A Concurrent Real-Time White Paper

2881 Gateway Drive
Pompano Beach, FL 33069
(954) 974-1700
www.concurrent-rt.com

Using ROS with RedHawk Linux on the NVIDIA Jetson TX2

By: Jason Baietto

Chief Systems Architect
Linux Group

June 2018

Overview

This paper provides details for installing and running ROS 2 under the RedHawk™ Linux® real-time kernel and tools available for the NVIDIA® Jetson TX2 and TX2i development boards. Also included is a look at RedHawk's real-time performance during the execution of the ROS Pendulum Control demo.

Installation

First, use Jetpack 3.2 to initialize a TX2 or TX2i development board with L4T 3.2 and then install RedHawk Linux 7.3.1 according to the instructions detailed in the *RedHawk 7.3.1 R28.2 for Jetson TX2/TX2i Release Notes*.

Next, perform the following steps to install the ROS 2 Ardent release onto a TX2:

1. Install curl utility

Issue the following commands to fetch and install curl from the Ubuntu repositories:

```
sudo apt update
sudo apt install curl
```

2. Import ROS 2 public key

Issue the following commands to fetch and install the ROS 2 public key to enable ROS 2 package signature verification:

```
curl http://repo.ros2.org/repos.key | sudo apt-key add –
```

3. Add ROS 2 package source definition

Create a package source file named `/etc/apt/sources.list.d/ros2-latest.list` with the following contents:

```
deb [arch=arm64] http://repo.ros2.org/ubuntu/main xenial main
```

4. Retrieve full list of ROS 2 packages

Issue the following commands to generate the full list of ROS 2 packages that are currently available for download and installation:

```
sudo apt update
apt-cache search 'ros-ardent-*' | cut -f1 -d' ' > full.txt
```

5. Remove ROS 1 packages from list

Certain ROS 2 packages depend upon ROS 1 packages in order to meet their package dependency requirements; installing ROS 1 packages would require additional package sources to be defined, however these packages are not needed and can be eliminated from the full package list with the following command:

```
grep -v -e ros1 -e turtlebot full.txt > packages.txt
```

6. Install ROS 2 packages

Issue the following command to download and install the modified list of ROS 2 packages:

```
sudo apt install $(<packages.txt)
```

Package download and installation should take approximately ten minutes to complete.

NOTE

The rest of this document will refer to ROS 2 Ardent simply as ROS for brevity.

Activation

Interacting with ROS software requires users to first define several environment variables, and this can be accomplished with the following command:

```
source /opt/ros/ardent/setup.bash
```

The above command can be invoked interactively as needed, or it can be appended to a user's `~/.bashrc` file to have these variables automatically defined upon user login.

Exploration

To verify that ROS has been installed and configured correctly, users can experiment with the newly installed ROS Intra-Process Communication demo. Full source code and complete details about this demo can be found at this URL:

https://github.com/ros2/ros2/wiki/Intra-Process-Communication

Briefly, this demo creates two independent ROS nodes that communicate with each other using the ROS publish/subscribe inter-process communication model. Start the demo with the following command:

```
ros2 run intra_process_demo two_node_pipeline
```

While this demo is running you will see output displayed indicating that messages are being published by one node, followed by acknowledgements that messages have been received by the other node. Issue the following command from a different bash shell to see the ROS nodes that are currently active:

```
ros2 node list
```

The output will show that two ROS nodes exist while the demo is running: one named `producer` and one named `consumer`.

There are many ROS tutorials and demonstrations available on the Internet that users are encouraged to seek out to learn more about ROS.

Benchmarking

The installed ROS Pendulum Control demo can be used to measure the real-time performance of a system; this demo simulates an inverted pendulum and it attempts to keep the pendulum aligned vertically by constantly adjusting a horizontal force at the base of the pendulum. The demo also runs a significant stress load in parallel on the system to more accurately determine the worst-case latency.

This demo cannot be run by the root user on the TX2 because it utilizes an 8 GB data structure and when run as root it attempts to lock down all 8 GB into memory to eliminate the impact of page faults on real-time performance. The TX2 only has a total of 8 GB RAM and does not have sufficient memory available to lock down all pages; running as root will cause the demo to trigger an out-of-memory error and the demo will abort.

The demo can successfully be run by a non-root user because it only accesses the 8 GB data structure sparsely during its execution, however it will not be able to run with boosted priority and its measurements will include the additional latencies caused by page faults.

Baseline Performance

First, invoke the Pendulum Control demo as the nvidia user without using any real-time features:

```
pendulum_launch.bash
```

You may see some errors about lacking the permissions required to lock down memory pages and boost priority at the start of the run, however the demo will ignore these errors and continue to run until it is interrupted. You should see many output samples like the following displayed:

```
Commanded motor angle: 1.570796
Actual motor angle: 1.567588
Current latency: 484711 ns
Mean latency: 765336.716988 ns
Min latency: 12416 ns
```

```
Max latency: 15679572 ns
Minor pagefaults during execution: 0
Major pagefaults during execution: 0
```

This output shows a sampled latency of 484 microseconds, an average latency of 765 microseconds, and a worst-case latency of 15 milliseconds; clearly the pendulum demo cannot achieve acceptable real-time performance levels on the TX2 without utilizing real-time features.

RedHawk Optimized Performance

While the pendulum demo cannot be run by the root user on the TX2, RedHawk shielding, process binding and priority boosting can still be used by non-root users to improve real-time performance.

RedHawk provides a capability plugin that can automatically grant individual users additional capabilities upon login. For more details on this extension read the *pam_capability*(8) and *capability.conf*(5) man pages by issuing the following commands:

```
export MANPATH=/usr/ccur/man
man pam_capability
man capability.conf
```

Perform the following steps to configure the capability plugin and utilize RedHawk real-time features to improve performance even when the demo is run by the nvidia user:

1. Assign extra capability to nvidia user

Edit the `/etc/security/capability.conf` file as the root user and add the following lines to the end of the file:

```
role demouser cap_sys_nice
user nvidia demouser
```

2. Activate capability plugin upon login

Edit the `/etc/pam.d/sshd` file as the root user and add the following line after the last `session` entry but before any `@include` lines:

```
session required /lib/aarch64-linux-gnu/security/pam_capability.so
```

To verify that you have configured the capability plugin correctly, ssh into the system as the nvidia user and issue the following command:

```
getpcaps $$
```

You should see output similar to the following:

```
Capabilities for `14554': = cap_sys_nice+eip
```

Note that you can similarly modify `/etc/pam.d/lightdm` to automatically grant capabilities to users upon graphical login.

3. Isolate one CPU core for real-time activity

Issue the RedHawk shield command to fully isolate CPU core 5 from all processes, interrupts and timer events:

```
shield -a 5
```

To verify that CPU core 5 has been fully shielded, issue the `shield` command again with no options and you should see output identical to the following:

```
    CPUID      irqs     ltmrs     procs
--------------------------------------------------
        0        no        no        no
        1        no        no        no
        2        no        no        no
        3        no        no        no
        4        no        no        no
        5       yes       yes       yes
```

The pendulum demo can now be run on CPU core 5 with real-time isolation.

4. Bind demo to shielded core and boost priority

You now need to modify the demo launch script to utilize RedHawk real-time features. Copy the launch script to a file in the nvidia user's home directory with the following command:

```
cp /opt/ros/ardent/bin/pendulum_launch.bash ~/demo.bash
```

Edit the `demo.bash` script and change this line:

```
pendulum_demo -i 0 &
```

to this line:

```
run -b 5 -s fifo -P90 pendulum_demo -i 0 &
```

This change will bind the `pendulum_demo` program exclusively to CPU core 5 and have it run with a real-time priority of 90 utilizing the SCHED_FIFO scheduling class.

5. Run the pendulum demo

Finally, invoke the modified pendulum launch script as follows:

```
~/demo.bash
```

You should now see many samples like the following printed:

```
Commanded motor angle: 1.570796
Actual motor angle: 1.570404
Current latency: 7005 ns
Mean latency: 8021.634300 ns
Min latency: 6555 ns
Max latency: 47488 ns
Minor pagefaults during execution: 0
Major pagefaults during execution: 0
```

This output shows a sampled latency of 7 microseconds, an average latency of 8 microseconds, and a worst-case latency of 47 microseconds; the pendulum demo achieves significantly improved real-time performance levels on the TX2 when utilizing RedHawk real-time features, even without locking down pages into memory.

While the real-time optimized version of the demo is running, issue the following command in a different bash shell:

```
run -n pendulum_demo
```

You will see output like the following displayed:

```
Pid    Tid    Bias   Actual   CPU   Policy   Pri   Nice   Name
19945  19945  0x20   0x20     5     rr       98    0      pendulum_demo
19945  19948  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19952  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19953  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19954  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19955  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19956  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19957  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19961  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19962  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19963  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19964  0x20   0x20     5     fifo     90    0      pendulum_demo
19945  19965  0x20   0x20     5     fifo     90    0      pendulum_demo
```

This output shows that all cores of the demo are now running on CPU core 5 with a boosted real-time priority.

Summary

The ROS Pendulum Control demo achieves significantly improved real-time response when utilizing RedHawk real-time features, however the demo was clearly not written to support the limited amount of RAM available on the TX2. It is hoped that future versions of the pendulum demo will dynamically adjust to the system's available RAM and reduce the number of memory pages required to be locked down, so that the worst-case latencies measured by the demo under RedHawk will be further reduced.