Virtual Memory

Virtual Address Space
- an abstraction of the physical address space of main (i.e., physical) memory
- programs access memory using virtual addresses
- hardware translates virtual address to physical memory addresses

Process
- a program execution with a private 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 ...

But, Address Space Use May Be Sparse
- 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

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

Solution
- currentAddressSpace is a hardware register
- the address space performs virtual-to-physical address translation

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?

But, There May Be No Room to Expand
- Issue
  - segments are contiguous chunks of physical memory
  - a segment can only expand to fill space between it and the next segment

Problem
- there is no guarantee there will be room to expand a segment
- the available memory space is not where we want it (i.e., adjacent to segment)
- this is the External Fragmentation problem

Solution
- maybe some room to expand
- maybe some room to expand
- But, Now We’re Stuck

But, Moving Segments is Expensive
- Issue
  - if there is space in memory to store expanding segment, but not where it is
  - could move expanding segment or other segments to make room
  - external fragmentation is resolved by moving things to consolidate free space

Problem
- moving is possible, but expensive
- to move a segment, all of its data must be copied
- segments are large and memory copying is expensive

Expand Segments by Adding Segments
- What we know
  - segments should be non-expandable
- size can not be effectively determined statically

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

Physical Address Space Collisions
- each program has assumed it is free to read/write anywhere in memory
- doesn’t work when multiple programs run at once

 synchronizes does not solve problem
- It’s a problem through the whole program
- not a short critical section with deliberate use of shared memory to communicate between threads

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
- currentAddressSpace is a hardware register
- the address space performs virtual-to-physical address translation

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 stacks

Problem
- there are large gaps of unused address space between these regions

Solution
- if there is space in memory to store expanding segment, but not where it is
- could move expanding segment or other segments to make room
- external fragmentation is resolved by moving things to consolidate free space
- moving is possible, but expensive
- to move a segment, all of its data must be copied
- segments are large and memory copying is expensive

Expand Segments by Adding Segments
- What we know
  - segments should be non-expandable
- size can not be effectively determined statically

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

Virtual Memory

Virtual Address Space
- an abstraction of the physical address space of main (i.e., physical) memory
- programs access memory using virtual addresses
- hardware translates virtual address to physical memory addresses

Process
- a program execution with a private 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 ...

But, Address Space Use May Be Sparse
- 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

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

Solution
- currentAddressSpace is a hardware register
- the address space performs virtual-to-physical address translation

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?
Translation with Many Segments

‣ What is wrong with this approach if there are many segments?
‣ Now what?
• is there another way to locate the segment, when segments are fixed size?
  
  Now what?
  • is there another way to locate the segment, when segments are fixed size?

Question

Consider this page table

Is 0x43a0 a valid virtual address and if so what is the corresponding physical address?

(E) 0x3a0

Hardware Enforced Encapsulation

‣ Goal
• define a set of interfaces (APIs) whose implementations are protected
• page table entries have protection flag (user or kernel)
• page table entries have protection flag (user or kernel)
• it encapsulates the implementation of these abstractions, including hardware

‣ Obstacle
• can not use language protection without excluding languages like C

Use Hardware for Protection

• virtual memory already provides a way to protect memory
• some virtual page frames have no physical page mapping
• some of these get data on demand from disk

The Operating System

‣ The operating system is
• a C/Assembly program
• implements a set of abstractions for applications

‣ The Operating System’s Address Space
• a part of every application's page table is reserved for the OS
• all code and data of OS is part of every page table (exact copies)

‣ Dual Protection Domains
• each address space splits into application and system protection domain
• CPU can run in one of two modes: user and kernel
• when in user mode, the OS part of virtual memory is inaccessible
• when in kernel mode, all of virtual memory is accessible

The Operating System

‣ The operating system is
• a C/Assembly program
• implements a set of abstractions for applications

‣ The Operating System’s Address Space
• a part of every application’s page table is reserved for the OS
• all code and data of OS is part of every page table (exact copies)

‣ Dual Protection Domains
• each address space splits into application and system protection domain
• CPU can run in one of two modes: user and kernel
• when in user mode, the OS part of virtual memory is inaccessible
• when in kernel mode, all of virtual memory is accessible

Summary

‣ Process
• a program execution
• 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

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
• need to be small to avoid internal fragmentation, so there are many of them
• since there are many, need indexed lookup instead of search