NanoDLP: Embedded Single Board Computer (SBC) Instructions

Though the most common hardware choice for embedded NanoDLP control servers is a Raspberry Pi of some kind, NanoDLP can also be run on other hardware (such as Pi-like SBCs) with some minor adjustments. Here, we outline instructions for installing and configuring NanoDLP on arm-based SBCs, according to the choice of base OS.

Debian

On an SBC running a basic install of Debian (e.g., no desktop environment), NanoDLP can be installed from the "ARM [32bit/64bit] without desktop" download, or "Linux 64bit" if the system is x86-64 based. The framebuffer /dev/fb0 will be used to control the display output. As of Debian 11 "Bullseye", NanoDLP does not have exclusive access to the framebuffer, meaning that any changes to the status of the terminal (e.g., typing inputs and responses from getty) or kernel messages will overwrite the corresponding area of the display output. If getty is disabled to prevent such issues, **then the machine will become effectively headless and require SSH** for Linux configuration.

Base install steps

1. Create a fresh install of Debian on your target SBC, using the images provided either by the Debian organization or by your machine's manufacturer.
2. Connect to the network. Ethernet is recommended for simplicity and reliability reasons, but wireless is available on Debian with the appropriate drivers and network manager of some kind.
3. Install sudo and wget
    a. If disabling getty, install ssh as well
4. Create user pi and add to sudoers
5. Login as user pi and create a new directory /home/pi/nanodlp/, then cd into the new directory
6. Download the appropriate NanoDLP archive from the NanoDLP download page with wget, e.g.

```
wget https://www.nanodlp.com/download/nanodlp.linux.arm64.stable.tar.gz
```

7. Unpack the compressed archive into the ~/nanodlp/ directory.

You may stop here and start NanoDLP by running ./run.sh while in the ~/nanodlp/ directory. However, NanoDLP is not yet running as a service, and terminal inputs and kernel events will still interrupt the framebuffer output. Additionally, the cursor in the terminal will continue to blink. These issues are addressed as follows:

Disabling cursor blink

Cursor blink can be reliably disabled by passing the parameter

```
vt.global_cursor_default=0
```

to the kernel. The exact method for doing this will depend on the install. For example, on Debian for Raspberry Pis, the OS install storage device will have a separate boot partition which is visible when connecting using RPi USB Boot or opening the SD card, depending on the Pi's storage type. Once the boot partition is open, open cmdline.txt and add the kernel parameter to the end of the file, i.e.

“[other kernel parameters] vt.global_cursor_default=0”

Preventing terminal outputs

If no input is provided directly over a keyboard and no user is logged in, then getty may be left enabled, but this approach is somewhat riskier than disabling it. To disable getty, issue

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

IMPORTANT: Disabling getty will disable local login, leaving only SSH for configuring Linux. Make sure to install SSH before disabling getty. If getty is only stopped temporarily by issuing “sudo systemctl stop getty@tty1.service”, then getty will restart after a reboot, but if getty is shut off using “systemctl disable” or shut off automatically by another service like in the example service file (see below), getty will **NEVER** reenable, and login without using SSH first will be impossible.

Preventing kernel messages

Even with getty disabled and no user keyboard inputs, the kernel will still output messages and overwrite the framebuffer. This especially will occur if there are device events, such as the connection or disconnection of a USB device, or changes in the network connection status. If you can ensure that neither of these events occur during prints, it is not strictly necessary to disable kernel messages, but best practice will be to prevent all messages except emergency messages from appearing using

```
sudo dmesg -n 1
```

Similar to getty, default behavior is for kernel message behavior to reset back to default when the system reboots.

Automatically run NanoDLP and prevent terminal overwriting

To automate the running of NanoDLP, the disabling of getty, and the disabling of kernel messages, a systemd unit may be created to automate these tasks and restart NanoDLP when the SBC reboots, or when NanoDLP is closed unexpectedly. An example of a systemd unit file is

```
[Unit]
Description=NanoDLP Service Module
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=.5
WorkingDirectory=/home/pi/nanodlp
ExecStartPre=systemctl stop getty@tty1.service
ExecStartPre=dmesg -n 1
ExecStart=/home/pi/nanodlp/run.sh
ExecStop=dmesg -n 7
ExecStop=systemctl start getty@tty1.service
```

```
[Install]
WantedBy=multi-user.target
```

where /home/pi/nanodlp is the folder where the NanoDLP download was unpacked. To use this service file, create a file in /etc/systemd/system folder named nanodlp.service and copy over the example file. To enable starting the service at boot, run

```
sudo systemctl enable nanodlp
```

As written this service file will stop getty and dmesg just prior to starting NanoDLP, and then reenable getty and dmesg after NanoDLP closes. This service file will also restart NanoDLP after it closes; note that in order to stop NanoDLP fully while the service is running, one cannot use the “Terminate nanoDLP” button in the Tools pane of the NanoDLP control webpage, but must instead issue

```
sudo service nanodlp stop
```