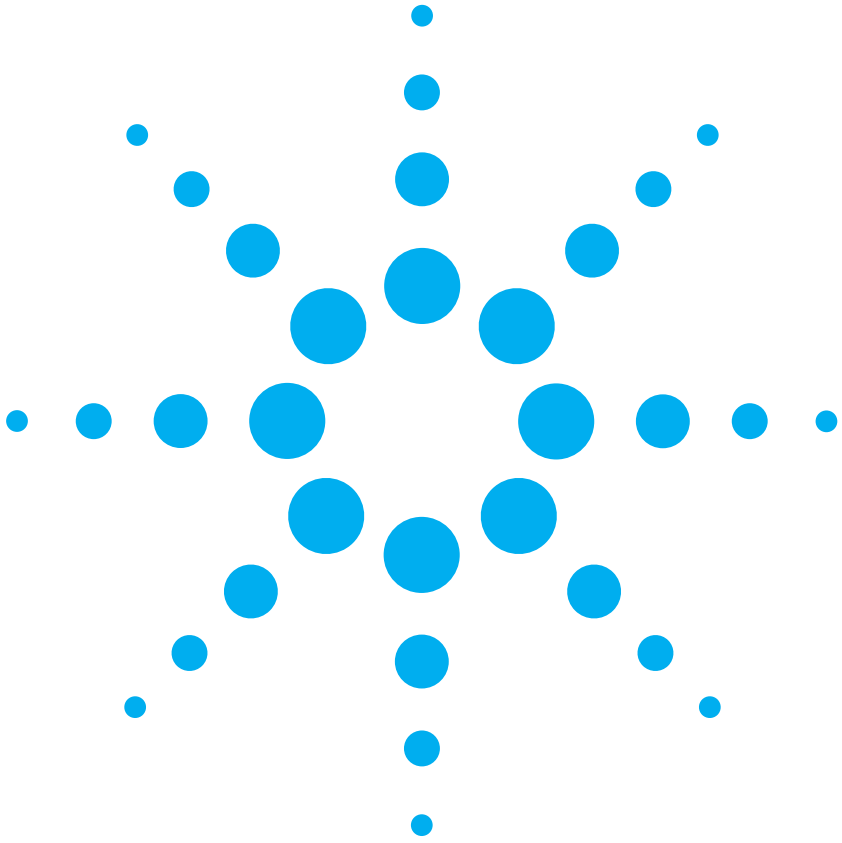
Tips for Optimizing Test System Performance in Linux Soft Real-Time Applications

Application Note 1465-31



Flexibility has always been a cornerstone of the Linux operating system. The Linux kernel can be tuned and modified more than most other operating systems to make it fit your application's requirements. With some fairly simple techniques, you can optimize your system's suitability for soft real-time applications, which often is desirable for simulation or measurement sequencing with exact timing. *Tips for Optimizing Test System Performance in Linux Soft Real-Time Applications* is part of a series of application notes designed to explain key aspects of using Linux in test and measurement applications.

Table of Contents

Agilent Technologies

Introduction

Effective functional testing often requires you to simulate the natural operating environment of your device under test (DUT), including realistic stimulus signals. Simulating the operating environment can be a challenge, especially if your DUT is a control module designed for a real-time application. To properly test the DUT, your test system needs to have real-time capabilities as well, so it can apply the right stimulus, based on the DUT's outputs, in a timely manner.

Real-time systems

What do we mean by "real-time"? A real-time system has the ability to react to an external event (such as a trigger signal) within a known amount of time. Unlike regular operating systems, real-time systems guarantee a certain reaction time, no matter what state the system is in when the external event occurs. A real-time system doesn't necessarily react quickly – but it is definitely reliable.

Let's look at some examples where you would need at least a degree of real-time capability in your test system. Let's suppose you need to apply a sequence of stimulus signals with exact timing. Whenever you control the timing in your test software using operating system calls such as sleep(), you rely on the operating system to awaken your process at the right time. A real-time system typically gives you more precise control over sleep times.

Simulating sensors in a dynamic environment is another example where you need real-time capability. The stimulus signal often needs to be generated on the fly, as a function of DUT state and other variable parameters. For smooth update of the stimulus signals, your software algorithm needs to be run at a certain minimum rate. A real-time system is able to guarantee that minimum rate.

Most operating systems, including off-the-shelf Linux and Windows® systems, are not real-time. They optimize the *average* processor time available to the user or to a process, but sometimes the system will simply become unavailable for a moment. This could happen, for example, when the operating system is performing a housekeeping task.

Soft real-time versus hard real-time systems

Depending on the nature of your application, you may not be able to live without hard real-time capabilities. For example, if you are controlling a chemical process using software-based closed-loop control, you will likely need a guaranteed update rate at all times. Another example would be controlling machinery or other moving parts. In these examples, real-time performance is critical and required under all circumstances.

Some applications, however, work well with what is known as soft real-time capability. In soft real-time, the system is optimized for reliable timing, but on a best-effort basis with degradation in performance on rare occasions. Test and measurement applications are often good candidates for soft real-time operation because of their non-critical nature (at least, compared to the process control example described above) and the possibility of repeating the test.

This application note offers several tips for optimizing the soft real-time performance of regular (off-the-shelf) Linux systems. Why would you choose this option instead of a hard real-time approach? Hard real-time systems (including Linux variants) are often proprietary and come with a substantial increase in system complexity. Consequently, as long as soft real-time is sufficient, you may want to avoid the cost and risk of using a specialized hard real-time variant.

Basic Tips for Optimizing Response Times

First, let's cover some tactics that might appear to be trivial but are important to keep in mind, nonetheless.

Avoid the challenge if you can. One potential solution to the real-time challenge is to just not play the game. For example, in manufacturing test, some people change the rules by loading special test software into their DUTs. This approach can substantially simplify testing. For example, the various pins, channels or subfunctions of the DUT oftentimes can be tested sequentially, without the need for simultaneous stimulus and real-time synchronization between the various system resources. The DUT test software communicates with the test application and supports testing by controlling individual resources or reporting status information.

Put the burden of real-time control on your instruments. Sometimes, you can avoid the challenge on the software side by putting the burden of real-time control on a suitable instrument (or set of instruments). For example, the Agilent 34980A multifunction switch measure unit can be programmed to perform a sequence of actions with exact timing and raise an alarm if a given measurement channel exceeds preset limits. Another example: the Agilent E5818A LXI class-B trigger unit uses IEEE-1588 (precision time protocol or PTP) over Ethernet for precise time-stamping of events and time-based triggering of conventional (non-PTP) instruments.

Using the real-time capabilities built into your instruments is a clever work-around. Obviously, it is only feasible if your instruments can be configured to handle the real-time requirements of your application. If your requirements call for intelligent and flexible stimulus and response, an implementation in software could be the only choice.

Use a fast PC with plenty of memory. A fast PC with ample processing power is a great starting point for soft real-time applications. Among the things that hurt you most in these applications is the housekeeping regularly done by the operating system, as well as the overhead involved in the start-up of additional programs or services. You probably cannot avoid housekeeping activities altogether, but a fast PC will complete such tasks more quickly and, as a result, there will be less of an interruption to your measurement task.

Shut down unused services. Make sure your PC spends its processing power on the tasks that are really important. A default Linux installation typically will have a number of services enabled you don't need. These services often come with a daemon that implements the service, and that daemon process might become active to do its work or just to perform some housekeeping. With regards to real-time, such processes are a source of uncertainty and increased response times. It is good practice to disable unused (network) services anyway

because they could be a security hole. The chkconfig(8) command allows you to list and enable/disable the services on your system. Some examples of services you might want to disable:

- The cron service allows you to schedule programs for later execution. It is usually used for routine tasks such as backup, removal of old temporary files, etc., that impact response times when running.

- Other services that might not be required include network file system (nfs, nfsserver), ntp (network time protocol), cups (printing server), as well as any firewall services (especially if you are using an isolated network with a standalone firewall).

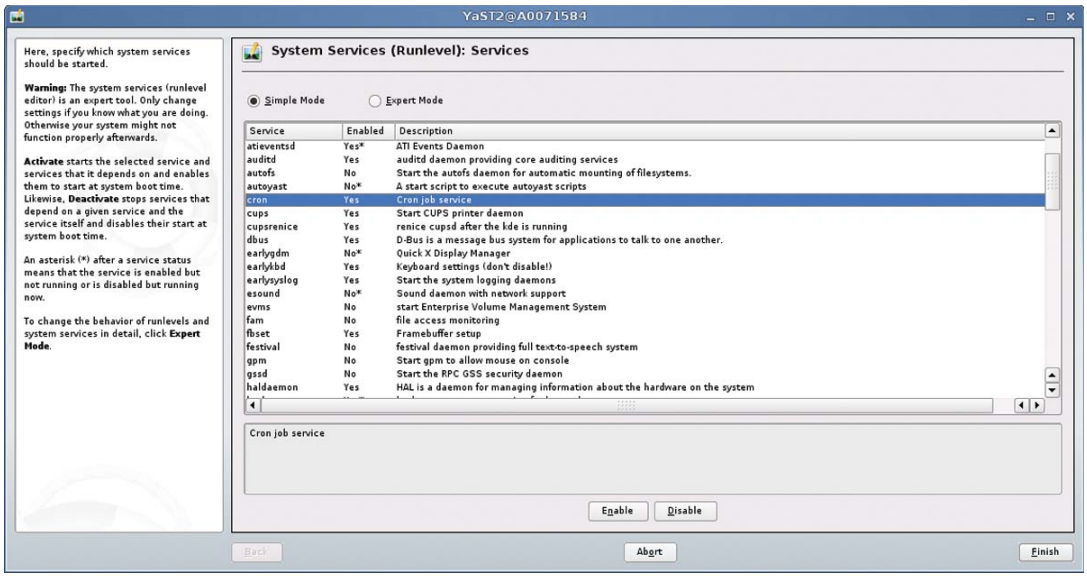
- Many basic network services (including telnet and ftp) do not register as an individual service but are started by inetd, the Internet daemon. These services are enabled or disabled by editing the file /etc/inetd.conf (when using the original inetd) or the files in the /etc/xinetd.x directory (when using the newer xinetd, the extended Internet daemon).

- As an alternative to using chkconfig(8) or editing the configuration files directly, most Linux distributions feature interactive system administration tools such as YaST, available with openSUSE (see Figure 1).

Isolate the real-time part of your application. Oftentimes, only a small part of the application or DUT requires real-time capabilities. For example, an ECU that requires some of its sensor inputs to be updated in real-time usually has a load of other inputs and outputs that are static.

It is often useful to separate the real-time and non-real-time parts of the application in separate processes. Doing so allows you to optimize each part for its special requirements and make different design decisions. For example, you could implement the real-time part using C and the rest using a higher-level language. You could also use different scheduling policies (see *Scheduling Policies and Priorities*, page 5).

Figure 1. Most distributions come with administration aids such as YaST (openSUSE)



Time Slice Values

Multitasking presents a tradeoff between responsiveness and overhead. The system's time slice value indicates how long a process is allowed to run before it needs to relinquish the CPU to the next process. If the time slices are short, the average waiting time is smaller and the system will be more responsive. At the same time, every task switch means overhead, and the efficiency of the system overall will decrease somewhat.

Larger time slices are good for number-crunching or server applications. Smaller time slices are better for desktop applications and interactive use. For real-time applications, you might want to choose even smaller time slices.

Figure 2 shows an excerpt of the file kernel/sched.c under the kernel sources tree. This is where the time slice values are defined. As you can see, in this example the default time slice is set to 100 ms. For real-time applications, you might want to try setting both the minimum and default values to 1 ms.

Again, most distributions offer interactive system configuration tools that allow you to tune various system parameters without modifying the kernel source files directly. Figure 3 shows the corresponding window in YaST (openSUSE). The time slice values are defined under System/Kernel in the system configuration tool.

Reducing the time slice values could help your system be more responsive overall. Note, however, that there is only an impact on your real-time application if it is running as a "normal" process (with static priority zero). See the following section for details.

Scheduling Policies and Priorities

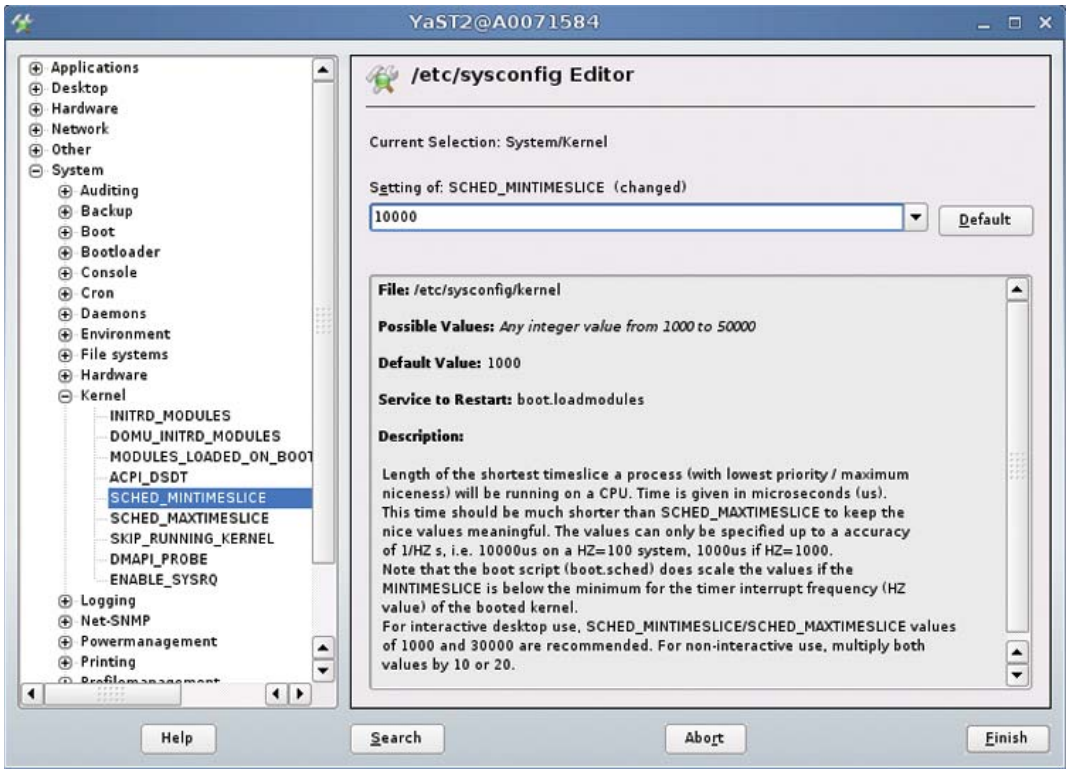
The Linux scheduler allocates processor time to the various processes based on their scheduling policy and priority. Most processes use the SCHED_OTHER policy which always goes along with a static priority value of zero (meaning, lowest priority). Within this group of (regular) processes, the scheduler uses additional dynamic priority values for fairness. For example, a process that is waiting for CPU time gets an increase in (dynamic) priority over time.

Figure 2. Time slice values, as defined in sched.c

```
/*
 * These are the 'tuning knobs' of the scheduler:
 *
 * Minimum timeslice is 5 msecs (or 1 jiffy, whichever is larger),
 * default timeslice is 100 msecs, maximum timeslice is 800 msecs.
 * Timeslices get refilled after they expire.
 */
#define MIN_TIMESLICE             max(5 * HZ / 1000, 1)
#define DEF_TIMESLICE             (100 * HZ / 1000)
#define ON_RUNQUEUE_WEIGHT         30
#define CHILD_PENALTY              95
#define PARENT_PENALTY            100
#define EXIT_WEIGHT                 3
#define PRIO_BONUS_RATIO           25
#define MAX_BONUS                 (MAX_USER_PRIO * PRIO_BONUS_RATIO / 100)
#define INTERACTIVE_DELTA           2
#define MAX_SLEEP_AVG             (DEF_TIMESLICE * MAX_BONUS)
#define STARVATION_LIMIT          (MAX_SLEEP_AVG)

#define NS_MAX_SLEEP_AVG          (JIFFIES_TO_NS(MAX_SLEEP_AVG))
```

Figure 3. Configuration of time slices using YaST (openSUSE)



To a certain degree, you can set up a process for preferred treatment by manually giving it a high (dynamic) priority using the nice(1) command (for new programs) or renice(8) (for existing processes). This will result in more processor time being available to the process on average, but is usually not sufficient for real-time applications. No matter what the dynamic priority is, the process is still preempted when processes with non-zero static priority are ready to run.

For time-sensitive processes, Linux offers the SCHED_FIFO and SCHED_RR policies. Processes that use these policies just use static priorities, usually ranging from 1 to 99 (highest priority). There's no fairness here: a lower-priority process is always preempted by a higher-priority one. On the positive side, you can usually get a decent level of soft real-time performance by using a high priority value.

SCHED_RR is very similar to SCHED_FIFO but adds time slicing between all processes sharing the same priority. Using the FIFO algorithm, a process goes to the end of the list only if it relinquishes the processor voluntarily or if it is blocked by an I/O request (or, of course, if a higher-priority process is ready to run). Using the RR (round robin) algorithm, the process loses the processor after it has been running for a full time quantum. So, with this policy, there is some fairness, at least within the same level of priority.

The code snippet shown in Figure 4 demonstrates how a process can change its own scheduling policy and priority. Obviously, this requires root privilege.

The graph in Figure 5 shows what effect this change can have on your application. The four traces show the execution time of a math function over time. The yellow and turquoise traces were taken while the system was running idle (no CPU-intensive processes other than the benchmark process). The traces show the execution time using the SCHED_OTHER and SCHED_FIFO policies, respectively. As you can see, the performance is very similar and reasonably constant.

The picture changes dramatically if the system is running under load. In this example, additional load is generated by simultaneously starting the OpenOffice Writer application and the Firefox Web browser. The blue trace shows that the execution time using the default scheduling policy fluctuates widely, with peaks where the time is more than double the idle level. Using the FIFO policy, however, the performance stays constant and reliable (magenta trace).

Figure 4. Code example for setting the scheduling policy and priority

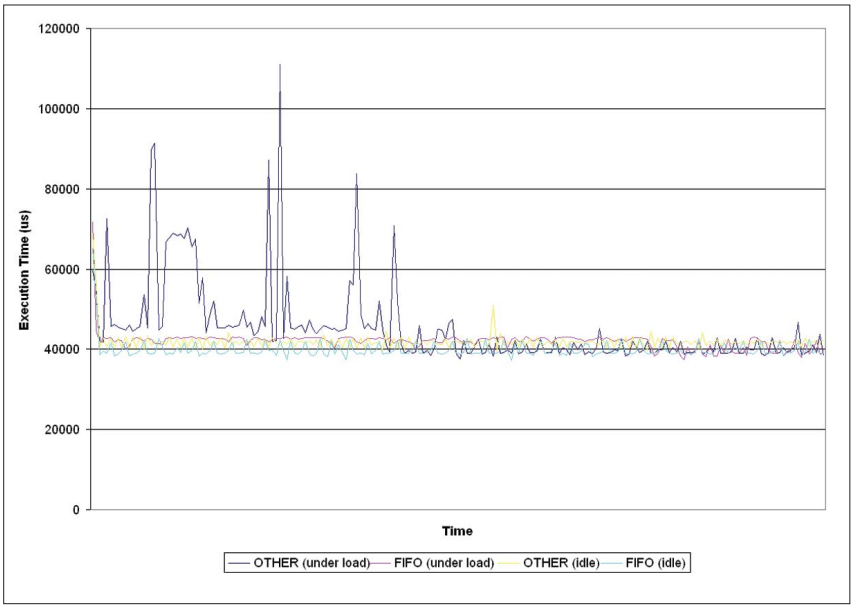
```
#include <stdio.h>
#include <sys/mman.h>
#include <sched.h>
#include <errno.h>
#include <sys/resource.h>
#include <time.h>

int MinPriority;
int MaxPriority;
struct sched_param prio;
int en;

// Get min and max priority values for SCHED_FIFO policy
MinPriority=sched_get_priority_min(SCHED_FIFO);
printf("Min priority for FIFO policy is %d\n",MinPriority);
MaxPriority=sched_get_priority_max(SCHED_FIFO);
printf("Max priority for FIFO policy is %d\n",MaxPriority);

// Set scheduling policy and priority value (maximum)
prio.sched_priority = MaxPriority;
if(sched_setscheduler(0,SCHED_FIFO,&prio)<0) {
         // There was a problem
         en=errno;
         printf("Error returned by sched_setscheduler: %s\n",strerror(en));
         exit(0);
}
```

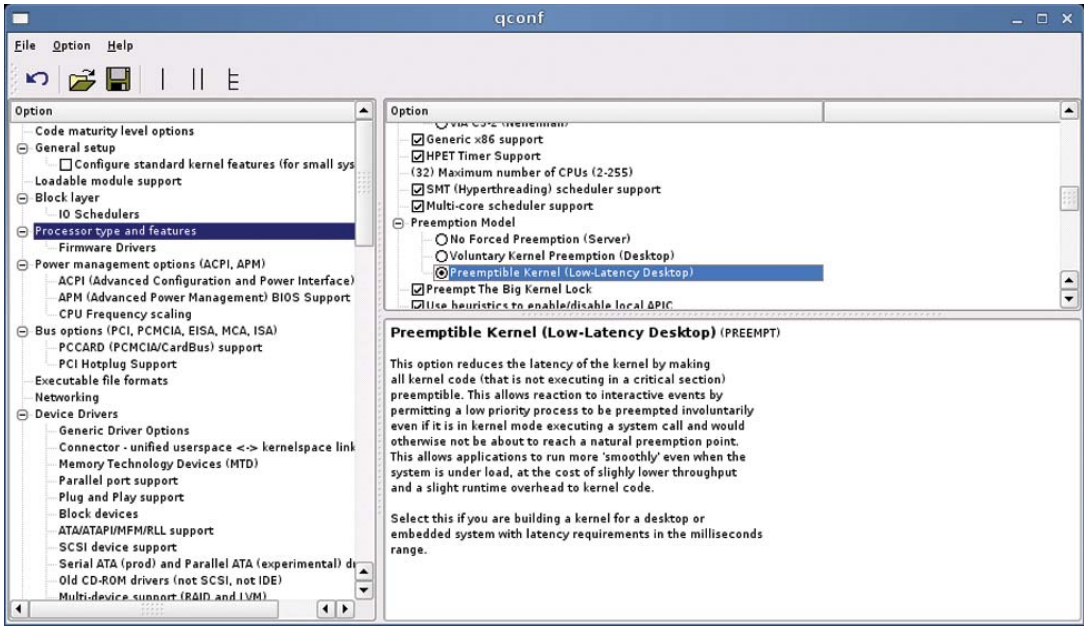
Figure 5. Impact of scheduling policy on execution time (example)



Using Preemptible Kernel

As shown in figure 5, Linux uses preemptive multitasking. This means that the operating system can take control away from a process if its share of time runs out or if a higher-priority process is ready to run. Historically, the Linux kernel has been an exception to that rule: code running in kernel space was excluded from preemption. This is an issue in real-time systems because regular processes can avoid preemption by making an operating system call (and some calls, such as fork(2), can take a fair amount of time to execute).

Figure 6. Entry for kernel preemption in XConfig tool



Luckily, modern Linux kernels can be configured to support preemption. Using this option, all kernel code is considered to be preemptible unless explicitly marked as a critical section. Figure 6 shows the corresponding entry in the XConfig[1] kernel configuration tool. Kernel preemption is found under the "Processor type and features" subtree.

1 To use XConfig, cd to /usr/src/linux and run the command **make xconfig.** Note that XConfig is based on the Qt library, so make sure all packages related to Qt are installed on your system. XConfig creates a kernel configuration (.config) file that is subsequently used by "make" when it is building the kernel. The complete process of building and installing a new kernel depends on your distribution and boot manager and is beyond the scope of this application note. Please refer to your distribution's documentation.

Figure 7. Locking a process in RAM

```
#include <sys/mman.h>
// Lock all current and future memory areas associated with the current process

if(mlockall(MCL_CURRENT|MCL_FUTURE)<0) {
// There was a problem
        en=errno;
        printf("Error returned by mlockall: %s\n",strerror(en));
        exit(0);
}

// Real-time stuff goes here.

// Unlock memory (optional, done automatically when process ends)
munlockall();
```

Virtual Memory and Paging

Like all modern operating systems, Linux offers virtual memory and paging, and can thereby offer a much larger address space than is physically available in RAM. When the system runs low on physical memory, it swaps currently unneeded pages out to its swap space on the hard disk. When the pages are accessed, an exception occurs and the pages are swapped back in.

Paging is one of the big issues with real-time applications because it introduces additional latency and uncertainty. The actual delays due to paging depend on memory usage and are usually hard to foresee.

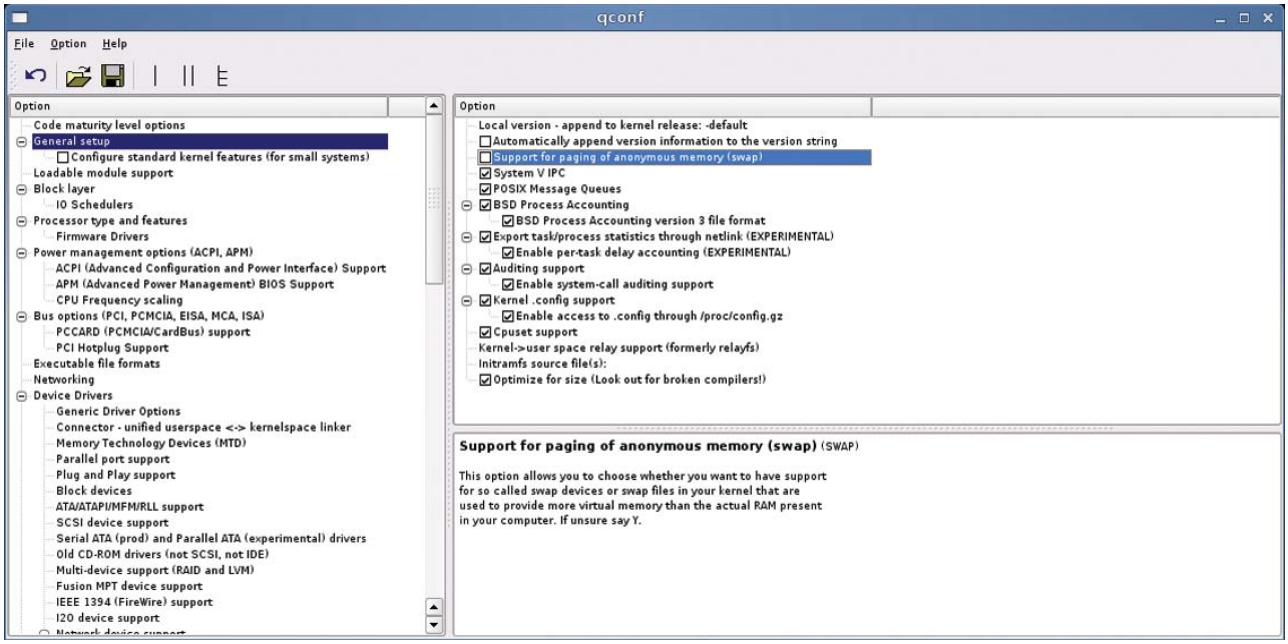
To limit the effects of paging, at the very least, you should lock your real-time application's process and associated data in RAM. The code snippet in Figure 7 shows how this can be done.

Although it is brute force, the most effective method of avoiding latency issues caused by paging is to exclude that option from the kernel altogether. You might want to choose this option if you have a good grasp of your application's current and future memory requirements. Embedded applications, for example, usually have well-known memory requirements and don't require paging. Figure 8 shows the corresponding option in the XConfig kernel configuration tool.

Summary

If your application requires soft real-time operation and you choose to implement the corresponding algorithms in software, you can benefit from the flexibility offered by Linux. Using real-time scheduling (SCHED_FIFO or SCHED_RR policy) along with a high static priority often makes a dramatic difference. By combining the latter three strategies (using real-time scheduling and preemptible kernel and avoiding paging) you can typically get a very decent level of soft real-time performance from off-the-shelf Linux distributions.

Figure 8. Disabling paging via XConfig



Related Agilent literature

The 1465 series of application notes provides a wealth of information about the creation of test systems, the successful use of LAN, WLAN and USB in those systems, and the optimization and enhancement of RF/microwave test systems:

- *Test-System Development Guide: A Comprehensive Handbook for Test Engineers*
  (pub no. 5989-5367EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-5367EN.pdf

Test System Development

- *Test System Development Guide:*
  Application Notes 1465-1 through 1465-8 (pub no. 5989-2178EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-2178EN.pdf
- *Using LAN in Test Systems: The Basics*
  AN 1465-9 (pub no. 5989-1412EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1412EN.pdf
- *Using LAN in Test Systems: Network Configuration*
  AN 1465-10 (pub no. 5989-1413EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1413EN.pdf
- *Using LAN in Test Systems: PC Configuration*
  AN 1465-11 (pub no. 5989-1415EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1415EN.pdf
- *Using USB in the Test and Measurement Environment*
  AN 1465-12 (pub no. 5989-1417EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1417EN.pdf
- *Using SCPI and Direct I/O vs. Drivers*
  AN 1465-13 (pub no. 5989-1414EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1414EN.pdf
- *Using LAN in Test Systems: Applications*
  AN 1465-14 (pub no. 5989-1416EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1416EN.pdf
- *Using LAN in Test Systems: Setting Up System I/O*
  AN 1465-15 (pub no. 5989-2409)
  http://cp.literature.agilent.com/litweb/pdf/5989-2409EN.pdf
- *Next-Generation Test Systems: Advancing the Vision with LXI*
  AN 1465-16 (pub no. 5989-2802)
  http://cp.literature.agilent.com/litweb/pdf/5989-2802EN.pdf

RF and Microwave Test Systems

- *Optimizing the Elements of an RF/Microwave Test System*
  AN 1465-17 (pub no. 5989-3321)
  http://cp.literature.agilent.com/litweb/pdf/5989-3321EN.pdf
- *6 Hints for Enhancing Measurement Integrity in RF/Microwave Test Systems*
  AN 1465-18 (pub no. 5989-3322)
  http://cp.literature.agilent.com/litweb/pdf/5989-3322EN.pdf
- *Calibrating Signal Paths in RF/Microwave Test Systems*
  AN 1465-19 (pub no. 5989-3323)
  http://cp.literature.agilent.com/litweb/pdf/5989-3323EN.pdf

LAN eXtensions for Instrumentation(LXI)

- *LXI: Going Beyond GPIB, PXI and VXI*
  AN 1465-20 (pub no. 5989-4371)
  http://cp.literature.agilent.com/litweb/pdf/5989-4371EN.pdf
- *10 Good Reasons to Switch to LXI*
  AN 1465-21 (pub no. 5989-4372)
  http://cp.literature.agilent.com/litweb/pdf/5989-4372EN.pdf
- *Transitioning from GPIB to LXI*
  AN 1465-22 (pub no. 5989-4373)
  http://cp.literature.agilent.com/litweb/pdf/5989-4373EN.pdf
- *Creating Hybrid Systems with PXI, VXI and LXI*
  AN 1465-23 (pub no. 5989-4374)
  http://cp.literature.agilent.com/litweb/pdf/5989-4374EN.pdf
- *Using Synthetic Instruments in Your Test System*
  AN 1465-24 (pub no. 5989-4375)
  http://cp.literature.agilent.com/litweb/pdf/5989-4375EN.pdf
- *Migrating System Software from GPIB to LAN/LXI*
  AN 1465-25 (pub no. 5989-4376)
  http://cp.literature.agilent.com/litweb/pdf/5989-4376EN.pdf
- *Modifying a GPIB System to Include LAN/LXI*
  AN 1465-26 (pub no. 5989-6824)
  http://cp.literature.agilent.com/litweb/pdf/5989-6824EN.pdf

Using Linux in Your Test Systems

Example code is available for download at **http://www.agilent.com/find/linux**

- *Using Linux in Your Test Systems: Linux Basics*
  AN 1465-27 (pub no. 5989-6715)
  http://cp.literature.agilent.com/litweb/pdf/5989-6715EN.pdf
- *Using Linux to Control LXI Instruments Through VXI-11*
  AN 1465-28 (pub no. 5989-6716)
  http://cp.literature.agilent.com/litweb/pdf/5989-6716EN.pdf
- *Using Linux to Control LXI Instruments Through TCP*
  AN 1465-29 (pub no. 5989-6717)
  http://cp.literature.agilent.com/litweb/pdf/5989-6717EN.pdf
- *Using Linux to Control USB Instruments*
  AN 1465-30 (pub no. 5989-6718)
  http://cp.literature.agilent.com/litweb/pdf/5989-6718EN.pdf

www.agilent.com/find/open

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent Direct

www.agilent.com/find/agilentdirect
Quickly choose and use your test equipment solutions with confidence.

Agilent Open

www.agilent.com/find/open
Agilent Open simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent offers open connectivity for a broad range of system-ready instruments, open industry software, PC-standard I/O and global support, which are combined to more easily integrate test system development.

LXI

www.lxistandard.org
LXI is the LAN-based successor to GPIB, providing faster, more efficient connectivity. Agilent is a founding member of the LXI consortium.

Remove all doubt

Our repair and calibration services will get your equipment back to you, performing like new, when promised. You will get full value out of your Agilent equipment throughout its lifetime. Your equipment will be serviced by Agilent-trained technicians using the latest factory calibration procedures, automated repair diagnostics and genuine parts. You will always have the utmost confidence in your measurements.

Agilent offers a wide range of additional expert test and measurement services for your equipment, including initial start-up assistance onsite education and training, as well as design, system integration, and project management.

For more information on repair and calibration services, go to:

www.agilent.com/find/removealldoubt

www.agilent.com

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas

Canada	(877) 894-4414
Latin America	305 269 7500
United States	(800) 829-4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe & Middle East

Austria	0820 87 44 11
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700* *0.125 € fixed network rates
Germany	01805 24 6333** **0.14 €/minute
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland (French)	41 (21) 8113811(Opt 2)
Switzerland (German)	0800 80 53 53 (Opt 1)
United Kingdom	44 (0) 118 9276201

Other European Countries:
www.agilent.com/find/contactus

Revised: October 24, 2007

Product specifications and descriptions in this document subject to change without notice.

Printed in USA, February 19, 2008
5989-6719EN

Agilent Technologies