READ-ONLY ROOT FILESYSTEM FOR LINUX-BASED SYSTEMS

Prepared by	Piotr Skrzypek
	TEC-SWF

Document Type	TN - Technical Note
Reference	
Issue/Revision	0 . 1
Date of Issue	21/07/2023
Status	For Information Only

CHANGE LOG

Reason for change	Issue Nr	Revision Number	Date
Initial version	0	1	21/07/2023

Table of Contents

1. INTRODUCTION

This technical note provides instructions how to convert an existing Linux root filesystem for read-only operation.

Whenever possible, it is recommended to generate a custom root filesystem (e.g., using Yocto[1] or Buildroot[2]), tailored for a specific use case. However, in certain situations (budget constraints, lack of existing recipes in Yocto or Buildroot), it might be better to use an existing image, delivered by the hardware vendor. In this case, the delivered system has to be screened for services and applications that require write access to the filesystem.

This guide will use the Raspberry Pi OS[3] as an example. This target has been selected due to enormous popularity of Raspberry Pi single board computers. However, the reasoning detailed in this guide should also apply to other Linux images.

Please note that this technical note addresses only tweaks to the root filesystem. Tailoring Linux kernel is not covered here. Tailoring of the root filesystem is highly dependent on the target application. This guide assumes the user wants a lightweight, clean filesystem with wifi and/or ethernet connectivity.

2. INSTALLATION

This part is strongly dependent on Raspberry Pi OS. If you are using a different image, please skip this part.

2.1. Downloading

To download and confirm image consistency, issue the following commands:

```
wget "https://downloads.raspberrypi.org/raspios_lite_arm64/images/"\
    "raspios_lite_arm64-2023-05-03/"\
    "2023-05-03-raspios-bullseye-arm64-lite.img.xz"

wget "https://downloads.raspberrypi.org/raspios_lite_arm64/images/"\
    "raspios_lite_arm64-2023-05-03/"\
    "2023-05-03-raspios-bullseye-arm64-lite.img.xz.sha256"

sha256sum -c 2023-05-03-raspios-bullseye-arm64-lite.img.xz.sha256
```

[1] https://www.yoctoproject.org/
[2] https://buildroot.org/
[3] https://www.raspberrypi.com/software/

```
xz -d 2023-05-03-raspios-bullseye-arm64-lite.img.xz
```

Note: links point to the latest version available when writing this document.

Copy the inflated image to an SD card and flush the storage cache. Assuming the SD card is identified as */dev/sda* in your host system, issue the following commands:

```
dd if=2023-05-03-raspios-bullseye-arm64-lite.img of=/dev/sda
sync
```

2.2. Headless configuration

Raspberry Pi OS allows to configure several system parameters before the first boot. The following steps are based on Raspberry Pi documentation[4].

First mount the filesystem copied to the SD card:

```
mount /dev/sda2 /mnt/
mount /dev/sda1 /mnt/boot/
```

Create a user:

```
echo -n 'YOUR-USERNAME:' > /mnt/boot/userconf.txt
echo 'YOUR-PASSWORD' | openssl passwd -6 -stdin >> /mnt/boot/userconf.txt
```

Note: previous versions of Raspberry Pi OS came with default user 'pi' and password 'raspberry'. This is no longer the case.

To enable the ssh server, add an empty file named *ssh* into /boot:

```
touch /mnt/boot/ssh
```

To make the Raspberry Pi OS connect to a specific wi-fi network automatically, add *wpa_supplicant.conf*. The 256bit PSK key can be generated with *wpa_passphrase*.

```
vi /mnt/boot/wpa_supplicant.conf
```

Example contents of wpa_supplicant.conf (adjust to your needs):

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=NL

network={
  ssid="YOUR-SSID"
```

[4] https://www.raspberrypi.com/documentation/computers/configuration.html#setting-up-a-headless-raspberry-pi

```
  psk=3bd98d9c569fe00e7354e94efed7a9e4138c567094f8684fbc7664e73ebea812
  key_mgmt=WPA-PSK
}
```

If desired, configure static IP address by editing dhcpcd.conf:

```
vi /mnt/etc/dhcpcd.conf
```

Add following lines to the configuration file. Adjust the IP addresses to your needs.

```
interface wlan0
static ip_address=YOUR-IP/YOUR-IPMASK
static routers=YOUR-GATEIP
```

Similar action could be done for *eth0*.

Modify the hostname. Remember to update 127.0.0.1 entry in */etc/hosts* as well.

```
vi /mnt/etc/hostname
vi /mnt/etc/hosts
```

Unmount the filesystem, put the SD card into Raspberry Pi board.

```
umount /mnt/boot
umount /mnt
sync
```

Power up Raspberry Pi board. If it has been configured to a static address, you know where to find it. If using DHCP, check your router or scan the network with *nmap*.

2.3. Configuration after the first boot

If you desire to use public key authentication, this must be explicitly enabled. Uncomment *PubkeyAuthentication* in *sshd_config* on the target.

```
sudo vi /etc/ssh/sshd_config
```

Your public key should be added to */home/YOUR-USERNAME/.ssh/authorized_keys.*

Set timezone

```
sudo timedatectl set-timezone Europe/Amsterdam
```

Update the system and reboot in case there were updates to the kernel:

```
sudo apt update
sudo apt upgrade
sudo reboot
```

3. SCREENING THE SYSTEM AND PREPARING FOR READ-ONLY OPERATION

While Raspberry Pi OS is still used as an example, this part should be applicable to wider range of root filesystems. In case you have a different set of packages, try to follow the same reasoning.

3.1. Checking running processes

The first step is to check running processes, in particular the init daemon. Init daemon is the first process created by the kernel. It is responsible for launching all services installed in the system. Popular init daemons are: *systemd*, *sysvinit*, *OpenRC*, *BusyBox*.

```
ps -eFH
```

The output of the *ps* is quite lengthy, so it is not copied here. We will analyse active processes to learn more about the system. We will skip the analysis of the kernel threads (surrounded by square brackets).

3.1.1. /sbin/init

This process with PID (process ID) 1 is the init deamon.

```
file /sbin/init
```

The command above reveals that this image is using Systemd[5] as init daemon. Systemd comes with several useful components: system logger, event timer, etc.

3.1.2. /lib/systemd/systemd-journald

Journald[6] is a Systemd's logger. By default, data is stored in a persistent location at /var/log/journal. However, it is possible to reconfigure journald to store logs only in RAM memory at /run/log/journal. If this is desired, follow the steps below. We will also set the maximum size to 100MB and clear any logs already written to the persistent storage.

Add *RuntimeMaxUse=100M:*

```
sudo vi /etc/systemd/journald.conf #add: RuntimeMaxUse=100M
```

Restart the service and flush previous logs:

[5] https://man7.org/linux/man-pages/man1/init.1.html
[6] https://man7.org/linux/man-pages/man8/systemd-journald.service.8.html

```
sudo systemctl restart systemd-journald.service
sudo journalctl --vacuum-time=1s
sudo rm -rf /var/log/journal
```

3.1.3. /lib/systemd/systemd-udevd

Udevd[7] is a service that listens to device events coming from the kernel and executes configured actions.

3.1.4. /lib/systemd/systemd-timesyncd

Timesyncd[8] is a service that can synchronise the system time with Network Time Protocol compatible server. It can also store the time on a file, so that systems without hardware Real Time Clock can have some approximation of the current time after a reboot. Inspection of /etc/systemd/timesyncd.conf shows that this feature is already disabled.

3.1.5. avahi-daemon

Avahi daemon[9] is used to advertise available services to the local network (compliant to Apple Bonjour). If we are not building a consumer-grade network device, we might choose to disable it.

```
sudo systemctl disable avahi-daemon
sudo systemctl stop avahi-daemon
sudo apt remove avahi-daemon
```

3.1.6. cron

Cron[10] allows to execute scheduled scripts. Systemd has timers which allow to achieve similar results. As such, we will remove cron. But first we need to check which services are scheduled and compare with systemd's timers.

```
ls /etc/cron*
systemctl list-timers
```

script	cron	systemd	function
e2scrub_all	X	X	Runs filesystem checks and recovery.
apt-compat	X		No effect – disabled in systemd systems.

[7] https://man7.org/linux/man-pages/man8/systemd-udevd.8.html
[8] https://man7.org/linux/man-pages/man8/systemd-timesyncd.service.8.html
[9] https://linux.die.net/man/8/avahi-daemon
[10] https://www.man7.org/linux/man-pages/man8/cron.8.html

dpkg	X		Backs up last 7 versions of dpkg database. Can be disabled in read-only system.
logrotate	X	X	Manages (mails, compresses, removes, etc.) system log files.
man-db	X	X	Builds index cache for manual pages.
fake-hwclock	X		Stores current time to a file, to compensate for lack of hardware RTC on Raspberry Pi (like timesyncd).
systemd-tmpfiles-clean.timer		X	Cleans up temporary files.
apt-daily.timer		X	Downloads latest package database.
apt-daily-upgrade.timer		X	Upgrades packages.
fstrim.timer		X	Discards storage blocks not used by the filesystem.

Looking at the table above, the conclusion is that there are no necessary scripts that are triggered by *cron* and not by *systemd. Cron* can therefore be removed from the system.

```
sudo systemctl disable cron.service
sudo systemctl stop cron.service
sudo apt remove cron
```

3.1.7. dbus-daemon, polkitid

dbus-daemon[11] is a system message bus daemon heavily used by the desktop environment. Even if we are not including desktop environment, there are several services depending on this infrastructure (including system) and we must keep it.

polkit[12] regulates the D-bus by applying policies defined in the system. Like *dbus-daemon*, we must keep it.

3.1.8. rsyslogd

Rsyslogd[13] is another utility implementing system logging. Its functionality overlaps with *journald*. As such, we will remove it.

```
sudo systemctl disable rsyslog.service
sudo systemctl stop rsyslog.service
sudo apt remove rsyslog
```

3.1.9. systemd-logind

[11] https://linux.die.net/man/1/dbus-daemon
[12] https://linux.die.net/man/8/polkitd
[13] https://man7.org/linux/man-pages/man8/rsyslogd.8.html

systemd-logind[14] is part of systemd infrastructure and its main function is to manage user logins and sessions. It is an integral part of the system and cannot be removed.

3.1.10. thd

This service is actually called triggerhappy[15] and is used to map hardware interface hotkeys with actions. Assuming we re not planning to hoop up a keyboard or other type of human input device, it is appropriate to remove this service.

```
sudo systemctl disable triggerhappy.service
sudo systemctl stop triggerhappy.service
sudo apt remove triggerhappy
```

3.1.11. wpa_supplicant

wpa_supplicant[16] is a daemon that implements network authentication. If the target system is expected to join a wireless or wired network that requires authentication, then it has to be kept.

3.1.12. rngd

rngd[17] process is used to feed sources of random events into kernel, to support random number generator. There are several areas where high entropy random stream is of high importance (incl. cryptographic functions) and therefore we will keep this process.

3.1.13. ModemManager

ModemManager[18] is a process that binds broadband modem hardware with the D-Bus, such that it can be later used by e.g., NetworkManager. Since our target system aims at simple low level control, we will not include this daemon.

```
sudo systemctl disable ModemManager.service
sudo systemctl stop ModemManager.service
sudo apt remove modemmanager
```

3.1.14. agetty

agetty[19] allows a user to log in via UART (tty1). If this is not needed, it can be disabled in the following way:

[14] https://man7.org/linux/man-pages/man8/systemd-logind.service.8.html
[15] https://github.com/wertarbyte/triggerhappy
[16] https://linux.die.net/man/8/wpa_supplicant
[17] https://linux.die.net/man/8/rngd
[18] https://www.freedesktop.org/software/ModemManager/man/latest/ModemManager.8.html
[19] https://man7.org/linux/man-pages/man8/agetty.8.html

```
sudo systemctl disable getty@tty1.service
sudo systemctl stop getty@tty1.service
```

3.1.15. sshd

sshd[20] is a secure shell daemon. If a remote ssh access is desired, this deamon must remain in the system.

3.1.16. hciattach, bluetoothd

These services implement the Bluetooth infrastructure. bluetoothd[21] manages all Bluetooth devices, while hciattach[22] can be used to connect to a device implementing the UART profile. Since we are not planning to use Bluetooth, we will disable these daemons.

```
sudo systemctl disable hciuart.service
sudo systemctl stop hciuart.service
sudo systemctl disable bluetooth.service
sudo systemctl stop bluetooth.service
sudo apt remove bluez
```

3.1.17. dhcpcd

dhcpcd[23] is a DHCP client daemon. If the system is required to obtain network settings from the DHCP server, then this service must remain installed.

3.2. Services installed in the system

ps -eEF displayed currently running processes. There might be more services installed in the system. Knowing that systemd is the init daemon, we can get the list of the system services using the following command:

```
systemd list-units
```

service	function	decision
alsa-restore	Used to re-init sound infrastructure	remove
console-setup	Allows to reconfigure console fonts and keymaps	remove
dbus	D-Bus communication bus	keep
dhcpcd	DHCP client daemon	keep

[20] https://linux.die.net/man/8/sshd
[21] https://linux.die.net/man/8/bluetoothd
[22] https://linux.die.net/man/8/hciattach
[23] https://manpages.org/dhcpcd/8

dphys-swapfile	Implements swap memory in a file	remove
fake-hwclock	Backs up current time in a file	remove
getty@tty1	Console on UART	keep
ifupdown-pre	Callback scripts for networking interfaces init/de-init	keep
keyboard-setup	Loads console keymap	remove
kmod-static-nodes	Generates info on the static device nodes	keep
networking	Manages networking interfaces	keep
polkit	D-Bus policy support	keep
raspi-config	Raspberry Pi OS – specific configuration utility	remove
rc-local	Supports automatic boot process	keep
rng-tools-debian	Random number stream	keep
rpi-eeprom-update	Raspberry Pi – specific EEPROM update utility	remove
ssh	Secure shell	keep
systemd-fsck-root	File system checks on the root filesystem	keep
systemd-fsck	File system checks on remaining file systems	keep
systemd-journal-flush	Manages journald logs	keep
systemd-journald	Logging utility	keep
systemd-logind	Manages user logins	keep
systemd-modules-load	Loads kernel modules	keep
systemd-random-seed	Initializes random number generator	keep
systemd-remount-fs	Mounts filesystems at boot	keep
systemd-sysctl	Configures kernel parameters at boot	keep
systemd-sysusers	Initializes system users and groups	keep
systemd-timesyncd	Synchronizes local clock with NTP	keep
systemd-tmpfiles-setup-dev	Manages temporary files	keep
systemd-tmpfiles-setup	Manages temporary files	keep
systemd-udev-trigger	Triggers actions linked with kernel events	keep
systemd-udevd	Monitors kernel events	keep
systemd-update-utmp	Updates utmp with the current system state	keep
systemd-user-sessions	Controls user logins	keep
user-runtime-dir@UID	Creates user runtime directory	keep
user@UID	Launches user processes	keep
wpa_supplicant	Network authentication	keep

Based on the table above, we can use the following commands to remove unwanted services:

```
sudo systemctl disable alsa-restore.service
sudo systemctl stop alsa-restore.service
sudo systemctl disable console-setup.service
sudo systemctl stop console-setup.service
sudo systemctl disable dphys-swapfile.service
```

```
sudo systemctl stop dphys-swapfile.service
sudo systemctl disable fake-hwclock.service
sudo systemctl stop fake-hwclock.service
sudo systemctl disable keyboard-setup.service
sudo systemctl stop keyboard-setup.service
sudo systemctl disable raspi-config.service
sudo systemctl stop raspi-config.service
sudo systemctl disable rpi-eeprom-update.service
sudo systemctl stop rpi-eeprom-update.service

sudo apt remove console-setup
sudo apt remove dphys-swapfile
sudo apt remove fake-hwclock

rm /etc/fake-hwclock.data
```

3.3. Timers installed in the system

It is time to have another look at the timers running in the system.

```
systemd list-units
```

script	function	decision
apt-daily-upgrade	Upgrades packages	remove
apt-daily	Downloads latest package database	remove
e2scrub_all	Runs filesystem checks and recovery	keep
fstrim	Discards storage blocks not used by the filesystem	remove
logrotate	Manages system log file	remove
man-db	Builds index cache for manual pages	remove
systemd-tmpfiles-clean	Cleans up temporary files	keep

Issue the following commands to remove unnecessary timers:

```
sudo systemctl disable apt-daily-upgrade.timer
sudo systemctl stop apt-daily-upgrade.timer
sudo systemctl disable apt-daily.timer
sudo systemctl stop apt-daily.timer
sudo systemctl disable fstrim.timer
sudo systemctl stop fstrim.timer
sudo systemctl disable logrotate.timer
sudo systemctl stop logrotate.timer
```

```
sudo systemctl disable man-db.timer
sudo systemctl stop man-db.timer
sudo apt remove logrotate
```

4. MAKING THE FILESYSTEM READ-ONLY

Remove packages that were installed as dependencies.

```
sudo apt autoremove
```

Configure fstab to mount filesystem as read only. Allocate space from RAM.

```
vi /etc/fstab
```

add ,ro to / and /boot, like this:

```
PARTUUID=fb0d460e-01 /boot      vfat  defaults,ro          0 2
PARTUUID=fb0d460e-02 /          ext4  defaults,noatime,ro  0 1
```

and the temp filesystem:

```
tmpfs                /tmp      tmpfs nosuid,nodev         0 0
tmpfs                /var/log  tmpfs nosuid,nodev         0 0
tmpfs                /var/tmp  tmpfs nosuid,nodev         0 0
```

Optionally restrict maximum size adding '*,size=100MB*'

Modify kernel boot parameters[24], set '*fsck.repair=no*' and add '*ro*'

```
vi /boot/cmdline.txt
```

Reboot into read-only filesystem:

```
reboot
```

5. USAGE TIPS

5.1. Write access

Whenever a modification needs to be done, issue the following command to enable write access:

```
sudo mount -o remount,rw /
```

And to revert to read-only:

```
sudo mount -o remount,ro /
```

[24] https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html

5.2. Check health and logs

Use the following commands to inspect whether the system is functioning properly:

```
systemctl
journalctl -p 0 -x
```

journalctl's severity can be adjusted from 0 (emergency) to 7 (debug).