Virtual Memory

Virtual Address Space
- an abstraction of the physical address space of a program
- programs access memory using virtual addresses
- hardware translates virtual address to physical memory address

Process
- a program execution with a virtual address space
- associated with authenticated user for access control & resource accounting
- running a program with 1 or more threads

MMU
- memory management unit
- the hardware that translates virtual address to physical address
- performs this translation on every memory access by program

Virtual Address Translation
- each program uses the same virtual address, but they map to different physical addresses

Implementing the MMU
- Let’s think of this in the simulator ...
  - introduce a class to simulate the MMU hardware

Multiple Concurrent Program Executions
- So far we have
  - a single program
  - multiple threads
- Allowing threads from different program executions
  - we often have more than one thing we want to do at once(s)
  - threads spend a lot of time blocked, allowing other threads to run
  - but, often there aren’t enough threads in one program to fill all the gaps
- What is a program execution
  - an instance of a program running with its own state stored in memory
  - compiler-assigned addresses for all static memory state (globals, code etc.)
  - security and failure semantics suggest memory isolation for each execution
- But, we have a problem
  - there is only one memory shared by all programs ...

Base and Bounds
- An address space is
  - a single, variable-size, non-expandable chunk of physical memory
  - named by its base physical address and its length
- As a class in the simulator

But, Address Space Use May Be Sparse
- Issue
  - the address space of a program execution is divided into regions
  - for example: code, globals, heap, shared-libraries and stack
  - there are large gaps of unused address space between these regions
- Problem
  - a single base and bounds mapping from virtual to physical addresses
  - means that gaps in virtual address space will waste physical memory
  - this is the Internal Fragmentation problem

Segmentation
- An address space is
  - a set of segments
  - A segment is
    - a single, variable-size, non-expandable chunk of physical memory
    - named by its base virtual address, physical address and length
- Implementation in Simulator

But, Memory Use is Not Known Statically
- Issue
  - segments are not expandable; their size is static
  - some segments such as stack and heap change size dynamically
- Problem
  - segment size is chosen when segment is created
  - too large and internal fragmentation wastes memory
  - too small and stack or heap restricted
- Solution
  - allow segments to expand?

Expand Segments by Adding Segments
- What we know
  - segments should be non-expandable
  - size can be effectively determined statically
- Idea
  - instead of expanding a segment
  - make a new one that is adjacent virtually, but not physically
- Problem
  - oh no! another problem! what is it? why does it occur?

Eliminating External Fragmentation
- The problem with what we are doing is
  - allocating variable size segments leads to external fragmentation of memory
  - this is an inherent problem with variable-size allocation
- What about fixed sized allocation
  - could we make every segment the same size?
  - this eliminates external fragmentation
  - but, if we make segments too big, we’ll get internal fragmentation
  - so, they need to be fairly small and so we’ll have lots of them

Broken Program
- Goal
  - translate any virtual address to a unique physical address (or none)
  - fast and efficient hardware implementation
- Let’s look at a couple of alternatives ...

CPSC 213
Introduction to Computer Systems
Unit 2d
Virtual Memory
Translation with Many Segments

- What is wrong with this approach if there are many segments?
  - New terminology
    - a small, fixed-sized (4-KB) segment
    - a virtual-to-physical translation table
    - physical page frame number
    - offset
    - virtual page number

- Now what?
  - is there another way to locate the segment, when segments are fixed size?

Paging

- Key Idea
  - Page Space is divided into fixed-size segments called pages
  - Page number = virtual address / page size

- Page Table
  - Indexed by virtual page number (vpn)
  - Stores base physical address (actually address / page size (pfn) to save space)
  - Valid flag, because some segment numbers may be unused

- Now what?
  - is there another way to locate the segment, when segments are fixed size?

Address Translation using a Page Table

- Class PageTableEntry {
  - boolean isValid;
  - int pfn;
}

- Class AddressSpace {
  - Segment segment[];
  - int translate (int va, int offset);
    - for (int i = 0; i < segments.length; i++) {
      - int offset = va - segments[i].bound();
      - if (offset > 0 & & offset < segment[i].bounds) {
        - pa = segment[i].basePA + offset;
        - return pa;
      }
    }
    - throw new IllegalAddressException (va);
  }
}

Summary

- Process
  - A program execution
  - A private virtual address space and a set of threads
  - Private address space required for static address allocation and isolation

- Virtual Address Space
  - A mapping from virtual addresses to physical memory addresses
  - Programs use virtual addresses
  - The MMU translates them to physical address used by the memory hardware

- Paging
  - A way to implement address space translation
    - Divide virtual address space into fixed, sized virtual page frames
    - Page table stores base physical address of every virtual page frame
    - Page table is indexed by virtual page frame number
    - Some virtual page frames have no physical page mapping
    - Some of these get data on demand from disk

Hardware Enforced Encapsulation

- Goal
  - Define a set of interfaces (APIs) whose implementations are protected
  - Implementation code and data can only be accessed through interface

- Obstacle
  - Can not use language protection without excluding languages like C

- Use Hardware for Protection
  - Virtual memory already provides a way to protect memory
  - Data in one address space can not be named by thread in another
  - So, we've got the protection implementation part
  - We'll need to add the interface part

The Operating System

- Key Idea
  - Some application data is not in memory
  - Can transfer data from disk to memory, only when needed

- Page Table
  - Only stores entries for pages that are in memory
  - Pages that are only on disk are marked invalid
  - Memory Map
    - A two-dimensional array of every virtual address space

- Page Fault
  - An exception raised by the CPU
  - When a virtual address is invalid
  - An exception is just an interrupt
  - Generated by the CPU not I/O device
  - Page fault handler runs each time a page fault occurs

- Memory Map
  - A two-dimensional array of every virtual address space
  - Divides virtual address space into regions, each mapped to a file

Hardware Vocabulary

- Page
  - A small, fixed-sized (4-KB) segment
- Page Table
  - Virtual-to-physical translation table

Demand Paging

- Virtual vs Physical Memory Size
  - VM can be even larger than available PM with demand paging

- Page Replacement
  - Pages can now be removed from memory transparent to program
  - Replacement algorithm chooses which pages should be resident and swap out others

Hardware and Encapsulation

- Hardware
  - Mode register (user or kernel)
  - Certain instructions only legal in kernel mode
  - Attempting to access a kernel page while in user mode causes a fault
  - Special instructions for switching between user and kernel modes

Inter-Process Communication

- IPC
  - Basic mechanism is send and receive unformatted messages
  - A message is an array of bytes
  - Sender and receiver have named endpoints, e.g., socket or port

Address Space Translation Tradeoffs

- Single, variable-size, non-expandable segment
  - Internal fragmentation of segment due to sparse address use

- Multiple, variable-size, non-expandable segments
  - Internal fragmentation of segments when size isn't known statically
  - External fragmentation of memory because segments are variable size
  - Moving segments would resolve fragmentation, but moving is costly

- Expandable segments
  - Expansion must be physically contiguous, but there may not be room
  - External fragmentation of memory requires moving segments to make room

- Multiple, fixed-size, non-expandable segments
  - Called pages
  - Needs to be small to avoid internal fragmentation, so there are many of them
  - Since there are many, need indexed lookup instead of search