# ICSI412 Operating Systems – Project 6 – Virtual Memory

## Overview

When our computer doesn’t have enough memory to fulfill all of the requests from userland programs, it uses disk space instead. While this is slower than running from memory, it does allow users to perform their tasks. The operating system can try to do a number of things to minimize the time-cost of this. We will not be modeling those.

This extends and changes the work from the Paging project. When a process attempts to allocate memory and there is no physical page available, we will now choose a random process (including our own!) and pick a random page from their page mapping. We will write that page out to disk (if necessary) and disassociate the physical page from its process. We will then associate that physical page with the current process. If the current process has written the page to disk, we will load the data into the physical page.

**Task 1** – Create a file

In your memory manager, get access to the fakefilesystem (ffs) object (I had to make some things public). On creation of the memory manager, use ffs to make a file (swapfile was the name I used). You will need an integer to track the page number (remember, a page = 1024 bytes) that is the “next page to write out”. We won’t be trying to reuse pages on the disk file.

**Task 2** – Enhance our in-process tracking of memory

Initially, we used an array of int in our PCB to map virtual->physical address. Now we need to track more information. Create a new object (VirtualToPhysicalMapping) with public members for “isDirty” (that is, this page has been written since it was last on disk (if ever)), physical page number and on disk page number. Change the int[] in the PCB to an array of 1024 of these classes. Fix the related code issues.

**Task 3** – Make allocation cheaper

Previously, when the user called sbrk, we did all of the virtual->physical mapping. We are going to remove that, since we now will have the ability to deal with physical pages that don’t exist. Instead, populate the VirtualToPhysicalMapping with a -1 physical page and a -1 disk page.

**Task 4** – Fix up accesses

If someone reads or write to memory, we need to check to see if the physical page exists before we do the array reference. If it does not exist, we need to fix it. We also need to consider the possibility that a user tries to access out of their sbrk region. I reused the RescheduleException for this (since we will crash the process anyway). This will cause you to need to add a bunch of “throws RescheduleException” clauses.

I created a new method:

Int VirtualToPhysicalMapping(int virtual) // return the physical page number

This allowed me to reuse the logic between ReadMemory and WriteMemory. Look up the VirtualToPhysicalMapping in the process. If the physical page is -1, then you need one. Pick a random process (I added a method to PriorityScheduler for this) that has memory. Then pick a random page with physical memory attached. Write that memory to disk (if necessary), unmap the page, load the data from the current process (if disk block is not -1) or zero out the page (so that you don’t leave data behind) and then assign it to be the physical page for the current process.

**Test your code!**

Test reading and writing (to make sure that you get the same value). Test extending your memory. Test trying to access memory that you shouldn’t be able to and make sure that your process is killed. Create a process that allocates and touches all of the physical pages. Make sure that your other processes still run.

***You must submit buildable .java files for credit.***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Swap file | None (0) |  |  | Is created using FFS (5) |
| Page object | None (0) | Is now a class (3) |  | Is a class with data items described (5) |
| Lazy Physical Page Allocation | Still allocating all (0) |  |  | Allocated on demand (10) |
| Out of bounds memory access kills process appropriately | None (0) |  |  | Throws exception (10) |
| Physical page access in method (not copy/pasted) | Copy/Paste (0) |  |  | In Method (5) |
| Lazy allocation works | None – all memory access fails (0) |  |  | Free physical pages are mapped (10) |
| Swapping uses a random process | No swapping (0) |  | Swapping picks a fixed process (5) | Random process used (10) |
| Swapping uses a random page | Fixed or first page (0) |  |  | Random page (5) |
| Swapping writes out dirty pages | No(0) |  | Writes dirty and clean pages (5) | Writes only dirty pages (10) |
| WriteMemory sets dirty flag | No (0) |  |  | Yes (5) |
| Physical pages are cleared when not loading | No (0) |  |  | Yes (10) |
| Dirty flag is reset on swap | No (0) |  |  | Yes (5) |
| Physical pages are loaded when they were previously written out | No(0) |  | Yes, but wrong (5) | Read correctly (10) |