ptrace()-friendly pidfds

Nick Alcock, Oracle Corporation

The problem

When using ptrace(), a common pattern is to loop in waitpid() so that signals dispatched to the process, syscalls it’s doing (when using PTRACE_O_TRACESYSGOOD) and other such things are processed rapidly.

But waitpid() does not compose with anything else: you can’t poll() at the same time etc, even if you want to do other things (like receive messages from other parts of your process, perhaps instructions to the ptracer) at the same time, in the same thread.

The problem

This is what pidfds exist to solve! But when a waitpid() is coming from ptrace(), it suddenly stops working: the rather strange thread-directed waitpids that ptrace produces are not dispatched to pidfds.

No obvious alternative

Casey Dahlin wrote a predecessor to pidfds called waitfd which was rejected in part on the grounds that it was redundant: you could always waitpid() in a separate thread.

This is almost right, but when ptracing, waitpids from ptraced children are directed to the *thread* that did the ptrace, not to the whole process.

So you cannot do this in a separate thread.

Implementation trivial?

Fixing this is very simple: see

https://github.com/oracle/dtrace-linux-kernel/commit/96db4343b9d94299f

But this is an unavoidable behavioural change: callers that are ptracing and using pidfds in the same thread will now see new wakeups they never did before. Is this sort of thing acceptable? Adding a flag to make the behaviour optional seems difficult, but equally it seems unlikely to break too many users

… but maybe still problems

The implementation at the link has problems, notably that if one non-ptracer has a pidfd on some process that a ptracer also has one on, both will be woken up.

Fixing this likely requires multiple *classes* of pidfds, so we can wake up only the ‘ptrace class’. This suddenly means pidfds are more than just stuffed into a private_data and need their own allocation lifetime and my head hurts.

… but maybe still problems

One last problem! Not only are ptrace waits thread-directed, you can ptrace *single threads*. This suddenly means that (this class of) pidfd might not have the thread-group-leader restriction that all others have. I don’t know what implications this might have. Why does this restriction exist in the first place?