A new API for robust futex list

André Almeida

andrealmeid@igalia.com / tonyk (IRC/Matrix)

Linux Plumbers Conference 2025

Windows on x86: the gaming ABI for SteamOS

- Most of desktop games at Steam store are compiled for Windows (x86)
- SteamOS is a Linux distro for x86 and Arm64

Windows on x86: the gaming ABI for SteamOS

- Most of desktop games at Steam store are compiled for Windows (x86)
- SteamOS is a Linux distro for x86 and Arm64
- A lot of fun things happens when you have all those translations layers
  - Proton, **FEX-Emu**
  - Filesystems, futex, memory management, ...

Windows on x86: the gaming ABI for SteamOS

- Most of desktop games at Steam store are compiled for Windows (x86)
- SteamOS is a Linux distro for x86 and Arm64
- A lot of fun things happens when you have all those translations layers
  - Proton, **FEX-Emu**
  - Filesystems, futex, memory management, ...
- We can discuss about all the implications of that later :)

Windows on x86: the gaming ABI for SteamOS

- **FEX-Emu** is a JIT for x86 (64 or 32-bit) binaries on top of Arm64
- Whenever it finds a `syscall` instruction, it needs to translate to the Arm64 syscall

Windows on x86: the gaming ABI for SteamOS

- **FEX-Emu** is a JIT for x86 (64 or 32-bit) binaries on top of Arm64
- Whenever it finds a `syscall` instruction, it needs to translate to the Arm64 syscall
- Not so easy for x86-32 syscalls
- FEX-Emu wiki: 32Bit Syscall Woes

What's set_robust_list()?

set_robust_list()

- If a (futex) lock owner dies, it can starve the waiters
- To avoid that, everytime a thread takes a lock, it adds it’s pointer to a linked list, the robust list

set_robust_list()

- If a (futex) lock owner dies, it can starve the waiters
- To avoid that, everytime a thread takes a lock, it adds it's pointer to a linked list, the robust list
- `set_robust_list()` is a syscall to the kernel what is the address of the head of the list

set_robust_list()

- If a (futex) lock owner dies, it can starve the waiters
- To avoid that, everytime a thread takes a lock, it adds it's pointer to a linked list, the robust list
- `set_robust_list()` is a syscall to the kernel what is the address of the head of the list
- In the kernel side exit path, the kernel wakes all threads waiting for that futex

set_robust_list()

- If a (futex) lock owner dies, it can starve the waiters
- To avoid that, everytime a thread takes a lock, it adds it's pointer to a linked list, the robust list
- `set_robust_list()` is a syscall to the kernel what is the address of the head of the list
- In the kernel side exit path, the kernel wakes all threads waiting for that futex
- The kernel marks the futex word with `FUTEX_OWNER_DIED`

set_robust_list()

- If a (futex) lock owner dies, it can starve the waiters
- To avoid that, everytime a thread takes a lock, it adds it's pointer to a linked list, the robust list
- `set_robust_list()` is a syscall to the kernel what is the address of the head of the list
- In the kernel side exit path, the kernel wakes all threads waiting for that futex
- The kernel marks the futex word with `FUTEX_OWNER_DIED`
- There's a special "pending" field on `struct robust_list_head` if the thread dies before adding to the list

Why do we need a new API?

Arm64 and the 32-bit entry point

- On x86_64, 64-bit apps can call both 64 and 32-bit syscalls
  - Arm64 cannot do that
  - But we want to emulate 32-bit apps...
  - If you give a 32-bit robust list to a 64-bit kernel, it cannot parse correctly

Arm64 and the 32-bit entry point

- On x86_64, 64-bit apps can call both 64 and 32-bit syscalls
  - Arm64 cannot do that
  - But we want to emulate 32-bit apps…
  - If you give a 32-bit robust list to a 64-bit kernel, it cannot parse correctly
- Can’t you run the emulator as a 32-bit app?
  - New Arm64 cores are removing the 32-bit ISA
  - Arm NEON (32-bit SIMD) is not good for emulating SSE and AVX

Arm64 and the 32-bit entry point

- On x86_64, 64-bit apps can call both 64 and 32-bit syscalls
  - Arm64 cannot do that
  - But we want to emulate 32-bit apps…
  - If you give a 32-bit robust list to a 64-bit kernel, it cannot parse correctly
- Can’t you run the emulator as a 32-bit app?
  - New Arm64 cores are removing the 32-bit ISA
  - Arm NEON (32-bit SIMD) is not good for emulating SSE and AVX
- We need a interface to tell to the kernel the **bitness of the list**

One robust list per task

- The current syscall can set only one robust list per task
- FEX uses robust futexes itself, but if the app uses as well… FEX needs to give up its robustness
- We need an interface to **set multiple lists heads per task**

Limit of elements on the robust list

- To avoid being trapped in a infinity loop, the kernel parses up to 2048 elements in the robust list

Limit of elements on the robust list

- To avoid being trapped in a infinity loop, the kernel parses up to 2048 elements in the robust list
- But userspace was never told that, it’s not part of headers or the uAPI

Limit of elements on the robust list

- To avoid being trapped in a infinity loop, the kernel parses up to 2048 elements in the robust list
- But userspace was never told that, it's not part of headers or the uAPI
- Bug: Robust mutexes do not take ROBUST_LIST_LIMIT into account
- Let's expose the limit or make it limitless and have some countermeasures to circular lists

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes.
3. Low level unlock (clear the futex and possibly wake waiters).

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes.
3. Low level unlock (clear the futex and possibly wake waiters).
4. Clear the "pending" slot in the thread's robust mutex list.

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes.
3. Low level unlock (clear the futex and possibly wake waiters).

Another thread takes the mutex lock, free it and alloc a memory/file on the same memory region

4. Clear the "pending" slot in the thread's robust mutex list.

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes.
3. Low level unlock (clear the futex and possibly wake waiters).

Another thread takes the mutex lock, free it and alloc a memory/file on the same memory region

~~4. Clear the "pending...~~ This thread dies before finishing the cleanup. The kernel writes `FUTEX_OWNER_DIED` in the memory, corrupting it

Race condition of unmap and robust list

The general procedure for unlocking a robust mutex is:

1. Put the mutex address in the "pending" slot of the thread's robust mutex list.
2. Remove the mutex from the thread's linked list of locked robust mutexes.
3. Low level unlock (clear the futex and possibly wake waiters).

Another thread takes the mutex lock, free it and alloc a memory/file on the same memory region

~~4. Clear the "pending...~~ This thread dies before finishing the cleanup. The kernel writes `FUTEX_OWNER_DIED` in the memory, corrupting it!!!!!!!!!!

Race condition of unmap and robust list

- We need to serialize the exit path with every mmap()/unmap()?
- Can we change the API to avoid that?
- Bug: File corruption race condition in robust mutex unlocking
  - Bonus idea: It's hard to get a reproducer for this, but maybe we could write a sched_ext scheduler that put things in the order that we want to

Finally, the new API!

```
set_robust_list2(struct robust_list_head *head, unsigned
int index, unsigned int cmd, unsigned int flags)
```

- cmd = CREATE_LIST_{32,64} returns a new unused index and set head
- cmd = SET_LIST_{32,64} + index = [0, 10] re-sets a list
- cmd = FUTEX_ROBUST_LIST_CMD_LIST_LIMIT, get the number of lists per task

Finally, the new API!

```
get_robust_list2(int pid, void **head_ptr, unsigned int
index, unsigned int flags)
```

- Returns the head for a giving pid + index

Join us!

https://www.igalia.com/jobs

Image credits: