Setup minimal kiosk environment with Alpine Linux

dev.to/nesterow/setup-minimal-kiosk-environment-with-alpine-linux-27b

Anton Nesterov

Anton Nesterov
Posted on Aug 7, 2019

💖 🦄 🤯 🙌 🔥
#raspberrypi #linux #tutorial #embedded
In this post I want to share a method of developing kiosk applications for boards like Raspberry PI and analogs.

Why Alpine Linux?

Almost every leading linux distributive provides builds for Raspberry PI. However, most of them come with useless weight especially when you need a system that suppose is to run a single application. Alpine Linux comes in a small size (~130mb) and provides you with simple (but powerful) tools to build your own system.
Alpine is designed to run from RAM, which makes it perfect for creating containers or embedded systems.

You'll need:

- Raspberry PI model 3
- SD Card, 1GB or more
- HDMI Cable
- Power source for Raspberry PI
- TV or Display with HDMI port
- A keyboard
- A SSH Terminal application (for Windows users)

Prepare SD Card

No need to flash disk images, installation process is as easy as copying files:

1. Download Alpine distribution for your model. We need the build for `aarch64` architecture. https://alpinelinux.org/downloads/
2. Format the SD card in Fat32 and make it bootable (Linux, Mac)
3. Unzip downloaded archive to the SD card `alpine-rpi-<version>-armhf.tar.gz`

Install Raspberry PI Firmware

Some drivers are not included within the standard Alpine distribution. You need to download them manually.

1. Open SD card and create directory named `firmware`
2. Copy directory called `brcm` from following repository: https://github.com/RPi-Distro/firmware-nonfree

Now the SD card is ready for Raspberry PI.

First boot

On the first boot Alpine will ask for login. Initially the system has only `root` user with an *empty password*.

```
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org>.

You can setup the system with the command: setup-alpine

login: root
...
```

Run setup script right after the first login:

```
alpine:~# setup-alpine
```

The setup process will help to get initial setting right. It is pretty straight-forward and intuitive.

Saving system configuration

Because Alpine runs from memory, all changes you would make to the system is lost after reboot. When you need to save any changes permanently Alpine provides a tool called Local Backup Utility (lbu).

After any significant changes, like editing configuration or installing new software always call `lbu commit`:

```
alpine:~# lbu commit

usage: lbu commit|ci [-nv] [<media>]

Options:
  -d    Remove old apk overlay files.
  -e    Protect configuration with a password.
  -n    Don't commit, just show what would have been committed.
  -p <password> Give encryption password on the command-line
  -v    Verbose mode.
```

Managing software

Package management in Alpine is simple, but slightly different from other distributions.

Updating and upgrading packages:

```
alpine:~# apk update && apk upgrade
```

First, let's install nano editor:

```
alpine:~# apk add nano
```

Now we need to add APK repositories to be able to install additional software.

Open `nano /etc/apk/repositories` and uncomment the link pointing to the community repository. The link should look similar to following:

```
http://mirror.example.com/mirrors/alpine/v3.10/community
```

Then update package index:

```
alpine:~# apk update
```

Don't forget to commit changes after adding new software:

```
alpine:~# lbu commit -d
```

Now we're ready to shape the system for our needs.

Configure SSHD

It is more convenient to manage the device from your laptop. Let's setup `sshd` to be able to open remote shells.

Open `nano /etc/ssh/sshd_config` and edit it as follows:

```
PermitRootLogin yes
PasswordAuthentication yes
PermitEmptyPasswords no
DenyUsers guest
```

Save the file and commit the system configuration:

```
alpine:~# lbu commit -d
```

Restart SSH daemon:

```
alpine:~# service sshd restart
```

Connect to the device from your desktop:

```
~#> ssh root@192.168.2.199
password: *********
```

Install X11

Next step is to setup essential display server and configure Raspberry PI GPU to enable 3D acceleration.

Setup Xorg server:

```
alpine:~# setup-xorg-base
(1/100) Installing ...
```

Install video drivers:

```
alpine:~# apk add mesa-dri-vc4 mesa-egl xf86-video-fbdev xf86-video-vesa xf86-
input-mouse xf86-input-keyboard dbus setxkbmap kbd xrandr xset
(1/100) Installing ...
```

Save changes to the disk:

```
alpine:~# lbu commit -d
```

Configure GPU

Raspberry PI doesn't have BIOS, but the essential hardware configuration can be performed through usercfg.txt file.

Insert the SD card and create `usercfg.txt` with following options:

```
dtoverlay=vc4-fkms-v3d
gpu_mem=256
```

Depending on the display model and resolution, you need to modify your settings. All possible options can be found here.

Create user

Now, in order to run kiosk we need to create an unprivileged user. This user will also be used to access the application directory using (SFTP)[https://en.wikipedia.org/wiki/SFTP]

We will keep application files at `/srv` directory and it also will be the user's home directory.

```
alpine:~# adduser -h /srv user
alpine:~# chown -R user /srv
alpine:~# chmod 701 /srv
alpine:~# lbu commit
```

The user's home directory is at `/srv`, however the files in this directory won't persist until this directory is in LBU index:

```
alpine:~# lbu add /srv
alpine:~# lbu commit
```

At last, the user must login automatically on system's boot. In order to achieve this behaviour edit `/etc/inittab` changing options for `tty1`:

```
tty1::respawn:/bin/login -f user
```

Creating kiosk

Install Chromium browser:

```
alpine:~# apk add chromium
alpine:~# lbu commit -d
```

Chromium is going to start after the user login. Let's create two init scripts for this purpose.

Create `/srv/.profile` file with following content:

```
#!/bin/sh
# start X server
exec startx
```

Create `/srv/.xinitrc` with following content:

```
#!/bin/sh

# turn off screensaver
xset -dpms
xset s off
xset s noblank


# screen size
width="1920"
height="1080"

# url
url="https://dev.to"

exec chromium-browser $url --window-size=$width,$height --window-position=0,0 --
kiosk --no-sandbox --full-screen --incognito --noerrdialogs --disable-translate --
no-first-run --fast --fast-start --ignore-gpu-blacklist --disable-quic --enable-
fast-unload --enable-tcp-fast-open ---enable-native-gpu-memory-buffers --enable-
gpu-rasterization --enable-zero-copy --disable-infobars --disable-
features=TranslateUI --disk-cache-dir=/tmp
```

Make init files executable and commit changes:

```
alpine:~# chmod u+x /srv/.profile
alpine:~# chmod u+x /srv/.xinitrc
alpine:~# lbu commit
```

The kiosk is ready! The chromium will start on system boot.
Total image size of this setup is ~235Mb which means you can run an Alpine Linux kiosk from a 256Mb SD card.

What next?

This setup provides pretty simple development environment for your kiosk applications. When developing an application you can point kiosk to a dev. server on your PC to debug and optimise it at realtime. It works well with Webpack and Gulp. In order to upload production builds just use `SFTP` and `lbu commit`.