

ptrace API extensions for BTS

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-12/msg02203.html>

- *From:* "Metzger, Markus T" <markus.t.metzger@xxxxxxxxx>
 - *Date:* Fri, 7 Dec 2007 09:11:04 -0000
-

Roland, Andi,

I would like to discuss the ptrace user interface for the BTS extension. In previous emails, Andi suggested a stream-like interface, but is also OK with an array-like interface (as far as I understood). Roland is dubious about the ptrace API additions.

I would like to settle the discussion and find an interface that everybody can agree to, so I can implement that interface and we can move forward with the patch.

Here's the link to the original patch:
<http://lkml.org/lkml/2007/12/5/234>.

Here are the facts:

- we need to provide access to an array (cyclic buffer) of BTS records
- the array can be quite big
- the most interesting part is the tail
- a BTS record can either describe a branch (from, to address) or a scheduling event (task arrives/departs at timestamp)

Let's look at the entire array, first. I see the following alternatives:

1. get the entire array in one command
 - + simple interface, like GET<arch-specific>REGS
 - a lot of (redundant) copying
2. array-like commands (get size, read element at index)
 - + allows precise reads; minimizes copying
3. stream-like commands (read, maybe seek) [read from back to front]
 - + favors most expected use cases
 - makes other uses much harder (e.g read from front to back)
 - harder to get the semantics right and intuitive (when to reset read pointer? e.g. when stepping between two reads)

Alternatives 1 and 3 require a reordering to turn the cyclic buffer into a sequential array or stream.

Alternative 2 would benefit from that, as well.

ptrace API extensions for BTS

When we reorder the array, the best order would be from back to front, so users can start reading the most interesting part first, and stop when they read enough.

I would recommend alternative 2. Number 1 may result in too much copying, and number 3 is better done in user space; the kernel API should be more flexible and not favor a single use case.

Let's look at the array size, next.

1. pre-defined array size
 - + most simple, no extra command
 - one size will not fit all users
2. user-defined array size
 - + most flexible for the user(need to set a system limit to restrain greedy users)

I would recommend alternative 2. A good citizen will only ask for the space he needs.

In the ideal case, the system limit would be variable (as Andi suggested).

Let's look at the array contents. Currently, we have 3 different record types.

1. self-describing union
 - + most extensible
 - + allows single bts array
 - may waste (user-space) memory
2. separate fixed-type arrays
 - + get command defines interpretation
 - need additional effort to describe relative order between array elements
 - extension requires new set of access commands

I would recommend alternative 1. It is most flexible and most easily extensible. And it is easier to use.

What extensions do we expect in the future?

1. more architectures
2. additional data

Regarding 2, a union would easily allow us to add additional data; at the cost of a few wasted bytes, if the data is not evenly sized. A user may look at the qualifier and either ignore records he does not understand, or bail out.

Regarding 1, we currently provide scheduling timestamps, which are arch independent, and from-to branch information, which should be available on all architectures for a similar feature. I could think of basic block

ptrace API extensions for BTS

from-to information as an alternative representation on some architectures. I could also imagine that other architectures provide additional information (like the predicted bit on Netburst that was dropped for later architectures). Both could be modelled using additional record types.

Additional architectures may want to (re)use and extend the x86 bts record, or they may want to invent their own format. In the former case, we may move the bts union and the bts commands to the generic ptrace header, and provide a default implementation for architectures that do not support it (basically pretend that the array is empty or return an error). In the latter case, they may copy parts of the x86 header.

I would postpone the decision until there are more arch's that wish to support this feature.

Thank you for reading until here.

regards,
markus.

Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen/Muenchen Germany
Sitz der Gesellschaft: Feldkirchen bei Muenchen
Geschaeftsfuehrer: Douglas Lusk, Peter Gleissner, Hannes Schwaderer
Registergericht: Muenchen HRB 47456 Ust.-IdNr.
VAT Registration No.: DE129385895
Citibank Frankfurt (BLZ 502 109 00) 600119052

This e-mail and any attachments may contain confidential material for the sole use of the intended recipient(s). Any review or distribution by others is strictly prohibited. If you are not the intended recipient, please contact the sender and delete all copies.

—
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>