INF-2201: Computer architecture and organization

Assignment 5

Tarek Lein

UiT id: Tle044@uit.no

GitHub users: TrakeLean

May 1, 2023

1 Introduction

In this project we were tasked with setting up a virtual memory map for our ongoing operating system, with this virtual memory mapping we also had to implement paging. With a virtual memory setup you trick the processes/threads to believing they have a large block of contiguous address space, but in reality the data is split up and mapped.

2 Technical Background

2.1 Virtual Memory Mapping

Virtual memory mapping allows us to "trick" the computer into thinking that it has more memory than it really does, when in reality some of its data might not even be loaded but on a hard-drive. This is done by creating a mapping between the logical addresses the programs use with the physical addresses in the memory. The program might think it's located between 0x0 and 0x7000, but in reality it's split up into pages which then are mapped to the physical memory, as you can see in figure 1.

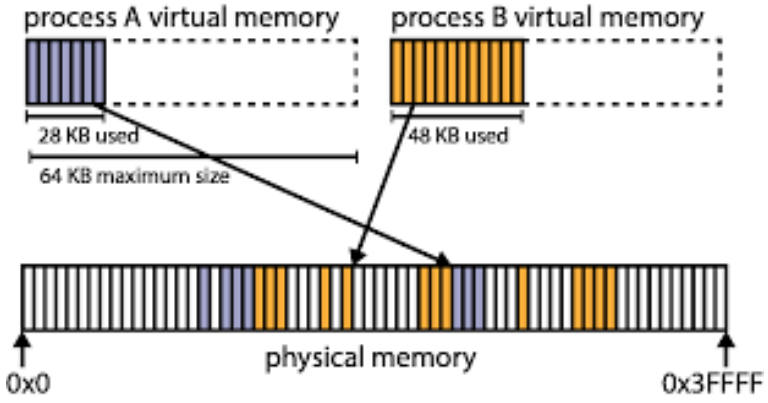

Figure 1: VMM visualised

2.2 Paging

Paging is a memory management technique used by operating systems, we use this to optimize memory usage. The memory is split up into pages which are typical in the sizes of 4KB or 8KB, they were 4KB in our project. We used multilevel paging with 2 levels, the first level would be the page directory, and the second level would be our page table, which then maps to data. Navigating through these pages can be quite complex as you can see in figure 2, you have to take use of multiple factors to compute your way towards what you're looking for.

When a program attempts to access a page that isn't currently in physical memory, the operating system performs a page fault and brings the required page from disk into memory. This allows the program to continue execution as if the data was always in memory. If there isn't enough free memory to accommodate the requested page, the operating system may choose to swap out another page from memory to make space for the new page.

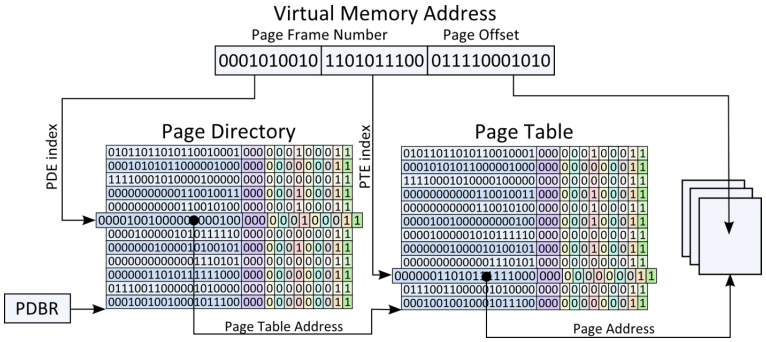

Figure 2: Paging visualised

2.3 Free List

We had to implement some sort of structure to be able to keep track of what pages that are free or used, this is where a simple free list comes in. It's a simple doubly linked list with a few extra variables to keep track of its address, index and if it is pinned.

3 Design

There are a few key functions that had to be implemented

- Init_memory
  This is where there program goes at the very beginning to setup it's kernel and make everything ready for virtual memory mapping and paging
- Make_common_map
  A common map is needed for all the processes and threads so they have access to the things they need, like the video memory so they are able to write to the screen. We do not want user level processes to have access to everything though, which is why we also dynamically set processes/threads user level her with the help of our "table_map_present" function which in simple words just sets a bit to 0 or 1
- Setup_page_table
  Yet another simple function that check if we are working with a thread or a process. Threads will just get the kernel directory as page directory and processes would create its own, which then uses the "make_common_map" function with the user level set to 1.
- Page_fault_handler
  Lastly we have the page fault handler which is here to do all the work. Here we kind of have to listen to the operating system and have it tell us exactly what it wants through error codes. This is where we setup tables and pages for our processes.

4 Implementation

There are a few ways of handling page faults which comes with their pros and cons. The way that we choose to implement it came down to that we really wanted to see what happened when the operating system counters a page fault.

When a process starts up, it only receives a directory with user access to the kernel. When it tries to continue it suddenly gets interrupted and end up in the page fault handler, at first this happens because there are no tables within the directory, so it simply tells us "Table is not present" and our job is then to create a table and answer with "It is now present, go ahead". The process then tries to start again, but the same thing happens only this time it found a table but not a page, so once again it says "Page is not present", so we allocate a page and fill that page with chunks of the process data, each time it gets a page fail we write 4096kb to the page requested.

Now the reason for doing it this way is that we want the operating system to show us exactly what it wants and when it wants it even though we already know the theory behind it. It is quite interesting seeing the theory come through in our program. The other way of setting up a process could be to give it a table and calculate how many pages it needs, then write to those pages all within the "Setup page table" function, but this approach handicaps the page fault handler, since you remove a lot of its purpose while also having to calculate your way to the pages if they fault.

We've also created a free list that keeps track of all the available pages the operating system has access too, which is needed later on when you want to switch out pages. Switching out pages is necessary if you want to have an operating system that doesn't crash after the resources are used up. This is where swapping enters the picture, swapping can occur when all the pages are used, but you need more space, it takes non pinned pages and write them to a deeper level of memory, like your hard disk to free up space in the RAM. This could slow down everything on the operating system since you could end up doing a lot of reads and writes from memory because every time the process wants to run again you'll have to bring that process back and write another one down in memory.

Pinning pages is also important to understand when all of this is taking place. Some pages are more important than others, like the pages the kernel uses. What would happen if a process swapped out the kernel pages and inserted its own? Well the operating system would lose its kernel and everything would crash, which is why you'll want to pin pages and then always check if pages are pinned before you swap them out, then don't swap if they're pinned. The way we choose which pages to pin and not was to check their user bit, if it was 0 it ment that it belonged to the kernel, which as we talked about above, always needs to be loaded.

5 Discussion

We're letting the process enter page fault for needing a table which means it goes through an extra cycle, which will have more "dead" time rather than if we set it up in the "setup page table" function, but in return we get to utilize the "page fault handler" function more, which we believe is more fitting considering this is a school project where we're trying to learn all about page faults, so a few extra cycles with more time spent is a solid trade off for better learning, if this was a live project ment to be distributed for people to use, we would probably have optimized this process a bit more.

There is a bug where if you try to load process 4 it won't show up on screen, as if it doesn't have access to the video memory, which we do not understand why. Process 4 speaks to process 3 through a mailbox, so you would think that when we load them both it wouldn't work since process 4 isn't showing, but to our surprise process 3 shows up and starts counting, which means that it is able to speak to process 4 and process 4 is able to receive and speak back.

6 Conclusion

Threads run perfectly fine, almost all the processes run fine and are able to load up if there are available pages, which we are satisfied with. The program crashes if you try to load something beyond the available pages since we haven't had time to implement swapping, it would be fun to see everything working and experience the operating system slow down when it has to read and write from the hard drive, but you can't always get everything right.

7 Sources

References

[1] https://wiki.osdev.org/Memory_management

[2] https://wiki.osdev.org/Paging

[3] https://wiki.osdev.org/Memory_Management_Unit

[4] https://wiki.osdev.org/TLB