White Paper
**Amarpreet Singh Ugal**
BIOS/Firmware Engineer
Intel Corporation

# Hard Real Time Linux* using Xenomai* on Intel® Multi-Core Processors

October 2009

322386

# Executive Summary

Linux* is not a hard real-time operating system as it does not guarantee a task to meet strict deadlines. The kernel can suspend a task when its time slice has completed and it can remain suspended for an arbitrarily long time (for example, when an interrupt is getting serviced). Some of the hard real-time approaches that can be applied to Linux are "Micro-Kernel Approach" and "Nano-Kernel Approach". This document covers a similar approach using Xenomai*.

The basic idea of making standard Linux hard real-time is that a small high-priority real-time kernel runs between the hardware and standard Linux. The real-time tasks are executed by this real-time kernel (run to completion) and normal Linux processes are suspended during this duration. The real-time scheduler of the real-time kernel treats the standard Linux kernel as an idle task, which when given a chance to run, executes its own scheduler to schedule normal Linux processes. But since the real-time kernel runs at a higher priority, the normal Linux processes can at any time be preempted by a real-time task.

Interrupt management is another factor handled by the real-time kernel. When an interrupt gets triggered during the execution of a real-time task, it is first received by the real-time kernel and stored. When the real-time kernel is done, the interrupt is handed over to the standard Linux kernel. If there is an associated real-time handler for the interrupt, it is executed by the real-time kernel. Otherwise if there are no more real-time tasks to run, the stored interrupt is passed to normal Linux. Different mechanisms are used to pass the interrupts from real-time kernel to normal Linux kernel. Xenomai uses an Interrupt pipeline from the  ADEOS Project.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc. §

# Contents

# Introduction

This document describes how Xenomai* can be configured on Intel® Core™ i7 multi- core Intel® architecture processors. It does not cover the internals of Xenomai or the programming of real-time applications using Xenomai.

## Intel® Core™ i7 Multi-core Intel® Architecture Processor

Intel® Core™ i7 is the first revision of the latest-generation micro-architecture processor released by Intel. It is a successor to Intel® Core™ micro-architecture.

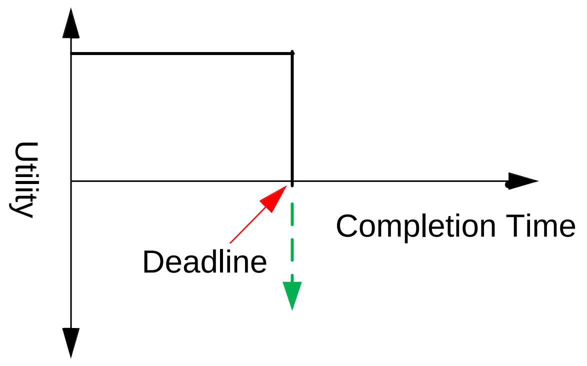
## Real-time Systems

Real-time systems in general are classified as hard real-time and soft real-time systems.

### Hard Real-time Systems

Hard real-time systems have strict timing requirements to meet deadlines or erratic behavior can occur. Figure 1 shows the timing constraints in a hard real-time system using a time-utility function. An example of a hard real-time system is an aircraft auto-pilot system. When the auto-pilot senses a potential collision with a nearby object, it changes direction of the aircraft immediately. Some of the major requirements of hard real-time systems are: minimal latency during task switching, minimal jitter, run-to completion, preemptive multi-tasking, priority inheritance and principally meet strict deadlines.

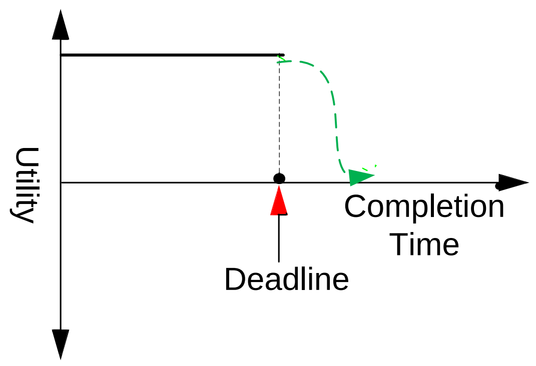
**Figure 1. Hard Real-time System Time-utility Function**



## Soft Real-time Systems

For soft real-time systems, meeting a deadline is important, but missing the same may not be catastrophic. Figure 2 shows the timing constraints in a soft real-time system using a time-utility function. The software misses the deadline. An example of a soft real-time system is a CD player, which, if ejected, may delay the actual time of responding to the eject request by few milliseconds.

**Figure 2. Soft Real-time System Time-utility Function**



## Hard Real-time Systems on Linux*

Linux* is not a hard real-time operating system as it does not guarantee a task to meet strict deadlines. The kernel can suspend a task when its time slice has completed and it can remain suspended for an arbitrarily long time (for example, when an interrupt is getting serviced). Some of the hard real-time approaches that can be applied to Linux are discussed in Micro-kernel Approach and Nano-kernel Approach.

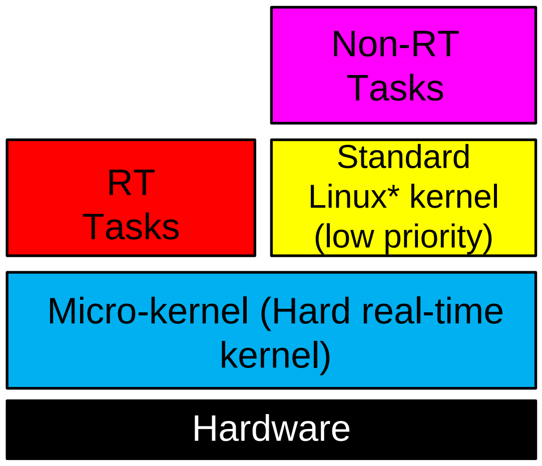
The basic idea of making standard Linux hard real-time is that a small high-priority real-time kernel runs between the hardware and standard Linux. The real-time tasks are executed by this real-time kernel (run to completion) and normal Linux processes are suspended during this duration. The real-time scheduler of the real-time kernel treats the standard Linux kernel as an idle task, which when given a chance to run executes its own scheduler to schedule normal Linux processes. But since the real-time kernel runs at a higher priority, the normal Linux processes can at any time be preempted by a real-time task. Interrupt management is another factor handled by the real-time kernel. When an interrupt gets triggered during the execution of a real-time task, it is first received by the real-time kernel and stored. When the real-time kernel is done, the interrupt is handed over to the standard Linux kernel. If there is an associated real-time handler for the interrupt, it is executed by the real-time kernel. Otherwise if there are no more real-time tasks to run, the stored interrupt is passed to normal Linux. Different mechanisms are used to pass the interrupts from real-time kernel to normal

Linux kernel. Xenomai uses an Interrupt pipeline from the ADEOS project (see References).

### Micro-kernel Approach

A hard real-time kernel called micro-kernel (see Figure 3) is provided between the standard Linux kernel and the hardware. The Micro-kernel is responsible for intercepting all the hardware interrupts and ensures that normal Linux kernel cannot suspend high-priority interrupts or tasks currently running on the micro-kernel. Besides, the standard Linux kernel runs at a lower priority in the background of micro- kernel. The micro-kernel also comes with a RT scheduler that schedules higher-priority tasks with minimal latency (for example RTLinux* or RTAI*).

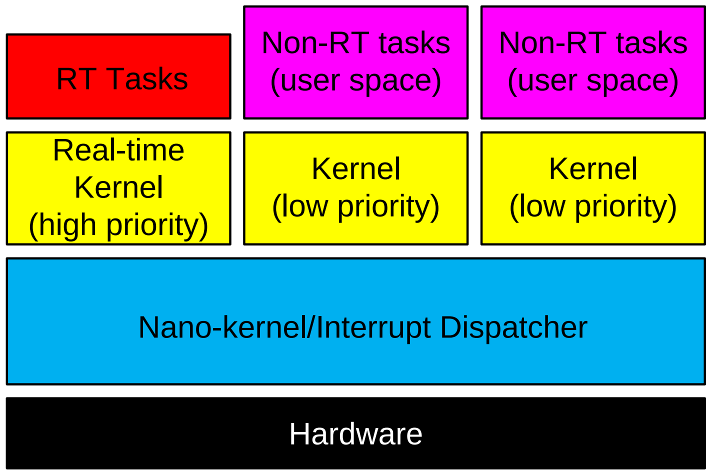
**Figure 3. Micro-kernel Approach to Hard Real-time**


Non-RT Tasks
RT Tasks
Standard Linux* kernel (low priority)
Micro-kernel (Hard real-time kernel)
Hardware

### Nano-kernel Approach

The nano-kernel approach (see Figure 4) is similar to micro-kernel in that it also provides a real-time kernel layer between the standard Linux kernel and hardware, with standard Linux kernel running in the background as a low-priority task. Nano-kernel is basically a Hardware Abstraction Layer that unlike micro-kernel, provides the facility to run multiple real-time and non-real-time operating systems on top of the nano-kernel (for example, ADEOS*).

**Figure 4. Nano-kernel Approach to Hard Real-time**


RT Tasks
Non-RT tasks (user space)
Non-RT tasks (user space)
Real-time Kernel (high priority)
Kernel (low priority)
Kernel (low priority)
Nano-kernel/Interrupt Dispatcher
Hardware

## Xenomai*

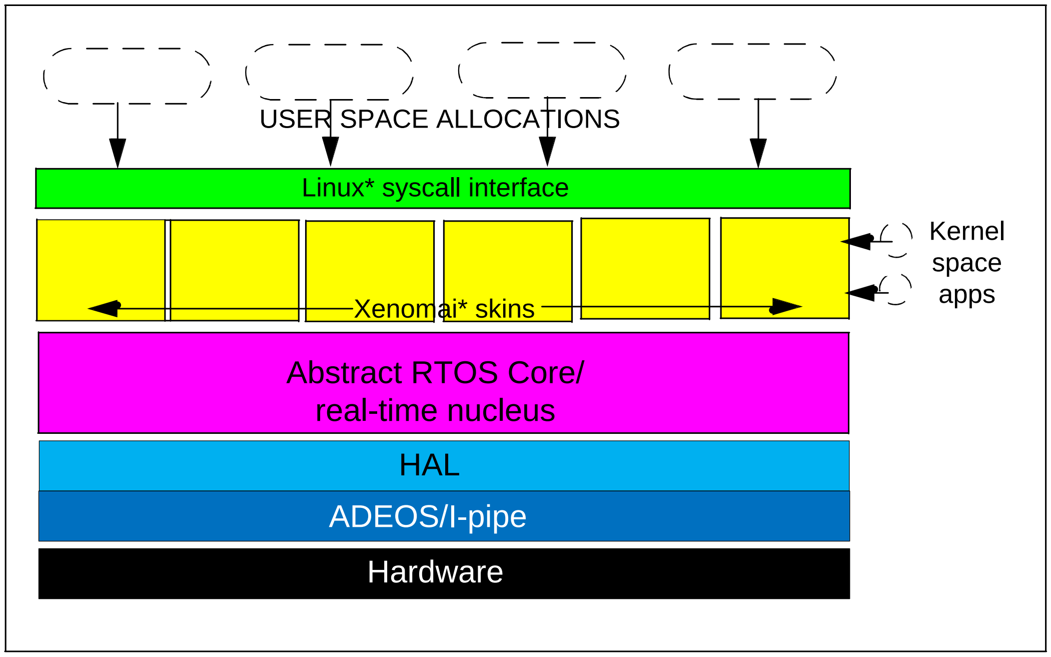
To support hard real-time capabilities to the Linux kernel, Xenomai* implements a micro-kernel between the hardware and the Linux kernel (see Figure 5). This micro- kernel is responsible for executing hard real-time tasks and intercepts interrupts, blocking them from reaching the Linux kernel, hence preventing the Linux kernel from preempting the hard real-time micro-kernel. Intrinsically, the Linux kernel executes in the background of this micro-kernel as a low-priority task that runs non-real-time tasks. Xenomai provides the hard real-time development framework with real-time support for user-space applications and implements the micro-kernel with a real-time scheduler. It also provides different APIs for creating RT tasks, timers, synchronization objects, etc., and simulates various APIs called skins (see Xenomai Skins) for real-time application, including POSIX* interface, RTAI, VxWorks*, etc., which allows easier porting of existing RT applications to Linux.

Xenomai uses the ADEOS real-time nano-kernel to handle real-time interrupt dispatching.

See References for further information on Xenomai and how to use it for creating hard real-time tasks.

***Note:*** There are many other hard real-time schemes available for Linux (for example, RTLinux (Wind River* Linux), RTAI, etc., but this document solely covers Xenomai).

**Figure 5. Xenomai* Architecture**



## Xenomai Skins

The Xenomai core is an abstract RTOS that provides operating-system resources and generic building blocks for building different RTOS interfaces called skins. These skins mimic the different RTOS APIs, allowing straightforward porting of existing applications to Xenomai. The building blocks form the Xenomai nucleus. Xenomai comes with the following skins. For further information click on the links below.

- POSIX*
- pSOS*
- RTAI*
- uITRON*
- VRTX*
- VxWorks*
- Xenomai native API*

# Installation and Configuration

## Required Packages

For finding the Xenomai package compatibility with the Linux kernel, see http://www.xenomai.org/index.php/Xenomai:News#2008-11-10_Xenomai_2.4.6

***Note:*** A Xenomai patch does not exist for Linux kernel version 2.6.18 at the time of writing this document.

Combinations of the following packages have been tested for the purpose of this document, but the same installation steps should apply if other compatible versions of the Xenomai and Linux kernel are used.

**Table 1. Required Packages**[1]

| Package | Version | Download Link |
|---|---|---|
| CentOS* | 5.2 | http://isoredirect.centos.org/centos-5/5.2/ |
| Linux* Kernel | 2.6.26.8 | http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.8.tar.gz |
| Xenomai* | 2.4.6 | http://download.gna.org/xenomai/stable/xenomai-2.4.6.tar.bz2 |

1. These packages were installed/patched on the CentOS* Linux 5.2 distribution.

## Installing and Configuring CentOS* 5.2

1. Download CentOS 5.2 Linux Distribution (see Table 1 for download link).

2. Install CentOS with default packages. Remove ‘Desktop-Gnome’ when asked for the packages to Install. This will avoid the overhead of X11.

***Note:*** X11 can also be disabled later so that it does not load at system bootup. To disable X11, go to /etc/inittab file and change the line: id:5:initdefault: to id:3:initdefault:. Disabling X-Windows saves a reasonable amount of RAM.

3. After installation, boot into CentOS.

4. Ensure the following requisite development packages are installed. Otherwise install them from the installation disk(s). If there are multiple installation disks, disk numbers where the packages can be found are highlighted in blue.

   a. `Mkdir/mnt/cdrom`

b. For each of the disks do the following, to mount/unmount and change to the install directory:

```
mount /dev/cdrom /mnt/cdrom (to mount the disk)

>> umount /dev/cdrom
(to remove the disk after package installation)

cd /mnt/cdrom/CentOS
```

c. Install the following in the listed sequence:

```
rpm -ivh kernel-headers-2.6.18-92.el5.i386.rpm (Disk 1)

rpm -ivh cpp-4.1.2-42.el5.i386.rpm (Disk 1)

rpm -ivh glibc-headers-2.5-24.i386.rpm (Disk 2)

rpm -ivh glibc-devel-2.5-24.i386.rpm (Disk 2)

rpm -ivh libgomp-4.1.2-42.el5.i386.rpm (Disk 2)

rpm -ivh gcc-4.1.2-42.el5.i386.rpm (Disk 2)

rpm -ivh elfutils-libs-0.125-3.el5.i386.rpm (Disk 2)

rpm -ivh redhat-rpm-config-8.0.45-24.el5.noarch.rpm (Disk
2)

rpm -ivh ncurses-devel-5.5-24.20060715.i386.rpm (Disk 2)

rpm -ivh elfutils-0.125-3.el5.i386.rpm (Disk 2)

rpm -ivh libstdc++-devel-4.1.2-42.el5.i386.rpm (Disk 2)

rpm -ivh doxygen-1.4.7-1.1.i386.rpm (optional, Disk 2)

rpm -ivh unifdef-1.171-5.fc6.i386.rpm (Disk 3)

rpm -ivh rpm-build-4.4.2-48.el5.i386.rpm (Disk 3)

rpm -ivh emacs-common-21.4-20.el5.i386.rpm (required for
emacs-nox, Disk 3)

rpm -ivh emacs-nox-21.4-20.el5.i386.rpm (optional,
Disk 4)
```

5. Stop the unnecessary services. A list of services currently running can be checked by issuing the following command:

```
chkconfig --list | more
```

6. The services can be shut down using:

```
chkconfig --levels 123456 servicename on/off
```

**Example:** `chkconfig --levels 123456 iptables off, will shutdown the iptables service.`

For the purpose of this document, the following services have been shut down:

```
iptables, ip6tables, yum-updatesd, sendmail, bluetooth,
cups, irda, acpid, atd, auditd, autofs, hidd and smartd.
```

The user determines the services to be shut down.

7. Remove the unnecessary cron jobs that are scheduled to run daily:

```
mkdir /etc/cron.backup
cd /etc/cron.daily
mv mlocate.cron ../cron.backup
```

***Note:*** There are few default cron jobs scheduled to run daily when CentOS is installed (for example, 0anacron, 0logwatch, cups, logrotate, makewhatis.cron, mlocate.cron, prelink, rpm and tmpwatch). Remove the ones not required by moving them to the */etc/cron.backup* directory. For further information on tuning Linux, please see Section 3 of http://download.intel.com/design/intarch/ep80579/320524.pdf.

## Installing and Configuring Linux Kernel 2.6.26.8 on CentOS* 5.2

1. Download Linux kernel source (linux-2.6.26.8.tar.gz). See Table 1 for download link. Extract and copy it to a folder (for example, */usr/src/kernels*)
2. *cd /usr/src/kernels*
3. tar -xvzf linux-2.6.26.8.tar.gz
4. mv linux-2.6.26.8 linux-2.6.26.8_xenomai-2.4.6
5. cd linux-2.6.26.8_xenomai-2.4.6
6. Edit the Makefile and change the EXTRAVERSION=.8 to EXTRAVERSION=.8_xenomai-2.4.6. This is just to differentiate this kernel from others, in case of a multi-boot kernel installation.
7. make mrproper (to configure the kernel from scratch)
8. Configure the kernel as below:

   -- make menuconfig

```
Device Drivers ---> USB support ---> OHCI HCD support <M>

<M>

Device Drivers ---> USB support ---> UHCI HCD (most Intel and
VIA) support
```

```
Device Drivers ---> USB support ---> EHCI HCD (USB 2.0)
support <M>

Device Drivers ---> USB support ---> USB Mass Storage support
<M>

Device Drivers ---> Serial ATA (prod) and Parallel ATA
(experimental)
drivers ---> AHCI SATA support <M>
```

***Note:*** The above configuration settings are built into the kernel by default. They need to be included as kernel modules; otherwise, the following warnings appear when the kernel is installed:

```
WARNING: No module ehci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module ohci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module uhci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module ahci found for kernel 2.6.26.8_xenomai-2.4.6,
continuing anyway

WARNING: No module ehci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module ohci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module uhci-hcd found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway

WARNING: No module usb-storage found for kernel 2.6.26.8_xenomai-
2.4.6, continuing anyway
```

Also, change the following kernel configuration settings:

```
Processor type and features ---> Preemption Model --->
(Preemptible Kernel(Low-Latency Desktop))
```

If memory is not over 4 GB, disable the configuration settings pertaining to *High Memory Support, PAE* and *64 bit Memory* and *IO resources*.

```
Processor type and features ---> High Memory Support ---> off

Processor type and features ---> PAE (Physical Address Extension)
Support -
--> disabled

Processor type and features ---> 64 bit Memory and IO resources
(EXPERIMENTAL) ---> disable
```

Choose the most suitable processor family for your system. To determine the type of processor being used, use the following command:

```
cat /proc/cpuinfo | grep "model name"
```

Running the above command for a two Quad Core Intel® Core™ i7 CPU configuration yields the following result:

```
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
model name      : Genuine Intel(R) CPU          @ 0000 @ 2.13GHz
```

Since, from the above output it can be seen that there are 16 CPUs, configure the processor type settings as follows:

```
Processor type and features ---> Processor family ---> (Core
2/newer Xeon)

Processor type and features ---> Symmetric multi-processing
support --->enabled

Processor type and features ---> sub-architecture type --->
Support for other sub-arch SMP systems with more than 8 CPUs
```

***Caution:*** The kernel configuration for "Processor type and features ---> sub-architecture type" should be selected correctly, otherwise Linux will fail to boot or a Kernel panic would occur. The "Support for other sub-arch SMP systems with more than 8 CPUs" is chosen for a system with two Quad Core Intel® Core™ i7 CPUs (16 virtual CPUs). For a single Quad Core Intel® Core™ i7 CPU or other standard PC or compatible, "PC-Compatible" setting would suffice.

```
Enable loadable module support ---> Forced module unloading --->
disable

Enable loadable module support ---> Module versioning support ---
> disable

Enable loadable module support ---> Automatic kernel module
loading --->enable
```

Save and exit the kernel configuration menu.

Build and install the kernel to verify if the kernel boots successfully with no errors:

```
make clean && make && make modules && make modules_install
&& make install
```

vi /etc/grub.conf

Change 'default' setting to point to 'CentOS (2.6.26.8_xenomai-2.4.6)'. Save and exit file.

***Note:*** The default setting can be altered by changing the default (highlighted below) in the '/ etc/grub.conf' file. In the example below, the Linux kernel will boot by default from '2.6.26.8_xenomai-2.4.6' Changing the default to 1 means that the kernel will boot from the 2.6.18-92.el5 kernel.

```
# grub.conf generated by anaconda

#

# Note that you do not have to rerun grub after making changes to
this file

# NOTICE: You have a /boot partition. This means that

#       all kernel and initrd paths are relative to /boot/, eg.

#       root (hd0,0)

#       kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00

#       initrd /initrd-version.img

#boot=/dev/sda default=0 timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz hiddenmenu
title CentOS (2.6.26.8_xenomai-2.4.6)

root (hd0,0)

kernel /vmlinuz-2.6.26.8_xenomai-2.4.6 ro
root=/dev/VolGroup00/LogVol00 rhgb pci=nommconf
crashkernel=128M@16M

initrd /initrd-2.6.26.8_xenomai-2.4.6.img title CentOS (2.6.18-
92.el5)
   root (hd0,0)

   kernel /vmlinuz-2.6.18-92.el5 ro root=/dev/VolGroup00/LogVol00
   rhgb quiet crashkernel=128M@16M

   initrd /initrd-2.6.18-92.el5.img
```

Change the runlevel so as to boot to the command line:

```
vi /etc/inittab

Change the line: id:5:initdefault: to id:3:initdefault:
```

Save and exit the file.

Reboot. This should boot into the new installed kernel: 2.6.26.8_xenomai-2.4.6

## Patching and Configuring Xenomai 2.4.6

1. Download Xenomai 2.4.6 (see Table 1). Extract and copy it to a folder (for example, */usr/src/kernels*).
2. *cd /usr/src/kernels*
3. tar -xvjf xenomai-2.4.6.tar.bz2
4. cd xenomai-2.4.6
5. Patch Xenomai using the following command:

```
./scripts/prepare-kernel.sh --linux=/usr/src/kernels/linux-
2.6.26.8_xenomai-2.4.6 -- adeos=./ksrc/arch/x86/patches/adeos-
ipipe-2.6.26.7-x86-2.0-16.patch
```

***Note:*** When asked for the architecture type, choose the relevant type. In this case, it is the i686.

6. cd /usr/src/kernels/linux-2.6.26.8_xenomai-2.4.6

7. For reducing latencies, change the following Linux kernel configuration settings:

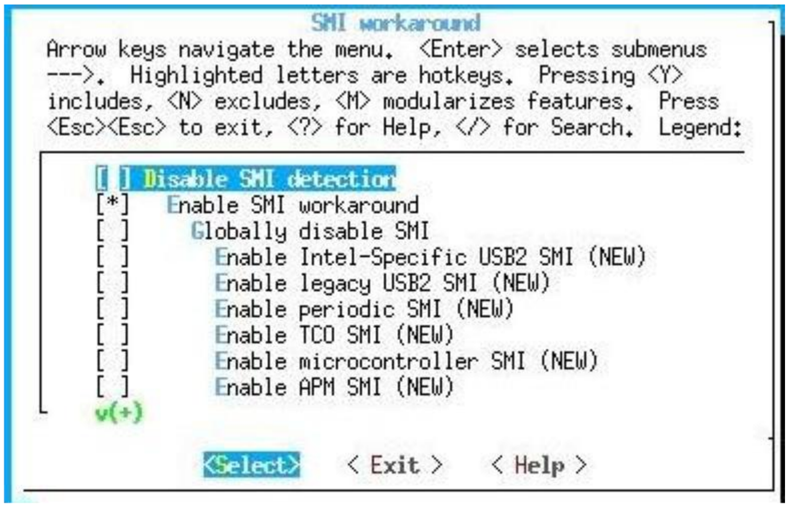
   --- make menuconfig

```
Power management options ---> Power Management support --->
disabled

Power management options ---> CPU Frequency scaling ---> CPU
Frequency scaling ---> disabled

Power management options ---> CPU idle PM support --->
disabled
```

***Note:*** It is important that System Management Interrupts (SMI) are disabled to avoid significant latencies. SMIs are being generated by the board power management hardware, and they can negate the hard real-time performance of the system. First, they can last for hundreds of microseconds, which, for many RT applications, cause unacceptable jitter. Second, they are the highest-priority interrupt in the system (even higher than the NMI). And third, an SMI cannot be intercepted because it does not have a vector in the CPU. Instead, when the CPU gets an SMI, it goes into a special mode and jumps to a hard-wired location in a special SMM address space (likely in BIOS ROM). Essentially SMI interrupts are 'invisible' to the operating system. Although SMI interrupts are handled one processor at a time, real-time responsiveness on dual-core/ SMP systems is affected. If the processor handling the SMI interrupt has locked a mutex or spinlock, which is needed by some other core, that other core has to wait until the SMI interrupt handler has been completed and a mutex/spinlock has been released.

Xenomai-related SMI options can be changed via the'Real-time sub-system' configuration setting. In case any of the SMIs are required, disable the "Globally disable SMI" configuration setting and choose from the other options (as shown in Figure 6).

**Figure 6. Options for System Management Interrupts**



```
                         SMI workaround
Arrow keys navigate the menu.  <Enter> selects submenus
--->.  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press
<Esc><Esc> to exit, <?> for Help, </> for Search.  Legend:

    [ ] Disable SMI detection
    [*]   Enable SMI workaround
    [ ]     Globally disable SMI
    [ ]       Enable Intel-Specific USB2 SMI (NEW)
    [ ]       Enable legacy USB2 SMI (NEW)
    [ ]       Enable periodic SMI (NEW)
    [ ]       Enable TCO SMI (NEW)
    [ ]       Enable microcontroller SMI (NEW)
    [ ]       Enable APM SMI (NEW)
    v(+)

              <Select>    < Exit >    < Help >
```

Also, change the following kernel configuration settings:

```
Processor type and features ---> Hign Resolution Timer Support
---> enable

Processor type and features ---> check for P4 thermal
throttling interrupt
---> disable

Processor type and features ---> Toshiba Laptop support --->
disable

Processor type and features ---> Dell laptop support --->
disable

Bus options (PCI etc.) ---> PCI Express Hotplug driver --->
disable

Bus options (PCI etc.) ---> Message Signaled Interrupts (MSI
and MSI-X) ---
> disable

Bus options (PCI etc.) ---> PCCard (PCMCIA/CardBus) support --
-> disable
```

```
Bus options (PCI etc.) ---> Support for PCI Hotplug --->
disable

Device Drivers ---> Memory Technology Device (MTD) support ---
> disable

Device Drivers ---> Fusion MPT device support ---> disable

Device Drivers ---> I20 device support ---> disable

Device Drivers ---> ISDN support ---> disable
```

Save and exit the kernel configuration menu.

8. Build and install kernel.

```
make clean && make && make modules && make modules_install &&
make install
```

9. Reboot (should boot into 2.6.26.8_xenomai-2.4.6 kernel).

***Note:*** After booting, check that the correct number of cores can be detected on the CPU. This can be done using "cat /proc/cpuinfo". If the correct number of cores is not visible, there can be a mismatch between the BIOS setting and the Linux kernel configuration. For the correct number of cores to be detected, enable SMP/multi-core detection support in the BIOS.

10. After booting, build Xenomai.

```
cd /usr/src/kernels/xenomai-2.4.6
```

11. Prepare the building of user space support using the autoconf script called 'configure'. The script also verifies that all necessary packages are available for Xenomai to build successfully.

```
./configure
```

***Note:*** The following x86 options can be passed to the 'configure' script:

```
Generic configure options

NAME              DESCRIPTION                   [BINDING,]DEFAULT(*)

--prefix          Installation directory        /usr/xenomai

--enable-debug    Enable debug symbols (-g)     disabled

--enable-smp      Enable SMP support            weak,disabled

Arch-specific configure options

NAME              DESCRIPTION                   [BINDING,]DEFAULT(*)

--enable-x86-sep   Enable x86 SEP instructions strong, disabled
                   for issuing syscalls.
```

```
                        You will also need NPTL

--enable-x86-tsc  Enable x86 TSC for timings  strong,enabled

                        You must have TSC for this.
(*) Each option enabled by default can be forcibly disabled by
passing --disable- <option> to the configure script.
```

Further information on how to use the options and resolve conflicts can be found in the README.INSTALL file in the '/usr/src/kernels/xenomai-2.4.6' directory.

12. Build and install the Xenomai binaries, libraries, and any support files into the appropriate locations:

    ```
    make && make install
    ```

13. Add the Xenomai installation paths to the PATH environment variable:

    ```
    Export PATH=
    /usr/xenomai/bin:/usr/xenomai/lib:/usr/xenomai/include:$PATH
    ```

Optionally, add the above command to .bashrc, so that it executes on launch.

***Note:*** The Kernel configuration settings described in this document are personalized settings for getting hard real-time Linux working on two Quad Core Intel® Core™ i7 CPUs. The end-user is free to modify them according to specific needs.

## Testing Installation and Running Instrumentation Examples

Xenomai comes with a number of tests that can be run to measure the performance of the real-time machine. The test cases are located in: /usr/xenomai/bin. Some of these are covered as below:

***Note:*** The test results for the individual tools presented below are for a particular dual-core machine. These results are examples showing how they can be interpreted to determine the real-time performance of the system. The results will vary for different machines with different kernel and real-time configurations and under different load conditions.

```
$cat /proc/cpuinfo | grep "model name" outputs:

   model name    : Intel(R) Core(TM)2 Duo CPU    E8500 @ 3.16GHz
   model name    : Intel(R) Core(TM)2 Duo CPU    E8500 @ 3.16GHz
```

## switchbench

The test **'switchbench'** measures the context switch latency between two real-time tasks.

```
usage: switchbench [options]

-h   - enable histogram

-p <period_us> - timer period

-n <samples> - number of samples to collect

-i <samples> - number of _first_ samples to ignore
```

Running switchbench produces the following output on a dual-core, Xenomai-patched Linux machine:

```
[root@ixa00321605 bin]# ./switchbench -n 100000 -h

== Sampling period: 100 us

== Do not interrupt this program

RTH|lat min|lat avg|       lat max|          lost

RTD|1.067|   1.131|1.451|   0

---|---range-|---samples

HSD| 1 - 2 | 99996

HSS|99996|   1.000|0.000
```

The above output shows that the average context switch latency is about 1 µs.

## switchtest

The test '**switchtest'** provides the number of task switches that can happen every second.

```
For usage of switchtest, run switchtest --help
```

For switchtest to work, change the following kernel configuration setting:

```
Real-time sub-system ---> Xenomai ---> Nucleus ---> Drivers --
-> Testing drivers ---> Context switch unit testing driver ---
> enable
```

Save configuration changes and rebuild the kernel.

Running switchtest produces the following output on a dual-core, Xenomai-patched Linux machine:

```
[root@ixa00321605 bin]# ./switchtest -T10

== Testing FPU check routines... r0: 1 != 2
r1: 1 != 2 r2: 1 != 2 r3: 1 != 2 r4: 1 != 2 r5: 1 != 2 r6: 1
!= 2 r7: 1 != 2
== FPU check routines: OK.

== Threads: sleeper_ufps-0 rtk-1 rtk-2 rtk_fp-3 rtk_fp-4
rtk_fp_ufpp-5 rtk_fp_ufpp-
6 rtup-7 rtup-8 rtup_ufpp-9

rtup_ufpp-10 rtus-11 rtus-12 rtus_ufps-13 rtus_ufps-14 rtuo-15
rtuo-16 rtuo_ufpp-17 rtuo_ufpp-18 rtuo_ufps-19

rtuo_ufps-20 rtuo_ufpp_ufps-21 rtuo_ufpp_ufps-22

RTT| 00:00:01

RTH|ctx switches|-------total

RTD|21643|   21643

RTD|21666|   43309

RTD|21666|   64975

RTD|21666|   86641

RTD|21666|   108307

RTD|21666|   129973

RTD|21666|   151639

RTD|21666|   173305

RTD|21666|   194971

RTD|21551|   216522
```

The above output shows that about 21666 context switches can happen in a second in a test duration of 10 seconds.

## cyclictest

The test '**cyclictest'** measures the time between configured timer expiration time and the actual expire time.

```
Usage:

cyclictest <options>

-b USEC --breaktrace=USEC send break trace command when
                          latency > USEC
```

```
-c CLOCK --clock=CLOCK      select clock

                            0 = CLOCK_MONOTONIC (default)

                            1 = CLOCK_REAL TIME

-d DIST  --distance=DIST    distance of thread intervals in us
default=500

-i INTV  --interval=INTV    base interval of thread in us
                             default=1000

-l LOOPS --loops=LOOPS      number of loops: default=0(endless)

-n        --nanosleep       use clock_nanosleep

-p PRIO  --prio=PRIO        priority of highest prio thread

-q        --quiet           print only a summary on exit

-r        --relative        use relative timer instead of
                             absolute

-t NUM   --threads=NUM      number of threads: default=1

-v        --verbose         output values on stdout for
                             statistics

               format: n:c:v n=tasknum c=count v=value in us
```

Running cyclictest (10 timer threads with priority 80 and 5000 loops) produces the following output on a dual-core, Xenomai-patched Linux machine:

```
[root@ixa00321605 bin]# ./cyclictest -t 10 -p 80 -n -i 5000 -l
5000

0.07 0.03 0.01 1/74 6256

T: 0 ( 6246) P:80 I:
1   5000 C:  5000 Min:      0 Act:      0 Avg:    0 Max:

T: 1 ( 6247) P:79 I:
4   5500 C:  5000 Min:      0 Act:      0 Avg:    0 Max:

T: 2 ( 6248) P:78 I:
5   6000 C:  5000 Min:      0 Act:      0 Avg:    0 Max:

T: 3 ( 6249) P:77 I:
7   6500 C:  5000 Min:      0 Act:      0 Avg:    0 Max:

T: 4 ( 6250) P:76 I:
8   7000 C:  5000 Min:      0 Act:      0 Avg:    0 Max:

T: 5 ( 6251) P:75 I:
10  7500 C:  5000 Min:      0 Act:      0 Avg:    0 Max:
```

```
T: 6 ( 6252) P:74 I:
11  8000 C:  5000 Min:      0 Act:      0 Avg:   0 Max:

T: 7 ( 6253) P:73 I:
12  8500 C:  5000 Min:      0 Act:      0 Avg:   0 Max:

T: 8 ( 6254) P:72 I:
14  9000 C:  5000 Min:      0 Act:      0 Avg:   1 Max:

T: 9 ( 6255) P:71 I:
15  9500 C:  5000 Min:      0 Act:      0 Avg:   0 Max:
```

The above output shows that Minimum and Average timer latencies for 10 threads running in parallel is 0 µs, Maximum timer latency is 15 µs for thread 9.

## clocktest

The test '**clocktest'** provides information on time offset using gettimeofday() as a reference, drift value and warp information.

```
usage: clocktest [options]

[-C <clock_id>]              # tested clock, default=0
                              (CLOCK_REAL TIME)
[-T <test_duration_seconds>] # default=0, so ^C to end
```

Running clocktest produces the following output on a dual-core, Xenomai-patched Linux machine:

```
[root@ixa00321605 bin]# ./clocktest

== Tested clock: 0 (CLOCK_REAL TIME)

CPU ToD offset [us] ToD drift [us/s]    warps   max delta [us]

--- --------------- ---------------  ---------  --------------

0        37598625.8            -0.359         0              0.0
1        37598625.5            -0.361         0              0.0
```

## latency

The test '**latency'** provides statistics for scheduling latency over a certain sampling period for a periodic real-time thread.

```
usage: latency [options]

   [-h]                             # print histograms of min
                                    avg, max latencies

   [-s]                             # print statistics of min,
                                    avg, max latencies
```

```
[-H <histogram-size>]           # default = 200, increase if
                                your last bucket is full

[-B <bucket-size>]              # default = 1000ns, decrease
                                for more resolution

[-p <period_us>]                # sampling period

[-l <data-lines per header>]  # default=21, 0 to suppress
                                headers

[-T <test_duration_seconds>]  # default=0, so ^C to end

[-q]                            # supresses RTD, RTH lines if
                                -T is used

[-D <testing_device_no>]       # number of testing device,
                                default=0

[-t <test_mode>]                # 0=user task (default,
                                1=kerneltask, 2=timer IRQ

[-f]                            # freeze trace for each new
                                max latency

[-c <cpu>]                      # pin measuring task down to
                                given CPU

[-P <priority>]                 # task priority (test mode 0
                                and 1 only)
```

Running latency produces the following output on a dual-core, Xenomai-patched Linux machine:

```
[root@ixa00321605 bin]# ./latency -t0 -p200 -f -T50

== Sampling period: 200 us

== Test mode: periodic user-mode task

== All results in microseconds warming up...

RTT| 00:00:01 (periodic user-mode task, 200 us period, priority
99)

RTH|---lat min|-lat avg|-lat max|-overrun|-lat best|-lat worst

RTD|    -0.220|    -0.115|    0.273|        0|    -0.220|     0.273

RTD|    -0.227|    -0.185|    0.196|        0|    -0.227|     0.273

RTD|    -0.224|    -0.110|    0.281|        0|    -0.227|     0.281

RTD|    -0.224|    -0.183|    0.141|        0|    -0.227|     0.281

RTD|    -0.222|    -0.111|    0.293|        0|    -0.227|     0.293
```

```
RTD|    -0.224|    -0.185|    0.159|        0|    -0.227|      0.293
RTD|    -0.223|    -0.113|    0.253|        0|    -0.227|      0.293
RTD|    -0.229|    -0.181|    0.363|        0|    -0.229|      0.363
RTD|    -0.224|    -0.117|    0.218|        0|    -0.229|      0.363
RTD|    -0.224|    -0.183|    0.203|        0|    -0.229|      0.363
RTD|    -0.220|    -0.113|    0.248|        0|    -0.229|      0.363
RTD|    -0.226|    -0.183|    0.238|        0|    -0.229|      0.363
RTD|    -0.223|    -0.118|    0.291|        0|    -0.229|      0.363
RTD|    -0.224|    -0.182|    0.202|        0|    -0.229|      0.363
RTD|    -0.221|    -0.118|    0.239|        0|    -0.229|      0.363
RTD|    -0.223|    -0.182|    0.393|        0|    -0.229|      0.393
RTD|    -0.222|    -0.115|    0.281|        0|    -0.229|      0.393
RTD|    -0.219|    -0.178|    0.183|        0|    -0.229|      0.393
RTD|    -0.221|    -0.116|    0.236|        0|    -0.229|      0.393
RTD|    -0.223|    -0.183|    0.387|        0|    -0.229|      0.393
RTD|    -0.221|    -0.115|    0.272|        0|    -0.229|      0.393
RTT| 00:00:22 (periodic user-mode task, 200 us period, priority
99)
RTH|---lat min|-lat avg|-lat max|-overrun|-lat best|-lat worst
RTD|    -0.221|    -0.180|    0.134|        0|    -0.229|      0.393
RTD|    -0.223|    -0.121|    0.237|        0|    -0.229|      0.393
RTD|    -0.227|    -0.179|    0.654|        0|    -0.229|      0.654
RTD|    -0.223|    -0.120|    0.439|        0|    -0.229|      0.654
RTD|    -0.112|    -0.075|    0.482|        0|    -0.229|      0.654
RTD|    -0.113|    -0.079|    0.387|        0|    -0.229|      0.654
RTD|    -0.107|    -0.074|    0.792|        0|    -0.229|      0.792
RTD|    -0.107|    -0.073|    0.359|        0|    -0.229|      0.792
RTD|    -0.106|    -0.073|    1.067|        0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.354|        0|    -0.229|      1.067
```

```
RTD|    -0.106|    -0.074|    0.661|         0|    -0.229|      1.067
RTD|    -0.107|    -0.073|    0.374|         0|    -0.229|      1.067
RTD|    -0.104|    -0.073|    0.399|         0|    -0.229|      1.067
RTD|    -0.106|    -0.074|    0.351|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.364|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.392|         0|    -0.229|      1.067
RTD|    -0.106|    -0.074|    0.384|         0|    -0.229|      1.067
RTD|    -0.107|    -0.073|    0.402|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.370|         0|    -0.229|      1.067
RTD|    -0.107|    -0.075|    0.395|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    1.019|         0|    -0.229|      1.067
RTT| 00:00:43 (periodic user-mode task, 200 us period, priority
99)
RTH|---lat min|-lat avg|-lat max|-overrun|-lat best|-lat worst
RTD|    -0.220|    -0.075|    0.373|         0|    -0.229|      1.067
RTD|    -0.224|    -0.181|    0.365|         0|    -0.229|      1.067
RTD|    -0.106|    -0.074|    0.440|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.441|         0|    -0.229|      1.067
RTD|    -0.105|    -0.071|    0.373|         0|    -0.229|      1.067
RTD|    -0.107|    -0.074|    0.502|         0|    -0.229|      1.067
RTD|    -0.106|    -0.074|    0.368|         0|    -0.229|      1.067
---|----------|----------|---------|---------|--------------------
RTS|    -0.229|     -0.113|    1.067|         0|00:00:50/00:00:50
```

As can be seen from the results above that running the latency test for 50 seconds, produces no overruns. Latency test should be run for longer periods of time and the machine flooded with processes to get the actual figures for the real-time task in consideration.

***Note:*** Latency test provides instrumentation for user-space task, kernel-space task and timer interrupt.

For enabling the timer interrupt latency instrumentation, the following configuration setting should be enabled in the kernel.

```
Real-time sub-system ---> Xenomai ---> Nucleus ---> Drivers --->
Testing drivers ---> Timer benchmark driver ---> enable
```

## xeno-test

The test '**xeno-test'** is a very useful test to determine the system's real-time state. It provides detailed information of a number of system parameters including faults found in all the CPUs present. It runs several test cases to monitor the real-time performance of the machine; it also runs the latency tests in all the three modes.

```
usage: xeno-test [options]

  runs latency test in all 3 test-modes

  -w <number>  spawn N workloads (dd if=/dev/zero of=/dev/null)
               default=1

  -d <device>  used as alternate src in workload (dd if=/dev/zero
               ..) The device must be mounted, and unfortunately)
               cannot be an NFS mount a real device (ex /dev/hda)
               will generate interrupts

  -W <script>  script is an alternate workload. If you need to
               pass args to your program, use quotes. The program
               must clean up its children when it gets a SIGTERM

  -P <cmd>     cmd is run before and after rt-tests
               (forex: 'ntpdate -b <host>' or 'ntpq -p')

  -L           writes logs to /tmp/test-2.6.26.1_xenomai-2.4.6-
               <timestamp>

  -N <name>    like -L, but writes to name-<timestamp> (in PWD)

               name can be full or relative pathname

  -v           verbose

  -M <email>   sends output to given addr

  -m           sends output to xenomai-data@gna.org

  -U <url>     uploads output to given URL

  -D <datefmt> alternate options to date, for timestamp (dflt:


# following options are passed thru to latency

  -s           print statistics of sampled data (default on)

  -h           print histogram of sampled data (default on,
               implies -s)
```

```
-q          quiet, dont print 1 sec sampled data (default on,
            off if !-T)

-T <sec test>            (default: 120 sec)

-l <data/header lines>   (default 21)

-H <bucketcount>         (default 100)

-B <bucketsize ns>       (default 1000 ns)

-p <sample_period_us>    (default 100 us)
```

# Conclusion

Xenomai is a generic solution to achieve hard real-time capabilities in Linux for any dimension of the project which uses Linux. It provides different APIs for creating RT tasks, timers, synchronization objects, etc., and simulates various APIs called skins for real-time application, including POSIX interface, RTAI, VxWorks, etc., which allows easier porting of existing RT applications to Linux.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. http://intel.com/embedded/edc.

## References

| | |
|---|---|
| A Quantitative Comparison of Real Time Linux Solutions | http://rtg.informatik.tu-chemnitz.de/docs/da-sa-txt/sa-franm.pdf |
| ADEOS | http://home.gna.org/adeos/ |
| Captain's Universe | http://www.captain.at/xenomai.php |
| Life with ADEOS | http://www.xenomai.org/documentation/branches/v2.0.x/pdf/Life-with-Adeos.pdf |
| Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application | http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04448543 |
| Xenomai API | http://www.xenomai.org/documentation/branches/v2.4.x/html/api/index.html |
| Xenomai documentation | http://www.xenomai.org/documentation/branches/v2.4.x/pdf/ |
| Xenomai FAQs | http://www.xenomai.org/index.php/FAQs |
| Xenomai official site | http://www.xenomai.org/index.php/Main_Page |
| Xenomai: A Tour of the Native API | http://www.xenomai.org/documentation/branches/v2.0.x/pdf/Native-API-Tour.pdf |
| Time/Utility Functions | http://www.real-time.org/timeutilityfunctions.htm |

## Authors

**Amarpreet Singh Ugal** is a BIOS/Firmware Engineer with ECG at Intel Corporation.

## Acronyms

| | |
|---|---|
| ACPI | Advanced Configuration and Power Interface |
| ADEOS | Adaptive Domain Environment for Operating Systems |
| BIOS | Basic Input/output System |
| CPU | Central processing unit |
| HAL | Hardware Abstraction Layer |
| POSIX | Portable Operating System Interface |
| RT | Real Time |
| RTAI | Real Time Application Interface |
| SMI | System Management Interrupt |
| SMP | Symmetric multiprocessing |