# ICSI412 Operating Systems – Project 5 - Paging

## Overview

In operating systems, we manage three key sets of resources – processor, devices, and memory. In paging, we start addressing the memory. Normally, a significant amount of the work is done in hardware. Our simulation is a little … unusual because we don’t have hardware.

We also have a new situation to handle – what if a process does something that can’t be allowed, like accessing memory it doesn’t have permissions to access? We are choosing to kill that process. Unfortunately, this is a little complex in Java. We will need to introduce some exceptions and exception handling through our code to make that work.

A modern CPU has variable page sizes, 4KB or bigger. To keep our model reasonable for Java, we will use 1KB pages and 1MB (1024 pages) of memory.

**Task 1** – bring in the new/changed interfaces and RescheduleException and make code changes to allow your code to compile.

**Task 2** – add a page array to KernelandProcess/PCB

This is a simple array of 1024 integers. We will use this to map physical pages to our virtual address space. This is a little bit of a wasteful way to do this. Create/update the constructor so that all integers default to -1.

**Task 3** – Create a MemoryManagement class that implements MemoryInterface

Start by making storage for the physical pages – a 2D array of byte. This will allow us to access it:

memory[page][offset]

Next, we have to keep track of which pages are in use and which are free. Java gives us a very convenient way to do this – the BitSet class.

Finally – in hardware, looking in memory (i.e. the page array we set up above) is too slow for every memory access. So in hardware, we have the TLB (translation lookaside buffer) – a stashed copy of some virtual🡪physical mappings. We will simulate that here with a single pair of ints – tlbVirtual and tlbPhysical. We need to make sure to set these to -1 initially **and** on every process switch.

**Task 4** – Finish MemoryManagement

With the infrastructure in place, you should be able to write ReadMemory and WriteMemory, making sure to use the tlb if appropriate.

Sbrk is a Linux function – it adds the requested amount of space to the end of the running processes’ address space. It is used by the operating system when a new process is created to make stack and heap space, but it can be called from userland to add more space. It returns the address of the **old** top of the memory space. So it will return 0 the first time you call it, then “amount” the second time.

You will also need some other methods:  
Some method to free memory when a process is deleted.

Some method to invalidate the TLB.

**Task 5** – integrate with OS and the scheduler

OS should pass through ReadMemory/WriteMemory/sbrk like we did for devices. When a RescheduleException is thrown, you should tell the scheduler to delete the current process. You may need to make some members public or provide accessors (I did).

Call your function to invalidate the TLB on process change. Make sure that your DeleteProcess handles deleting the **currently running** process. Also remember that deleted processes need to have their memory freed.

**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.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Page table in PCB | None (0) |  |  | Exists and is right sized (5) |
| Memory Management – Memory array | None (0) |  |  | Exists and is right sized (5) |
| Memory Management – tlb | None (0) |  |  | Exists and is right (5) |
| Memory Management – free list | None (0) |  |  | Exists and is right sized (5) |
| Memory Management – ReadMemory | None (0) | Gets virtual page, gets physical page (5) | Gets virtual page, gets physical page, checks TLB (10) | Gets virtual page, gets physical page, checks TLB and returns data correctly (15) |
| Memory Management – WriteMemory | None (0) | Gets virtual page, gets physical page (5) | Gets virtual page, gets physical page, checks TLB (10) | Gets virtual page, gets physical page, checks TLB and writes data correctly (15) |
| Memory Management – sbrk | None (0) | Finds number of pages to add, adds mapping to the end of existing virtual space (7) | Finds number of pages to add, adds mapping to the end of existing virtual space, marks physical pages as in use (13) | Finds number of pages to add, adds mapping to the end of existing virtual space, marks physical pages as in use, returns correct value (20) |
| MemoryManagement – FreeMemory | None (0) |  |  | Memory is freed when a process is deleted (10) |
| TLB Cleared | None (0) |  |  | TLB cleared when process switched (10) |
| Process killed when accessing out of bounds | None (0) |  | Process is killed (5) | Process is killed and memory is freed (10) |