glibc and system call wrappers

Florian Weimer, Red Hat Platform Tools Team

Virtual Linux Plumbers, 2020-08-28

Outline

- Why do we have system call wrappers?
- How can we add them to glibc?
- Do we actually want to do that?
- What can the kernel do to make things easier?

- Poll: What do *you* work on?
  - **A**: kernel, **B**: userspace toolchain (compiler, core libraries), **C**: other userspace, **D**: something else

What are system call wrappers?

- `off64_t lseek(int fd, off64_t off, int whence);`

```
lseek: movl $8, %eax          /* syscall number */
       syscall
       cmpq $-4096, %rax      /* magic errno limit */
       ja 1f                  /* handle error */
       ret
1:     movq __libc_errno@gottpoff(%rip), %rdx
       negl %eax
       movl %eax, %fs:(%rdx) /* update errno TLS */
       movq $-1, %rax
       ret
```

Why bother?

- Can we just use a generic wrapper?
- `syscall(__NR_lseek, fd, 0, SEEK_SET);`

Why bother? Portability!

- Need to use the correct types:
  `syscall(__NR_lseek, fd, (off64_t) 0, SEEK_SET);`
- Need to use the correct system call:

  ```
  off64_t off;
  syscall(__NR__llseek, fd, 0L, 0L, &off, SEEK_SET);
  ```
- This is more common than you would think (`open` → `openat`, `futex` → `futex_time64`).

glibc `lseek` (without symbol management)

```
off64_t lseek (int fd, off64_t offset, int whence)
{
#ifdef __NR__llseek
  loff_t res;
  int rc = INLINE_SYSCALL_CALL (_llseek, fd,
      (long) (((uint64_t) (offset)) >> 32),
      (long) offset, &res, whence);
  return rc ? rc : res;
#else
  return INLINE_SYSCALL_CALL (lseek, fd, offset, whence);
#endif
}
```

glibc implementation options

- C with `INLINE_SYSCALL_CALL`: automatic `errno` handling
- C with `INTERNAL_SYSCALL_CALL`: no `errno` updates
- Auto-generated assembler via `syscalls.list`
- Manual assembler (only required in exceptional cases)

glibc's system call wrapper requirements

- Copyright assignment
- Determining the appropriate header file and API scope (POSIX/standard vs GNU vs Linux)
- Should the wrapper imply a cancellation point? (No.)
- Finding the right place in the source tree: `misc` or `sysdeps/unix/sysv/linux`
- `Versions` file and ABI list updates
- Minimal test case
- Update to the glibc manual (GFDL-licensed)
- `NEWS` file update

Contributing wrappers: Help wanted!

- I would have liked to include a tutorial here, but even now, every system call is a little bit different:
  - Adding new header file customization points for GNU vs Linux variance (e. g. for `<unistd.h>`)
  - Writing entirely new sections in the manual explaining concepts that can be referenced (`*at` functions)
  - Container-based testing might be needed, maybe with test harness enhancements.
  - It's still difficult to predict what you might encounter.
- But we will help you if you want to implement a wrapper and walk you through the process.

State on the glibc side

- There is consensus for adding wrappers, unless the system call is obsolete or breaks core userspace invariants.
- Case in point: `gettid` (finally added in glibc 2.30)
- There is still a substantial backlog.
- Manual updates for core undocumented concepts (such as `*at`-based pathname resolution) are under way.
- So far, we ignore the downsides of adding wrappers.

Downsides of wrappers

- New wrappers add new symbols to the glibc ABI.
- Current policy is that the ABI does not change within one glibc release.
  - Up to six months waiting time.
- Distributions do not backport wrappers.
  - `/lib64/libc.so.6: version 'GLIBC_2.30' not found` when trying to run a program that uses `gettid` on glibc 2.28.
  - Backports are difficult for some RPM-based distributions due to their dependency management.
  - Up to three years waiting time, maybe more.

Downsides of wrappers

- Emulation in userspace is tempting, but rarely a good idea. Latest example was `copy_file_range`.
- Potential exception: Call the flag-less system call variant if the caller passes a zero flag.
  - Even that does not always work, see `nanosleep` vs `clock_nanosleep`
- Adoption of new system calls breaks browsers, `systemd-nspawn` (the `EPERM` vs `ENOSYS` issue). Availability of wrappers may speed this up.

Downsides of wrappers

- glibc's wrappers cannot be used in all contexts, e. g., missing thread control block (TCB) after `clone`.
  - Reporting failure via `errno` needs the TCB for TLS.
  - Stack protector instrumentation needs the TCB for the canary on many targets.
  - `setxid` broadcast
  - POSIX cancellation handling
  - Lazy binding might call into the dynamic loader.
- Even experienced programmers do not know of these restrictions.
  - This topic is related to asynchronous signal safety and asynchronous cancellation safety.
  - (`syscall` shares some of these problems.)

New kind of wrappers for glibc?

- `syscallresult64 _G_lseek(int,off64_t,int);`
- In-line error signaling is used, like the usual kernel/userspace ABI.
- The wrappers are statically linked hidden functions symbol.
  - No ABI change to shared objects helps with backporting.
- The wrappers are built specifically for no TCB dependency at all.
- They are not cancellation points.
- They are usable after `clone`.
  - This avoids `posix_spawn` feature creep.

Can the kernel make this easier?

- No more multiplexers, please.
  - `syscall(__NR_FUTEX, &futex, FUTEX_WAIT_PRIVATE, 1, NULL, NULL, 0);`
  - `futex(FUTEX_WAIT_PRIVATE, 1, NULL, NULL, 0);`
- It still needs porting to `futex_time64`, even though `struct timespec` is not actually used.
- Multiplexers can break with ILP32 target variants if variadic arguments are not promoted correctly for use with the kernel/userspace ABI.
- Lazy Linux interface emulators break probing.

```
int sync_file_range(int, off64_t, off64_t, unsigned)
{
  // There are no observable side effects, right?!
  return 0;
}
```

Can the kernel make this easier?

- Enable generic system calls for all architectures at the same time.
  - Already much improved, I think.
- Use appropriate types.
  - `unsigned` for flag arguments (not `long`).
  - `size_t` for byte sizes (not `int`)
- Pass 64-bit arguments in memory.
  - `off64_t *` in `copy_file_range` is nice.

Can the kernel make this easier?

- Conventions for extensions with which programmers become familiar over time (see Christian Brauner's talk).
- But do we actually need extensible system calls? How costly is it to add more system calls instead?
- Feature bitmaps may help imperfect emulators (indicating `vfork`-as-`fork`, for example).
- Maybe the kernel can do something to help with the sandboxing issues surrounding new system calls.