Chromium OS Build system

Mitsuhiro Kimura
TOSHIBA CORPORATION

2012/ 3/23
CELF Japan Technical Jamboree #40
Chrome OS History

2009.07.07 Chrome OS announcement

2009.11.19 source code exposure

2010.12.07 Cr-48 reference laptop (not retail sales)

2011.05.11 Chromebook (Acer, Samsung) at Google I/O

Changed significantly especially build system

2010.02 “徹底解説ChromeOSの全貌” by 日経Linux
Chromium

- Chromium is an open source version of Chrome
- The Chromium Projects include Chromium browser and Chromium OS

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Official</td>
<td>Chrome</td>
<td>Chrome OS</td>
</tr>
<tr>
<td>Open source</td>
<td>Chromium</td>
<td>Chromium OS</td>
</tr>
</tbody>
</table>

http://www.chromium.org/chromium-os/chromium-os-faq
Chromium

• Chromium is an open source version of Chrome
• The Chromium Projects include Chromium browser and Chromium OS

<table>
<thead>
<tr>
<th>Browser</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Official</td>
<td>Chrome</td>
</tr>
<tr>
<td>Open source</td>
<td>Chromium</td>
</tr>
</tbody>
</table>

Today’s topic
Information

• The Chromium Projects (http://www.chromium.org/)

• Chromium OS (http://www.chromium.org/chromium-os)
  • Developer Guide
  • FAQ
  • Overview of the source

• Gitweb (http://git.chromium.org/)
  • Repo + Git

• developer wiki (http://code.google.com/p/chromium-os/)
  • for developer-contributed or unofficial content about the Chromium OS project
Version

Active branches: Stable/Beta/Developer preview

Stable: 0.16.1193.194.0 (2012/1/24)
Beta: 0.18.1660.34.0 (2012/2/16)
Dev: 0.18.1660.20.0 (2012/2/12)

<MAJOR.MINOR.BUILD.PATCH>

(0.16.1193.194: MAJOR=0, MINOR=16, BUILD=1193, PATCH=194)

MAJOR and MINOR track updates to the Google Chrome stable channel.

Each branch version grows as Dev → Beta → Stable

http://www.chromium.org/releases/version-numbers

This presentation is based on the survey about 0.15.877
Features

- Log in to your Google account
  - Multiple people can log in alternately to one device
- Chrome Browser is started after login
  - Available web application by Chrome Web Store
  - Share bookmark across multiple devices
- Mechanism of fast boot
  - Chrome OS original BIOS
Architecture

- Specializes in operating Chrome browser
  - improved boot performance by removing a lot of complexity that is normally found in PC firmware

http://www.chromium.org/chromium-os/chromiumos-design-docs/software-architecture
**Topic**

- Chromium OS Build system
  - Get toolchain ~ Build image

<table>
<thead>
<tr>
<th>Recommended environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 10.04 (Lucid) 64bit</td>
</tr>
<tr>
<td>9.10(Karmic) is not to work</td>
</tr>
<tr>
<td>4GB RAM</td>
</tr>
<tr>
<td>linking the browser uses 4GB</td>
</tr>
<tr>
<td>4GB USB memory</td>
</tr>
<tr>
<td>For writing a boot image</td>
</tr>
<tr>
<td><strong>sudo access</strong></td>
</tr>
<tr>
<td>to run <strong>chroot</strong></td>
</tr>
</tbody>
</table>

Getting source/toolchain

```bash
$ mkdir chromiumos
$ cd chromiumos
$ repo init -u
  https://git.chromium.org/chromiumos/manifest.git
  -m minilayout.xml
  --repo-url https://git.chromium.org/external/repo.git
$ repo sync
$ ls
AUTHORS LICENSE chromite/ src/
$ ls src/
overlays/platfrom/ repohooks/ scripts/ third_party/
$ ls src/third_party/
chromiumos-overlay/ portage/ portage-stable/
```

Download minimum src/tool
Other can get when you need
Chromium OS build steps

1. $ chromite/bin/cros_sdk
   To make sure everyone uses the same exact environment and tools chroot

2. (cr) ./setup_board --board=x86-generic --default
   Initialize the build for a board

3. (cr) ./build_packages
   The rough equivalent of make all in a standard Makefile system

4. (cr) ./build_image --noenable_rootfs_verification
   Build a disk image for a board

5. (cr) ./image_to_usb.sh --to=/dev/sdc
   Put a disk image on a USB disk
1. cros_sdk

To make sure everyone uses the same exact environment and tools chroot.

At first time, this command downloads sdk (521MB) and creates the chroot.

Second and subsequent, check update of sdk and enter the chroot. (and you’ll be in the ~/trunk/src/scripts)

http_proxy is taken over when created chroot by cros_sdk
ftp_proxy, all_proxy are also. But no_proxy is not.
http_proxy is necessary to download sdk or packages through proxy.
If you share packages inside proxy network at the same time, no_proxy also necessary.
Patch for `no_proxy`

Take over `no_proxy` in the same way as `http_proxy`, `ftp_proxy`, `all_proxy`

```
src/scripts/sdk_lib/make_chroot
170 bash_chroot "echo Defaults env_keep += http_proxy >> /etc/sudoers"
171 bash_chroot "echo Defaults env_keep += ftp_proxy >> /etc/sudoers"
172 bash_chroot "echo Defaults env_keep += all_proxy >> /etc/sudoers"
173 bash_chroot "echo Defaults env_keep += no_proxy >> /etc/sudoers"
174 bash_chroot "echo %adm ALL=(ALL:) ALL >> /etc/sudoers"
175 bash_chroot "echo root ALL=(ALL:) ALL >> /etc/sudoers"
176 bash_chroot "echo $USER ALL=NOPASSWD: ALL >> /etc/sudoers"
```

```
src/scripts/sdk_lib/enter_chroot.sh
389 CHROMEOS_VERSION_DEVSERVER=${CHROMEOS_VERSION_DEVSERVER}"
390
391 # Pass proxy variables into the environment.
392 for type in http_proxy ftp_proxy all_proxy GIT_PROXY_COMMAND GIT_SSH; do
393     value=$(env | grep ${type}_proxy || true)
394     if [ -n "${value}" ]; then
395         CHROOT_PASSTHRU="${CHROOT_PASSTHRU} ${type}=${value}"
396     fi
397 done
398 for type in http_proxy ftp_proxy all_proxy no_proxy GIT_PROXY_COMMAND GIT_SSH; do
399     value=$(env | grep ${type}_proxy || true)
400     if [ -n "${value}" ]; then
401         CHROOT_PASSTHRU="${CHROOT_PASSTHRU} ${type}=${value}"
402     fi
403 done
```
2. setup_board

Choose a board for your first build in src/overlays/
Built packages put into chroot/build/<board>

- x86-generic
  - board-overlay
  - Built packages

- arm-generic
  - board-overlay
  - Built packages

- x86-pineview
  - board-overlay
  - Built packages

- tegra2
  - board-overlay
  - Built packages

configuration files
Use for build
Boot image
board, variant

Different classes of computers are referred to by Chromium OS as different target "boards"

- board
- variant
  variant of board

ex.)
- arch: x86, arm
- board: x86-pineview, tegra2, x86-generic, arm-generic
- variant: tegra2-seaboard, tegra2-dev-board

cf. src/overlays/README
overlay

- Separate common config, arch, board

Easily support different models
Avoid redundant description
Example

• x86-generic overlay

- Common
  - Compiler
  - Libc
- x86
  - SSE
  - VIDEO_CARDS
- x86-generic
  - Kernel config
  - Add
  - VIDEO_CARDS

Overlay
overlay-x86-generic/make.conf

VIDEO_CARDS = VIDEO_CARDS + intel, nouveau, radeon

CHROMEOS_KERNEL_SPLITCONFIG="chromiumos-i386"

# PORTAGE_BINHOST is pulled in from prebuilt.conf
source prebuilt.conf

VIDEO_CARDS="${VIDEO_CARDS} intel nouveau radeon"

MARCH_TUNE="-march=core2 -mfpmath=sse"
CFLAGS="-O2 -pipe ${MARCH_TUNE} -ggdb"
CXXFLAGS="${CFLAGS}"
LDFLAGS=""
# Copyright (c) 2009 The Chromium OS Authors. All rights reserved.
# Use of this source code is governed by a BSD-style license that can be
# found in the LICENSE file.

(...omission...)

# Recommended x86-specific USE flags.
USE="_${USE} mmx sse sse2 dri hardened"

# Recommended MARCH_TUNE, CFLAGS, etc.
MARCH_TUNE="-march=atom -mtune=atom -mfpmath=sse"
CFLAGS="-O2 -pipe ${MARCH_TUNE} -ggdb"
CXXFLAGS="${CFLAGS}"
LDFLAGS=""

VIDEO_CARDS="intel vmware vesa"
INPUT_DEVICES="evdev synaptics"

# Allow a board to override or define additional settings.
source make.conf.board
## Linux Kernel in Chromium OS

**Version:** 2.6.38.3 (Chromium OS version 0.15.877)

**Kernel src:** src/third_party/kernel/files/

**config:** files/chromeos/config/  
(Hierarchical)

<table>
<thead>
<tr>
<th>Family</th>
<th>Architecture</th>
<th>Flavour</th>
</tr>
</thead>
<tbody>
<tr>
<td>(common)</td>
<td></td>
<td>(platform)</td>
</tr>
</tbody>
</table>

**script:** files/chromeos/scripts/

- kernelconfig
- prepareconfig
- splitconfig

## Kernel config hierarchy

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ChromeOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>config.common.chromeos</td>
<td>x86_64</td>
<td>pineview</td>
</tr>
<tr>
<td></td>
<td>config.common.x86_64</td>
<td>config.flavour.chromeos-intel-pineview</td>
</tr>
<tr>
<td></td>
<td>i386</td>
<td>pinettrail</td>
</tr>
<tr>
<td></td>
<td>config.common.i386</td>
<td>config.flavour.chromeos-pinettrail-i386</td>
</tr>
<tr>
<td></td>
<td>armel</td>
<td>chromeos-arm</td>
</tr>
<tr>
<td></td>
<td>config.common.armel</td>
<td>config.flavour.chromeos-arm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chromiumos-arm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>config.flavour.chromiumos-arm</td>
</tr>
</tbody>
</table>

Edit kernel config

1. Look for the kernel config option you want to edit
   If you want to add new option, edit Flavour.

2. Edit that file to change the config

3. Run script to recreate configs based on your changes
   $ kernelconfig oldconfig
3. build_packages

- Portage of Gentoo Linux is adopted
- Pre-built packages are distributed

build_packages \(\rightarrow\) emerge

\[\text{ChromiumOS}\]

\[\text{kernel} \quad \text{browser} \quad \text{Wifi}\]

\[\text{X window} \quad \text{adobe-flash}\]

Can skip the download / compile source code
Can be compiled code only the parts that need to change
3. build_packages

It will take a long time in the first time you run

It must build every package, and also download about 1.7GB of source packages and 1.3GB of binary packages.

(around 90 minutes on a four core machine)

By default, packages other than Chrome will be compiled in debug mode.

For remove debugging constructs, supply --nowithdebug

Built packages put into chroot/build/<board>

x86-generic

board-overlay

Built packages

chroot/build/x86-generic
### 4. build_image

Build a disk image for your board

Supply argument to specify the type of image to build

Lower order of priority of options

<table>
<thead>
<tr>
<th>kind</th>
<th>feature</th>
<th>argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer image</td>
<td>Include developer packages</td>
<td>(default)</td>
</tr>
<tr>
<td>Pristine Image</td>
<td>Without developer packages</td>
<td>--nowithdev</td>
</tr>
<tr>
<td>Test image</td>
<td>Start sshd on boot</td>
<td>--test</td>
</tr>
<tr>
<td></td>
<td>Can run run_remote_tests.sh through the ethernet.</td>
<td></td>
</tr>
<tr>
<td>Factory image</td>
<td>Run factory test automatically</td>
<td>--factory</td>
</tr>
<tr>
<td></td>
<td>Include packages that Test image include.</td>
<td></td>
</tr>
<tr>
<td>Factory install image</td>
<td>Install factory image to device</td>
<td>--factory_install</td>
</tr>
</tbody>
</table>

`--noenable_rootfs_verification` turns off verified boot allowing you to freely modify the root file system at the time of version 0.15.877
4. build_image

Build a disk image for your board
Supply argument to specify the type of image to build

<table>
<thead>
<tr>
<th>kind</th>
<th>feature</th>
<th>argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer image</td>
<td>Include developer packages</td>
<td>dev</td>
</tr>
<tr>
<td>Pristine Image</td>
<td>Without developer packages</td>
<td>base</td>
</tr>
<tr>
<td>Test image</td>
<td>Start sshd on boot</td>
<td>test</td>
</tr>
<tr>
<td></td>
<td>Can run run_remote_tests.sh through the ethernet.</td>
<td></td>
</tr>
<tr>
<td>Factory image</td>
<td>Run factory test automatically</td>
<td>factory_test</td>
</tr>
<tr>
<td></td>
<td>Include packages that Test image include.</td>
<td></td>
</tr>
<tr>
<td>Factory install image</td>
<td>Install factory image to device</td>
<td>factory_install</td>
</tr>
</tbody>
</table>

--noenable_rootfs_verification

turns off verified boot allowing you to freely modify the root file system

Copyright 2012, Toshiba Corporation.
5. image_to_usb.sh

Put your image on a USB disk

(cr) ./image_to_usb.sh --to=/dev/sdc

Boot from your USB disk

after that, you can install to HDD by
/usr/sbin/chromeos-install from console

(Another way) Build an image to run in a virtual machine

(cr) ./image_to_vm.sh
Install to HDD

If your image includes developer packages, you can open console by ‘Ctrl+Alt+F2’, and install image to HDD by run `/usr/sbin/chromeos-install`.

`build_image --nowithdev`

not include the developer packages

→ can not open console

→ can not install image to HDD

   by `/usr/sbin/chromeos-install` from console

→ use factory_install image

This operation wipes HDD clean
Factory_install image

Prepare 3 images
1. factory_install image
2. factory_test image
3. release image (finally installed image)

A) Make a image that integrates the three images
   (cr) ./make_factory_packages.sh
   --usbimg output_image
   --install_shim factory_install_image
   --factory factory_test_image
   --release release_image

B) Prepare the server that provide a image to be installed
   (cr) ./make_factory_packages.sh
   --factory factory_test_image
   --release release_image

Start server: python2.6 devserver.py --factory_config miniomaha.conf

Boot target with factory_install image
Factory_install image

Prepare 3 images
1. factory_install image
2. factory_test image
3. release image (finally installed image)

A) Make a image that integrates the three images
   (cr) ./make_factory_packages.sh
   --usbimg output_image
   --install_shim factory_install_image
   --factory factory_test_image
   --release release_image

B) Prepare the server that provide a image to be installed
   (cr) ./make_factory_packages.sh
   --factory factory_test_image
   --release release_image
   Start server: python2.6 devserver.py --factory_config miniomaha.conf

Boot target with factory_install image

If test fails, release image is not installed

Not support legacy BIOS (0.15.877)
Factory_install image

Prepare 3 images
1. factory_install image
2. factory_test image
3. release image (finally installed image)

A) Make a image that integrates the three images
   (cr) ./make_factory_packages.sh
   --usbimg output_image
   --install_shim factory_install_image
   --factory factory_test_image
   --release release_image

B) Prepare the server that provide a image to be installed.
   (cr) ./make_factory_packages.sh
   --factory factory_test_image
   --release release_image
   Start server: python2.6 devserver.py --factory_config miniomaha.conf

Boot target with factory_install image

You can write any image to HDD instead of factory test image.

If test failed, Don’t install release image
## Partitions of boot image

<table>
<thead>
<tr>
<th>partition</th>
<th>type</th>
<th>purpose</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ext3</td>
<td>Stateful partition</td>
<td>1GB</td>
</tr>
<tr>
<td>2</td>
<td>kernel A</td>
<td>(bootloader &amp; kernel for Chrome OS BIOS)</td>
<td>16MB</td>
</tr>
<tr>
<td>3</td>
<td>ext2</td>
<td>rootfs A (include kernel for EFI BIOS)</td>
<td>858MB</td>
</tr>
<tr>
<td>4</td>
<td>kernel B</td>
<td>(use for upgrade)</td>
<td>16MB</td>
</tr>
<tr>
<td>5</td>
<td>rootfs B</td>
<td>(use for upgrade)</td>
<td>512B</td>
</tr>
<tr>
<td>6</td>
<td>kernel C</td>
<td>(placeholder for future use only)</td>
<td>512B</td>
</tr>
<tr>
<td>7</td>
<td>rootfs C</td>
<td>(placeholder for future use only)</td>
<td>512B</td>
</tr>
<tr>
<td>8</td>
<td>OEM Customization</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Future use                                                   Empty when you create the image</td>
<td>512B</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Future use                                                   Empty when you create the image</td>
<td>512B</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Read/Write firmware                                              Empty when you create the image</td>
<td>8MB</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>vfat</td>
<td>EFI System Partition (temporary)</td>
<td>16MB</td>
</tr>
</tbody>
</table>

Cf. src/scripts/chromeos-common.sh
# Partitions of boot image

<table>
<thead>
<tr>
<th>partition</th>
<th>type</th>
<th>purpose</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ext3</td>
<td>Stateful partition</td>
<td>1GB</td>
</tr>
<tr>
<td>2</td>
<td>kernel A</td>
<td>(bootloader &amp; kernel)</td>
<td>16MB</td>
</tr>
<tr>
<td>3</td>
<td>ext2</td>
<td>rootfs A (include kernel for EFI BIOS)</td>
<td>858MB</td>
</tr>
<tr>
<td>4</td>
<td>kernel B</td>
<td>(use for upgrade)</td>
<td>16MB</td>
</tr>
<tr>
<td>5</td>
<td>rootfs B</td>
<td>(use for upgrade)</td>
<td>512B</td>
</tr>
<tr>
<td>6</td>
<td>kernel C</td>
<td>(placeholder for future use only)</td>
<td>512B</td>
</tr>
<tr>
<td>7</td>
<td>rootfs C</td>
<td>(placeholder for future use only)</td>
<td>512B</td>
</tr>
<tr>
<td>8</td>
<td>OEM Customization</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Future use</td>
<td>512B</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Future use</td>
<td>512B</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Read/Write firmware</td>
<td>8MB</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>vfat</td>
<td>EFI System Partition (temporary)</td>
<td>16MB</td>
</tr>
</tbody>
</table>

- Use for factory_test image
- Use for release image
- Empty when you create the image
- Cf. src/scripts/chromeos-common.sh
Boot with NFSROOT

- Boot from USB disk + use NFSROOT as rootfs
  - use chroot/build/<board> as NFSROOT

- Necessary operation:
  - Exclude the impact of building factory_test packages (later)
  - add `CONFIG_IP_PNP=y, CONFIG_IP_PNP_DHCP=y, CONFIG_R8169=y` to src/third_party/chromiumos-overlay/sys-kernel/chromeos-kernel/files/nfs.config
  - edit script so that you can specify kernel cmdline `root=/dev/nfs` (later)
  - (cr) `USE=“nfs” ./build_image --boot_args=“noinitrd pci=noacpi rw nfsroot=xxx.xxx.xxx.xxx:/path_to_chroot/chroot/build/<board> ip= dhcp” --noenable_rootfs_verification`
  - disable iptables (later)

<table>
<thead>
<tr>
<th>content of --boot_args</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pci=noacpi</td>
<td>Need to be recognized NIC Root cause is unknown</td>
</tr>
<tr>
<td>rw</td>
<td>mount rootfs with readable/writeable</td>
</tr>
<tr>
<td>xxx.xxx.xxx.xxx</td>
<td>IP address of NFS server</td>
</tr>
<tr>
<td>ip=dhcp</td>
<td>get IP address by DHCP</td>
</tr>
</tbody>
</table>
--- a/create_legacy_bootloader_templates.sh
+++ b/create_legacy_bootloader_templates.sh
@@ -111,6 +111,13 @@ EOF
   fi
   info "Emitted $ {SYSLINUX_DIR} /default.cfg"

+ if [[ ${common_args} == *nfsroot=* ]]; then
+   FLAGS_usb_disk="/dev/nfs"
+   usb_root="/dev/nfs"
+ else
+   usb_root="/dev/sdb3"
+ fi
+
+ cat <<EOF | sudo dd of="${SYSLINUX_DIR}/usb.A.cfg" 2>/dev/null
  label chromeos-usb.A
  menu label chromeos-usb.A
@@ -191,7 +198,7 @@ menuentry "verified image B" {

    # FIXME: usb doesn't support verified boot for now
    menuentry "Alternate USB Boot" {
-      linux (hd0,3)/boot/vmlinuz ${common_args} root=/dev/sdb3 i915.modeset=1 cros_efi
+      linux (hd0,3)/boot/vmlinuz ${common_args} root=${usb_root} i915.modeset=1 cros_efi
    }
EOF
   if [[ ${FLAGS_enable_rootfs_verification} -eq ${FLAGS_TRUE} ]]; then

specify kernel cmdline
root=/dev/nfs
Exclude the impact of building factory_test packages

--withfactory options in build_packages is enabled by default. (add --nowithfactory to disable)

Changed boot sequence in build_packages --withfactory
→Get in the way of NFSROOT

Once build_packages --withfactory, need to do below for exclude the impact of building factory_test packages.

(cr) emrge-x86-generic --depclean chromeos-base/chromeos-factoryinstall # uninstall chromeos-base/chromeos-factoryinstall
(cr) emrge-x86-generic --depclean chromeos-base/factorytest_init       # uninstall chromeos-base/factorytest_init
(cr) sudo rm -f /build/x86-generic/root/.leave_firmware_alone           # remove files that chromeos-base/chromeos-factoryinstall creates
(cr) emerge-x86-generic -av chromeos-base/chromeos-init                 # reinstall chromeos-base/chromeos-init tofix init script
(cr) emerge-x86-generic -av app-laptop/laptop-mode-tools               # reinstall app-laptop/laptop-mode-tools to restore 99-laptop-mode.rules
disable iptables

edit two files

chroot/build/x86-generic/etc/init/iptables.conf
chroot/build/x86-generic/etc/init/ip6tables.conf

--- iptables.conf.org
+++ iptables.conf
@@ -6,7 +6,7 @@
 author          "chromium-os-dev@chromium.org"

 # We must run eventually even if the UI doesn't come up correctly.
-start on starting failsafe
+start on never #start on starting failsafe

script
 iptables -P INPUT DROP
Other than NFSROOT

If the test image, such as sshd is up and running...

- `image_to_live.sh` update image and reboot via ethernet
- `update_kernel.sh` update kernel and reboot via ethernet
- `gmerge` (run on target) request packages to devserver via ethernet
  - (cr) ./start_devserver on host to use devserver

Two ways to start sshd on Developer image

- change before build
  add below to `create_base_image()` in build_image
  ```bash
  sudo sed -i 's/#for_test //" "$ {ROOT_FS_DIR} /etc/init/openssh-server.conf"
  ```
- change after boot
  add below to `/etc/init/openssh-server.conf`
  ```bash
  start on stopped iptables and ip6tables
  ```

http://www.chromium.org/chromium-os/testing/running-tests
WiFi is activated prior to the display login prompt. failsafe(B) will be executed after 30 seconds even if failsafe(A) can not be executed. iptable and hotkey configuration in failsafe(A/B)

Boot time 8.38 seconds

firmware 4.98
kernel 1.06
system 1.09
chrome 1.25

(Celeron B800 Note PC)
Conclusion/reference

• Conclusion
  • 5 scripts to build
  • Build system using Portage
  • Can use NFSROOT
  • Mechanism of fast boot

• Reference
  • http://www.chromium.org/chromium-os

• Trademarks
  Chrome is a trademark of Google Inc.
  Chrome OS is a trademark of Google Inc.
  Chromium is a trademark of Google Inc.
  Chromium OS is a trademark of Google Inc.
  Ubuntu is registered trademarks of Canonical Ltd.
  "Gentoo" is a trademark of Gentoo Foundation, Inc.
  Other trademarks and registered trademarks not listed above may be used in this slide.