Operating Systems Review

Muyao Xiao
email xiaomuyao@outlook.com

October 18, 2024

Contents

1 Threads and Process

1.1 What's the difference of a thread and a process?

There's a classical interview question: *What's the difference of a thread and a process?*.

Personally I think the answer of this question is largely based on the context, but theres is a beautiful explaination from *Linus Torvalds* himself (The complete version can be found here or here.):

On Mon, 5 Aug 1996, Peter P. Eiserloh wrote:

> We need to keep a clear the concept of threads. Too many people seem to confuse a thread with a process. The following discussion does not reflect the current state of linux, but rather is an attempt to stay at a high level discussion.

NO!

There is NO reason to think that "threads" and "processes" are separate entities. That's how it's traditionally done, but I personally think it's a major mistake to think that way. The only reason to think that way is historical baggage.

Both threads and processes are really just one thing: a "context of execution". Trying to artificially distinguish different cases is just self-limiting.

A "context of execution", hereby called COE, is just the conglomerate of all the state of that COE. That state includes things like CPU state (registers etc), MMU state (page mappings), permission state (uid, gid) and various "communication states" (open files, signal handlers etc).

Traditionally, the difference between a "thread" and a "process" has been mainly that a threads has CPU state (+ possibly some other minimal state), while all the other context comes from the process. However, that's just_one_ way of dividing up the total state of the COE, and there is nothing that says that it's the right way to do it. Limiting yourself to that kind of image is just plain stupid.

The way Linux thinks about this (and the way I want things to work) is that there _is_ no such thing as a "process" or a "thread". There is only the totality of the COE (called "task" by Linux). Different COE's can share parts of their context with each other, and one _subset_ of that sharing is the traditional "thread"/"process" setup, but that should really be seen as ONLY a subset (it's an important subset, but that importance comes not from design, but from standards: we obviusly want to run standards-conforming threads programs on top of Linux too).

In short: do NOT design around the thread/process way of thinking. The kernel should be designed around the COE way of thinking, and then the pthreads _library_ can export the limited pthreads interface to users who want to use that way of looking at COE's.
...

Besides which, I also found some links might be helpful:

- Is it true that fork() calls clone() internally?: `fork()` doesn't call `clone()`, both are functions that use the syscall `clone`.
  - It's also interesting to check the clone syscall, it has arguments to decide whether to share virtual address space, the table of file descriptors, etc. Basically in my understanding, we distinguish a so-called process or a thread based on those arguments.