INFO-0940: Operating Systems

Project 2: Adding system calls - Report

Goffart Maxime
180521

Joris Olivier
182113

Academic year 2020 - 2021

1 Implementation

First, we had to make the kernel aware of the 2 new system calls. Thus, we had to modify to existing files: `arch/x86/entry/syscalls/syscall_32.tlb` and `include/linux/syscall.h`.

Afterwards, we had to implement the 2 new system calls. In order to be able to retrieve the files easily, we decided to put all our new files inside a folder `custom/` at the root of the source code. This folder contains 1 file per system call and 2 files (module & header) that allow to fetch processes that have a given name and store the pages that are read-only and present in memory.

When calling `getnbpages`, we are calling the `find_proccesses` function[1] that will fetch the processes that have a name matching the given one and will store the pages that are read-only and present in memory of these processes. The return of this function is a structure `task_struct_list`[2].
As explained in section 2, we need to check all the `task_struct` inside the kernel memory, in order to find the processes that have a name matching the given one, because each one represents a process. When we have the list of the processes that we are considering, we need to go through all their `vm_area_struct` because each one of these represents multiple pages. For each page, we need to check if it is read-only and present in memory. If it is the case, we are storing it in order to be able to set the bitmap in the `getreadpages` system call.
The structure `task_struct_list` is a linked list of `task_struct_list_node` where each one of the latter represents a process that have a name equal to the given one. Each `task_struct_list_node` contains a linked list of `page_list` which represents a read-only page that is present in memory.
In order to be able to access the generated structure when calling `getreadpages`, we are storing a pointer to it inside the `mm_struct` of the process with pid equal to 1 (systemd).

When calling `getreadpages`, we are fetching the pointer stored inside the `mm_struct` of process with pid 1 and checking if the process's name on which getnbpages was called is the same as the one provided. Then, we have to walk across the linked lists that were stored. The top-level linked list store one linked list per process with the embedded linked list storing the pages that read-only and present in memory. Each node inside the embedded linked list store the index of the page. Thus, by walking through all the linked lists, we are able to fetch all the indexes that should be set to 1 inside the bitmap.

2 How the memory of a process is managed by the kernel ?

Processes are implemented in the kernel as instances of `task_struct`. This struct contains a *mm* field which is an instance of `mm_struct` that represents a summary of the process memory. This *mm* field contains a *mmap* field that is an instance of `vm_area_struct` that represents a memory area. This *mmap* field contains 3 fields *vm_start*, *vm_end*, and *vm_next*. The first one represents the logical address corresponding to the first address within the virtual memory area, the second one represents the first address outside the virtual memory area, and the third one is a reference to another `vm_area_struct` which contains higher segment begin and end addresses.

[1] Implemented in custom/get_processes.c.
[2] Defined in custom/get_processes.h.

To translate a logical address to a physical address, a page table is used. This table allows to obtain the physical address corresponding to a given logical address using multi-level paging in order to reduce the size of the table that is stored in physical memory. Moreover, the offset of the logical address is retained according to the page size and only the first bits of the logical address are used to look at the mapping in the table in order to reduce the size of the entries in the table. Some entries do not refer to any physical page: this means that they have the present flag clear. This could be because their contents have been swapped out or because they never have been touched.

This management of the process memory by the kernel is represented on the Figure 1.

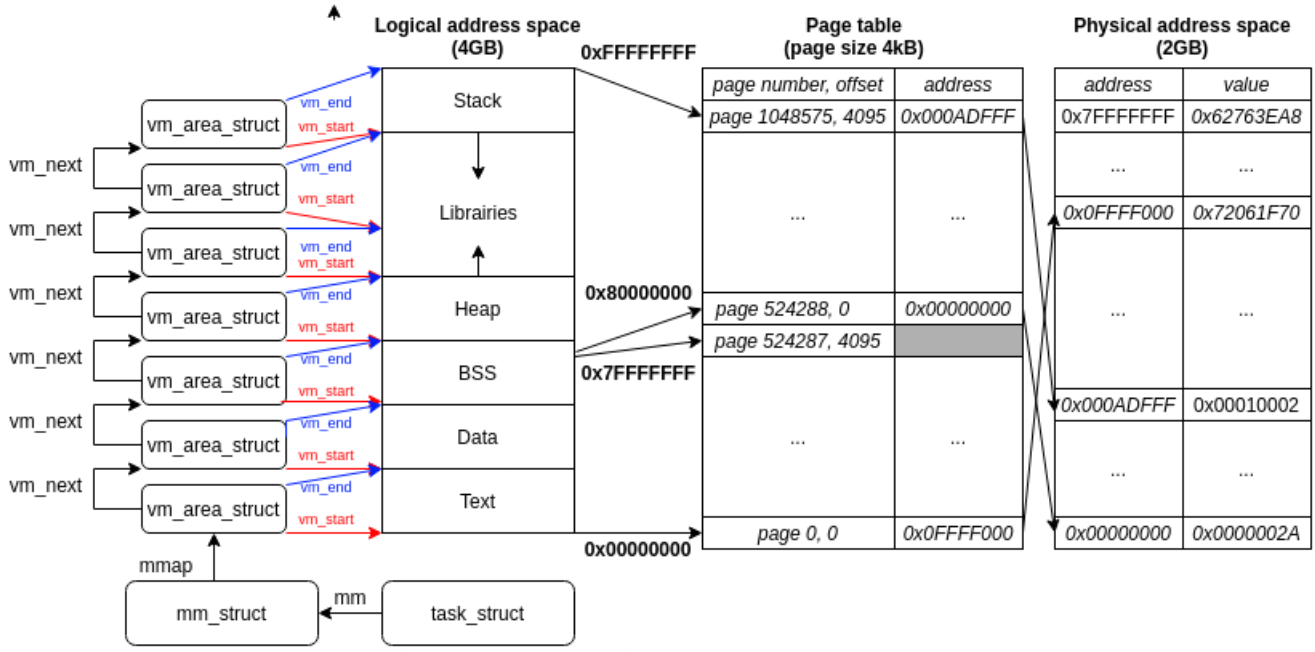

Figure 1: Management of the process memory by the kernel.