

Adding C Library Wrappers for all Syscalls – Part 2

Linux Plumbers Conference 2026 Update

Presenter:

Carlos O'Donnell

Distinguished Engineer

Presenter:

Florian Weimer

Senior Principal Software Engineer

What problem are we trying to solve?

- Make it easy to use kernel functionality exposed via syscalls

How do we do that today?

- Provide libraries that encapsulate groupings of functionality
 - liburing, libibverbs, liburcu, libaio, ... libc
- Use the syscall() function provided by libc
- Assembly for custom per-arch per-syscall calling conventions



LPC 2025: What should we agree to?

- **Agree to review, coach, and mentor syscall wrappers through the process.**



[RFC PATCH 0/2] Linux: Add futex_waitv syscall wrapper



From: "André Almeida" <andrealmeid@igalia.com>
To: libc-alpha@sourceware.org, carlos@redhat.com,
adhemerval.zanella@linaro.org
Cc: "André Almeida" <andrealmeid@igalia.com>
Subject: [\[RFC PATCH 0/2\] Linux: Add futex_waitv syscall wrapper](#)
Date: Fri, 14 Aug 2026 15:36:10 -0300 [\[thread overview\]](#)
Message-ID: <20260814183612.1780736-1-andrealmeid@igalia.com> ([raw](#))

Hi folks,

Based on a talk on last LPC by Carlos O'Donell[1], I'm sending this patch as a starting point to add the futex_waitv() syscall wrapper.

I tried to base myself on the work done for others wrappers, but I see that there are missing details in my implementation, so I'm seeking guidance on how to approach this.

Thanks!

André

[1] <https://lpc.events/event/19/contributions/2220/>

André Almeida (2):

Linux: Add futex_waitv

misc: tst-futex: Create futex test

```
include/sys/futex.h          |  4 ++
sysdeps/unix/sysv/linux/Makefile |  3 +
sysdeps/unix/sysv/linux/futex_waitv.c | 32 ++++++++
sysdeps/unix/sysv/linux/sys/futex.h | 24 ++++++
sysdeps/unix/sysv/linux/tst-futex.c | 93 +++++++++++++++++++++
5 files changed, 156 insertions(+)
create mode 100644 include/sys/futex.h
create mode 100644 sysdeps/unix/sysv/linux/futex_waitv.c
create mode 100644 sysdeps/unix/sysv/linux/sys/futex.h
create mode 100644 sysdeps/unix/sysv/linux/tst-futex.c
```

--

2.55.0



- Architecture: **x86_64**
 - Linux master has **386** syscalls (glibc missing **fchroot**)
- How many Linux syscalls have glibc wrappers? Starting place.
 - **292** syscalls have same-named global symbols.
 - **16** syscalls have different-name global symbols e.g. **pselect6** is **pselect**, **eventfd2** is **eventfd**, **newfstatat** is **fstatat**/**fstatat64**.
 - **77** syscalls have no exported wrapper!



Syscalls without glibc wrappers (opinionated highest priority first):

Core kernel functionality	concurrency: futex, futex_waitv, futex_wake, futex_wait, futex_requeue, membarrier, rseq_slice_yield process: clone3, kcmp scheduling: ioprio_set, ioprio_get isolation: seccomp, listns mm: userfaultfd, memfd_secret, cachestat fs: quotactl_fd, listmount, statmount, setxattrat, getxattrat, listxattrat, removexattrat, open_tree_attr, file_getattr, file_setattr security: map_shadow_stack
Domain-specific functionality	io_uring: io_uring_setup, io_uring_enter, io_uring_register numa: mbind, set_mempolicy, get_mempolicy, migrate_pages, move_pages, set_mempolicy_home_node kernel: kexec_load, kexec_load_file, finit_module, keyring: add_key, request_key, keyctl observability: perf_event_open, bpf security: landlock_create_ruleset, landlock_add_rule, landlock_restrict_self, lsm_get_self_attr, lsm_set_attr, lsm_list_modules aio: io_setup, io_destroy, io_getevents, io_submit, io_cancel, io_pgetevents
libc / kernel interface kernel internals	set_thread_area, get_thread_area, set_tid_address, set_robust_list, get_robust_list, rseq restart_syscall, uretprobe, uprobe
Legacy / Dead - Do not implement	getdents, sysfs, _sysctl, afs_syscall, tuxcall, security, tkill, lookup_dcookie, epoll_ctl_old, epoll_wait_old, lookup_dcookie, vserver



What principles could we apply to selecting from and working on the list?

- Avoid C library/kernel header conflicts
 - Replicating complex kernel headers in C headers is not desirable.
 - e.g. userfaultfd, seccomp, bpf, io_uring etc.
- Complete existing functionality
 - Make it easy to use certain new functionality
- Focus on system calls with strong C library dependency (e.g., via TCB)
 - Help developers avoid C library specific hacks e.g. update TCB after clone.
- Prioritize system calls with non-portable signatures
 - Help developers avoid architecture specific syscall invocation.



Applying the principles

- Complete existing functionality
 - `map_shadow_stack`, `membarrier`, `rseq_slice_yield`
- Focus on system calls with strong C library dependency (e.g., via TCB)
 - `futex`, robust mutexes (`robust_list_head` offset from TP)
 - Clone system call variants
- Prioritize system calls with non-portable signatures
 - `futex` (due to `time64` support)
 - System calls with many arguments, 64-bit arguments
 - Historic examples: `mmap64`, `lseek`



Improving the generic syscall wrapper

- **Three variants: default, returns-twice, cancelable**
 - `_syscall (...), _syscallr (...), _syscallc (...)`
- Use the correct register type (not long)
- **Use in-band signaling (not errno/TCB)**
- Minimize C header dependencies
- **Not variadic, to simplify use from FFI**
 - e.g. `_syscall10 (), _syscall10r (), _syscall10c () ...`
- May reduce pressure to add new wrappers



Full access to clone system call capabilities

- If you don't need the TCB, use the new `syscall` wrappers.
- If you need a libc-compatible TCB, that does not work.
 - Idea: **Expose `clone3` functionality via `pthread_create`.**
 - `pthread_create` allocates TCB and (optionally) the stack.
 - The application is mostly responsible for flag compatibility.
 - We need to figure out how to deallocate TCB etc. after a vfork-style `pthread_create`. Similar to detached threads?



“Do you need any additional syscalls to make your life easier?”

- Whole process group permission transition e.g. setxid broadcast
 - `run_callback_on_tid()` run a callback on a different task using the current tasks stack, while blocking the current task
- `exit_signal()` to exit a process with a given signal

Proposals should go to: <https://github.com/uapi-group/kernel-features>



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



linkedin.com/company/red-hat



youtube.com/@redhat



facebook.com/RedHat



x.com/RedHat