Cool Things VM Provides

• Learning Objectives
  • Explain how virtual memory provides abstractions such as:
    • Contiguous allocation of memory
    • Processes
    • Fork
    • Mmap
  • Explain how virtual memory enables process isolation using:
    • Per-process page tables
    • Protection bits in PTEs
    • Faults
    • Validating user addresses (avoid the confused deputy problem)
Pointer Arithmetic (It’s all Lies!)

• Recall how nicely we can calculate the addresses of data.
• For example
  
  ```c
  int array[10];
  ```
• Let’s say that this array is allocated at address 0x1FFFC.
• What is &array[6]?
• We know that C allocates this array contiguously.
• BUT – it is only contiguous in the virtual address space.
• Need it be contiguous in physical memory?
No!

- We’ve learned that virtual pages map individually to physical pages, so your address space might look like this:
Moving on: Address Spaces

At the very beginning of the semester, we introduced an address space. In the context of virtual memory, what exactly is an address space?
Next up: fork

- When we introduced fork, we said that it “creates a new process with its own address space.”
- Now that we understand virtual memory – what exactly does that mean?
And … `mmap`

- In assignment 3, we introduced `mmap` and we saw that it:
  - Allows us to pretend that a file’s data is directly accessible in a process’s address space, and
  - Allows us to share memory between two processes.

- In the context of VM, what does this mean?
Let’s Talk About Process Isolation

• How does virtual memory protect processes from one another and the kernel from user processes?
• Protect processes from one another:

• Protect kernel from processes:
Let’s Talk About Process Isolation

• How does virtual memory protect processes from one another and the kernel from user processes?
• Protect processes from one another:
  • Each has its own page table.
  • The operating system must ensure that a process’s pages are not accessible from another process’s page table (unless they are intended to be share).
• Protect kernel from processes:
Let’s Talk About Process Isolation

• How does virtual memory protect processes from one another and the kernel from user processes?
• Protect processes from one another:

• Protect kernel from processes:
  • The kernel (OS) runs in privileged mode
  • The kernel’s memory is marked as being accessible only to code that runs in privileged mode.
Bad Processes

• If the OS sets everything up correctly, when a process tries to violate process isolation:
  • Touch kernel memory
  • Touch another process’s memory
  • Write hardware registers it’s not supposed to

• What happens?
Bad Processes

• If the OS sets everything up correctly, when a process tries to violate process isolation:
  • Touch kernel memory
  • Touch another process’s memory
  • Write hardware registers it’s not supposed to

• What happens?
  • The processor generates a fault.
  • When the processor takes a fault, the OS gains control.
  • The OS could do whatever it wants:
    • Kill the process
    • Skip the instruction
The Confused Deputy Problem

- When privileged code acts on behalf of unprivileged code and the unprivileged code tricks the privileged code into doing something bad.
- Who is the deputy here?
- How could a process confuse the deputy?
The Confused Deputy Problem

- When privileged code acts on behalf of unprivileged code and the unprivileged code tricks the privileged code into doing something bad.
- Who is the deputy here?
  - The OS
- How could a process confuse the deputy?
  - While a process can’t write into privileged memory, the OS can.
  - What if a process could somehow convince the OS to write something bad into a location that the process cannot write, but the kernel can!?
  - How do we avoid that?
Verifying Process Addresses

- Whenever a process passed an address to the operating system (e.g., a buffer, a string, etc), the operating system must verify that the process has the proper permissions to use the address in the way the kernel is being asked to.

- Examples:
  - Ensure that the address is a valid address in the process’s address space.
  - Ensure that if the process is trying to write the location, the page is writable.
PTEs: The heart of VM protection

- Page table entries are at the heart of the operating system and hardware’s ability to maintain process isolation.

- Recall a virtual address (on 32-bit x86)

<table>
<thead>
<tr>
<th>20 bit (virtual) page number</th>
<th>12-bit offset</th>
</tr>
</thead>
</table>

- The PTE must contain a page number; in addition it contains special bits.

  | 20 bit (physical) page number | 12 bits of metadata |
PTE Meta-Data

• Both L1 and L2 page tables have three critical bits that provide protection:
  • Bit 0: Present Bit
    • 0 indicates that the entire entry is invalid
    • 1 indicates the entry is valid
  • Bit 1: Read/Write Bit
    • 0 indicates that the page (or entire set of pages represented by the referenced L2 page table) is read only.
    • 1 indicates that the page(s) are writable.
  • Bit 2: User/Supervisor bit
    • 0 indicates that the page is accessible only to privileged code.
    • 1 indicates that the page is accessible to unprivileged code.
Wrapping Up

• Virtual memory is a cooperative arrangement between the OS and the hardware.

• Process isolation is provided by proper management of virtual memory.
  • Each process has its own page table
  • Pages in the page table are described by present, read/write, and privilege bits. Setting these bits correctly prevents processes from doing bad things.
  • Whenever a process sends an address to the OS, the OS must ensure that the address is valid for the intended operation.