FAST SCALAR DATA BUFFERING INTERFACE IN LINUX 2.6 KERNEL

A. Homs#, ESRF, Grenoble, France.

Abstract

Key instrumentation devices like counter/timers, analog-to-digital converters and encoders provide scalar data input. Many of them allow fast acquisitions, but do not provide hardware triggering or buffering mechanisms. A Linux 2.4 kernel driver called *Hook* was developed at the ESRF as a generic software-triggered buffering interface. This work presents the portage of the ESRF *Hook* interface to the Linux 2.6 kernel. The interface distinguishes two independent functional groups: trigger event generators and data channels. Devices in the first group create software events, like hardware interrupts generated by timers or external signals. On each event, one or more device channels on the second group are read and stored in kernel buffers. The event generators and data channels to be read are fully configurable before each sequence. Designed for fast acquisitions, the *Hook* implementation is well adapted to multi-CPU systems, where the interrupt latency is notably reduced. On heavily loaded dual-core PCs running standard (non real time) Linux, data can be taken at 1 KHz without losing events. Additional features include full integration into the /sys virtual file-system and hot-plug devices support.

INTRODUCTION

Many scientific experiments carried out at the ESRF beamlines (BL) are based on the *scan* concept, where the evolution in time of one or more magnitudes is measured, typically while scanning the values of one or more parameters. Two modes of scan execution are identified. In "step-by-step mode", scan steps and measurements are performed sequentially (parameter change does not overlap in time with detector integration), starting and stopping the involved hardware at each point. In "continuous mode", both scan and measurement are executed in parallel, increasing the duty cycle of the X-ray beam utilisation, and/or notably reducing the total scan duration. As a drawback, time errors in the synchronisation of the readout of all the involved magnitudes affect the precision of the continuous scan.

Initial support to continuous mode at the ESRF was based on a "soft" synchronisation. In the first versions, a scan loop sequentially read, as fast as possible, motors and counters in a VME crate from a remote control workstation. The next approach read all these scan magnitudes sequentially inside a program running in the VME crate, removing the network from the sources of synchronisation error. A third performance improvement used a combination of hard and soft synchronisations. A hardware event triggered an interrupt request (IRQ), and during its IRQ service routine (ISR) all the scan scalar parameters (implemented as registers in instrumentation boards) were read and stored in a kernel buffer.

This IRQ *Hook* mechanism, originally developed in OS/9, was extended in Linux 2.2 [1] and 2.4 kernels with a flexible configuration interface. A dedicated TACO device server allowed the configuration and reading of the kernel buffer by the BL control program, running on the same PC or on a remote workstation.

In the migration of the VME and PCI instrumentation control at the ESRF to Linux 2.6, the new kernel device abstraction was extensively used [2]. In particular, a robust VME infrastructure was developed for Symmetric-Multi-Processor (SMP) environments, supporting device hot-plug functionality.

HOOK INTERFACE

Device Abstraction

Virtually all the common instrumentation hardware providing scalar parameters can fit in the simple device/channel concept. A hardware unit (VME/PCI board, serial line/GPIB/Ethernet instrument) is considered in this context a device. It groups control registers or commands, and exports one or more scalar channels. In addition, a device can generate events, typically hardware IRQs or operating system (OS) signals. These events are used in the *Hook* structure to trigger the readout of an arbitrary set of channels (belonging to the same and/or other devices), which are recorded in dedicated buffers.

Like its Linux 2.4 predecessor, the *Hook* interface is seen as a "kernel service" that instrumentation drivers register to using their names. This interface has been completely redesigned in Linux 2.6 to explicitly distinguish device, event, and channel structures. The same kernel "device" concept already identified by those drivers by a *major:minor* pair, like a specific PCI board structure associated to a kernel *pci_dev*, can contain an analogue *hook_dev* structure. Each *hook_dev* is the parent of a group of *hook_chan* and/or *hook_event* structures, depending of the board functionality.

Once all *hook_devs*, with their *hook_events* and *hook_chans*, are registered at drivers' start-up, they are ready to be used. The "user" side of the *Hook* "service" configures for an acquisition (*Hook* buffer) the event generator and the data channels to be read at each event. The event setup is performed through the board-specific drivers because of its complex nature. As a result of this setup, the event generator's driver installs the *hook_event* on the specified *Hook* buffer. The data channel setup is performed through the *Hook* driver interface, by using a list of *<drv_name, minor, chan_idx>* triplets that reference the channels to store on the buffer.

#alejandro.homs@esrf.fr

Each *hook_chan* has an associated *hook_chan_functs*; the structure defining the methods that carry out its specific functionality. During buffer channel setup a *prepare* method is called for each selected channel to perform its hardware initialisation. If it succeeds, the board driver module is locked to avoid to be removed from the kernel, and the *hook_chan_functs.lock* method is called to notify the board driver that the channel is in use. On each point the event generator driver calls the *Hook* interface, and the *read* method is called for all the involved *hook_chans* to trigger their readout. In case the channel value is not available immediately (it is called in kernel interrupt context), the *read* method can return a delayed completion request. Two solutions are identified in such a condition: asynchronous response or "read-before-next". In the former, the board driver is responsible of calling an asynchronous buffer write *Hook* function once the value is ready. The latter is requested if the driver does not provide asynchronous events; it can only ensure that the value will ready before the next event. Then, the *read_before_next* method is called to retrieve the previous channel value. Upon acquisition end the *unlock* method is called to release the channel.

A device can have channels of different classes, and hence have different set of methods, which might be the case of a multi-purpose counter/timer, analogue and digital I/O PCI board.

The current *Hook* implementation is Linux 2.6 kernel is limited to VME and PCI boards, which generate hardware IRQs and can be controlled by standard I/O operations. Communication-based devices using serial line, GPIB or Ethernet interfaces are not covered (yet). This limitation is due to both the complexity in accessing these platforms from the kernel interrupt context and the latency normally associated to those instruments. Nevertheless, Linux 2.6 kernels are so flexible and well structured that the support of those interfaces would be, in principle, possible.

Buffer Functionality

As we have seen, a fast acquisition involves an event generator, a set of data channels and a kernel buffer. Multiple acquisitions, each with its own buffer, can be performed in parallel on the same PC; they might share data channels if the *prepare* and *lock* methods' implementations allow it. The *Hook* driver creates 5 buffers by default, but this can be changed at module load time.

Two buffer filling modes are implemented: linear, where the acquisition stops once the buffer is full; and circular, where a ring buffer is implemented and data must be read continuously to not get overwritten. Data overrun is not considered a fatal error, but it is always reported.

In addition to kernel memory allocation and management, each *Hook* buffer is in turn a *hook_dev*. In this sense it exports two *hook_chans*: the point index and elapsed time in microseconds, both measured since the beginning of the acquisition. These two values are very important to determine if an event has been lost, either by the kernel (elapsed time between adjacent points higher than expected) or by the user-space program reading buffers (discontinuity in point index). Each buffer also exports a *hook_event*: a Linux software timer, whose period can be set with a time resolution given by the kernel HZ timer frequency (HZ=1000 in RedHat EL 4: 1 ms resolution).

A simple user-space library allows to configure the kernel interface and to read the buffer contents in an efficient way. As mentioned before, a TACO server on top of the library allows clients like SPEC [3], the central BL control application, to retrieve the data on the buffers. For ring-buffer acquisitions with intervals below 1 ms, the readout must be performed continuously due to the limited length of kernel buffers. For these fast acquisitions, the TACO server provides a bigger, user-space buffer and an auxiliary thread to empty the kernel buffer accordingly. This relaxes the time constrains of the remote SPEC main acquisition loop.

Virtual sysfs Support

The Linux 2.6 */sys* virtual file-system gives a useful user-space visibility of kernel structures. The *class* interface is used in the *Hook* to export information about the registered elements, replicating its tree structure of *hook_drv*, *hook_dev*, *hook_chan* and *hook_event*. In this direction, additional *read_desc* methods in *hook_chan* and *hook_event* are foreseen to provide a human-readable description through */sys* interface.

The *Hook* driver allows code tracing and debugging through messages on the standard kernel *syslog* interface. However, this is far from optimal for very fast acquisitions. In order to provide real time status to user space, the */sys* interface also publishes the instantaneous buffer information. It includes buffer setup (size, filling mode, list of channels) and buffer state (total number of events, number of points to be read and elapsed time since start).

IMPLEMENTATION

Supported Hardware

The standard ESRF BL instrumentation PCI and VME boards used for fast acquisition has been ported to the Linux 2.6 Hook structure:

- ESRF PCI P201 Counter/Timer
- ESRF VME VCT6 Counter/Timer
- Compcontrol CC133 Incremental Encoder
- ADAS ICV150 Analog-to-Digital Converter (ADC)

The first two boards implement both data channels and event generators; the last two only export data channels.

SMP and VME Issues

Linux 2.6 gives an improved support to hot-plug dynamics, that is, devices that can appear and disappear without system control. This is the case of VME devices connected to PCI/VME bus couplers; if the remote crate is disconnected the corresponding VME devices no longer exist on the system. It becomes an important issue in SMP

systems, because one or more CPU/cores could be referring to one device while another is trying to delete it. Such situation is even more critical in the *Hook* environment, which works on interrupt context and the system can immediately hang on invalid operations. Special care has been taken to protect all the data structures against these different situations.

Nevertheless, some crashes or conflicts are unavoidable if performance must be kept as first design priority. One example is found on scans with a PCI IRQ-based event generator that reads channels in VME boards. The conflict arises if the remote VME bus access (channel *read)* is done while another CPU/core is performing an Interrupt-Acknowledge cycle on the same VME bus. This condition is forbidden by the PCI/VME bus coupler, and implementing additional protection mechanisms would sacrifice the performance of fast acquisitions.

PERFORMANCE

Measurement Details

The current ESRF standard instrumentation control computer is based on 4U industrial PCs with Intel Core 2 Duo E6400 @ 2.13 GHz. The OS is RedHat EL 4 [Update 6]; the official Linux kernel 2.6.9-67.ELsmp has not been modified. Measurements have been performed on that system with two PCI counter/timer boards, and one SBS Bit-3 PCI/VME bus coupler with all the supported VME boards mentioned before. The event generator is either a PCI and/or VME timer programmed in free-run (cyclic counting), with end-of-count (EOC) IRQ activated. Another counter of the same board is set to count microseconds, and to latch its value on timer EOC. By reading the running counter value as first data channel and then its EOC-latched value, it is possible to calculate the effective IRQ latency. Moreover, by specifying the running counter value twice consecutively in the channel list, the channel access time is obtained, quantifying the systematic component of the synchronisation error.

Single Acquisitions

For the first tests the VCT6 channel 1 was configured as free-run timer for event generation and channel 2 as a microseconds counter, as explained before. The *Hook* buffer was setup to store 10 channels:

- VCT6 channel 2 running counter (read 4 times)
- VCT6 channel 2 latch counter
- *Hook* buffer 0 point index
- *Hook* buffer 0 time stamp (read twice)
- ICV150 channels 1 & 2

The measurement results for a timer event frequency of 5 KHz are shown in Table 1. Trying to measure at higher event frequencies resulted in data point lost.

A common conclusion for all the measured magnitudes is their small standard deviation (SD), showing that, in most of the time, the Linux SMP kernel is very regular in scheduling. The notable difference in VCT6 and P201 average IRQ latencies comes from the delay added by the PCI/VME bus coupler hardware and software layers.

Table 1: VCT6 IRQ Latency and Read Delays (µs)

Magnitude	Min.	Ave.	Max.	SD
VCT6 IRQ latency	15	24	126	0.8
VCT6 channel read delay	0	2.5	150	0.8
Hook channel read delay	2	3.1	22	0.7

The results for the same measurements using the P201 are shown in Table 2. Again, 5 KHz is the maximum readout frequency for stable acquisitions.

Table 2: P201 IRQ Latency and Read Delays (µs)

Magnitude	Min.	Ave.	Max.	SD
P201 IRQ latency	3	4.2	105	0.6
P201 channel read delay	0	0.9	7	0.25
Hook channel read delay	2	3.1	26	0.7

When analysing the maximum IRQ latency values, it should be noted that these conditions are close to the data sampling reliability limits. As it will be seen in the next section, lower sample frequencies give smaller differences between min./max. values. Indeed, the higher sample frequency, the larger amount of data per unit time must be transferred from kernel to user-space, and then to the TACO client via the Ethernet device, reducing the total idle times of CPUs, chipset and hardware busses.

In this sense, IRQs shared with active devices like network and disk controllers are a proven source of instabilities. The limited number of 4 available PCI IRQs in standard systems makes sharing difficult to avoid. It is better then to choose inactive devices like embedded USB or graphics controllers, unused in most of our applications.

It should be noted that jitter in ISR scheduling is important if the channels to be read are not hard synchronised with the event generator. This is the case of many encoders and ADCs. If, on the contrary, the boards allow data latching on hardware events, like VCT6 and P201 counters, this jitter does not contribute to the measurement error, and it is only important to guarantee that no point is lost by the kernel.

Parallel Acquisitions

It was explained before that the *Hook* has been designed to support parallel acquisitions. This can be useful when a monitoring process turns in the "background" during fast scans, or when devices with different speed capabilities are brought together on the same measurement.

We have performed both uncorrelated and correlated parallel acquisitions with the VCT6 and the P201 reading VME and PCI channels, respectively. In the uncorrelated case, both timers generate IRQs at the same frequency but

with unlinked phase, creating both coincidence and anti-coincidence conditions during very long scans due to the independence of their internal oscillators. In the second case, the VCT6 is the master timer and two P201 boards have been chained as slaves, so the VME and PCI IRQs synchronisation is at a sub-microsecond level.

Unlike previous measurements, where the acquisition was the only running task in the system, the parallel uncorrelated scans have been performed under heavy system load. Besides the fast acquisition, the system was executing:

- RedHat kernel recompilation using rpmbuild
- Intensive network data transfer with *iperf*-like program
- Linux root (/) partition imaging in a file on a user partition using basic OS tools.

Each of these tasks was continuously executed in *bash* with the basic loop *while true; do …; done*. The OS load ranged from 2.5 – 4.5, with an average of 3.5. The uncorrelated statistics under these conditions are shown in Table 3.

Table 3: IRQ Latencies (µs) in Uncorrelated Parallel Scans @ 1 KHz on Heavily Loaded System

Magnitude	Min.	Ave.	Max.	SD
VCT6 IRQ latency	14	26	80	3.0
P201 IRQ latency	3	6	110	8.5

Like in the previous cases, the tests ran for 24 hours, and no scan point is missing. Again, no specific tuning, or kernel patching has been applied; the default RedHat configuration with the *CONFIG_PREEMPT=no* kernel option has been kept.

The correlated measurements were performed on an unloaded system to double the scan rate. Table 4 shows the corresponding results.

Table 4: IRQ Latencies (µs) in Correlated Parallel Scans @ 2 KHz on Unloaded System (2x P201)

Magnitude	Min.	Ave.	Max.	SD
VCT6 IRQ latency	14	25	70	1.6
2nd P201 IRQ latency	45	50	104	2.0

The shift in the P201 IRQ latency comes from the fact that it is measured on the second board, on a different PCI slot that shares IRQ with a larger number of devices.

DEPLOYMENT AT THE ESRF

The Linux 2.6 *Hook* interface has been in production on ID31 Powder Diffraction ESRF BL for more than one year. It is the core of the BL experiments, which are normally carried out between 60 – 300 Hz sample rates (3 – 15 ms). System instabilities have been completely fixed during this time, including vulnerabilities against crashes of the remote BL control workstation. It is planned to be deployed to the other of ESRF BLs that use fast scans during the next long shutdown, in the framework of a global SuSE 7.2 → RedHat EL 4 migration campaign.

CONCLUSIONS

A generic kernel *Hook* interface, providing fast software trigger and scalar data buffering, has been ported to Linux 2.6. Good performance results are obtained with standard (non real time) RedHat EL 4 kernels running on mid-level dual-core CPUs. Up to two parallel, uncorrelated acquisitions can be performed at 1 KHz on heavily loaded systems without data lost. Support to */sys* virtual file-system provides useful information for *Hook* configuration, monitoring and debugging. The new module has been in production at the ESRF for more than one year; its deployment to the rest of ESRF BLs is foreseen soon.

ACKNOWLEDGEMENTS

The author wants to thank F. Sever, M. Perez, E. Papillon, G. Berruyer, J.M. Clement and H. Gonzalez from the Instrumentation Services and Development Division (ISDD) at the ESRF for their support, help and useful discussions during the development of this work.

REFERENCES

[1] A. Homs-Purón, D. Beltrán, A. Beteva, M. C. Domínguez, P. Fajardo, A. Götz, J. Klora, E. Papillon, M. Pérez, V. Rey, "Linux/PCI: The ESRF Beamline Control System Modernisation", ICALEPCS'03, Gyeongju, October 2003, MP565, p. 162 (2003), http://www.JACoW.org.

[2] A. Homs, F. Sever, "Generic VME Interface for Linux 2.6 Kernels", PCaPAC'08, Ljubljana, October 2008, TUP001, p. 77 (2008), http://www.JACoW.org.

[3] SPEC - X-Ray Diffraction and Data Acquisition Software, G. Swislow, Certified Scientific Software, http://www.certif.com