Growing CPU register state without breaking ABI (much)

Linux Plumbers 2017
Los Angeles, CA

Dave Martin <Dave.Martin@arm.com>
September 15, 2017
What am I doing here?

- Arm recently published its next-generation SIMD architecture: the Scalable Vector Extension (SVE). [a-profile]
- I’m writing patches for it... [sve-patches], [sve-git]
- Oops, the register state got large.
- → ABI breaks ahoy!
- Other arches likely do / will have problems in this area too.
The SVE register set

- 32 vector registers Z0–Z31, each $128q$ bits
  Low 128 bits of Z$n$ alias V$n$ from ARMv8.
- 16 predicate registers P0–P15, each $16q$ bits
- 1 special-purpose predicate FFR, $16q$ bits
- Maximum value of $q_{\text{max}}$ determined by the hardware, $1 \leq q_{\text{max}} \leq 16$
- OS / firmware can also constrain $q$ via system registers, $1 \leq q \leq q_{\text{max}}$

That’s $546q$ bytes of user registers, i.e., up to 8736 bytes.
Might grow even larger in the future.
The SVE register set

128\(q - 1\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
128\((q - 1)\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
128  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
0  
\[\begin{array}{|c|c|c|}
\hline
V31
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
Z31
\end{array}\]  

128\(q - 1\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
128\((q - 1)\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
128  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
0  
\[\begin{array}{|c|c|c|}
\hline
V1
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
Z1
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
V0
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
Z0
\end{array}\]  

16\(q - 1\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
16\((q - 1)\)  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
16  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
0  
\[\begin{array}{|c|c|c|}
\hline
FFR
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
FPCR
\end{array}\]  
\[\begin{array}{|c|c|c|}
\hline
FPSR
\end{array}\]  

P15  
\[\begin{array}{|c|c|c|}
\hline
\end{array}\]  
P0  
\[\begin{array}{|c|c|c|}
\hline
1 \leq q \leq 16
\end{array}\]  
up to 8736 bytes total state
So what?
Affected ABI areas

Key ABI impacts I will focus on in this session:

- Signal frame
- ucontext

More straightforward areas (see backup slides):

- ptrace: add a new regset (with some hacks)
- coredumps: add `user_regset.get_size()` for regset size discovery
- KVM: enumerate new regs in `ioctl()` interface.
Signal frame
Signal frame overview

- Data structure encoding user task state during signals
- Pushed on the user stack during signal delivery, popped by sigreturn.
- Partly generic(ish), partly arch-specific.
- POSIX says that the entire signal frame fits in MINSIGSTKSZ bytes:
  - A #define → changing it breaks ABI.
  - Kernel definition often overridden by libc headers.
  - Excludes userspace compiler / libc overheads.
- POSIX also specifies SIGSTKSZ, which includes ‘typical’ overheads, and is used more often. Still a fudge, and still not extensible.
- Userspace relies on these #defines for sizing thread and sigaltstack(2) stacks.
Arm64 signal frame today

- SP
- generic(ish) rt_sigframe
- ucontext
- siginfo
- mcontext

- _reserved(4K)
- integer regs

- arm64 sigcontext

- arm64 extensions
  - exception syndrome
  - size2
  - ESR_MAGIC
  - size1
  - FPSIMD_MAGIC

- © 2017 Arm Limited
Extending arm64 sigcontext

arm64 extensions

extra stack data

0
0

size base addr
size2 ESR_MAGIC
size3 EXTRA_MAGIC
exception syndrome
floating-point regs

size1 FPSIMD_MAGIC

SVE registers

size4 SVE_MAGIC
New arm64 signal frame

extra data

ucontext

siginfo

SP
Architectures affected

- Number-crunching extensions tend to add a lot of register state:

<table>
<thead>
<tr>
<th>Arch extension</th>
<th>Data size</th>
<th>Official signal frame size</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 AVX-512</td>
<td>2K</td>
<td>2K</td>
</tr>
<tr>
<td>power VSX</td>
<td>1K</td>
<td>2K–4K</td>
</tr>
<tr>
<td>arm64 SVE</td>
<td>0.5–8.5K</td>
<td>4K–5K</td>
</tr>
</tbody>
</table>

...  

- No common approach to this problem for now.

- Generally, hack `MINSIGSTKSZ / SIGSTKSZ` and hope for the best...
Damage limitation

If an arch lives long enough, it may eventually exceed its `MINSIGSTKSZ`. For arm64, 4K seemed huge, but it’s not always enough for SVE. So:

- → Inhibit enlarged signal frames by default.
- → Turned on by selecting CPU features that need a larger frame.
- → For SVE, using vectors longer than 64 bytes may enlarge the frame.
Reporting the signal frame size to userspace

When features are enabled that require an enlarged signal frame, userspace needs a way to discover the real frame size.

We could do this:

- Add a new `AT_MINSIGSTKSZ` entry to the aux vector (thanks Will)
- Fall back to `MINSIGSTKSZ` #define when absent
- Fixed for the lifetime of a process: must describe the maximum possible size on this hardware and kernel.
- Could expose through POSIX `sysconf(3)` for a more portable interface.
ucontext
Ucontext

- A task context abstraction, ‘task_struct for userspace’
- More featureful than `siglongjmp(3)` and friends, supports full task switching (in theory)
- Turned out that full portability is almost impossible
- Largely superseded by proper threads
- Some legitimate niche uses, like coroutines
- Incorporates arch `sigcontext` definition
- Welded into the `sigaction(2)` API...
What to do about Ucontext?

- Fixed-size structure with arch context data in the middle of it somewhere. So, totally inextensible.
- **Copying** `ucontext_t` objects can’t work properly if the signal frame is extended.
  - → For arm64, make the reference to the extra context data a **pointer** to give a chance of detecting copying.
  - → Might convince libc / POSIX folks to add proper support, if there’s consensus:
    - `dupcontext()` could copy the entire thing, etc.
  - → Or try to justify letting ucontext finally die ...?
Conclusion
Conclusion

Major register set extensions can be supported by Linux, but there are some pain points:

- signal frame and ucontext are the most problematic
  - inextensible by design
  - largely depends on luck whether an arch’s initial signal frame has enough space.
- ptrace regsets are a bit awkward, but can be extended without ABI breaks
  - NT_ARM_SVE may be a useful template

If enough people care, we could push for:

- common signal frame size reporting via AT_MINSIGSTKSZ
- common libc interface for signal frame size via sysconf
- suitable ucontext API additions (or deprecate ucontext harder)

What should we require from new arches / extensions?
Discussion
Backup / Extra material
ptrace — background

- ptrace regset API, `PTRACE_GETREGSET`, `PTRACE_SETREGSET` etc. much friendlier to extensibility.
- Easy to add new regsets.
- Semantics ill-defined in some areas:
  - size of a register
  - size of a regset
  - behaviour of short reads and writes
- Already some ‘abuse’:
  - e.g., arm64 wedges 32-bit FPSR and FPCR into one 128-bit ‘register’ of `PR_FPREG`
  - But arch-specific knowledge needed in order to manipulate the register data anyway
  - → these aren’t major problems in practice.
PTRACE FOR SVE

- **NT_ARM_SVE** defined with a **variable** number of 128-bit pseudo-registers.
- ‘Register 0’ contains a fixed **struct user_sve_header**, encoding:
  - size of the whole regset
  - maximum possible size of the regset
  - SVE-specific metadata (current and maximum vector length, plus some flags)

- Reading / writing the header by itself is explicitly allowed:
  Caller ran read the size, allocate memory, then read the whole regset.

- Other arches could reuse this model if needed.

- Not clear whether it’s worth formalising as a common model for variable-sized regsets (?)
ELF coredumps

A free gift from the core code!

- An ELF note per thread is automatically added to the coredump, for each regset the arch defines.
- Regset size is statically determined
- To be future-proof, theoretical maximum size of NT_ARM_SVE is stupidly large: roughly $\frac{1}{4}$MB of padding in the coredump per thread

However, making the size dynamic seems pretty non-invasive:

- → Add a `.get_size()` method to `struct user_regset`:
  - If NULL, size calculated statically from `.n` and `.size` (as done currently)
  - If non-NULL, `.get_size()` returns the current size.
- → Should work for any other variable-sized regset in the future (?)
And finally:

- Virtual CPU registers exposed to userspace via ioctls
- (SVE support currently under construction)
- ... but KVM_GET_REG_LIST, KVM_GET_ONE_REG, KVM_SET_ONE_REG quite friendly for extensibility.
- Userspace can save / restore without needing to understand individual registers at all.

For SVE:

- Variable-size registers not natively supported,
  but register IDs don’t need to be assigned contiguously:
- Easy to leave space for future expansion.
[a-profile]  Arm Architecture A-Profile Specifications
   https://developer.arm.com/products/architecture/a-profile/docs

[sve-git]  SVE Git tree on linux-arm.org
   http://linux-arm.org/git?p=linux-dm.git;a=shortlog;h=refs/heads/sve/v2
   git://linux-arm.org/linux-dm.git sve/v2

[sve-patches]  SVE v2 Linux patch posting
   http://lists.infradead.org/pipermail/linux-arm-kernel/2017-August/529575.html
Thanks for listening

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

© 2017 Arm Limited