Team 08
s192865 KATOUZIAN Pouria
s201354 GERARD Manon

INFO0940-1 - Project 2 (2024)

Overview of your approach

We used the template provided in the 4^{th} tutorial session for our code. This template facilitated the initialization and exit of the module with the associated pseudo file system and associated functions to call for user interaction.

During module initialization in `module_start`, a data structure is populated with information about all currently running process information on the system. We opted for a hash list structure since we have to store the information linked to a set of process(es) with the same name. Therefore, using a hash list which uses the name as a key is the best approach. It's worth noting that the hash list in Linux differs from conventional implementations, maintaining its original size and mapping different nodes to the same hash in case of collisions. Despite these differences, it remains an efficient approach. This hash list is eventually freed in the `module_stop` function.

To populate this structure, we used the `populate_process_memory_hashlist` function. This function accesses all the processes using `for_each_process`, providing a *task_struct*. For the processes with pages, we check if there is an existing set of processes with the same name or if we need to add a new entry to the hash list. In both cases, we add the process pid to a list within the structure that contains all the pids for the set of processes. We implemented the `count_pages` function to fill in the field related to pages. For the total number of pages, we used the *total_vm* field in the memory descriptor (`mm_struct`) of the `task_struct`. We then accessed each virtual memory area, iterating from start to end by accessing one page at a time. For each page, we traverse the different page tables (pgd, p4d, pud, pmd and pte) by using the `pXX_offset` functions. These provide the virtual address of the required page table entry containing the related physical address of either the page table or the final physical page. We ensure these entries are not empty and that they are suitable. After this, we checked that the page was present in RAM using `pte_present`. If present, we increment the valid page counter and then check if the page is readable. If readable, we access it with `pte_page` and hash the page. Within a process set, we maintained a `count_hashlist` to store for each hashed page, the page and a counter of identical pages. When we get the hash of the page, we can therefore identify if it is an identical page by looking at `pages_identical` which compares the content. While populating this hash list, we incremented the group counter whenever a new group is created.

Finally, we handle user commands in the `write_msg` function, where the data is copied from the user space to kernel space. Then we identified to which command it corresponds, if it exists. For the "RESET" command, we free the hash list storing the processes and repopulate it. For "ALL" and "FILTER" commands, we use two functions: `calculate_buffer_size` and `generate_process_info_message`. These functions compute the message size for a set of processes and fill a buffer with the message for the set of processes. The "ALL" command processes the entire hash list, computing the buffer size and generating the message for all the sets of processes. The "FILTER" command is a simplified version that finds the specified process in the hash list, then computes the size of the buffer and fills the buffer similarly. Finally, for the "DEL" command, we locate the process in the hash list and free it.

Feedback

- Difficulty: The difficulty of this project lies in identifying the appropriate functions compatible with the version of the kernel we are working with, as well as understanding the kernel's behavior in response to errors and warnings.
- Amount of work: It took us two weeks to fully understand and code the implementation of this kernel module.
- Other: Implementing a kernel module is a great idea for a project, but it would be beneficial to have more examples and explore other possibilities with kernels. It could be a good idea to publish projects from previous years just to see how other tasks could be implemented in a kernel, such as making modifications to specific system calls or functions etc.