pidfds: Process file descriptors on Linux

Tuebix, 2019
Tübingen, Germany

Christian Brauner
Senior Engineer
Canonical Ltd.

christian@brauner.io
christian.brauner@ubuntu.com
@brau_ner
https://brauner.io/
https://people.kernel.org/brauner

Processes, Threads, PIDs, TIDs, fds, and other assorted nonsense

A pidfd is a file descriptor referring to a process or a thread*.

* At some point in the future

Processes

- dumb definition would be "executing instance of a program"
- a binary image
- memory (stack, heap)
- resources (file descriptors etc.)
- attributes (uid, gid, capabilities, etc.)

- fork() + exec() to spawn new process
  load binary image

```
struct linux_binprm
exec_binrprm()
-> search_binary_handler(<aout, elf, elf_fdpic, em86, flat, script>)
```

Processes

What does fork() do?

- **duplicates/copies** the calling process
- duplicated/copied process -> parent
- duplicate/copy -> child
- typical code pattern (abbreviated at the end):

```
int ret;
pid_t pid, wpid;

pid = fork();
if (pid < 0)
        _exit(EXIT_FAILURE);

/* child */
if (pid == 0) {
        printf("I am the child");
        _exit(EXIT_SUCCESS);
}

/* parent */
wpid= waitpid(pid, NULL, 0);
```

Threads

- lightweight process
- processes exclusively own their resources (for the most part)
- threads share their resources (for the most part)

- clone() to create a new thread by specifying a bunch of flags; at least: CLONE_THREAD, CLONE_FILES, CLONE_FS, CLONE_SYSVSEM, CLONE_SIGHAND, CLONE_SETTLS, CLONE_VM
- userspace library (part of libc) to manage threads: pthreads

Processes and Threads

- vague terminology
- kernel usually uses "task" which is best translated to "thread"
- single thread ("single-threaded") -> process
- multi-threaded -> multiple threads with one thread-group leader

PIDs, TIDs, and for the sake of confusion TGIDs

- PID -> process identifier
- TID -> thread identifier
- TGID -> thread-group identifier

- single threaded processes PID == TID == TGID
- multi-threaded process TGID == PID != TID for non-thread-group leaders
- thread-group leader: first process that created a new thread
- thread-group leader stays zombie until all other threads in its thread-group have exited
  (that has various interesting consequences… for some definition of "interesting")

Signals

- Signals are one way for userspace to interact with processes and threads
- Can also be used for IPC, i.e. can be used to communicate between processes (or threads)

- Signals can be targeted at a whole thread-group or at a single thread.

Killing the wrong target

You really wanted to kill that pesky little resource hungry bastard
but instead you killed a really essential little sucker.

PID allocation

- PIDs are allocated cyclically
  ```
  alloc_pid()
  -> idr_alloc_cyclic()
  ```
- The standard PID limit is 32768 but can be up to 2^22

PID recycling

- PID recycling is built into the allocation algorithm
- on high pressure systems it's rather easy to operate on the wrong process…

- possible mitigations:
  - make PID allocation algorithm random
  - bump limit to 2^22
  - introduce UUIDs

- "but either way, I don't like correctness guarantees based on timing, especially if they might affect security"

pidfds

- file descriptors:
  - handle on a file {file, directory, device, etc…}
  - private, stable reference
- skipping over a few details, an fd references a `struct file` in the kernel
  - multiple fds can refer to the same `struct file`
  - multiple `struct file` can refer to the same inode

```
int fd1 = open(something);
int fd2 = dup(fd1);
int fd3 = open(something);
```

pidfds

- every `struct file` comes with a member called `f_op` which refers to a `struct file_operations`
  - f_op essentially defines what "type" a file is
    standard pattern in kernel comparing file operations to determine type of file, e.g.
    `file->f_op == &pidfd_fops`
  - struct file also comes with another member called `private_data`
    - used to stash file specific data, e.g. pidfds stash away a `struct pid` (the kernel's abstraction on top of tasks

- `struct pid` is the kernel-internal stable handle on a process

pidfds: A New Hope^wAPI

- building a new process management API
- at some point you can manage processes completely without using PIDs

- Linux 5.1 *(released)*
  - `pidfd_send_signal()`

- Linux 5.2 *(to be released tomorrow or next Sunday)*
  - `CLONE_PIDFD`

- Linux 5.3 *(upcoming merge-window)*
  - `pidfd_open()`
  - polling support

- Linux 5.4
  - `WPIDFD`

Can we do more? Maybe… ;)

pidfds: User? User!

- systemd

- Android's LMKD
  Google is already backporting the complete pidfd work
  to all of their LTS kernels:
  - 4.9: https://android-review.googlesource.com/q/topic:%22pidfd+polling+support+4.9+backport%22
  - 4.14: https://android-review.googlesource.com/q/topic:%22pidfd+polling+support+4.14+backport%22
  - 4.19: https://android-review.googlesource.com/q/topic:%22pidfd+polling+support+4.19+backport%22

- LXC and LXD

pidfds: A Linux invention?

- No!
- Solaris and the BSDs have them too

pidfds: Process file descriptors on Linux

Tuebix, 2019
Tübingen, Germany

Christian Brauner
Senior Engineer
Canonical Ltd.

christian@brauner.io
christian.brauner@ubuntu.com
@brau_ner
https://brauner.io/
https://people.kernel.org/brauner