Designing Extensible Syscalls

Aleksa Sarai
Senior Software Engineer
@lordcyphar
<cyphar@cyphar.com>

Why?

- Trade-off between more (specific) syscalls and fewer (extensible) syscalls.

`ioctl` ↔ `openat2` ↔ `statx` ↔ `open` ↔ `creat`

(most extensible) (least extensible)

- Extensible syscalls allow for more flexible userspace code.
  - Libraries *can* allow usage (without modification) of newer features.
  - Applications *can* allow usage (without modification) of newer features.
  - Ideally, they should be *both* forward and backward compatible.

Bit Flags

- Traditional UNIX-like extensibility.

```
int open(const char *path, unsigned int flags, ...);
```

- Very simple to add new boolean flag bits.
  - Cannot add new arguments easily and idiomatically (see `O_TMPFILE`).
- Unfortunately, some new(ish) syscalls don't include flags.
  - `renameat` (and the later `renameat2`) is a classic example.
  - `readlinkat` doesn't take flags either...

Fixed-Size Structs

- Slightly more modern way to “add more arguments”.

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- Simple to add new fields.
  - Requires clairvoyance to predict the right size of the struct.
  - Further extensions are possible, but not “nice” to use or implement.
    - Similar problems to adding syscall arguments.

Extensible Structs

- Better than sliced bread.

```
int openat2(int dfd, const char *path,
            struct open_how *how, size_t size);
```

- New fields can be added trivially ad infinitum.
  - Userspace passes struct *and struct size.*
  - Field additions are both forward and backward compatible.
  - *Userspace doesn't need to care about them.*
- Currently used by five newer syscalls (and a few other interfaces).

Extensible Structs

- Rules to ensure backward and forward compatibility:
  - Extension fields are always appended to the structure.
  - Future fields must have their zero value be “the old behaviour”.
  - (ksize == usize) → Copy the struct verbatim.
  - (ksize > usize) →
    Copy usize bytes, zero-fill trailing (ksize – usize) bytes.
  - (ksize < usize) →
    Copy ksize bytes, check if trailing (usize – ksize) bytes are zeroed.
    If non-zero bytes are present, return `-E2BIG`.
  - Implemented in `copy_struct_from_user()`.

openat2 / clone3 / bpf

```
int openat2(int dfd, const char *path,
            struct open_how *how, size_t size);


struct open_how {
  u64 flags;            // openat(2) flags
  u64 mode;             // openat(2) mode
  u64 resolve;          // RESOLVE_* flags
  // future fields go here
};
```

sched_setattr / perf_event_open

```
int sched_setattr(pid_t pid, struct sched_attr *attr,
                  unsigned int flags /* must be 0 */);


struct sched_attr {
  u32 size;           // size of sched_attr (set to ksize, on -E2BIG return)
  u32 sched_policy;   // policy (SCHED_*)
  u64 sched_flags;    // flags
  s32 sched_nice;     // nice level
  u32 sched_priority; // priority
  u64 sched_runtime;  // SCHED_DEADLINE runtime
  u64 sched_deadline; // SCHED_DEADLINE deadline
  u64 sched_period;   // SCHED_DEADLINE period
  // future fields go here
};
```

Future Work: CHECK_FIELDS

- Finding out which flags and fields are supported *sucks*.
  - Userspace has to do a bunch of retries.
  - Have to come up with flag combinations which fail (not `-EINVAL`) or no-op.
  - Field-supported checks are even less fun.
    - Requires a binary search on `-E2BIG` for extensible structs.
    - Requires a checking each reserved field for fixed-size structs.

Future Work: CHECK_FIELDS

- **Idea:** Add a flag which causes a syscall to fill the struct with all supported bits.
  - Returns `-ENOANO` to avoid confusion with a successful return.
    - `-ENOANO` is the least used errno value (but could just add a new errno).
  - Use same flag bit (1 << 63) for all extensible-struct syscalls.
  - Similar rules to `copy_struct_from_user()`.
    - (ksize > usize) → only fill first usize bytes.
    - (ksize < usize) → zero-fill trailing (usize – ksize) bytes.

Future Work: CHECK_FIELDS

```
struct open_how how =
       { .flags = CHECK_FIELDS };

openat2(AT_FDCWD, “”, &how, sizeof(how));
switch (errno) {
   case EINVAL:
      // CHECK_FIELDS unsupported (fallback to old method)
      find_all_openat2_supported_bits(&how);
      break;
   case ENOANO:
      // CHECK_FIELDS supported
      resolve_no_automount_supported = (how.flags & RESOLVE_NO_AUTOMOUNT);
      cwd_field_supported = (how.cwd_fd != 0);
      break;
   default:
      abort();
}
```

Caveats

- Not all syscalls need to be *this* extensible.
  - "Trivial" syscalls (`readlinkat`) would only ever need bitflags.
  - Arguably, ioctls shouldn't be extensible in this fashion.
    - Though `SECCOMP_IOCTL_NOTIF_*` does it anyway.
- Initial struct layout and planning is still important.

Questions?

Time to break out the pitchforks!