CSE221 Lecture 6

Aronya Baksy

October 2024

1 Virtual Memory Management in VAX

- 32-bit address space for each process: (2 bit to specify which address space, 21 bit virtual page number, 9 bit offset, 512 byte pages)
- The address space is split into 4 regions
  - 00 and 01 describe the program ($P0$) and control ($P1$) regions of the program address space. they grow in opposite directions (this allows $P0$ and $P1$ to grow flexibly without pre-defining sizes)
  - 10 and 11 describe the system-wide address space. all virtual addresses here refer to the same physical page. only the lower address part is used (rest is *reserved*)
- Each region has a page table. Page table entry is 32 bits wide:
  - **valid bit**: check if the entry has valid info
  - 4 bits for **protection** to list permissions for that page
  - **modify** bit to indicate if page has been written to (dirty bit)
  - 5 bits for **OS use**
  - **physical frame number** 21 bits
- Each page table has a base address register and a length register
- P0 and P1 page tables are in system memory (hence registers contain virtual addresses, and they can themselves be paged out)
- Hardware provides a split TLB for caching virtual to physical translations. (for per-process and system pages, only per-process TLB needs to be flushed per context switch)
- user calls OS routines like any procedure. addresses of these routines are in the first pages of the system address space
- prevent access to 1st page to catch program errors executing on null pointers

1.1 Memory Management Implementation

- two main components: **pager** and **swapper**

1.1.1 Pager

- Pager is an OS routine that does page management
- Pager design prioritizes reducing processor usage over reducing memory consumption
- The pager maintains a *resident set* of pages for each process. each process can load limited number of pages called the *resident set limit*
- **Local page replacement**: Instead of global LRU, the page evicted is seelected from the **resident set** of the **requesting process** in a **FIFO** manner
- When a page is paged out, it moves to either the free set or the modified set based on the modify bit value, called second chance algorithm
- These lists act as caches with much faster access time than the main memory

- VMS resolves the problem of addition I/O caused by paging, by clustering pages together to load/store (this is caused by small page size)
- The cluster size depends on several factors, can be specified while programs are being linked
- Clustered reads use the free-page list and clustered writes use the modified page list

1.1.2 Paging in System Addr Space

- The OS contains pageable and non-pageable memory (some system pages cannot be paged out)
- The system has its own free and modified lists where paged-out pages reside
- The process-specific page tables are kept on the process' cache after being paged out (even though they are in system addr space)

1.1.3 Swapper

- Swaps entire resident sets to a swap-file
- ensures that **highest priority processes** are **always resident**, and avoids cold-start problems with recently-resumed processes
- The swap operation may happen in pieces (esp. if there was some ongoing I/O op)
- When a new process is created, the swapper swaps in a process shell, which provides the initial environment

2 Virtual Memory Management for Uni- and Miltiprocessor Systems: the Mach kernel

- Provide machine-independent support for paging with complete UNIX capability
- Add support for:
  - Large, sparse virtual address spaces
  - copy-on-write
  - memory-mapped files

2.1 Main Abstractions

- *Task*: Exec environment where threads run, has a virtual addr space, protected access to resources
- *Thread*: Single program counter running inside a task, smallest unit of program exec
- *Port*: communication channel protected by kernel
- *Message*: data objects sent over a port, may contain pointers/typed capabilities
- *Memory object*: collection of data that can be mapped into a task's addr space

2.2 Basic Virtual Memory Operations

- A Mach physical page size is not the same as a page as defined by particular h/w (is a power of 2 multiple of the same)
- Entire address spaces can be shared using CoW
- Child tasks inherit their parent's address space based on the inheritance type (*shared* for read and write, *copy* for CoW semantics or *none*)
- Default semantics are copy-on-write. UNIX fork is implemented this way
- Each group of pages also has permissions whose enforcement is supported by hardware
- Data structures used for memory mgmt:
  - **resident page table**: track machine-independent pages

  - **address map**: map range of addresses to a memory object.
  - **memory object**: unit of storage managed by kernel/user task
  - **pmap**: machine-dependent memory mapping structure

- Addresses within a task address space are mapped to byte offsets in memory objects by a data structure called an **address map**
- Address map entries store protection information for the range of virt addresses
- The **resident page table** is similar to an inverted page table, mapping Mach pages to address map entries
- A **virtual memory object** is a repository for data, indexed by byte, upon which various I/o operations can be performed
- Memory objects have reference counts and are **garbage collected** when refcount becomes 0
- Modified pages created by CoW faults are held in a memory object called a **shadow object**
- Shadow objects do not contain all the pages of the memory object they reference. It is possible to follow the trail of shadow objects to retrieve any page from the original object
- The **sharing map** is used to list pages that are shared read/write with other tasks