FAQ: Working with 32/64-bit linuxRT OS and EPICS R3-14-12-4_1-x

While linuxRT is a relatively established RTOS at SLAC ICD, it has mostly been used only in conjunction with the uTCA platform.

Recently linuxRT has been 'ported' and running successfully on a few COTS linux servers such as the Dell PowerEdge servers and SuperMicro Industrial PCs.

Till Straumann built a newer version of linuxRT to support the new hardware.

These linux servers are all Intel-based targets that run linuxRT (embedded linux) with uClibc and PREMPT_RT patch.

The important change to the linuxRT RTOS was the addition of support for the Broadcom NICs which are ethernet network chipsets found in these servers.

The other important change was the setup of these servers to boot linuxRT targets as diskless clients.

The BIOS boot order was modified to over-ride 'boot out of hard-disk' or 'CD-ROM' which are the default in most cases, and instead perform network booting as the first choice.

Additionally, BIOS was modified to redirect console output to the first serial port instead of the monitor.

This enables us to observe the PC boot process remotely, like all our IOCs, via the 'screen' process using the 'iocConsole' script.

This FAQ is mostly confined to describing the steps needed for building and running a linuxRT IOC on a COTS Linux Server using the latest EPICS base.

We will use 'MyTest' as an example EPICS application and 'vioc-b34-bd32' as an example IOC.

To keep up with ongoing linuxRT infrastructure improvement efforts, visit Krist's folder in slacspace:

ICD --> Controls Operations & Maintenance Documents --> Centralized Computer Infrastructure ---> LinuxRT

linuxRT infrastructure related documents in slacspace

As we continue to test this new system and learn more, these documents will also evolve.

Frequently Asked Questions

1. What version of EPICS do I use?

The current version of EPICS to use for our 64-bit linuxRT ioc is R3-14-12-4_1-0

That is, EPICS BASE = base-R3-14-12-4_1-0

When the version of EPICS BASE is selected, that selects the cross-compiler's tool chain.

This means you are now locked into a specific tool chain that only supports, "RHEL6 64-bit"

This tool chain is buildroot and the current version is buildroot-2014.11.

/afs/slac/g/lcls/package/linuxRT/buildroot-2014.11/

The linuxRT kernel should match the toolchain.

So, in terms of development unless you have your own RHEL6 64-bit workstation setup properly; always log into **lcls-dev3.**

More about the make process and cross compilers for targets explained under Question 6.

Next, to make sure that your ENV is setup for our version of EPICS BASE as explained under Question 2.

source /afs/slac/g/lcls/epics/setup/go_epics_3-14-12-4_1-0.bash

^^

First a note of CAUTION:

This project does not and will not support RHEL5 32-bit hosts

After the summer 2015 downtime, Jingchen will have fully migrated to RHEL6 64-bit.

^^

2. How do I setup my bash shell to use the new EPICS?

From a bash shell on an ICD Development host (e.g. lcls-dev2 for 32-bit; lcls-dev3 for 64-bit), type the following:

source /afs/slac/g/lcls/epics/setup/go_epics_3-14-12-4_1-0.bash

The above script sets up several EPICS environment variables in your bash shell. It is recommended that in the .bashrc script, you add a case statement to check for development hosts and source the epics setup so that this step is automatic upon login.

Some enviroment variables that are now avilable to you are paths to ' 'TOP-level' directories of interest. Below are a few examples ***:***

EPICS_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0

EPICS_BASE_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/base

EPICS_MODULES_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/modules

EPICS_EXTENSIONS=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/extensions/extensions-R3-14-12

EPICS_IOC_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/iocTop

EPICS_MBA_TEMPLATE_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/modules/icdTemplates/icdTemplates-R1-2-0

PACKAGE_TOP=/afs/slac/g/lcls/package

EPICS_IOC_TOP is the top level application directory for all EPICS iocs that use this EPICS version.

It is here where all new EPICS applications must be created. EPICS_IOC_TOP is the same as the APP variable.

Other environment variables setup in your bash shell are:

IOC=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/iocCommon

$IOC is the directory where the startup scripts for all the iocs are located.

IOC_DATA=/nfs/slac/g/lcls/epics/ioc/data

$IOC_DATA is the data directory. This is where the outputs (and in some case inputs) from iocs are stored.

3. Where are the linuxRT kernels and kernel modules?

Currently there are several versions of linux kernels that have been built for several host architectures.

They are all here under *BUIDLROOT_HOME*: /afs/slac/g/lcls/package/linuxRT/buildroot-2014.11

The tool chain used to build these kernels is 'buildroot' and version 2014.11 was used to build these kernels.

These kernels are available for 32 bit (linuxRT_glibc-x86 and linuxRT-x86) and 64-bit (linuxRT_glibc-x86_64) host architectures.

Further, they have been built with either uClibc (C library for embedded LInux) or glibc (GNU C LIbrary).

uClibc or glibc are libraries which defines "system calls" and other basic facilities such as open, malloc, printf, exit etc.

Based on which libraries the kernels have been built with, currently there are four options for linuxRT kernels to choose from:

buildroot-uclibc-x86 (for 32-bit i680 architectures with uClibc)

buildroot-glibc-x86 (for 32-bit i680 architectures with glibc)

buildroot-uclibc-x86_64 (for 64-bit x86_64 architectures with uClibc)

buildroot-glibc-x86_64 (For 64-bit x86_64 architectures with glibc)

uClibc and 32-bit architectures will be phased out in the near future at ICD - hence it is recommended to use 64-bit glibc kernels for new applications.

For our test application, we use 64-bit kernel built with glibc.

The various kernel modules built for the various kernels are all here:

/afs/slac/g/lcls/package/linuxKernel_Modules

Example: EVR kernel module is pci_mrfevr_linuxRT

Under this directory, there are three different kernel modules - each built for some corresponding kernels as listed above.

Ensure that you use the right version of the kernel module that matches with your kernel.

4. Where are the EPICS modules for this new base?

/afs/slac/g/lcls/epics/R3-14-12-4_1-0/modules

A handful of EPICS modules have been built for R3-14-12-4_1-0.

Most of them have been cross-compiled for several targets including the following:

linuxRT-x86

linuxRT-x86_64

linuxRT_glibc-x86_64

linux-x86

linux-x86_64

5. What do I do if I need a new module or a specific version for an existing module built for the new base?

The python script makeModules.py. This script is located in /afs/slac/g/lcls/tools/script,, which is in PATH.

This script automatically discovers the module dependencies and builds an entire module tree in the correct order for all targets.

As of date this script is 'reasonably' functional.

If the python tool is problematic, use "eco" to checkout modules into your local sandbox and build.

TO DO list: Make the script makeModules.py user-friendly

Contact Murali or Ernest to have the modules built and installed in the meanwhile.

6. Where and how do I create a new EPICS application? How do I setup a new IOC to use linuxRT-86_64 OS?

(a) Create directories as shown below using cpu-b34-bd32 as an example:

$IOC/cpu-b34-bd32

$IOC_DATA/cpu-b34-bd32

$IOC_DATA/vioc-b34-bd32

$IOC_DATA/vioc-b34-bd32/autosave

$IOC_DATA/vioc-b34-bd32/autosave-req

$IOC_DATA/vioc-b34-bd32/iocInfo

TO DO list: Generate a script to create above directories automatically via a single python or perl script.

Note: this script is available from Ken Brobeck on softegr@lcls-srv01:/usr/local/admin/scripts/setup-ioc-dir

$APP/MyTest (look at this as an example)

Now create an EPICS application in your home directory called, 'MyTest' , using standard templates and scripts in the above example.

(b) In your home directory, create a sandbox area called 'MyTest', which will contain your EPICS IOC application.

```
cd ~
mkdir MyTest
```

(c) cd MyTest

You are currently in:

~/MyTest

(d) Now, create a new EPICS IOC Application called,'MyTest' as follows:

makeBaseApp.pl -t slac MyTest

The perl script, makeBaseApp.pl populates your application using the icdTemplates module that has standard ICD templates.

These templates are used by ICD to support our style of IOC Application Building for all facilities we support.

The location for icdTemplates module was defined, when you sourced go_epics_3-14-12-4_1-0.bash

EPICS_MBA_TEMPLATE_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/modules/icdTemplates/icdTemplates-R1-2-0

If you would like to know more about makeBaseApp.pl type:

makeBaseApp.pl -help

(e) Now, open the configure/RELEASE file in your new IOC Application

(i) Notice how the following line appears towards the beginning:

```
# EPICS Templates
TEMPLATE_TOP=/afs/slac/g/lcls/epics/R3-14-12-4_1-0/modules/icdTemplates/icdTemplates-R1-2-0
```

(ii) Notice the following line next:

```
include $(TOP)/RELEASE_SITE
```

The above sets the paths for EPICS base, modules, packages, tools etc. to point to version R3-14-12-4_1-0.

(iii) The file has the following line towards the end of it:

```
# Location of Linux Kernel Modules:
LINUX_KERNEL_MODULES=$(PACKAGE_SITE_TOP)/linuxKernel_Modules
```

(iv) Modify configure/RELEASE as necessary to use the version of EPICS modules that your application needs.

Ensure that the specific versions included in your RELEASE file DO EXIST under the modules directory and are consistent.

They must also have been built for your target kernel (such as linuxRT_glibc-x86_64). If something is missing, contact Ernest.

In our example, we will be needing the EVENT module for our EVR:

```
EVENT_MODULE_VERSION=event-R4-1-6
```

(f) Open MyTestApp/src/Makefile and notice that the IOC application is referred to as 'MyTest"

(g) Now it is time to add your linuxRT_glibc-x86_64 virtual ioc (vioc) startup scripts. vioc-b34-bd32 is used here as an example.

From $(TOP) type the following command to create a boot directory for 'vioc-b34-bd32':

```
makeBaseApp.pl -i -t slac vioc-b34-bd32
```

When prompted to choose target architecture, choose linuxRT-x86 or inuxRT_glibc-x86_64.

Now iocBoot directory has been created under $(TOP) and underneath it 'vioc-b34-bd32' has been created.

Two files are modified based on this selection.

(1) iocBoot/vioc-b34-bd32/st.cmd

Open iocBoot/vioc-b34-bd32/Makefile and notice that ARCH is set to the target platform you you chose - linuxRT-x86 or linuxRT_glibc-x86_64.

(2) iocBoot/vioc-b34-bd32/Makefile.

Open iocBoot/vioc-b34-bd32/st.cmd and notice that this script is setup to pick the binary from your target's directory - linuxRT-x86 or linuxRT_glibc-x86_64 directory.

When prompted with 'Application name?' type 'MyTest'.

This will use the application name MyTest that was created in MyTestApp/src/Makefile as described in step (f).

If you just hit 'Enter' without specifying your application name, then your ioc name vioc-b34-bd32 will be used as the default for the dbd file in the st.cmd

(h) icdTemplates created MyTestApp/src/Makefile.

Open this file and notice how it is populated.

It has support for base, iocAdmin, autosave-restore which all IOCs must include.

It also includes support for Till Straumann's Cexp shell.

Look at Cexp Shell Examples to see how to invoke Cexp and how to call C-functions from this interactive shell. For more information on Cexp, read What is Cexp?

Additionally, the Makefile includes 'system.dbd' - this is mainly for linux-based IOCs.

For linuxRT, 'system' command is useful for changing kernel thread priorities for kernel drivers at run-time. This is explained later under rtPrioritySetup.cmd.

(i) If you do not have an EVR in your system, comment out all lines pertaining to EVR in MyTestApp/src/Makefile:

MyTest_DBD += evrSupport.dbd

MyTest_DBD += devMrfEr.dbd

...

MyTest_LIBS += evrSupport

MyTest_LIBS += devMrfEr

....

MyTest_LIBS += mrfVme64x

....

MyTest_LIBS += drvMrf

For a PMC EVR230 or PCI-e EVR300 that is installed in your system, the above is all you need.

Modify MyTestApp/src/Makefile adding more libraries and DBDs as needed for your application.

(j) Replace the macros in the EPICS database files under MyTestApp/Db or add more databases as needed for your application.

Modify MyTestApp/Db/Makefile to ensure that you have included all your databases for the make process.

(k) Now 'make' your application from the top level directory $(TOP) to ensure all your changes are good:

There should be no build errors.

Refer to notes from Ernest regarding building the application for several targets here: linuxRT targets and cross compiler notes.

(l) Having tested this phase, you can now uninstall the binaries and cleanup the make-generated files using the following command:

make clean uninstall

Do the above before committing the files to CVS, and make sure to cram your IOC application

(m) Add your application to CVS and commit the source files.

Here are some slides that explains a few guidelines for importing new applications into CVS:

cvs-vendor-import.txt

CVS-Importing-Merging.pdf

CVS-At-SLAC.pdf

TO DO list: icdTemplates module that provides default templates is a work in progress.

Make changes to the MyTestApp/Db/* files and iocBoot/vioc-b34-bd32/st.cmd as needed for your application.

Follow the example MyTest and create similar database files and st.cmd.

The EVR-related screens have been added for vioc-b34-bd32.

You can see them when the vioc-b34-bd32 is running as follows from lcls-dev2:

lclshome -> User Dev Displays.... -> Test IOCs/Dev Event IOCs....

The buttons for the various panels for VIOC:B34:BD32 are towards the bottom of this screen.

For easy access to your IOC's EVR panels in LCLSDEV, add the necessary buttons to this page.

The 'Pattern' button brings up the 'Pattern Diagnostics' screen.

'PMC EVR:B34:BD32' button brings up the Low Level 'Diagnostics' screen for EVR;B34:BD32

'Triggers' button brings up 'Trigger Configuration' setup screen.

7. What is PXE, DHCP, TFTP and NFS and why are they needed by my linuxRT IOC?

LinuxRT is installed on our system using the **Preboot Execution Environment (PXE)** method of network booting.

We enable the PXE/network-booting method in the BIOS.

In order to use PXE there is a boot-server that will allow our client system to :
(a) Request an IP address (via DHCP)
(b) Download a kernel (via TFTP)

With both of these services in place any system which supports PXE/network-booting should be able to gain an IP address, fetch a kernel, and boot without an installed operating system.

PXE uses three distinct network protocols that map to three server processes to perform the installation.
In our case, some of the processes run on lcls-dev1 (LCLSDEV daemon)

Both DHCP and TFTP services run on the LCLSDEV host '**dhcp3**' maintained by SCCS.

(a) Dynamic Host Configuration Protocol (DHCP)

PXE uses DHCP to deliver initial network configuration options to client nodes.
The DHCP server supplies the PXE boot plug-in with
(i) IP address
(ii) TFTP server address
(iii) Stage 1 image boot-loader name from which to download and execute the image.

As the supplied PXE installation environments are non-interactive and will unconditionally reinstall a client machine,
we have the client associate its MAC address with a specific OS installation before starting the PXE boot.

The configuration information, in our case, in addition to IP/MAC address, includes a hostname and a pointer to the Master Starupt script in afs for our IOC.
It has an optional root-path variable pointing to the afs area which hosts the boot image that is served via TFTP.

This can be over-ridden as will be seen later.

When the Linux server is rebooted or power-cycled, PXE will attempt the network booting method first and as a first step it will contact the DHCP server to retrieve the network configuration information.

Hence, every new linuxRT ioc (host) needs to be added to the DHCP server configuration file in afs.

This file is in /afs/slac/service/dhcp-pxe/dhcpd.conf

Note that the DHCP service running on **dhcp3** is intended only for booting embedded devices like our linuxRT servers that are connected to the LCLSDEV and SSRL subnets.

The MAC addresses for such devices must be registered in CANDO and assigned fixed IP addresses.

The IP/MAC address of the primary ethernet that will fetch the linuxRT boot image is defined here.

To add a new host to the DHCP configuration, contact Thuy.

After a new ioc is added to dhcpd.conf, the DHCP service must be restarted.

To restart DHCP Server, from a Unix command line type:

remctl dhcp3 dhcp check

remctl dhcp3 dhcp restart

For help:

remctl dhcp3 dhcp help

Currently only Thuy, Ernest and a couple of others have permissions to perform this restart on dhcp3.

Here's is an example - cpu-b34-bd32:

```
host cpu-b34-bd32 {
# SuperMicro (INTELx86)
#
hardware ethernet 00:25:90:D1:95:1E;
fixed-address 134.79.218.190;
option host-name "cpu-b34-bd32";
if ( substring( option vendor-class-identifier, 0, 5 ) = "udhcp" ) {
filename "/afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/startup.cmd";
option root-path " afsnfs1:/afs/slac:/afs/slac ";
}
}
```

To find out more about how our linux server boots up linuxRT, click on the following link:

How does the Linux Server boot up linuxRT?

(b) Trivial File Transfer Protocol (TFTP)

PXE uses TFTP that defines a simple UDP protocol for delivering files over a network.

PXE delivers kernels and initial bootstrap software to client nodes using TFTP.

tftpboot is mounted in afs area in LCLSDEV on the server dhcp3, which runs both the TFTP an DHCP services:

/tftpboot -> /afs/ slac.stanford.edu/service/dhcp-pxe/tftpboot/

The iocs retrieve the linuxRT boot image from the TFTP server from the following location:

/afs/slac/g/lcls/tftpboot/linuxRT/boot

In this location, there are several linuxRT bootimages.

These were custom-built by T.Straumann for the various Linux Servers/IPCs that we currently have setup to boot with linuxRT OS.

Of these images, we will use '3.14.12-rt9_x86_64_glibc'

It has in-built support for the Broadcom networking ethernet chipset that are used in our dev linuxRT Servers.

(c) Network File System (NFS)

The NFS service is used by the installation kernel to read all of the packages necessary to the installation process.

NFS services run on the LCLSDEV hosts **afsnfs1** and **afsnfs2** maintained by SCCS.

This service makes available the boot directory to all linuxRT targets that boot as diskless clients.

All clients have read-only permissions to this directory.

The linuxRT iocs need some additional NFS Mount Points to write their data some where.

surrey04b is an NFS Appliance and is used by iocs for data and have both read and write permissions to the data directory ($IOC_DATA).

In LCLSDEV, we must obtain permissions for the iocs to write to the $IOC_DATA directory.

Fill out a Service Request abd submit to SCCS to obtain permissions for your ioc to write to this directory :

https://www-rt.slac.stanford.edu/rt3/SelfService/Forms/IocNfs.html

Jingchen has created a diagram that shows linuxRT boot process' communication flow:

(1) LinuxRT boot process

The system boot process communication flow ends when IOC startup.cmd is executed.

It is then picked up by the IOC application startup process that is further explained in following sections.

They are also captured in more diagrams as below:

(2) liunxRT Boot Sequence and Scripts

The above flow chart is also under Question 15.

(3) For a more up-to-date and complete overview of the linuxRT system in ICD development area, refer to Alisha's block diagram below:

linuxRT System Overview in LCLSDEV subnet

8. How do I start my CPU? Where is my host's statrup.cmd?

There are a few scripts that automate this process.

To begin with, there is the 'ipxe.ini' script in the tftp boot area /afs/slac/g/lcls/tftpboot/linuxRT/boot that PXE will run.

This is where the version of (linuxRT) kernel to run is specified as follows:

set vers 3.2.13-121108

This version number can be over-ridden by a chained, host-specific pxe init script to load an image different from the above:

chain ${hostname}.ipxe ||

For example, we have defined a script specifically for our ioc cpu-b34-bd32.ipxe which chooses to load the latest linuxRT image:

set vers 3.14.12-rt9_x86_64_glibc

This is also the place to over-ride the 'root-path' option specified in the DHCP configuration file dhcpd.conf.

For example, we may decide to over-ride the afsnfs1 server and instead choose to get my boot image from lcls-dev3 server:

set extra-args brd.rd_size=524288 ROOTPATH=lcls-dev3:/afs/slac:/afs/slac BOOTFILE=/afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/startup.cmd

The 'ipxe.ini' script loads the linuxRT kernel via the TFTP protocol:

kernel --name linux tftp://${next-server}/linuxRT/boot/${vers}/bzImage && initrd tftp://${next-server}/linuxRT/boot/${vers}/rootfs.ext2 || shell
imgargs linux debug idle=halt root=/dev/ram0 console=ttyS0,115200 BOOTIF_MAC=${net0/ mac:hex } ${extra-args} || boot || shell

After linuxRT boot image is downloaded to the target and linuxRT starts to run, additional nfs mounts will be done.

The afs to nfs translator service makes available the directory structure, to all clients that have mounted this nfs space.

NFS File Servers for LCLSDEV is primarily lcls-dev3. Alternately, it can use afsnfs1 or afsnfs2 but these will be phased out soon.

One of the arguments to the kernel process is the location of the BOOTFILE that does the mounting.

The 'filename' argument (which can be over-ridden by the BOOTFILE argument for linuxRT) is as follows:

"/afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/startup.cmd"

This script is similar to and modeled after RTEMS startup.cmd. The standard startup.cmd script is a soft link to ../skel/startup_cpu.cmd.

If you use the above startup_cpu.cmd script, you can skip the rest of this section.

The remaining standard host level configuration involves setting up iocConsole (question 9) and hardware specific kernel modules (question 10).

Then system startup continues with IOC application startup (questions 11 through 14)

If non-default behavior is desired, the soft link may be replaced by a different startup.cmd. The standard startup_cpu.cmd sets up the host-wide environment by calling the generic script common/linuxRT_startup_cpu.sh.

The linuxRT_startup_cpu.sh script defines standard environment variables such as $HOSTNAME and $T_A, and calls a script for facility specific environment setup, facility/linuxRT_env.sh.

When linuxRT loads and starts, the kernel process is run as "root" user.

Hence it has permissions to setup the nfs mounts which is done by the following line in linuxRT_startup_cpu.sh:

linuxRT_nfs.cmd

Additional NFS Mount Points for linuxRT pertaing to the ioc data directory $IOC_DATA are mounted as well.

The next line in the linuxRT_startup_cpu.sh file loads the system specific linuxRT kernel modules.

More on kernel modules under question (10).

This can also be done only by the "root" user:

/afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/kernel-modules.cmd

Next we must start the caRepeater process:

(http://www.aps.anl.gov/epics/base/R3-14/12-docs/CAref.html#Repeater)

"When several client processes (virtual iocs) run on the same host it is not possible for all of them to directly receive a copy of the server beacon messages when the beacon messages are sent to unicast addresses, or when legacy IP kernels are still in use. To avoid confusion over these restrictions a special UDP server, the CA Repeater, is automatically spawned by the CA client library when it is not found to be running. This program listens for server beacons sent to the UDP port specified in the EPICS_CA_REPEATER_PORT parameter and fans any beacons received out to any CA client program running on the same host that have registered themselves with the CA Repeater. If the CA Repeater is not already running on a workstation, then the "caRepeater" program must be in your path before using the CA client library for the first time."

So we have added the following lines to linuxRT_startup_cpu.sh to start a caRepeater for all EPICS VIOCs that may be hosted by this CPU:

su $IOC_USER -c $(which caRepeater) &

Once linuxRT_startup_cpu.sh completes system-wide configuration, it is possible to automatically startup one or more EPICS IOCs and detach them using the linux screen program. The standard startup script uses linuxRT_cpu_load_iocs.sh for this. It searches screeniocs for viocs configured to run on this host and starts them.

There are other options to start IOCs when the standard automatic startup is not wanted. Rather than using screeniocs to determine what to start, a custom startup.cmd may call linuxRT_cpu_load_iocs.sh after defining environment variable $LOCAL_IOCS and optionally $IOC_EXECUTABLES to list which IOCs to start.

More directly, we can start another shell such that the ioc user account, usually "laci", can start the IOC process instead of the "root" user:

su laci -c /afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/startup-epics-bd32.cmd

or

su laci -c /afs/slac/g/lcls/epics/iocCommon/vioc-b34-bd32/startup.cmd

More on startup-epics-bd32.cmd under Question (13).

Under linuxRT, a few real time processes that need real time scheduler and kernel memory locking features, can be specifically run as such.

The screen process does not need or have RT priority.

The _MAIN_ thread in epics application process, which is started doesn't have the RT priority either.

But, other threads which are created by the _MAIN_ thread may need RT priority.

Kukhee provides some information about this to find out which processes are running with real-time priorities: Command to look up thread priorities

9. How do I monitor the server remotely as it boots up - after a power-cycle or after a 'reboot' command?

The BIOS of all our linux servers used for linuxRT development have been setup by Thuy to re-direct the console (monitor) output to one of the serial ports.

This allows us to watch the boot process remotely via our standard 'iocConsole' python script. iocConsole uses the 'screen' process to accomplish this re-direction.

Add iocConsole access by adding a line for your host to $IOC/screeniocs like:

```
cpu-b34-bd32        ts-b34-nw04   2006      lcls-dev1     # Super Micro PC - PC92965
```

where the fields are cpu-name, terminal server name, terminal server port, host where the screen process will run.

Type the following command from any LCLSDEV host:

```
iocConsole cpu-b34-bd32
```

This will establish a serial connection with the linux server from lcls-dev1 via a DIGI Terminal Server port as follows:

```
: ssh -x -t -l laci lcls-dev1 bash -l -c " pyiocscreen.py -t HIOC cpu-b34-bd32 ts-lclsdev05 2001 "
```

You can monitor the linuxRT as it goes through the PXE network booting process.

Finally, you will get the login prompt:

```
Welcome to Buildroot

cpu-b34-bd32 login:
```

Login as 'root'. No password is required.

At the shell, type the following to ensure that you are running the correct version of liunxRT for your ioc. This has the real-time Preempt RT patch.

```
# uname -a
 Linux cpu-b34-bd32 3.14.12-rt9 #1 SMP PREEMPT RT Sat Oct 11 17:27:39 PDT 2014 i686 GNU/Linux
```

To see the PCI devices in your system type the following command:

```
# lspci
```

10. What are kernel modules and how are they loaded in linuxRT?

In the linux world, kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand.
They extend the functionality of the kernel without the need to reboot the system.
For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system.
Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image.
Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

linuxRT too, lets you load and unload kernel modules dynamically.

Now we are ready to load some kernel modules essential to our ioc - like EVR.

In our 'startup.cmd' script, we have the following line which lets us customize and load our kernel modules:

/afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32/kernel-modules.cmd

The location for kernel modules is specified as an environment variable in linuxRT:

KERNEL_DRIVER_HOME=/afs/slac/g/lcls/package/linuxKernel_Modules

There are several linux and linuxRT drivers in this directory: Linux Kernel Modules

The PCI EVR230 driver is here. The following driver version used in our application has been built for the linuxRT 3.14.12-rt9_x86_64_glibc:

EVR230_PCI_EVR_DRIVER=$KERNEL_DRIVER_HOME/pci_mrfevr_linuxRT/buildroot-glibc-x86_64

The kernel drivers are installed (loaded) dynamically as follows:

```
# Load the MRF EVR230 Kernel Module for timing
insmod $EVR230_PCI_EVR_DRIVER/pci_mrfevr.ko
$EVR230_PCI_EVR_DRIVER/module_load
```

There are a couple of things to note:

1. Currently the EVR kernel modules SW has the restriction that if there are both a PMC EVR230 and a PCI-e EVR300 in a linux box, then the PMC EVR230 MUST BE initialized as card 0 and loaded first. EVR300 must be initialized as card 1.

Additionally, due to hard-coded device names in the module, it is essential to setup the following links:

```
ln -s /dev/er3a0 /dev/erb0
ln -s /dev/er3a1 /dev/erb1
ln -s /dev/er3a2 /dev/erb2
ln -s /dev/er3a3 /dev/erb3
```

2. If only one EVR (either PMC EVR230 or PCI EVR300) is installed in your system, then the above restriction does not apply and soft links are not needed.

Take a look at the module_load and module_unload scripts to see what they do. The latest scripts are here:
/afs/slac/g/lcls/package/linuxKernel_Modules/pci_mrfev300_linuxRT/buildroot-2014.08/module_load

Notice how kernel modules are loaded as device drivers under the /dev/ in linuxRT much like linux.

For convinience, you can setup a soft link to your ioc's home directory in kernel-modules.cmd as below:

```
ln -s /afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32          /home/laci/cpu-b34-bd32
```

11. How do I create a startup script for my ioc?

The third and final script is specifically to setup and start your EPICS ioc. The standard startup is done entirely with 3 soft links. The commands to set up vioc-b34-bd32 would be:

```
> cd $IOC

> mkdir vioc-b34-bd32

> cd vioc-b34-bd32

> ln -s ../skel/startup_ioc.cmd startup.cmd

> cp ../facility/screenrc screenrc [then edit the log file path to match the vioc name]

> ln -s ../../iocTop/MyTest/Development iocSpecificRelease
```

If the directory name where the app-specific st.cmd is located is the same as the name of this direcotry, and its name is st.cmd, nothing more is required in this directory and you can skip the rest of this question.

These defaults may be overridden in a custom startup.cmd. Rather than creading the soft link, make a copy ("cp ../skel/startup_ioc.cmd startup.cmd") and at the top define environment variables $EPICS_IOC_APP or $ST_CMD. That is, the default values that startup.cmd passes to common/linuxRT_st.cmd are EPICS_IOC_APP is iocSpecificRelease/iocBoot/$VIOC, and ST_CMD is "st.cmd", but you can define them to be something else like

epicsEnvSet("EPICS_IOC_APP", "iocSpecificRelease/iocBoot/accl/vioc-b34-bd32")

epicsEnvSet("ST_CMD", "st.dev.cmd")

Note that because startup_ioc.cmd is a generic script, there is no "!#" at the top, so it is not executable. It is intended to be used by the linuxRT_startup_ioc.sh (script that linuxRT_cpu_load_iocs.sh calls for each IOC at boot) or linuxRT_viocConsole.sh scripts, which determine the executable. It may also be run directly as an argument to the executable on the command line, e.g. "./iocSpecificRelease/bin/linuxRT_glibc-x86_64/MyTest startup.cmd".

You may also create more customized scripts. For example the 'startupConsole-bd32.cmd' lets you start your EPICS ioc as a foreground process in your host. This lets you observe the ioc startup process to catch errors and interact with the ioc shell via iocConsole. Generally linuxRT_viocConsole.sh would be used instead, as it does the same basic thing.

This is useful during development for debugging. Once the ioc has been tested fully, you can automatically start and run the process in background.

Under linuxRT, your EPICS ioc must run as a real-time process. It must lock the kernel in memory.

The following command in your 'startupConsole-bd32.cmd' does that:

ulimit -l unlimited

The following line is also needed to run the ioc with real-time priorities:

ulimit -r unlimited

Finally, you will be running your ioc as a specific user called as 'laci' who has permissions to run this ioc:
Setup the permissions for this user 'laci':

umask 002

Now you are ready to start your IOC and have it run as a foreground process.

Create a directory called as 'vioc-b34-bd32' for your ioc as below:

mkdir $IOC/vioc-b34-bd32

cd $IOC/cpu-b34-bd32

ln -s ../vioc-b34-bd32

cd $IOC/cpu-b34-bd32/vioc-b34-bd32

Set up a soft link to the 'bin' directory of your IOC app that you created in step (6). This is where your executable is:

ln -s /afs/slac/g/lcls/epics/R3-14-12-4_1-0/iocTop/MyTest/Development/bin/linuxRT_glibc-x86_64 bin

Create an ASCII text file called 'screenrc' with the following lines in it:

deflog on

logtstamp on

defscrollback 10000

logfile /data/vioc-b34-bd32/screenlog.0

'screenrc' is passed as an argument to the 'screen' process and allows us to customize a few parameters such as number of lines stored in history buffer that we can scroll back for viewing.

In the same directory $IOC/ioc-b34-my01/vioc-b34-my01, add a startup script 'startup.cmd' for vioc-b34-my01:

It setups some shell environment variables used by all iocs, and starts the st.cmd file in your ioc boot directory.

The EPICS environment variables that are set in this script can be used by your application's st.cmd script.

The default st.cmd script generated by the module icdTemplates, expects the following environment values to be defined somewhere.

iocStartup.cmd may be a good place to define it:

epicsEnvSet("IOC_NAME","VIOC:B34:BD32")

epicsEnvSet("IOC_APP","/afs/slac/g/lcls/epics/R3-14-12-4_1-0/iocTop/MyTest/Development")

epicsEnvSet("IOC_BOOT","${IOC_APP}/iocBoot/vioc-b34-bd32")

epicsEnvSet("IOC_COMMON","/afs/slac/g/lcls/epics/iocCommon")

epicsEnvSet("SYS_FACILITY","SYS0")

In addition to setting these environment variables, this script also executes some scripts common to all EPICS IOCs, such as below:

${IOC_COMMON}/All/Dev/linuxRT_pre_st.cmd

${IOC_COMMON}/All/Dev/linuxRT_post_st.cmd

The IOC application startup occurs between the pre and post scirpts:

cd ${IOC_BOOT}

< st.cmd

12. What are the guidelines for creating App-specific st.cmd?

Work is still in progress regarding standardizing App-specific st.cmd.

The IOC engineer has freedom to be creative with this script during development.

But the following are expected as a minimum when the ioc is installed in production.

Refer to IOC Preferred Practices document for further guidance on creating IOCs.

That is the reason they are included in the default template provided by the icdTemplates module, as a guideline:

a. Making available several key paths and variables to your ioc by setting the environment variables.
   These were auto-generated during the application 'make' process and stored the 'envPaths' file under your ioc's boot directory.
   st.cmd should set all these variables using 'envPaths'.
   < envPaths
b. Set the '<TOP>' variable. You could do this via the macro ${IOC_APP} which was defined in 'iocStartup.cmd' as describe before.
   epicsEnvSet(TOP,"${IOC_APP}")
c. Load Timing-related databases using standard templates.
d. Load iocAdmin and iocRelease databases using standard templates
e. Load database for autosave using standard templates and use "makeAutosaveFiles" to autogenerate the PVs to be autoasaved /restored.

While the above is common guideline for all iocs, the following is specific to linuxRT.

You can set up real-time priorities after iocInit() for your driver threads and this can be done with a script such as rtPrioritySetup.cmd .

cd ${IOC_BOOT}

system("/bin/su root -c `pwd`/rtPrioritySetup.cmd")

Look at rtPrioritySetup.cmd as an example.

13. How do I start my ioc and monitor it?

(a) If the ioc is defined in screeniocs and automatically started, use iocConsole. See question 14 for details on automatic startup.

(b) You can directly ssh to your ioc as 'laci' from any LCLSDEV host and use linuxRT_viocConsole.sh:

ssh laci@cpu-b34-bd32

linuxRT_viocConsole.sh vioc-b34-bd32

Alternately, you can get a console to the target cpu-b34-bd32 using iocConsole and ssh as user "laci" from there:

iocConsole cpu-b34-bd32

When prompted to enter the "login" username as below, enter "laci".

Welcome to Buildroot

cpu-b34-bd32 login:

No password is needed. Just hit Enter an you will get the shell prompt "$".

Ensure the caRepeater process is running on this CPU.

(b) Ensure your IOC is not already running as below:

```
$ screen -ls
No Sockets found in /tmp/uscreens/S-laci.
```

You can also look for your application under all running processes and make sure one is not already running:

If it was running already, then you may get something as below:

```
$ ps -ef | grep MyTest
 2085 laci       0:07 bin/MyTest iocStartup.cmd
```

(c) Change to $IOC/cpu-b34-bd32 and from there start your ioc as a foreground process as below:

```
$ pwd
/home/laci
$ ls -lt
total 0
lrwxrwxrwx    1 root     lcls           45 Nov  5 15:11 bld -> /afs/slac/g/lcls/epics/iocCommon/cpu-b34-bd32
$ cd bld
```

Ensure that ***'startupConsole-bd32.cmd'*** is in your current directory.

```
$ ls -lt | grep startupConsole-bd32.cmd
-rwxrwxr-x    1 8396     1020          1225 Nov  5 15:22 startupConsole-bd32.cmd
```

Start the ioc as a foreground process:

```
./startupConsole-bd32.cmd
```

You can observe the vioc ~~b34~~bd32 go through the boot process .

If you had an EVR230 and it was initialized correctly, you may see something as below before iocInit():

```
ErConfigure(0, 0, 0, 0, 1)
Try EvrOpen, device = /dev/era3
EVR Found with Firmware Revision 0x11000007
Found a PMC_EVR /dev/era3
```

Verify if all your drivers, modules and databases loaded correctly.

If you used iocConsole, you can scroll back and forth in the screen using <CTRL> A-[and <CTRL>-A-]

If everything went well, you must get your interactive ioc shell prompt as below:

```
        vioc-b34-bd32>
```

(d) Once the ioc is started, you can interact with vioc-b34-bd32 from any LCLSDEV host via ssh laci@cpu-b34-bd32 :

To attach a screen session:

```
        screen  -r  vioc-b34-bd32
```

To detach from a screen session:

```
        <CTRL> A D
```

(e) To exit an EPICS IOC Process type exit at the IOC Shell and you will be back at the "$" prompt:

```
        vioc-b34-bd32>exit
```

If the IOC hangs too long while trying to exit, use <CTRL> C

14. How do I start my ioc automatically every time my server boots up?

You can start your ioc as a background process that will start up everytime your Linux server reboots, as follows:

Add a line to screeniocs for your vioc, e.g.

vioc-b34-bd32 cpu-b34-bd32 laci /afs/slac/g/lcls/epics/iocCommon/vioc-b34-bd32/iocSpecificRelease/bin/linuxRT_glibc-x86_64/MyTest

where the fields are vioc name, host name, IOC user, executable path. At startup, the cpu/startup.cmd script calls linuxRT_cpu_load_iocs.sh, which scans screeniocs for these lines and starts the ones with the correct host name.

The rest of this answer describes the older approach, which will still work and may be desirable in some circumstances.

In $IOC/cpu-b34-bd32, you can add another script ' ***startup-epics-bd32.cmd*** ' that lets you start your EPICS-based ioc as a background process in your host.

Invoke this script from ***$IOC/cpu-b34-bd32/startup.cmd*** :

su laci -c /usr/local/lcls/epics/iocCommon/ecpu-b34-bd32/startup-epics-bd32.cmd

The above line starts another shell so that the user called "laci" can start the IOC Process as opposed to the "root" account

Remember, the kernel modules can only be loaded onto the kernel with "root" permissions but iocs can be run by users.

'startup-epics-bd32.cmd' executes the EPICS-based IOC startup script and detaches it using the linux screen program

It contains the following lines which are similar to what we have in startupConsole-bd32.cmd:

```
export EPICS_IOCS=/afs/slac/g/lcls/epics/iocCommon

# Lock Kernel in memory
ulimit -l unlimited
# Run with real-time priorities
ulimit -r unlimited

# Set umask for the laci user
umask 002

# EPICS IOC Application Real-time Process

cd $EPICS_IOCS/cpu-b34-bd32/vioc-b34-bd32

  screen -h 8192 -c screenrc -dm -S vioc-b34-bd32 bin/MyTest iocStartup.cmd
```

The above line will automatically start your ioc and have it running in the background, every time your server reboots

To view the console output from the running ioc, you can attach the screen process to it, as described below:

(1) ssh into the ioc-b34-b32 as user "laci":

ssh laci@cpu-b34-bd32

(2) To see the list of all EPICS iocs running on this CPU, at the console type:

screen -ls

(3) To attach to vioc-b34-b32 that is running MyTest application:

screen -r vioc-b34-b32

15. Which script calls what and what parameters are passed from one to another?

Here's a flowchart that shows which script calls what and some of the important parameters that can be passed from one to another:

liunxRT Boot Sequence and Scripts

References

Page Revision History:

Rev 1.0: Shantha Condamoor Date: 15-Oct-2014

Initial Version

Rev 1.1: Kristi Luchini Date: 24-Mar-2015:

1) added a note listing Ken Brobeck's script on lcls-prod that generates the IOC_DATA directories automatically.

Only a couple of mods need to be made to make this work for all facilities, including development.

2) Had the user generate their sandbox in their home directory rather than $EPICS_IOC_TOP/MyTest Once you have a working version on development, the users can use "cram push" or "cram push --freshen"

to push the working version to $EPICS_IOC_TOP/<application>. You don' t have to CVS before using cram to push files.

3) Add a note to cram IOC application before CVSing, and provide a link for the cramming.

Rev 1.2: Shantha Condamoor Date: 10-Apr-2015:

1) added link to Alisha's linuxRT System Overview Block Diagram at end of FAQ 7.

2) added link to Kristi's linuxRT folder in slacspace to the Overview section.

Rev 1.3: Garth Brown Date: 4-Jun-2015:

Updated with the new automatic startup script info that uses the screeniocs file.

Garth's steps for creating a new linuxRT ioc:

1) Make cpu (or ioc) and vioc directories under $(IOC). If you like the vioc subdirectories under your cpu directory, make soft links.

2) in the cpu directory, ln -s ../skel/startup_cpu.cmd startup.cmd

3) in the cpu directory, create kernel-modules.cmd (or it already exists, probably doesn't need any change)

4) in the vioc directory, ln -s ../skel/startup_ioc.cmd startup.cmd

5) in the vioc directory, ln -s ../facility/screenrc screenrc (then edit the log file path to match the vioc name)

6) in the vioc directory, ln -s ../../iocTop/[whatever cram wants this to be for your app, just like you're used to] iocSpecificRelease

7) add your cpu and viocs to screeniocs

8) boot

Rev 1.4: Shantha Condamoor Date: 15-Jun-2015:

1) Added Ernest's notes about building for various targets under Question 6. Cross Compiler settings explained.

2) added link to Alisha's updated linuxRT System Overview Block Diagram at end of FAQ 7.

Rev 1.5: Garth Brown Date: 4-Nov-2015:

1) Updated path names after we decided it would be more clear to keep facility-specific files in "facility" rather than "common", and facility-agnostic files in "common".

2) Corrected how to handle screenrc files. I haven't been able to figure out a way to make a generic one that has the log file path, since screen doesn't process macros.

Rev 1.6: Alisha Babbitt Date: 23-Nov-2015:

1) Updated linuxRT System Overview Block Diagram and reattached new version at the end of FAQ 7 (version 22 replacing version 16)

Rev 1.7: Alisha Babbitt Date: 15-Dec-2015:

1) Updated linuxRT System Overview Block Diagram and reattached new version at the end of FAQ 7 (version 23 replacing version 22)

.