Realtime OS

Installation and Evaluation Guide

ELECTGON
www.electgon.com
ma_ext@gmx.net

09.06.2018

Contents

Abstract

This guide is illustrating practical steps for building and evaluation of Linux based Realtime Operating System. It is discussing only installation procedure and how to measure performance of the built system. So it is not discussing actually any theory about Realtime Operating System. This guide then may be useful for people who want to start hacking or changing some kernel parameters. It is assuming actually that the build process is targeting the same host platform. i.e. no cross compile procedure is discussed here. Cross Compiling will be easy step if main steps were successfully applied. At beginning, basic kernel configuration for Realtime response is shown then Linux Preempt_RT patch and also RTAI are implemented and explained here. Thus, you will find three possibilities for having Realtime system based on Linux.

1 Introduction

Driving a hardware is mainly what embedded systems engineers want. Operating systems are built then for that purpose. Some contexts are defining operating systems as a platform that consists of specific libraries and programs that are needed to host applications and to organize work and interaction between these applications. Not only but also provides needed interface between these applications and hardware devices if needed. This hardware interfacing function is done by what is called kernel.

So kernel can be defined as part of the operating system that mediates access to system resources (CPU, memory, disk I/O, networking). You can also consider the kernel as a core of the operating system that everything in the operating system is built on this core.

For embedded systems engineers they are interested with an operating system with minimum libraries and programs that typically fit their target hardware. In that sense they may work with the kernel only that can drive their limited hardware. In this case a kernel is considered an operating system.

The term real time operating system "RTOS" adds special specification for some operating systems. Simply it is required from that RTOS to do what you expect it to do when you expect it to do it. It is related mainly with timing of executing processes and threads. i.e. timing and speed of processes execution. That is why usually embedded systems engineers need minimum kernel as they can because it fits their limited hardware and also to meet this processing timing requirement.

For that reason managing the kernel is the base of building successfully RTOS. There are mainly 3 categories of kernels: microkernel, monolithic kernel, hybrid kernel. Monolithic kernel is best kernel that can be used for real time applications. Of course other types can be used, but this depends on how fast you want your RTOS.

Linux systems are monolithic kernels. and is used for many critical real time industries. In this document we need then to show how to use Linux distributions as a RTOS. First step is to learn how to configure the kernel of Linux correctly to meet this real time specification.

2 Kernel Development

Linux kernels are open source kernels that you can use to customize your own operating system. Generally, to develop a kernel you need a hosting system. This hosting system acts as workspace in which you can customize the kernel. To customize the kernel you have to be aware of your target i.e. the hardware in which the kernel will be running. So we have to *build* a kernel on a *host* machine to *target* specific system. For example we can build a Linux kernel that will be installed on ARM processor. So our build then is a Linux kernel, target is ARM processor, host machine will be any running Linux PC that we can work to develop our kernel. In this example we are doing cross build because target and host are different systems. Figure 1 shows different structures for kernel development.

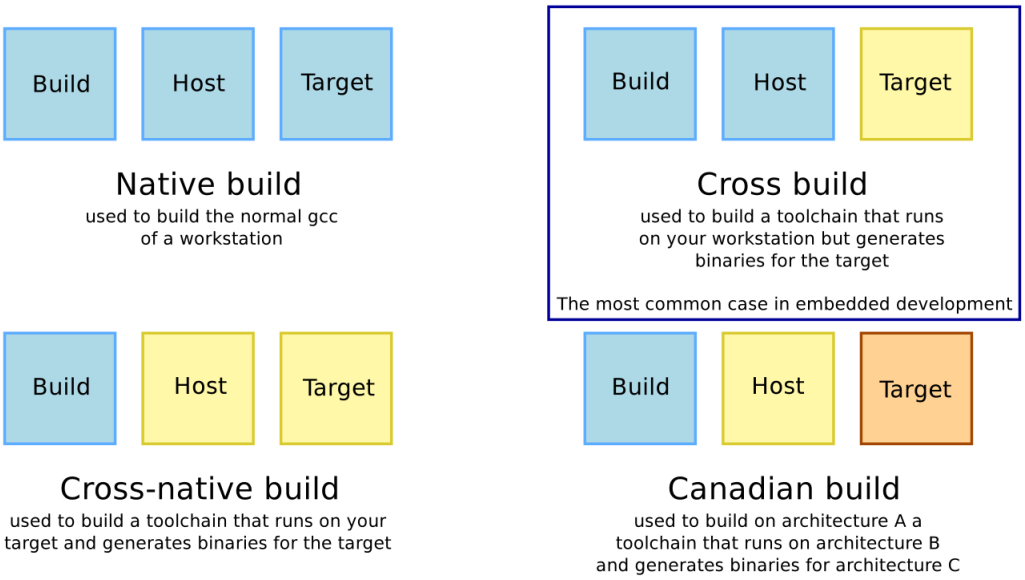

Figure 1: Compiling Kernel Platform [3]

3 Kernel Image Management

Kernel is configured in Linux environment using shell commands `make config` or `make menuconfig` or `make oldconfig` and many other options. Difference between these commands comes in the way how configuration options are displayed to the user.

In this tutorial we are going to use make menuconfig option as it displays configuration options in colored menus and radiolist. For this option ncurses must be installed in your Linux machine.

```
$sudo apt-get install libncurses5-dev
```

More details about how to run `menuconfig` is demonstrated step by step in later section. For the moment, we need to illustrate possible options that we can use for building a RTOS.

Performance of the kernel is subjected to many trade-offs, Speed-security-power-memory...etc. That is why configuring the kernel is producing many options for the developers. They are free to choose what best fit their target. In this tutorial I found it is better to put some considerations in order. I decided to consider security issues at first, then speed then power consumption then memory footprint. That means if some options give faster performance with cost of security, I will avoid these options then.

When running menuconfig the following screen (figure 2) will appear.

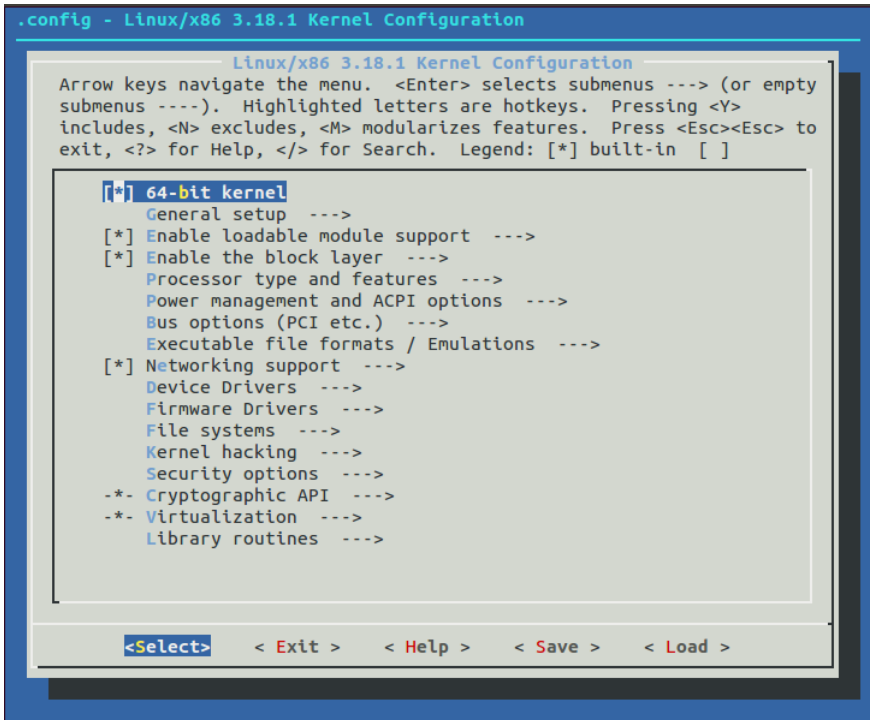

Figure 2: Main Configuration Panel

This is the main menu it is split out into the following sections:

3.1 64-bit kernel

This section asks if you want to build this kernel for 64 bit or 32 bit system. In this tutorial we work with 64-bit.

3.2 General setup

Provides overall Linux options. In this section, it is advisable to choose the following options in figure 3

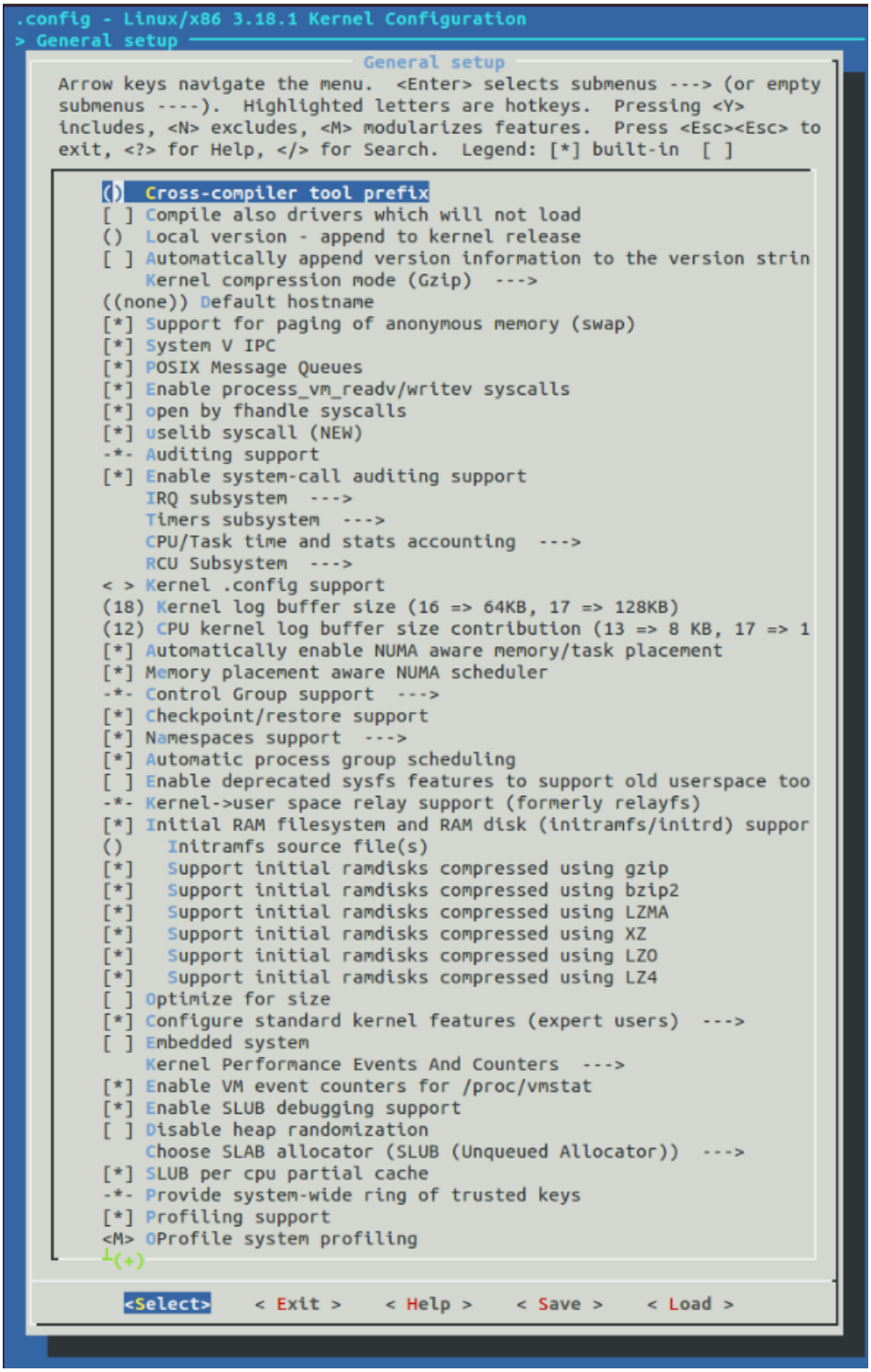

Figure 3: General Setup Options

In this General setup make sure of the following

- "support for paging of anonymous memory (swap)": enable this feature.
- "System V IPC": this is important for interprocess communication
- "Posix Message queue": this for giving priorities for message queues

- "IRQ subsystem": no option is needed.
- "Timers subsystems": use options as shown in figure 4.

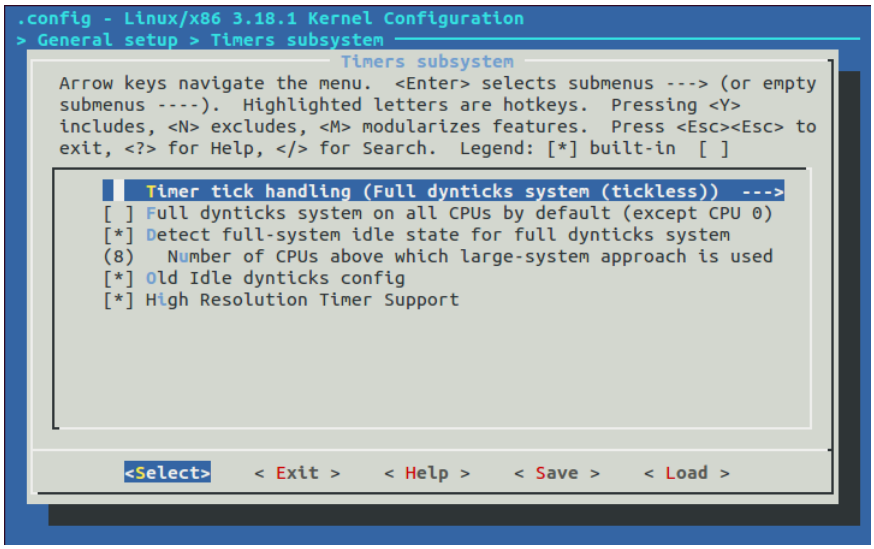

Figure 4: Timers Subsystem Settings

Make sure that "timer tick handling" option is set to tickless, this will enable a tickless system. This means that the timer interrupts will be used as needed. Timer interrupts allow tasks to be executed at particular timed intervals. Also use "High resolution timer support".

- "CPU/Task time and stats accounting ": Use options as shown in figure 5

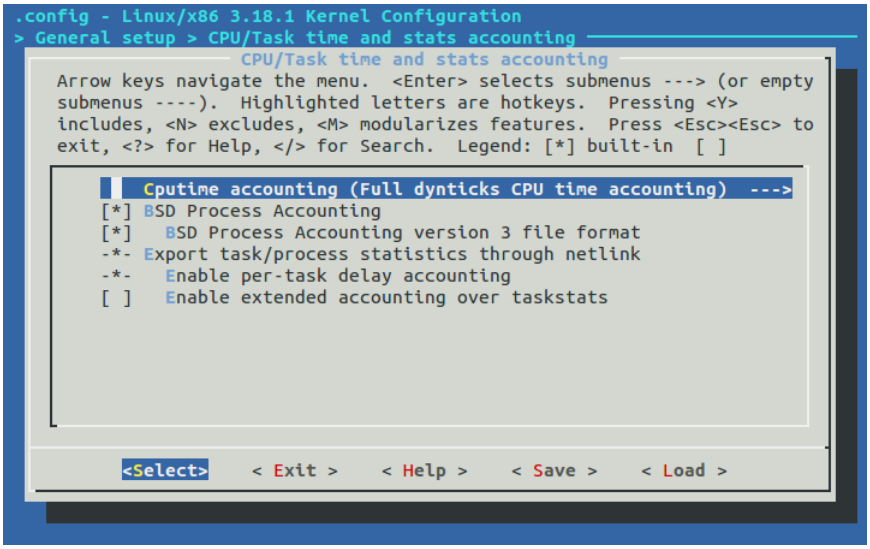

Figure 5: CPU and Task Time Settings

"BSD Process Accounting" is chosen. This feature logs a variety of information for each process that closes.

"Enable Extended accounting over taskstats" is unchecked, since this option collects extra accounting data that may cause more overhead.

Now back to the General setup main panel, With respect to NUMA options, make sure you use it. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors).

- "Automatic process group scheduling": enable this option.
- "Embedded Systems": use this option for working with embedded systems.
- "Optimize very unlikely/likely branches": use this option, this reduces overhead.
- "Stack Protector Buffer Overflow Detection": I chose none, because probability that overflow happens to stack is low since memory specs nowadays are getting better. Choosing other options (Regular or Strong) may cause un-needed interrupt to running processes, this is on cost of time.

3.3 Enable loadable module support

Provides the ability to load kernel modules. Use options as shown in figure 6.

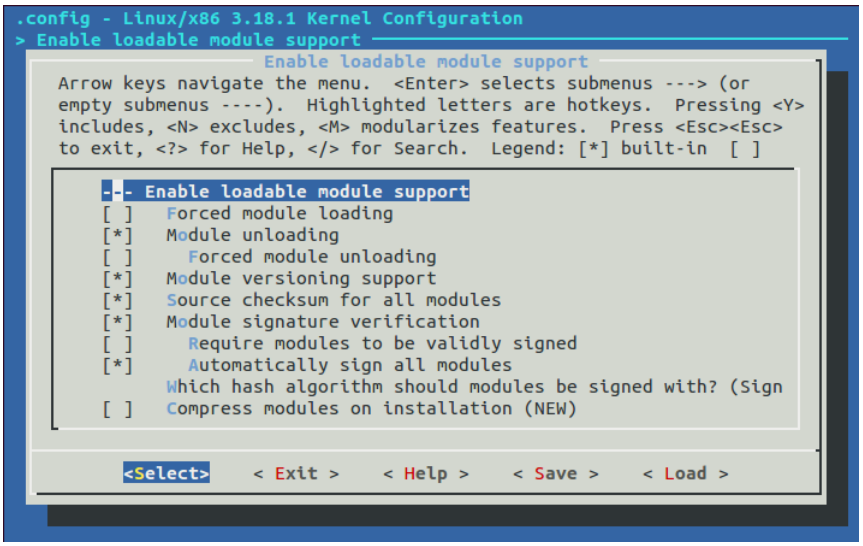

Figure 6: Enable Loadable Module Support

3.4 Enable the block layer

This needs to be enabled to be able to mount any disk drive.

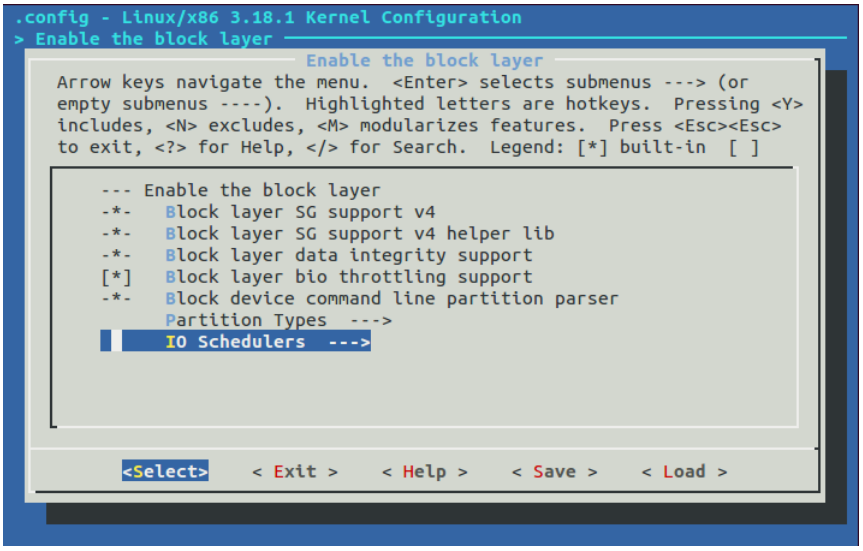

Figure 7: Enable Block Layer

It is important to check IO Schedulers. Choose options like the following shot. Enable Deadline I/O scheduler also CFQ scheduler. Use Deadline as the default scheduler

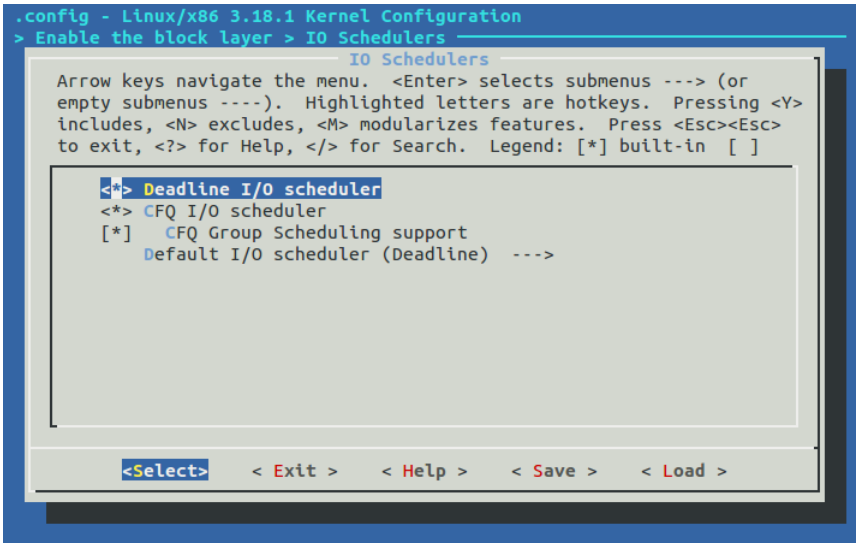

Figure 8: I/O Scheduler

3.5 Processor type and features

The defaults will set most of these properly for your hardware, but you may want to disable options that may not apply such as Multi-core scheduler support. You can also set the number of CPUs that the kernel supports. You can also set support for some specific laptop brands.

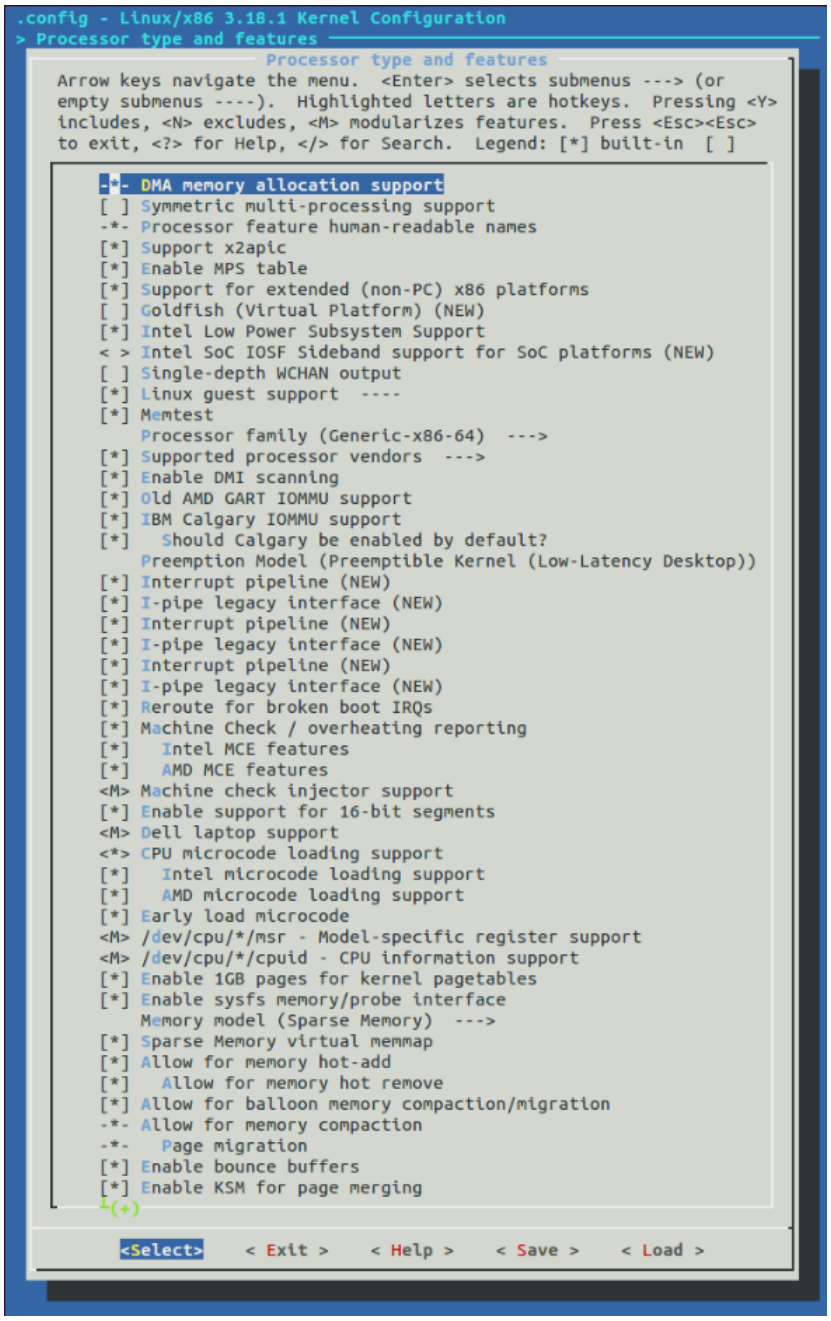

Figure 9: Processor Type and Features

- For systems with more than one CPU, it is best to enable SMP "Symmetric multi-processing support (SMP)". For single processor devices, the kernel will execute faster with this feature disabled. So we can disable at in this tutorial.
- "Single-depth WCHAN output": disable this option as it causes some overhead.
- "Preemption model": choose Low-latency Desktop

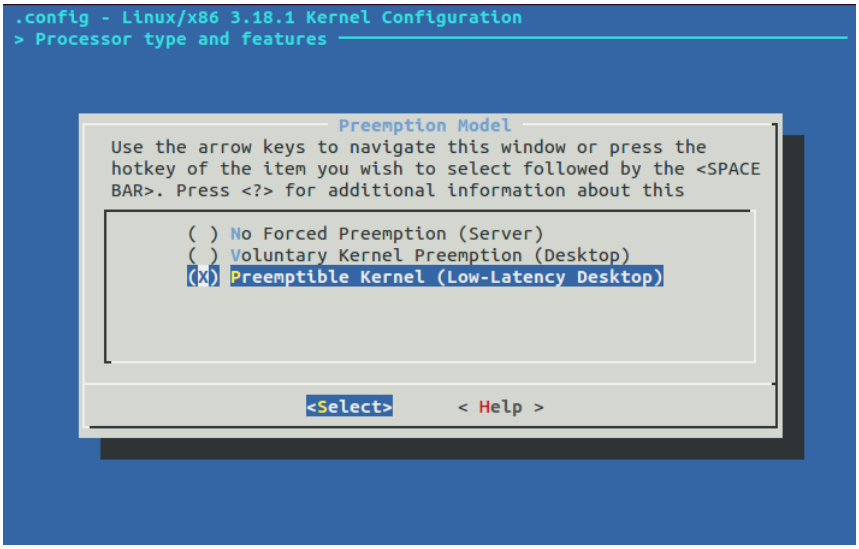

Figure 10: Preemption Model

- "Dell laptop Support": disable.
- "Transparent Hugepage support": disable for embedded systems applications.
- "Timer Frequency": this is time between interrupt checking. The time between interrupts on a timer frequency of 100HZ is 10ms, 250HZ is 4ms, and 1000HZ is 1ms. Now, many developers will instantly think that 1000HZ is the best. Well, it depends what effects you will be fine with. A large timer frequency means more power consumption and with more energy being utilized, more heat will be produced. More heat means the hardware may wear down faster. In our case here we are concerned with performance more. So we will work with 1000 HZ.

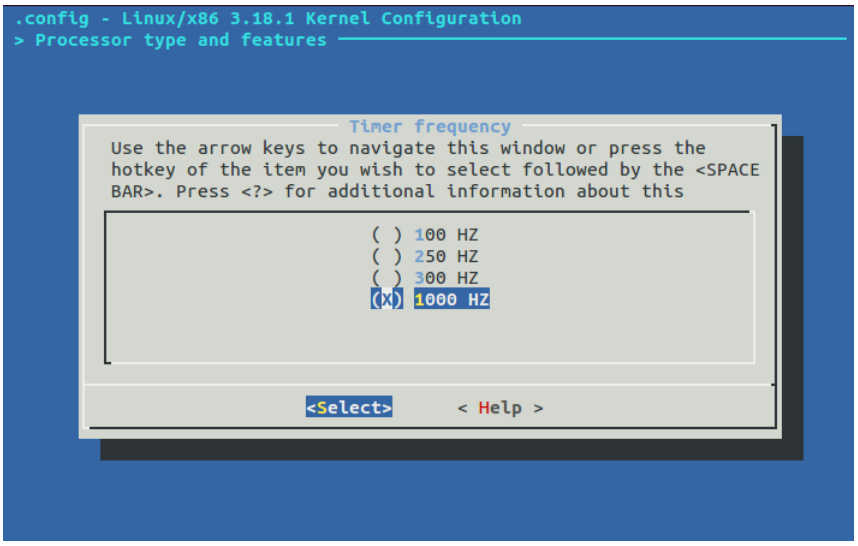

Figure 11: Timer Frequency

3.6 Power management and ACPI options

Controls ACPI (Advanced Configuration and Power Interface) . These options are most useful on laptops for power consumption optimization.

- "Power Management Debug Support": Disable this option.
- "ACPI (Advanced Configuration and Power Interface) Support": Enable this option.
- "SFI (Simple Firmware Interface) Support": Disable this option.
- "Frequency Scaling": Disable this option.
- "CPU Idle": Disable this option.

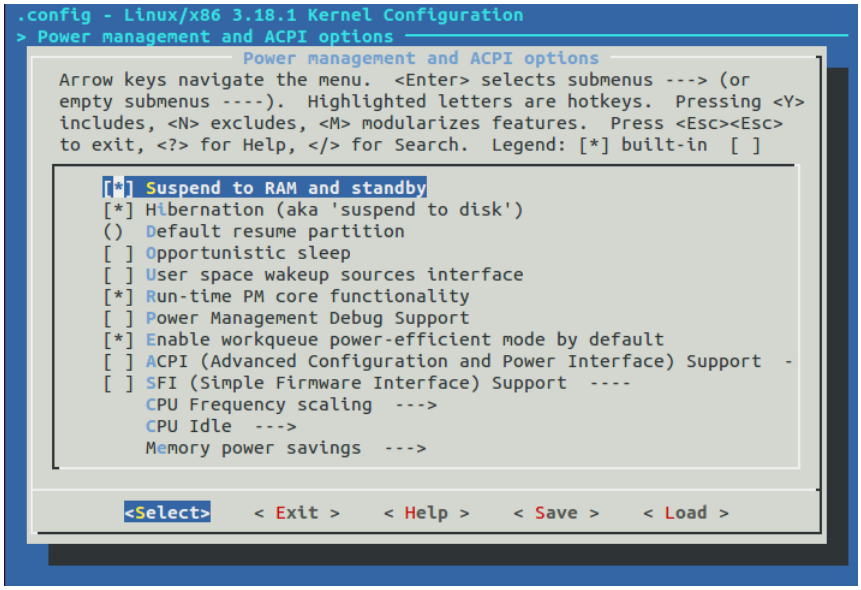

Figure 12: Power Management Options

3.7 Bus options (PCI etc)

A bus is a physical connection between several devices. The most popular bus technology within a computer nowadays is PCI (or PCI Express)

"PCI Express ASPM Control"–>"Default ASPM Policy": use Performance.

3.8 Executable file formats / Emulations

Within this section you can select what binaries (format for executable files with machine instructions inside) Linux should support. Use default options.

3.9 Networking support

This is where networking (including wireless) is enabled. Netfilter (firewall) capabilities are also defined here. The defaults are generally satisfactory. Use default options.

3.10 Device Drivers

This is one of the most important configuration areas. If you want the hardware to work, it has to be enabled with a driver. Check your devices on a currently running system with ‘lspci -v‘ to confirm what hardware you have. Enable any network or usb devices that you may have. Video drivers and sound cards are also enabled here.

3.11 Others

In the remaining sections it is OK to use the default options.

- Firmware Drivers
- File systems
- Kernel hacking
- Security options
- Cryptographic API
- Virtualization
- Library routines

To summarize what are the important options, refer to table 1 as a checklist while configuring a kernel.

Section	Features	Sub-features	Action
64-bit kernel			Enable
General Setup	Support for paging of anonymous memory (swap)		Enable
	System V IPC		Enable
	Posix Message queue		Enable
	IRQ subsystem		No action
	Timers subsystems	Timer tick handling	Tickless
		High resolution timer support	Enable
	CPU/Task time and stats accounting	Enable Extended accounting over taskstats	Disable
	Automatic process group scheduling		Enable
	Embedded Systems		Enable
	Optimize very unlikely/likely branches		Enable
	Stack Protector Buffer Overflow Detection		None
Enable Loadable module support	Module Unloading		Enable
	Source Checksum for all modules		Enable
Enable Block Layer	IO Schedulers	Deadline I/O scheduler	Enable
		CFQ I/O scheduler	Enable
		Default I/O scheduler	Deadline
Processor Type and features	Symmetric multi-processing support (SMP)		Disable
	Single-depth WCHAN output		Disable
	Preemption model (if you are not using any realtime patch)		Preemptible
	Interrupt Pipeline		Enable
	Dell laptop Support		Disable
	Transparent Hugepage support		Disable
	MTRR Support		Enable
	Timer Frequency		1000 HZ
Power Management and ACPI options	Power Management Debug Support		Disable
	ACPI (Advanced Configuration and Power Interface) Support		Enable
	SFI (Simple Firmware Interface) Support		Disable
	Frequency Scaling		Disable
	CPU Idle		Disable
Bus Options	PCI Express ASPM Control	Default ASPM Policy	Performance
ANY OTHER SECTION USE DEFAULT OPTIONS			

Table 1: Kernel Important Configurations

4 Building RTAI

Previous illustration described how to tweak the Linux kernel in order to act hardly for realtime prerequisites. However, there are some ready Linux distribution that were developed already to be used for realtime applications. RTAI is one of these Linux based realtime system. It can be considered as a hard Realtime Linux distribution that was produced in April 1999. In this section we will try to build and use it.

To build an RTAI, it is simply by applying RTAI patch to a Linux kernel. You need support of a running OS to do that. At time of this tutorial writing Linux.13.03.0-40 was the latest version of Linux. So this was the operating system used to build RTAI. You still need the following packages to build RTAI easily and successfully.

- Compiler: gcc and g++ compilers are needed during kernel configuration. Check your compiler version by typing

```
$gcc --version
$g++ --version
```

  in our demo here we had version 4.8.2. Note that usually these compilers are installed in usr/bin/

- Basic Kernel Configuration Menu: To launch a menu for kernel configuration, we need package called libncurses5-dev install this package by

```
$apt-get install libncurses5-dev
```

- Module loader: These tools will be needed to load kernel .ko modules, such as the rtai_hal.ko install it by:

```
$apt-get install module-init-tools
```

After installing needed packages and tools, we will go through the following steps to complete RTAI installation.

1. Create directory for your installation work, lets assume that you create directory called RTAI.

```
$cd /home/your/preferred/directory
$mkdir RTAI
```

2. Get your RTAI version, go to www.rtai.org to find your needed version. At time of this document, latest version was 4.0. save it in your working directory. In terminal window open this directory

```
$cd RTAI
```

3. Unpack this tarball file in your working directory

```
$tar xvf rtai-4.0.tar.bz2
```

this should extract this tarball file in a folder called rati-4.0

4. Important step is to check which linux kernel versions are supported by rtai-4.0, you can do that by

```
$cd rtai-4.0/base/arch/x86/patches
$ls
```

In this folder there are patch files for different Linux kernel versions, what I have found already:
hal-linux-3.4.67-x86-4.patch
hal-linux-3.4.6-x86-4.patch
hal-linux-3.5.7-x86-4.patch
hal-linux-3.8.13-x86-4.patch
So you can guess that rtai-4.0 is working with those kernels only (3.4.6, 3.4.67, 3.5.7, 3.8.13).

5. Based on previous step, choose your Linux kernel. In this tutorial I worked with linux-3.4.67. Go to the following link to find your suitable version of Linux kernel
https://www.kernel.org/pub/Linux/kernel/
download it, save it in your working directory. In terminal window go to this working directory

```
$cd /home/your/preferred/directory/RTAI
```

6. Unpack this Linux kernel-package

```
$tar xvf linux-3.4.67.tar.bz2
```

7. In this step we will apply RTAI to Linux kernel

```
$cd /home/your/preferred/directory/RTAI/linux-3.4.67
$patch -p1 -b < ../rtai-4.0/base/arch/x86/patches/hal-linux-3.4.67-x86-4.
    patch
```

8. So far, RTAI has been applied to Linux.3.4.67 kernel. To build an image of this Linux kernel, we start with configuring this kernel. You can use ready configuration file that you know it matches your hardware specs. For our demo here I used configuration file of the running OS (Linux.13.03.0- 40)

```
$cp /boot/config-3.13.0-40-generic .config
```

this command should take a copy of your running OS and put it in /home/your/preferred/directory/RTAI/linux-3.4.67 with the name .config

9. Run kernel configuration menu

```
$make menuconfig CC=/usr/bin/gcc-4.8.2 CXX=/usr/bin/g++-4.8.2
```

Note: make sure of versions of your gcc and g++ compilers.

10. At this step, we need to choose our configuration options. Please refer to previous section “Kernel Image Management” about recommended options for real time operation.

11. Compile the kernel: we will create .deb package. You can use this .deb package and install in any hardware having the same processor architecture. In order to do that the following packages are needed.

```
$apt-get install kernel-package fakeroot
```

Now, the following commands should be run to clean and compile the kernel (I am assuming that you still in /home/your/preferred/directory/RTAI/linux-3.4.67 directory)

```
$make-kpkg clean
$fakeroot make-kpkg --initrd --append-to-version=-you-can-write-your-
    preferred-name-here kernel_image kernel_headers
```

Note that the previous command takes more than **one hour** to be finished. When it is finished go up for one directory, you should see .deb packages. To install it in your current machine type the following

```
$cd ..
$dpkg -i *.deb
```

If everything went successfully, this means that you have installed your RTOS. Now it is time to experience it. Reboot your system to load this new RTOS.
For more reference about installing RTAI, you check it in this reference [4].

5 Building PREEMPT_RT

It is typically same steps that we followed while installing RTAI. We can quickly repeat it here.

1. Make sure of your compiler versions

```
$gcc --version
$g++ --version
```

2. Make sure you have libncurses5-dev package. If not, install it by

```
$apt-get install libncurses5-dev
```

3. Make sure you have Module loader. If not, install it by

```
$apt-get install module-init-tools
```

4. Create directory for your installation work

```
$cd /home/your/preferred/directory
$mkdir PREEMPT_RT
```

5. Get your PREEMPT_RT version. At time of writing this document, patch-3.14.29-rt26.patch.gz was the latest version. I have used it. You can get latest version of PREEMPT_RT from https://www.kernel.org/pub/linux/kernel/projects/rt/

6. Unpack this package

```
$tar xvf patch-3.14.29-rt26.patch.gz
```

7. As its name implies, this version of PREEMPT_RT is working only with linux-3.14.29 kernel. So go to kernel.org and download suitable kernel version.

8. Unpack your downloaded kernel version

```
$tar xvf linux-3.14.29.tar.bz2
```

9. Apply the patch

```
$cd linux-3.4.67
$patch -p1 -b < ../path/to/patch-3.14.29-rt26.patch
```

10. Get a copy of a configuration file

```
$cp /boot/config-3.13.0-40-generic .config
```

11. Run kernel configuration menu

```
$make menuconfig CC=/usr/bin/gcc-4.8.2 CXX=/usr/bin/g++-4.8.2
```

12. Choose configuration options. Please refer to previous section “Kernel Image Management” about recommended options for real time operation.

13. Compile the kernel: The following packages are needed

```
$apt-get install kernel-package fakeroot
```

Then start the compile process which takes so long time.

```
$make-kpkg clean
$fakeroot make-kpkg –initrd –append-to-version=-you-can-write-your-
    preferred-name-here kernel_image kernel_headers
```

When it is finished go up for one directory, you should see .deb packages. To install it in your current machine type the following

```
$cd ..
$dpkg -i *.deb
```

14. Reboot your system to work with new PREEMPT_RT RTOS.

6 How to remove un-needed kernel

If it happened and some error occurred in your build or you may mistakenly applied the new kernel to your system. You want then to remove that installed kernel. In order to do that, find out all installed kernel versions

```
$dpkg –list | grep linux-image
$sudo apt-get remove linux-image-xx.xx.xx
```

where xx.xx.xx is your kernel version that you would like to remove.

7 RTOS Evaluation

We can categorize RTOS performance metrics into the following three main categories

- Memory footprint.
- Latency.
- Service Performance.

Figure 13 illustrates these categories, below is some explanation about these metrics.

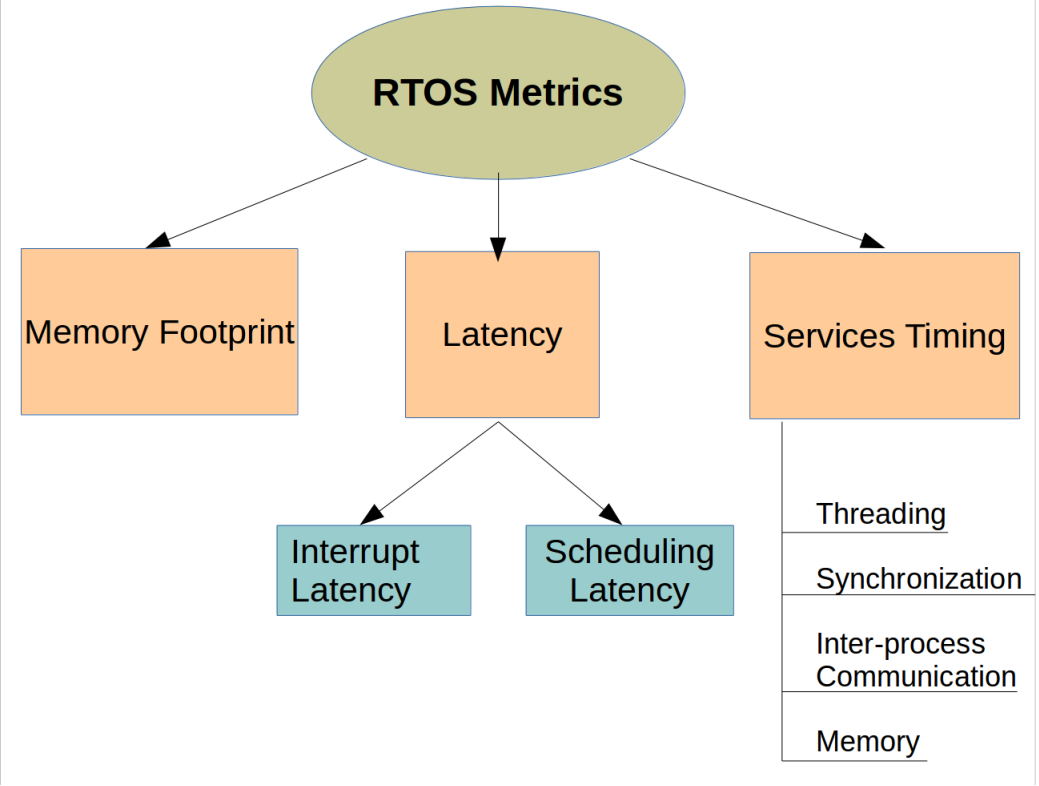

Figure 13: RTOS Performance Metrics

Memory footprint

Memory footprint is an estimate of RAM and ROM requirements of an RTOS on a specific platform. To measure footprint of OS, This can be done using command-lines like
'free', '/proc/meminfo', 'vmstat', 'top', etc [6].

Latency

It is the time taken by the system to respond to an interruption or another task. To measure Interrupt Latency, this should be based on creating some running tasks, then causing and interrupt then, measure response time. The point is to create different running task with different priorities to see how your OS is able to interrupt it. Also another aspect is to work with different preemptive or non-preemptive schemes. One of the methods also is to measure context switching time which can be done by calling a process while another process is running.

Services Performance

What is meant here also is the timing needed by the OS to complete a service. This service may be creating or stop a thread. It may be also an Interprocess Communication. It can be also memory allocation or deallocation.

7.1 Test Tools

We need to have a tool with which we can measure the performance of RTOS. You are free to develop your own techniques that you see it enables you measuring that. You can find also ready tools included with your RTOS, by running these tools (or scripts), you can get some info about performance of your RTOS. Like RTAI, it is included with some ready tests that you can run it for performance evaluation (you can find these test under install_directory/testsuite/kern).

For our context here, we are using two test tools that are free and available online: Cyclictest and LTP. There are different tools that you can use also, you can find it in

https://rt.wiki.kernel.org/index.php/RT:Benchmarks

7.2 Cyclictest

In Linux, rt-test (real-time test) is used. It's a tool that provides a mechanism to measure the latency of the processor N number of times. It creates an M thread that checks and rechecks how much time (in microseconds) the processor takes to respond during a period of time.

The idea is we want to measure time from periodic event to task wakeup time (measuring worst case response time of realtime task)

- Periodic event can be software timer or nanosleep
- Creates a number of tasks with different priority

You can find more about Cyclictest in https://rt.wiki.kernel.org/index.php/Cyclictest

To work with Cyclictest, First install the rt-tests source using Git

```
$cd /choose/your/preferred/working/directory
$git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git
$cd rt-tests
$make all
```

then to put a copy in your bin folder to be able to run it anytime

```
$cp ./cyclictest /usr/bin/
```

You may get an error about missing numa.h, which is required for the cyclictest. You may see something like this

```
rt-tests # make
cc -D VERSION_STRING=0.84 -c src/cyclictest/cyclictest.c -D_GNU_SOURCE -Wall  -
    Wno-nonnull -Isrc/include -O2 -DNUMA
In file included from src/cyclictest/cyclictest.c:37:0:
src/cyclictest/rt_numa.h:23:18: fatal error: numa.h: No such file or directory
    compilation terminated.
make: *** [cyclictest.o] Error 1
```

To resolve this, do the following

- If your system is Debian-based systems: Install the libnuma-dev package.
- If your system is RHEL/CentOS systems: Install the numactl-devel package.

Output of Cyclictest

When running cyclictest:

- The AVG Represents the average latency is being measured on the system.
- The MAX Represents the maximum latency detected on the system.
  Note: The values showed in cyclictest are represented in microseconds.

Max Latency Range (in microseconds):

- ~0 to 3000: Excellent Lower Latency than the default latency of our Drivers
- 3000 to 8000: Good Low Latency, Require to change the default latency of our Drivers
- 8000 to 15000: Fair Mid Latency, Require to change the default latency of our Driver
- 15000 to 25000: Problematic High Latency, Need to determine why the system has that latency
- 25000 to ∞: Bad Our Drivers won't be able to work[7]

7.3 LTP (Linux Test Project)

LTP is test suite that is built by IBM for testing performance of RTOS. This project can be found in this link

https://rt.wiki.kernel.org/index.php/LTP%28Realtime_Test_Tree%29

This test suite has many ready tests that measure different metrics of the RTOS. To install LTP project, do the following

The package libcap is needed. So make sure that this package is installed

```
$sudo apt-get install libcap libcap-devel
```

Then start installing LTP

```
$wget http://downloads.sourceforge.net/ltp/ltp-full-20090531.tgz
$tar xvzf ltp-full-20090331.tgz
$cd ltp-full-20090531
$./configure
```

Note that in last step we used ./configure without any option just for declaration, However you have to run configure with enabled realtime test suite first.

```
$./configure --with-realtime-testsuite
$make
$cd testcases/realtime
$make
```

For additional knowledge about how to use this test suite, you can check this link https://github.com/linux-test-project/ltp/wiki/GettingStarted

8 RTOS Test Execution

In the following, Some test cases were implemented to evaluate different RTOS using previously mentioned tools.

8.1 Test Methodology

All test execution was based on measuring system metrics using test tools (Cyclictest, LTP) while running program/script which generates sufficient amount of CPU activity and IO operations (example: reading/writing to disks) to keep the system busy 100% of the time. A shell script is used for that. (which is used from source: http://www.versalogic.com/). For our demo here we will call it System_load_script_code.sh.

```
#!/bin/bash
# This script is used from: http://www.versalogic.com/
defaultseconds=60
seconds="$1"

if test -z "$seconds" || test "$seconds" -lt 1
then
seconds=$defaultseconds
fi

cd /path/to/rt-tests

# Copy file zero in dev to null
(while true; do dd if=/dev/zero of=/dev/null bs=1024000 count=1000; done) &
    echo "Started DD"

#running ping
ping -q localhost & echo "Started ping flood"

#start hackbench which makes stress on the cpu
(while true; do nice ./hackbench >/dev/null; done) & echo "Started hackbench."

#display size of all files, store result in /dev/null
(while true; do du / 1>/dev/null 2>&1; done) & echo "Started DU..."

while test $seconds -gt 0
do
echo -n -e \\rStill $seconds seconds to wait ...
sleep 1
seconds=`expr $seconds - 1`
done
echo
echo "Script Terminated..."
```

Note: you have to be root for running all these tests. So before beginning your test, type

```
$sudo -s
```

8.2 Test Cases

8.2.1 Memory footprint

In this test we simply will run command line

```
$vmstat -s
```

We will use the Read/write shell script in parallel to make system busy as much as we can. Go to directory which has system load script code and run it ./System_load_script_code.sh Don't forget chmod of the file before running it for the first time.

8.2.2 Interrupt Latency Test

First Latency test is Interrupt Latency test, this test will be using cyclictest. As explained before, cyclictest is running periodic task (nanosleep) then trying to interrupt it with another task that has different priority. We will use the same Read/write shell script in parallel also with cyclictest to make system busy as much as we can. Go to directory which has system load script code and run it ./System_load_script_code.sh

In another terminal type

```
$cyclictest -p80 -t5 -i10000 -l10000 -n
```

The previous command shall create 5 tasks with priority 80. These tasks will interrupt the nanosleep. This operation will be iterated 10000 times.

8.2.3 Scheduling Overhead

This test to measure timing consumed by preemption overhead. In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime
$./run.sh -t func/async_handler -l 5
```

The last command measures the latency involved in asynchronous event handlers. Specifically it measures the latency of the pthread_cond_signal call until the signaled thread is scheduled.

8.2.4 Context Switching

In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime
$./run.sh -t func/sched_jitter -l 5
```

This test measures scheduling jitter w/ realtime processes. It spawns a realtime thread that repeatedly times how long it takes to do a fixed amount of work. It then prints out the maximum jitter seen (longest execution time - the shortest execution time).

It also spawns off a realtime thread of higher priority that simply wakes up and goes back to sleep. This tries to measure how much overhead the scheduler adds in switching quickly to another task and back.

8.2.5 Threading Test

This test to measure time to create and start a thread. In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime/func/gtod_latency
$./run_auto.sh
```

gtod_latency is a simple program to measure the time between several pairs of calls to gettimeofday(). If the average delta is greater than just a few microseconds on an unloaded system, then something is probably wrong.

8.2.6 Synchronization

In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime
$./run.sh -t func/matrix_mult -l 5
```

In this test we compare running sequential matrix multiplication routines to running them in parallel to judge multiprocessor performance.

8.2.7 Interprocess Communication - Signals between threads

In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime
$./run.sh -t func/pthread_kill_latency -l 5
```

pthread_kill_latency measures the latency involved in sending a signal to a thread using pthread_kill. Two threads are created: the one that receives the signal (thread1) and the other that sends the signal (thread2). Before sending the signal, the thread2 waits for thread1 to initialize, notes the time and sends pthread_kill signal to thread1. thread2, which has defined a handler for the signal, notes the time it receives the signal. The maximum and the minimum latency is reported.

8.2.8 Interprocess Communication - Semaphore time for lower priority threads

In this test we will use LTP test suite. Go to your LTP directory

```
$cd /Your/LTP/DIR/ltp-full-yourversion/testcase/realtime
$./run.sh -t func/pi_perf -l 5
```

The pi_perf creates a scenario with one high, one low and several medium priority threads. Low priority thread holds a PI lock, high priority thread later tries to grab it. The test measures the maximum amount of time the high priority thread has to wait before it gets the lock. This time should be bound by the duration for which low priority thread holds the lock.

Bibliography

[1] http://www.linux.org/threads/linux-kernel-reading-guide.5384/

[2] http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt

[3] http://free-electrons.com

[4] RTAI Installation Complete Guide, Jo ao Monteiro, version 1 -2/2008

[5] https://rt.wiki.kernel.org/index.php/Main_Page

[6] http://www.binarytides.com/linux-command-check-memory-usage/

[7] http://kb.digium.com/articles/Configuration/How-to-perform-a-system-latency-test

[8] http://swift.siphos.be/linux_sea/kernelbuilding.html