OpenLinux

App Auto Startup Configuration Guide

Issue 1.1 Date 2020-03-13

Notice

This document provides guide for users to use OpenLinux.

This document is intended for system engineers (SEs), development engineers, and test engineers.

THIS GUIDE PROVIDES INSTRUCTIONS FOR CUSTOMERS TO DESIGN THEIR APPLICATIONS. PLEASE FOLLOW THE RULES AND PARAMETERS IN THIS GUIDE TO DESIGN AND COMMISSION. NEOWAY WILL NOT TAKE ANY RESPONSIBILITY OF BODILY HURT OR ASSET LOSS CAUSED BY IMPROPER OPERATIONS.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE DUE TO PRODUCT VERSION UPDATE OR OTHER REASONS.

EVERY EFFORT HAS BEEN MADE IN PREPARATION OF THIS DOCUMENT TO ENSURE ACCURACY OF THE CONTENTS, BUT ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

Neoway provides customers complete technical support. If you have any question, please contact your account manager or email to the following email addresses:

Sales@neoway.com

Support@neoway.com

Website: http://www.neoway.com

Contents

About This Document

Scope

This document is applicable to OpenLinux modules, including the A70 series and the N720 series. It describes how to configure applications in the module to start automatically.

Audience

This document is intended for system engineers (SEs), development engineers, and test engineers.

Change History

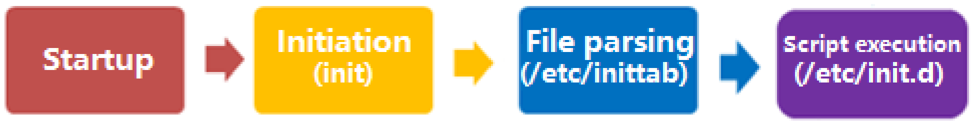
Issue	Date	Change	Changed By
1.0	2019-11	Initial draft	Tommy Sun
1.1	2020-03	Modified the description of Chapter 1	Tommy Sun

Conventions

Symbol	Indication
❗	This warning symbol means danger. You are in a situation that could cause fatal device damage or even bodily damage.
⚠	Means reader be careful. In this situation, you might perform an action that could result in module or product damages.
ℹ	Means note or tips for readers to use the module

1 Overview

After an OpenLinux module starts up, its kernel starts the initialzation process, during which files in **/etc/inittab** are parsed. Then the scripts that are stored in the **/etc/init.d** directory are executed after the **rc** scripts are initiated.



The **/etc** directory of Linux OS contains seven subdirectories **rcN.d**. Each subdirectory includes a series of scripts (run commands) that control processes and symbolic links that direct to the control scripts in **/etc/init.d/**. The rc scripts initiate the scripts in **/etc/init.d/** according to the running level specified in the **inittab** file.

To enable the automatic startup of custom apps, create symbolic links in the **/etc/rcN.d/** directory. The symbolic link is in the form of ***S|K + nn + script***.

- **N**: running level in the Linux OS, ranging from 0 to 6.

  It is rc5 by default in the OpenLinux system.
  To check the current value, execute **runlevel**.
- **S|K**: execution type of scripts

  S indicates that the script is executed when the module is started
  K indicates that the script is executed when the module is shut down
- **nn**: priority, ranging from 0 to 100

  The prority decrease as the number is smaller.
- **script**: script that the symbolic link directs to.

This document describes how to enable an application to start automatiically after the OpenLinux module boots. The auto-startup of an application can be configured through two methods:

- ADB shell commands
- Source code

2 Through ADB Commands

In the debugging phrase, tollow the steps below to enable an application to start automatically when the module boots.

Step 1: Send the ADB commands to push the application to the file system of the OpenLinux module.

1. Remount the **/** system readable and writable.

```
adb shell mount -o remount,rw /
```

2. Push the application to /etc/init.d/.

```
adb push <local path of the application> /etc/init.d/
```

3. Change the permission of the application.

```
adb shell chmod 755 /etc/init.d/<application name>
```

Only the data directory is readable and writeable by default in the OpenLinux system. The other directories are read-only.

To push an application to other directories, remount the **/** system readable and writable.

Step 2: Create a symbolic link in **init.d app.sh** is an example).

```
adb shell                          // Navigate to the ADB shell
cd /etc/init.d                     // Navigate to the init.d directory
ln -sv app.sh ../rc5.d/S98app.sh  // Create a symbolic link, priority level: S98
```

3 Through Source Code

When compiling source code, you can custom the file system to create a firmware that embeds automatic startup applications.

Follow the steps below:

Step 1: Create an **etc** directory in **APP_SDK/usrdir/add**. Create two subdirectories **init.d** and **rc5.d** in **etc**.

```
mkdir -p etc/init.d
cd etc
mkdir -p rc5.d
```

Step 2: Place the application to the **etc/init.d** directory (**app.sh** is an example).

Step 3: Create a symbolic link in **rc5.d**.

```
cd /etc/rc5.d/
ln -sv ../init.d/app.sh S99app.sh
```

Step 4: Build the system image in the **root** directory of APP_SDK.

```
./build.sh perf system
```

Ensure all modules of the perf version have been built before executing the Step 4.

Step 5: Check whether the system image is built successfully.

Check if the application and its symbolic linkare included in the following two path:

```
APP_SDK\poky\build\tmp-glibc\work\mdm9607-oe-linux-gnueabi\mdm-perf-image\rootfs\etc\init.d
APP_SDK\poky\build\tmp-glibc\work\mdm9607-oe-linux-gnueabi\mdm-perf-image\rootfs\etc\rc5.d
```