Windows 000 and Linux Memory Management

Memory Management
- Sophisticated virtual memory (VM) management
  - Assumption is that underlying hardware supports virtual-to-physical address translation, paging, and other VM features
- The VM manager in 2000 uses a page-based management scheme with a page size of 4 KB
- VM manager uses 32 bit addresses, so each process has a 4 GB virtual address space
- Upper 2 GB are identical for each process and lower 2 GB are distinct for each process
- Two-step memory allocation procedure
  1. Reservation a portion of the process’ address space
  2. Commitment of the allocation by assigning space in the OS paging file

Virtual Memory Layout

Virtual Memory Manager (Cont.)
- The virtual address translation in 2000 uses several data structures.
  - Each process has a page directory that contains 1024 page directory entries of size 4 bytes.
  - Each page directory entry points to a page table which contains 1024 page table entries (PTEs) of size 4 bytes.
  - Each PTE points to a 4 KB page frame in physical memory.
- A 10-bit integer can represent all the values form 0 to 1023, therefore, can select any entry in the page directory, or in a page table.
- This property is used when translating a virtual address pointer to a byte address in physical memory.
- A page can be in one of six states: valid, zeroed, free standby, modified and bad.
Virtual-to-Physical Address Translation

- 10 bits for page directory entry, 10 bits for page table entry, and 12 bits for byte offset in page.

Page File Page-Table Entry

- 5 bits for page protection, 20 bits for page frame address, 4 bits to select a paging file, and 3 bits that describe the page state. V = 0

Page File Page-Table Entry

Bits | Value
---|---
0 | Not used
1 | Global
2 | Large
3 | Page is dirty
4 | Page has been accessed
5 | Valid page table entry

A page table entry for a mapped page on the Pentium

Fundamental Concepts (2)

- Mapped regions with their shadow pages on disk
- The 4k.dll file is mapped into two address spaces at the same time

Memory Management System Calls

<table>
<thead>
<tr>
<th>Win32 API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualAlloc</td>
<td>Reserve or commit a region</td>
</tr>
<tr>
<td>VirtualFree</td>
<td>Release or decommit a region</td>
</tr>
<tr>
<td>VirtualProtect</td>
<td>Change the read/execute protection on a region</td>
</tr>
<tr>
<td>VirtualQuery</td>
<td>Inquire about the status of a region</td>
</tr>
<tr>
<td>VirtualLock</td>
<td>Make a region memory resident (i.e., disable paging for it)</td>
</tr>
<tr>
<td>VirtualUnlock</td>
<td>Make a region pageable in the usual way</td>
</tr>
<tr>
<td>CreateFileMapping</td>
<td>Create a file mapping object and (optionally) assign it a name</td>
</tr>
<tr>
<td>MapViewOfFile</td>
<td>Map (part of) a file into the address space</td>
</tr>
<tr>
<td>UnmapViewOfFile</td>
<td>Remove a mapped file from the address space</td>
</tr>
<tr>
<td>OpenFileMapping</td>
<td>Open a previously created file mapping object</td>
</tr>
</tbody>
</table>

The principal Win32 API functions for mapping virtual memory in Windows 2000

Programmer Interface - Memory Management

- Virtual memory:
  - VirtualAlloc reserves or commits virtual memory.
  - VirtualFree decommits or releases the memory.
- These functions enable the application to determine the virtual address at which the memory is allocated.
- An application can use memory by memory mapping a file into its address space.
  - Multistage process.
  - Two processes share memory by mapping the same file into their virtual memory.
Linux Memory Management
- Linux’s physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory.
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes.

Managing Physical Memory
- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request.
- The allocator uses a buddy-heap algorithm to keep track of available physical pages.
- Each allocatable memory region is paired with an adjacent partner.
- Whenever two allocated partner regions are both freed up they are combined to form a larger region.
- If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator).

Virtual Memory
- The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required.
- The VM manager maintains two separate views of a process’s address space:
  - A logical view describing instructions concerning the layout of the address space.
  - A physical view of each address space which is stored in the hardware page tables for the process.
Virtual Memory (Cont.)

- Virtual memory regions are characterized by:
  - The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (demand-zero memory).
  - The region’s reaction to writes (page sharing or copy-on-write).
- The kernel creates a new virtual address space
  1. When a process runs a new program with the exec system call
  2. Upon creation of a new process by the fork system call

Virtual Memory (Cont.)

- On executing a new program, the process is given a new, completely empty virtual-address space; the program-loading routines populate the address space with virtual-memory regions.
- Creating a new process with fork involves creating a complete copy of the existing process’s virtual address space.
  - The kernel copies the parent process’s VMA descriptors, then creates a new set of page tables for the child.
  - The parent’s page tables are copied directly into the child’s, with the reference count of each page covered being incremented.
  - After the fork, the parent and child share the same physical pages of memory in their address spaces.

Virtual Memory (Cont.)

- The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else.
- The VM paging system can be divided into two sections:
  - The pageout-policy algorithm decides which pages to write out to disk, and when.
  - The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed.

Virtual Memory (Cont.)

- The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use.
- This kernel virtual-memory area contains two regions:
  - A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code.
  - The remainder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory.

Executing and Loading User Programs

- Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made.
- The registration of multiple loader routines allows Linux to support both the ELF and a.out binary formats.
- Initially, binary-file pages are mapped into virtual memory; only when a program tries to access a given page will a page fault result in that page being loaded into physical memory.
- An ELF-format binary file consists of a header followed by several page-aligned sections; the ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory.

Memory Layout for ELF Programs

- The memory layout for ELF programs includes:
  - Memory segments for the program text, initialized data, uninitial data, stack, and the OračO pointer.
  - The kernel virtual memory is memory invisible to user mode code.
Static and Dynamic Linking

- A program whose necessary library functions are embedded directly in the program’s executable binary file is statically linked to its libraries.

- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions.

- Dynamic linking is more efficient in terms of both physical memory and disk-space usage because it loads the system libraries into memory only once.

Acknowledgements