FROM I/O PORTS TO PROCESS MANAGEMENT

3rd Edition
Covers Version 2.6

Understanding the LINUX KERNEL

O'REILLY®

DANIEL P. BOVET & MARCO CESATI

"fork-bomb" processes, the function considers the amount of memory eaten by all children owned by the parent, too.)

- Killing the victim should lose a small amount of work—it is not a good idea to kill a batch process that has been working for hours or days.
- The victim should be a low static priority process—the users tend to assign lower priorities to less important processes.
- The victim should not be a process with root privileges—they usually perform important tasks.
- The victim should not directly access hardware devices (such as the X Window server), because the hardware could be left in an unpredictable state.
- The victim cannot be *swapper* (process 0), *init* (process 1), or any other kernel thread.

The `select_bad_process()` function scans every process in the system, uses an empirical formula to compute from the above rules a value that denotes how good selecting that process is, and returns the process descriptor address of the "best" candidate for eviction. Then, the `out_of_memory()` function invokes `oom_kill_process()` to send a deadly signal—usually `SIGKILL`; see Chapter 11—either to a child of that process or, if this is not possible, to the process itself. The `oom_kill_process()` function also kills all clones that share the same memory descriptor with the selected victim.

The Swap Token

As you might have realized while reading this chapter, the Linux VM subsystem—and particularly the PFRA—is so complex a piece of code that is quite hard to predict its behavior with an arbitrary workload. There are cases, moreover, in which the VM subsystem exhibits pathological behaviors. An example is the so-called *swap thrashing* phenomenon: essentially, when the system is short of free memory, the PFRA tries hard to free memory by writing pages to disk and stealing the underlying page frames from some processes; at the same time, however, these processes want to proceed with their executions, hence they try hard to access their pages. As a consequence, the kernel assigns to the processes the page frames just freed by the PFRA and reads their contents from disk. The net result is that pages are continuously written to and read back from the disk; most of the time is spent accessing the disk, hence no process makes substantial progress towards its termination.

To mitigate the likelihood of swap thrashing, a technique proposed by Jiang and Zhang in 2004 has been implemented in the kernel version 2.6.9: essentially, a so-called *swap token* is assigned to a single process in the system; the token exempts the process from the page frame reclaiming, so the process can make substantial progress and, hopefully, terminate even when memory is scarce.

The swap token is implemented as a `swap_token_mm` memory descriptor pointer. When a process owns the swap token, `swap_token_mm` is set to the address of the process's memory descriptor.

Immunity from page frame reclaiming is granted in an elegant and simple way. As we have seen in the section "The Least Recently Used (LRU) Lists," a page is moved from the active to the inactive list only if it was not recently referenced. The check is done by the page_referenced() function, which honors the swap token and returns 1 (referenced) if the page belongs to a memory region of the process that owns the swap token. Actually, in a couple of cases the swap token is not considered: when the PFRA is executing on behalf of the process that owns the swap token, and when the PFRA has reached the hardest priority level in page frame reclaiming (level 0).

The grab_swap_token() function determines whether the swap token should be assigned to the current process. It is invoked on each major page fault, namely on just two occasions:

- When the filemap_nopage() function discovers that the required page is not in the page cache (see the section "Demand Paging for Memory Mapping" in Chapter 16).
- When the do_swap_page() function has read a new page from a swap area (see the section "Swapping in Pages" later in this chapter).

The grab_swap_token() function makes some checks before assigning the token. In particular, the token is granted if all of the following conditions hold:

- At least two seconds have elapsed since the last invocation of grab_swap_token().
- The current token-holding process has not raised a major page fault since the last execution of grab_swap_token(), or has been holding the token since at least swap_token_default_timeout ticks.
- The swap token has not been recently assigned to the current process.

The token holding time should ideally be rather long, even in the order of minutes, because the goal is to allow a process to finish its execution. In Linux 2.6.11 the token holding time is set by default to a very low value, namely one tick. However, the system administrator can tune the value of the swap_token_default_timeout variable by writing in the */proc/sys/vm/swap_token_default_timeout* file or by issuing a proper sysctl() system call.

When a process is killed, the kernel checks whether that process was holding the swap token and, if so, releases it; this is done by the mmput() function (see the section "The Memory Descriptor" in Chapter 9).

Swapping

Swapping has been introduced to offer a backup on disk for unmapped pages. We know from the previous discussion that there are three kinds of pages that must be handled by the swapping subsystem: