CS61 Scribe Notes - Oct. 30, 2014
Hannah Blumberg, Ryan Grossman, Kelwen Peng

<u>Process Isolation: What features? How? Why?</u>

Practically, process isolation involves separating each process in the unique context of several different "resources" the machine has at its disposal. This isolation is done at two levels within each of these contexts: the level of the hardware and the level of the Kernel. These are closely interlinked with each other as seen in the descriptions below.

- Resource: CPU time
    - What does isolation mean in terms of CPU time?
        - Every run-able process eventually runs
    - Hardware support:
        - periodically interrupt
    - Kernel support:
        - (1) needs to catch the interrupt
        - (2) needs to schedule the next process
- Resource: Interrupts
    - What does isolation mean in terms of interrupts?
        - One process's interrupt can't affect another process
        - *Can a process prevent an interrupt when it's running?*
            - No, otherwise it'll prevent process isolation
        - Process cannot disable interrupts or change interrupt handlers. *K*
            - *K* mean that kernels are allowed to do such things, but a smart kernel would never change timer interrupts (recall from last lecture when we allowed our program to turn off the timer interrupt)
    - Hardware support:
        - hardware has privilege bits
        - processor refuses to execute some processes when in unprivileged mode; (`cli, lidt`) are dangerous
        - hardware needs a correct definition and understanding of privilege bits (W*ho defines this?* Intel and other processor corporations make this decision)
    - Kernel support:
        - ensures that processes run in unprivileged mode
- Resource: Registers
    - What does isolation mean in terms of registers?
        - One process changing the register shouldn't affect other registers
        - Processes behaves as if it owns all registers
        - Registers of one process and registers of another process are disjoint and aren't related
    - Hardware support:

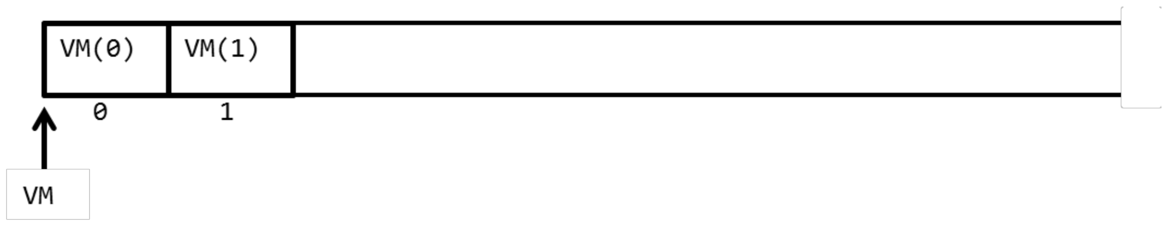
        - Save registers during exceptional control flow (refer to exceptional control transfer part of the textbook)
    - Kernel support:
        - provide each process with its own set of registers
        - *How is this done?*
            - By saving the registers in memory
        - Process descriptor contains process's registers
- Resource: Memory
    - What does isolation mean in terms of memory?
        - No access to memory of the kernel and other processes
        - But under the guise that it has a lot of memory
        - Processes can't read or write kernel memory or other process memory
    - Hardware support:
        - need a mechanism that prevents processes from reading or writing into other processes or kernel's memory → this is called virtual memory
        - Indirection layer that prevents illegal accesses by unprivileged code
            - Indirection: example is a pointer; rather than looking directly at a data, we look for the place that gives us the location of where the data is
    - Kernel support:
        - sets up the layer/virtual memory that isolates processes (memory accesses)

<u>Virtual Memory</u>

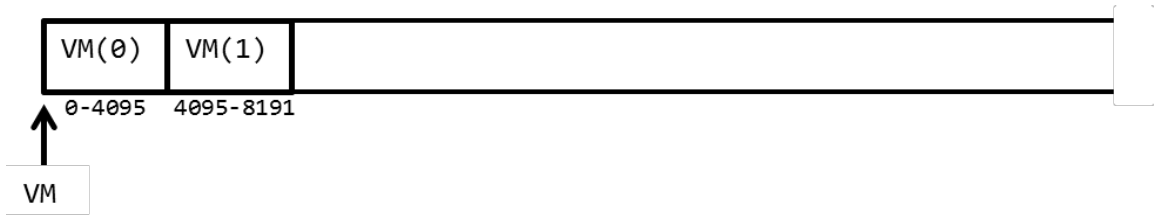
- Virtual Memory, abstractly, is a function that provides a mapping to a
- Physical address: example is a home address
- VM makes everything be like PO boxes
- Hardware is needed for virtual memory
    - if the kernel executes and oversees virtual memory, processes slow down

<u>Potential VM Functions</u>
begin with a struct with a `boolean isfault` and `uintptr_t pa;`
(all functions can be found in `~/cs61-lectures/l16/fakevm.c`)

0. Byte-indexed array
    a. Have an enormous array with special registers that only the kernel can change (in the same way only the kernel can only change the privilege level)
    b. Index 0 is VM(0), index 1 is VM(1), etc…
    c. Problems: array is WAY too big: 2^{35} (uses more bytes of memory than we can address)

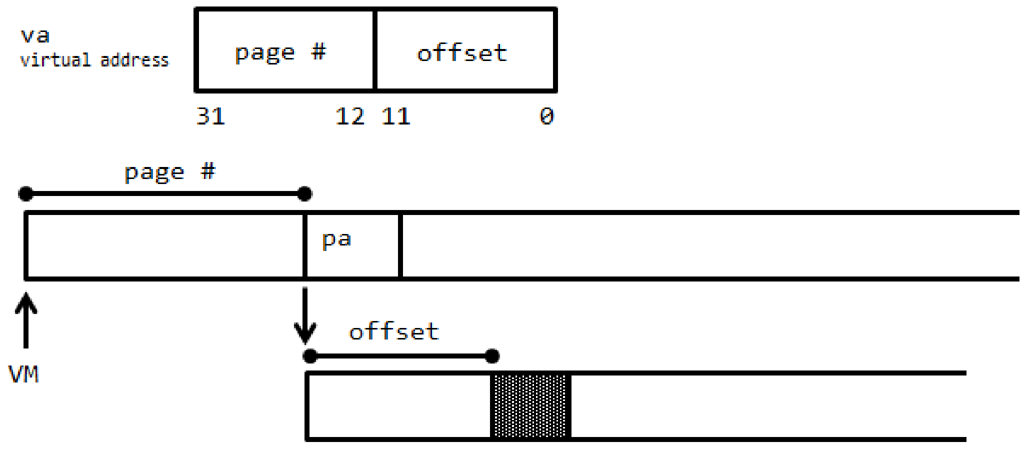


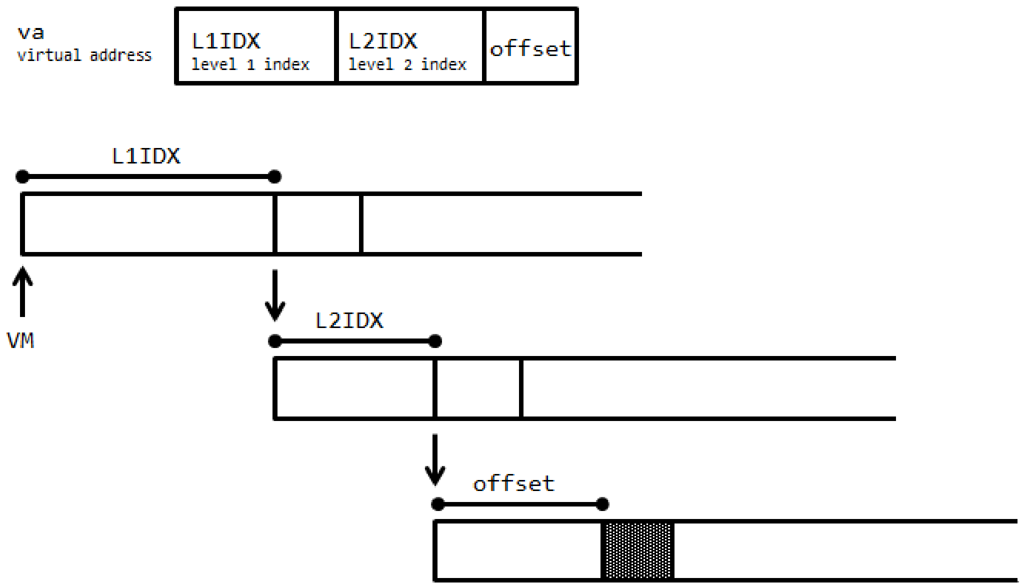
1. Paged-indexed array
    a. Instead of every byte, we divide the memory into units called PAGES
    b. All address in a page map similarly
    c. In example: pagesize = 4096 (it should be a power of 2 and equal or greater than 64 because 64 bytes is a the size of a cache line)
        i. 4096 is a good choice because it's the pagesize of an x86 processor
    d. This decreases the size of the array needed to 10^{22}



2. Page-indexed array with integrated `isfault` bit
    a. (we skipped over this function in class, but the idea is that the fault bit should not take an entire byte)

3. Radix tree
    a. Each level has 2^{10} children
    b. This greatly decreases the amount of space required to store these pages
    c. In the best case, the array has a size of size 2^{13}; however, it does not improve the worst case scenario (still has a size of 10^{22})




va
virtual address
L1IDX
level 1 index
L2IDX
level 2 index
offset
L1IDX
VM
L2IDX
offset