Ideas to improve glibc and Kernel interaction

Adhemerval Zanella <adhemerval.zanella@linaro.org>

Overview

- This is RFC session to check kernel features glibc lacks (such as termios2), some features glibc might require to implement correctly some standards (such as pthread cancellation), and how to improve communication between kernel and gnu toolchain developers.

- Each slide basically describes a current glibc issue, improvement, and/or idea to improve kernel support

Syscalls wrappers

- The main issue is glibc might not provide the latest Linux syscalls

- There is a recent discussion on libc-alpha about provide thin syscall wrappers to accomplish
  - glibc handles all standards requirements for syscalls (errno, pthread cancellation, kernel optional flags, etc.)
  - Some syscalls uses fallback implementation or use optimized implementations (vDSO)
  - What about different calling conventions depending of build flags (off_t/off64_t and time_t/time64_t in near future)?
  - Some exported APIs differ from kernel API (termios for instance)

- Does make sense to export all possible syscalls?
  - What about deprecated ones (sysctl, socketcall, osf_*, etc.)?
  - What about possible interoperation issues with glibc expectation (set_robust_list for instance)?

Formal description of syscall interface

- It would be interesting to follow up on this such that glibc, and other consumers can use this data without recreating
- Declarative, machine-readable approach to obtain system call numbers and error numbers from the kernel sources.
- This was discussed at LPC 2016.
  - https://lkml.org/lkml/2016/11/6/126

Consistent kernel support for arch-independent system calls

- New system calls which are not arch-specific should be wired up at the same time for all architectures
  - Avoid issues like a syscall added to unistd.h for an architecture, but not added to the syscall table
  - Or a syscall added to the native syscall table for some ABI but not added to the corresponding compat syscall table

- Even now we stumble on issue on syscalls being wire-up on different version on different architectures

Kernel and glibc header coordination

- Create and grow a set of tests in glibc or in the kernel to verify key headers work in both include directions.
- Continue to support libc_compat.h as the method to support inclusion header guards where possible.
- Not all header inclusion ordering is valid
  - we should not attempt to just solve this by permuting all headers
  - there is too much invalid garbage produced in that kind of result.

Have identified points of contact

- glibc maintains points of contact for all major distributions
  - https://sourceware.org/glibc/wiki/MAINTAINERS#Distribution_Maintainers
  - We use the points of contact to discuss things like switching to Python 3.4 for build scripting and tooling.

- We need something to get clarification of userspace ABI evolution (future plans) and how to deal with ABI changes after the fact.

- Do we need similar to discuss interaction between libc and kernel?
  - Maybe Linux API <linux-api@vger.kernel.org> ?

Integrate termios2 interface with a GNU extension

- Linux has supported arbitrary baud rates for **12 years**
  - Added in kernel version 2.6.20
  - New ioctl()s TCGETS2/TCSETS2

- glibc only supports POSIX defined baud rates (up to B38400) plus some Linux specific one (B4000000)
  - Also only uses old ioctl interface

- Current work is to cleanup glibc termios implementation and add a GNU extension (cf[gs]et[io]bps)
  - H. Peter Anvin work

- Tracked on sourceware bugzilla: **https://sourceware.org/bugzilla/show_bug.cgi?id=10339**

Pthread cancellation handling

- The idea is to use musl-libc strategy to use a syscall entrypoint along with address markers
  - glibc check if cancelled syscall has side-effects by comparing the SA_SIGINFO returned IP against the markers
  - It works well on almost every architecture

- The issue is x86-32 will require to use int80 instead of vDSO, slowing down cancellable syscalls in multithreaded environments
  - Recent kernel discussion about providing a new way to accomplish stalled

- Tracked on sourceware bugzilla: **https://sourceware.org/bugzilla/show_bug.cgi?id=12683**

Fork async-signal-safeness

- This requires a way to determine when a function is executed in a signal handler
  - So we an act on atfork handler properly

- On some other system this is accomplished by a pure-userspace solution (for instance FreeBSD).
  - A possible solution on glibc might be add signal handler wrappers
  - This imposes some serialization and less scalability due lock/atomic operations
    - My WIP work basically uses a recursive plus enable/disable cancellation for SA_NODEFER

- Could kernel help us by providing an atomic update on a variable when signal handler is executed, similar to tid set in clone?
  - One global wrapper (set via prctl) which is called by the kernel with the handler function as an argument would be sufficient

Save and restore errno around signal handler?

- Can we extend a possible solution for atomic update on a variable when signal handler is executed to save/restore arbitrary values?
  - This will stop the mistakes that users commit by not saving and restoring errno on their own. It could also be conditionally disabled if users didn’t care about it.

Making EPERM friendlier

- Additional error delivery information for file operations and mmap
  - For exemplo explanations of failures from security modules, nftables, etc.
  - Obtained from a GNU extension

- Does kernel currently supports provide such information?

- Previous discussion at https://lwn.net/Articles/532771/

Robust Mutexes Fundamental Flaw

- There is a race where the kernel will corrupt a mapping in the same place as the previous robust TID that is has to zero out.
  - This is ABA issue, where the share memory referenced by a robust mutex is deallocated and reallocated to another object
  - Propose solution on glibc BZ#14485 is to add a global lock on mmap/munmap and the robust operations to remove the mutex from the thread's linked list, unlock, and clear the "pending" slot

- Do we need to redesign the protocol between userspace and the kernel to fix this?

- Tracked on sourceware bugzilla: **https://sourceware.org/bugzilla/show_bug.cgi?id=14485**