takes 5 major steps to boot.

1. Parse parameters and setup working area
2. Setup outside resource limitation
3. Change namespace
4. Preparing new world
5. Dive into the new world

Overview of MINC boot process

Minc container takes 5 major steps to boot.



Structure: Building Container Like a Parfait!

Build it from bottom :)

Code commentary of MINCS

Let's see how minc boot into a container.

- Start from simplest case, and see how optional features are enabled.
- Not from the code, but from the execution log.

$ sudo minc --debug echo "hello mincs"

```
+ export MINC_DEBUG=1
+ [ 2 -ne 0 ]
+ cmd=echo
+ break
+ TRAPCMD=
+ [ -z  ]
+ :
+ : Setup temporary working directory for this container
+ :
+ [ -z  ]
+ mktemp -d /tmp/minc1505-XXXXXX
+ export MINC_TMPDIR=/tmp/minc1505-EaRzSD
+ :
+ : Trap the program exit and remove the working directory
+ :
```

Comments mostly explain what happens :-)

Step 1

Parse parameters and setup temporary working directory as below;

```
+ export MINC_DEBUG=1
+ [ 2 -ne 0 ]
+ cmd=echo
+ break
+ TRAPCMD=
+ [ -z  ]
+ :
+ : Setup temporary working directory for this container
+ :
+ [ -z  ]
+ mktemp -d /tmp/minc2798-XXXXXX
+ export MINC_TMPDIR=/tmp/minc2798-ZtvWh7
+ :
+ : Trap the program exit and remove the working directory
+ :
+ [ 0 -eq 0 ]
+ TRAPCMD=rm -rf /tmp/minc2798-ZtvWh7
+ trap rm -rf /tmp/minc2798-ZtvWh7 EXIT
+ trap  INT
+ /usr/local/libexec/minc-exec echo hello mincs
```

Make a directory and remove it when exit.

And call minc-exec as a child process

Step 2

Setup outside resource limitation (normally, minc does nothing.)

```
+ :
+ : Ensure parameters are set
+ :
+ test / -a -d /tmp/minc2798-ZtvWh7
+ [  ]
+ TRAPCMD=
+ IP_NETNS=
+ [  ]
+ /usr/local/libexec/minc-cage --prepare 2803
+ CAGECMD=
+ [  ]
+ :
+ : Prepare cleanup commands
+ :
+ trap  INT
+ trap rm -f /tmp/minc2798-ZtvWh7/pid; EXIT
```

Remove pid file after exit
(pid file will be made in phase4)

Step 3

Enter new namespace using “unshare” command

```
+ :
+ : Enter new namespace and execute command
+ :
+ UNSHARE_OPT=
+ [   ]
+ unshare -iumpf /usr/local/libexec/minc-core echo hello mincs
```

Invoke unshare with minc-core as a child process

At this moment, minc and minc-exec will wait for container exit as parent process

Minc
(grand parent)
Cleanup tempdir

Fork
& wait

minc-exec

(parent)
Cleanup pidfile

Unshare
& wait

minc-core

(current)
PID=1 in this
namespace

Step 4

Biggest part of this process, minc-core does the followings

1. Save PID in pidfile
2. Make a private mount namespace
3. Mount layered filesystem as a new rootfs
4. Setup new rootfs
    a. Bind user-defined mountpoints
    b. Prepare device files under /dev
    c. Prepare special files in /proc
    d. Prepare sysfs and tmpfs
5. Kick the minc-leash to phase-5

Step 4 - 1 Save PID in Pidfile

Access /proc/self to get self PID of outside of namespace (since $$ is 1)

```
+ :
+ : Get the PID in parent namespace from procfs
+ : (At this point, we still have the procfs in original namespace)
+ :
+ cut -f 4 -d  /proc/self/stat
+ export MINC_PID=2810
+ echo 2810
```

Get the PPID of ‘cut’ command
== PID of this script

NOTE: Until remounting /proc, original procfs instance is shown in new PID namespace.

Step 4 - 2 Make mount namespace private

Mount operation is shared across namespaces by default

- --make-rprivate makes it private recursively under given mountpoint

```
+ :
+ : Make current mount namespace private
+ :
+ mount --make-rprivate /
+ :
+ : Do not update /etc/mtab since the mount is private
+ :
+ export LIBMOUNT_MTAB=/proc/mounts
```

LIBMOUNT_MTAB env-var is used for updating mtab file, so it also should be hidden.

Step 4 - 3 Mount Layered Root Filesystem

Mount new rootfs under working directory using overlayfs

Tempdir

Rootdir

```
+ :
+ : Setup overlay rootfs by minc-coat
+ :
+ /usr/local/libexec/minc-coat bind /tmp/minc2798-ZtvWh7 /
[...]
+ :
+ : Make working sub-directories
+ : RD is mountpoint, UD is for upper layer, WD is working space
+ :
+ RD=/tmp/minc2798-ZtvWh7/root
+ UD=/tmp/minc2798-ZtvWh7/storage
+ WD=/tmp/minc2798-ZtvWh7/work
+ mkdir -p /tmp/minc2798-ZtvWh7/root /tmp/minc2798-ZtvWh7/storage
/tmp/minc2798-ZtvWh7/work
+ :
+ : Mount overlayed root directory
+ :
+ mount -t overlay -o
upperdir=/tmp/minc2798-ZtvWh7/storage,lowerdir=/,workdir=/tmp/minc2798-ZtvWh7/work
overlayfs /tmp/minc2798-ZtvWh7/root
```

Overlayfs requires upper, lower and workdir

Mounts given rootfs on tempdir/root

Step 4 - 4 Setup New Rootfs (1)

Setup /dev directory

```
+ :
+ : Prepare root directory
+ :
+ RD=/tmp/minc2798-ZtvWh7/root
+ mkdir -p /tmp/minc2798-ZtvWh7/root/etc /tmp/minc2798-ZtvWh7/root/dev
/tmp/minc2798-ZtvWh7/root/sys /tmp/minc2798-ZtvWh7/root/proc
[...]
+ :
+ : Make a fake /dev directory
+ :
+ mount -t tmpfs tmpfs /tmp/minc2798-ZtvWh7/root/dev
+ mkdir /tmp/minc2798-ZtvWh7/root/dev/pts
+ [  ]
+ mount devpts -t devpts -onoexec,nosuid,gid=5,mode=0620,newinstance,ptmxmode=0666
/tmp/minc2798-ZtvWh7/root/dev/pts
+ ln -s /dev/pts/ptmx /tmp/minc2798-ZtvWh7/root/dev/ptmx
+ :
+ : Bind fundamental device files to new /dev
+ :
+ bindmounts /dev/console /dev/null /dev/zero /dev/random /dev/urandom
```

Mount devpts for hide host pty

Step 4 - 4 Setup New Rootfs (2)

Setup /proc, /sys, /tmp and kick the minc-leash

Make original procfs readonly

```
+ :
+ : Do not bind procfs, since it shows outside pids
+ :
+ mount -t proc -o ro,nosuid,nodev,noexec proc /proc
+ mount -t proc -o rw,nosuid,nodev,noexec,relatime proc /tmp/minc2798-ZtvWh7/root/proc
+ bindmounts /proc/sys /proc/sysrq-trigger /proc/irq /proc/bus
[...]
+ [ -z  ]
+ bindmount /sys
+ test -e /sys
+ touch /tmp/minc2798-ZtvWh7/root/sys
+ mount --bind /sys /tmp/minc2798-ZtvWh7/root/sys
+ :
+ : /tmp is used for application working area in container
+ :
+ mount -t tmpfs tmpfs /tmp/minc2798-ZtvWh7/root/tmp
+ :
+ : Exec leash (chroot/capsh) to run command in new rootfs
+ :
+ exec /usr/local/libexec/minc-leash /tmp/minc2798-ZtvWh7/root echo hello mincs
```

These files are read-only

Step 5 Dive Into the New World (1)

Preparing Capsh (or chroot) options

```
+ which capsh
+ :
+ : Drop cap_sys_chroot to prohibit chroot-breakout
+ :
+ MINC_DROPCAPS=,cap_sys_chroot
+ :
+ : Check capsh --exec is supported
+ :
+ CAPSH_EXEC=
+ capsh -h
+ grep -q ^ *--exec
[...]
+ [ -z ,cap_sys_chroot ]
+ :
+ : If we need to drop capabilities, use capsh to run given command
+ :
+ RUN= capsh --chroot=/tmp/minc2798-ZtvWh7/root  --drop=,cap_sys_chroot --
```

Root user can break out chroot easily with chroot
http://pentestmonkey.net/blog/chroot-breakout-perl

Step 5 Dive Into the New World (2)

Cleanup environment variables, and launch it

- Capsh (capability shell wrapper) is the key to **"cap"** the container (Not to escape outside)

```
+ :
+ : Wash out the environment variables for MINCS
+ :
+ wash ^MINC_
+ env
+ + grep ^MINC_
cut -f 1 -d=
+ unset MINC_DEBUG_PREFIX
[...]
+ unset MINC_DIRECT
+ :
+ : If capsh does not support --exec, run it with sh -c
+ :
+ exec capsh --chroot=/tmp/minc2798-ZtvWh7/root --drop=,cap_sys_chroot -- -c exec echo
hello mincs
hello mincs
```

"exec" makes "echo" PID = 1

Options of minc

There some major options for minc

--root DIR

- Specify rootfs instead of ‘/’

--direct

- Don’t use overlayfs

--tempdir DIR

- Use given directory instead of “mktemp -d”

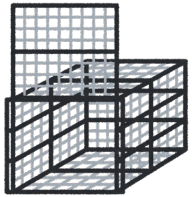
--keep

- Do not remove the tempdir when exit

--bind DIR1:DIR2

- Bind host directory (DIR1) to container directory (DIR2)

Options: cgroups resource limitation



Use **minc-cage** to make CPU/memory Cgroups

```
# minc --debug --mem-limit 512M echo hello minc
[...]
+ /usr/local/libexec/minc-exec echo hello minc
[...]
+ /usr/local/libexec/minc-cage --prepare 32547
[...]
+ TRAPCMD=cleanup_cages  /sys/fs/cgroup/memory/mincs/minc32544-F8yU4K;
+ :
+ : Setup a "cage" for the container
+ :
+ . /usr/local/libexec/minc-cage
[...]
+ :
+ : Set cgroups memory limit
+ :
+ test -d /sys/fs/cgroup/memory/mincs/minc32544-F8yU4K
+ echo 512M
+ [  ]
+ echo 32547
```

Prepare cleanup command for used cgroups (executed at exit)

Written to **$MEMCG/memory.limit_in_bytes**

Written to **$MEMCG/tasks**

Options: --cross (or --arch)

This option allows to run cross-arch container using qemu-user-static

- Ex) Run aarch64 container rootfs on x86_64 Linux

```
# minc --debug -r /opt/debian/arm64 --cross arm64 echo hello minc
[...]
+ export MINC_ARCH=aarch64
+ grep interpreter /proc/sys/fs/binfmt_misc/qemu-aarch64
+ cut -f 2 -d
+ export MINC_CROSS_QEMU=/usr/bin/qemu-aarch64-static
[...]
+ :
+ : Bind qemu-user-mode for cross-arch container
+ :
+ mkdir -p /tmp/minc625-8ug6g3/root/usr/bin/
+ [ -x /tmp/minc625-8ug6g3/root//usr/bin/qemu-aarch64-static ]
+ bindmount /usr/bin/qemu-aarch64-static
+ test -e /usr/bin/qemu-aarch64-static
+ touch /tmp/minc625-8ug6g3/root/usr/bin/qemu-aarch64-static
+ mount --bind /usr/bin/qemu-aarch64-static
/tmp/minc625-8ug6g3/root/usr/bin/qemu-aarch64-static
```

In minc, find appropriate qemu-user-static from binfmt_misc

In minc-core, **bind** the interpreter binary on new rootfs (not copy)

Ermine: --qemu and --um options

Run a container in qemu-system or on user-mode linux. (like as clear container)

Enter qemu after Step 4 - 3 (mount layered filesystem),

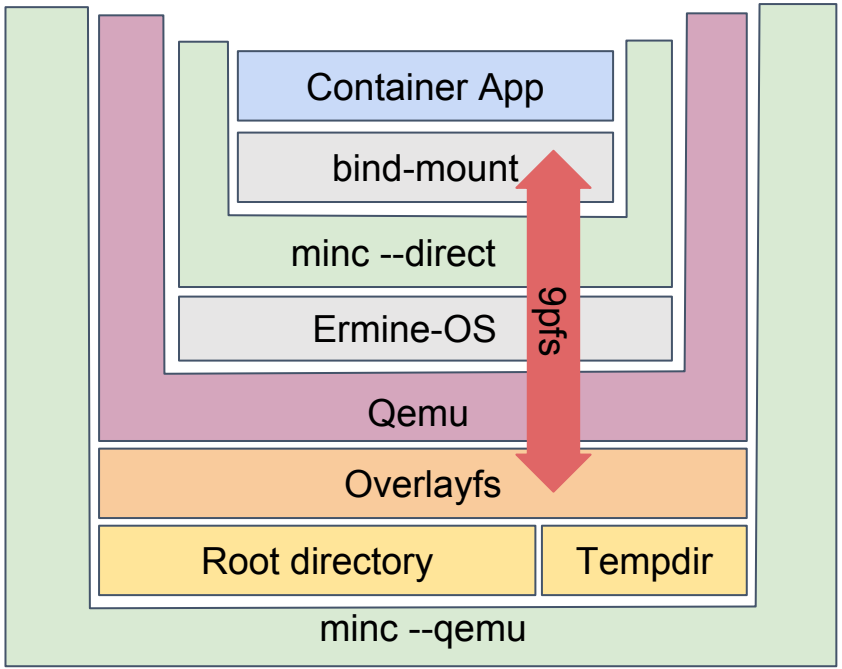
```
+ :
+ : Enter qemu-system or user-mode-linux container (ermine)
+ :
+ . /usr/local/libexec/minc-moult
+ hostname
+ MINC_GUEST_OPT=-r /mnt/root --name devnote
+ :
+ : Since host mounts overlayfs on rootfs, guest skips it.
+ :
+ MINC_GUEST_OPT=-r /mnt/root --name devnote -D --debug
+ :
+ : Prepare run.sh for qemu/um internal container
+ :
+ echo #!/bin/sh
+ tput lines
+ tput cols
+ echo stty rows 24; stty cols 80
+ echo minc -r /mnt/root --name devnote -D --debug "echo hello world"
+ minc_moult x86_64 /tmp/minc2798-ZtvWh7 ro quiet
```

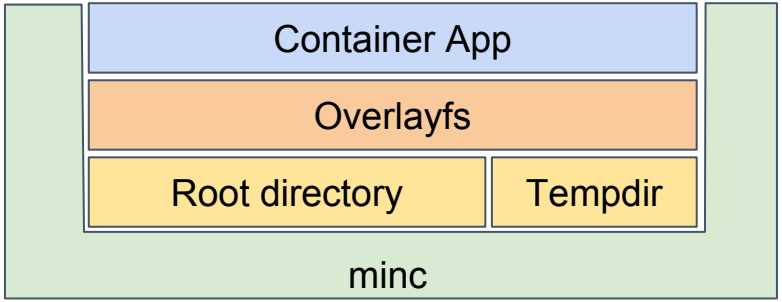
These commands are executed in qemu as a shell script

Ermine Structure

Ermine's layered fs is in host-side

- Use 9pfs(virtio-9p) to pass the layered rootfs to container in qemu
- You can reuse same rootfs and tempdir on minc





User can share the same rootfs image

Why Ermine?

Ermine changes its fur by season

Summer

By Steve Hillebrand, USFWS [Public domain], via Wikimedia Commons

Moult

Winter

By 4028mdk09 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Ermine in MINCS

Change its kernel by arch: minc with --qemu and --cross option

X86
(host)

By Steve Hillebrand, USFWS [Public domain], via Wikimedia Commons

minc-moult

Arm
(Guest qemu)

By 4028mdk09 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

ermine-breeder

Minimum root system (Ermine-OS) build script

- Rootfs(initramfs) and kernel for qemu
  - Linux + Busybox + libcap (for capsh) + MINCS
- Download, configure, build and install are done by one command.
  - And user-configurable.
- For x86, tinyconfig make it smaller than **4MB** in total
- X86-64, i386, arm, arm64, um (user-mode linux) are supported

E.g. build ermine images for arm64

```
# ermine-breeder build --arch arm64
```

And test to run (for debugging)

```
# ermine-breeder testrun --arch arm64
```

Other tools in MINCS

- marten
  - Docker-like image management
- polecat
  - Make a minimum application container

marten

Marten : **M**anager of image **ar**chives and **te**mporary-co**n**tainers

- Maintain docker-like container images(rootfs) and containers(tempdir) based on UUID and name
- Pull/import images from docker
  - Easy to setup new container, and able to use many distro images
- Minc command can use these images/containers with UUID

Examples:

```
# marten pull fedora
Pulled. Importing image: library/fedora
691bc14ee27487db536172a1fcdbbf956f460d1e1e1b201828e3a2bab81c5ec8
# marten li
691bc14ee274   249M fedora
# minc -r fedora
e46748c6b2c6196e426601de4650f0f76d8df0ad9c1a32eafdb8618162161412
bash-4.3# cat /etc/fedora-release
Fedora release 25 (Twenty Five)
```

polecat

Polecat: **Po**rtab**le** **C**ontainer'ed **A**pplica**t**ion

- Shrink down a container based on binary/file dependencies
- Pack it on squashfs and add self-executable script header
  - which can directly mount on host, no need to expand on temporary directory

Example:

```
# polecat -o top.sh /opt/debian/x86_64 top
(hit 'q' to quit top)
Install /sbin/capsh
...
Done:1.6M /home/mhiramat/ksrc/mincs/top.sh
# ./top.sh
```

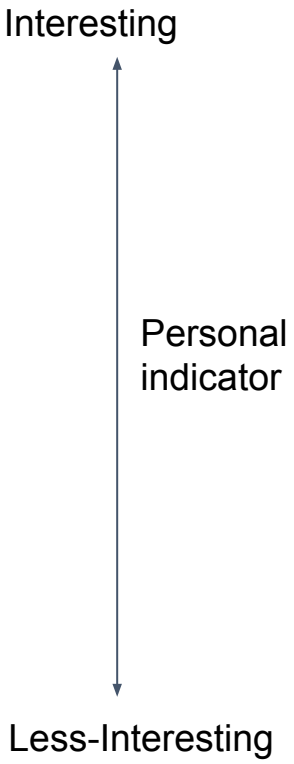
```
top - 09:42:43 up 5 days,  5:59,  0 users,  load average: 0.39, 0.62, 0.63
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.7 us,  1.7 sy,  0.0 ni, 93.7 id,  0.9 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  16150776 total, 15351268 used,   799508 free,     4864 buffers
KiB Swap:        0 total,        0 used,        0 free.  9350956 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   21952   2472   2096 R   0.0  0.0   0:00.05 top
```

Summary

- MINCS itself is a small and portable container engine
- Explained how to make a container engine
  - You can make your own container engine with your favorite language!
- Ermine allows you to run container in qemu
- Useful tools are included :)

Future work

- Improve network namespace
  - Firewalld/iptables would cause a headache...
- Networking support in qemu container
- Background mode support
- Improve non-privilege mode (user-ns, shiftfs)
- Non-x86 host --qemu mode (Aarch64)
- Support new arch
  - OpenRISC? MIPS? POWER?
- Better capsh is needed
  - Seccomp support
- Container package encrypt/signing
- selinux/AppArmor support
- OCI standard support



Questions?

Thank you!

Please visit https://github.com/mhiramat/mincs and fork it!

Backup slides

Libcap enhancements

For better supporting MINCS, I've also made enhancements on libcap

- Use **$SHELL** environment variable instead of /bin/bash
  - It is needed for some container image which has only /bin/sh
- **--exec** option support, which invokes command directly by execve(2)
  - No more need /bin/sh, run command directly in the container

You can get the patches here

- https://github.com/mhiramat/libcap
- Already sent to libcap's author (because there is no mailing list…), and have no response yet.

Kernel testing with Ermine-breeder

Ermine-breeder testrun command does;

1. Boot kernel with console output
2. Mount ./work directory
3. Run ./work/run.sh if exists

So, if we put a test script as ./work/run.sh, it automatically runs the test on qemu.

I'm usually running ftracetest on new kernel using ermine-breeder :-)

E.g.

```
# ermine-breeder build --config samples/ermine/ftracetest.config
# cp -r work/linux/linux*/tools/testing/selftests/ftrace work/
# cp samples/run/ftracetest.sh work/run.sh
# ermine-breeder testrun
```

Container is Yet Another SAL

System Abstraction Layer

- It doesn't "abstract" system, but keep it compatible
  - Containers can work on docker installed system
- You can package a software with its dependencies by container
  - Including old vulnerable libraries…
- It's not perfect solution, but very handy
  - NO software changes required
  - You can just PULL the image and run it
- Anyway, if there are enough amount of users, it becomes better and better.
  - Like Linux is...

Pet or Cattle?

Of course MINC is Pet model, or Wild Life :-)

The Background of MINCS

When I met the Docker (2014), I started to learn the backend technology of the container.

It was using several technologies provided by linux kernel itself.

- Namespace
- Cgroups
- Capabilities and/or LSM
- Layered filesystem (aufs)

And found “aufs” and “dockerhub” are special feature of the Docker. Except for aufs, it seems to be able to make similar container from command line.

And I met overlayfs (2015) on upstream kernel

It’s Time to Try

I’ve started to learn how to make namespaces using unshare and overlayfs.

- At this moment, it is called “chns”, like chroot.
- This is just like “bocker” (docker implementation by bash script)

I continued to implement features which I hit or got from other projects.

- Minimum application self-executable container
- Multi-layer filesystem image management from docker
  - Finally it is able to import image from dockerhub directly.
- Cross-arch container
- Qemu-based container from ClearLinux
  - UML-based container too

Additional script: build-debian-rootfs.sh

Shell-script to install debian rootfs using debootstrap

- This supports cross-arch (arm, arm64, i386, ppc64) install
- Also supports debian version and additional package

Example:

```
# mkdir -p /opt/debian/arm64
# samples/scripts/build-debian-rootfs.sh /opt/debian/arm64 --arch arm64 --deb stretch
I: Keyring file not available at /usr/share/keyrings/debian-archive-keyring.gpg; switching to
https mirror https://mirrors.kernel.org/debian
…
I: Configuring ca-certificates...
I: Base system installed successfully.
# ls /opt/debian/arm64/
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
# minc -r /opt/debian/arm64 --cross arm64 uname -a
Linux devnote 4.10.6 #40 SMP Wed Mar 29 16:43:19 JST 2017 aarch64 GNU/Linux
```