Operating Systems Security I

Tom Kelliher, CS 325

Oct. 3, 2011

1 Administrivia

Announcements

Assignment

Read 4.3–4.4.

From Last Time

Outline

1. Protected objects and general protection mechanisms.

2. Memory protection

3. CPU execution modes and kernel virtualization of CPU.

4. Server virtualization.

Coming Up

Operating Systems Security II.

2 Protected Objects and General Protection Mechanisms

With today's operating systems, multiple processes/threads run together "simultaneously," necessitating that protection mechanisms exist for the following resources:

1. Memory.

2. Shared I/O devices (disk drives, etc.).

3. Serially reusable I/O devices (printers, etc.).

4. Sharable programs and libraries.

5. Networks.

6. Sharable data.

Separation is the basis of protection:

1. Physical separation: Run sensitive processes on separate systems.

2. Temporal separation: Run sensitive processes at different times.

3. Logical separation: OS/hardware provide a logical separation of sensitive processes.

4. Cryptographic separation: Sensitive processes conceal their data and computations cryptographically.

How can processes share data and code?

1. No protection: acceptable when temporal separation is in use.

2. Isolate: one process is completely unaware of the existence of another process. (Logical separation)

3. Share all or share nothing: objects declared public (access by all) or private (access by no one else).

4. Share via access limitation: Maintain a list of (user, access rights) pairs for each object in the system.

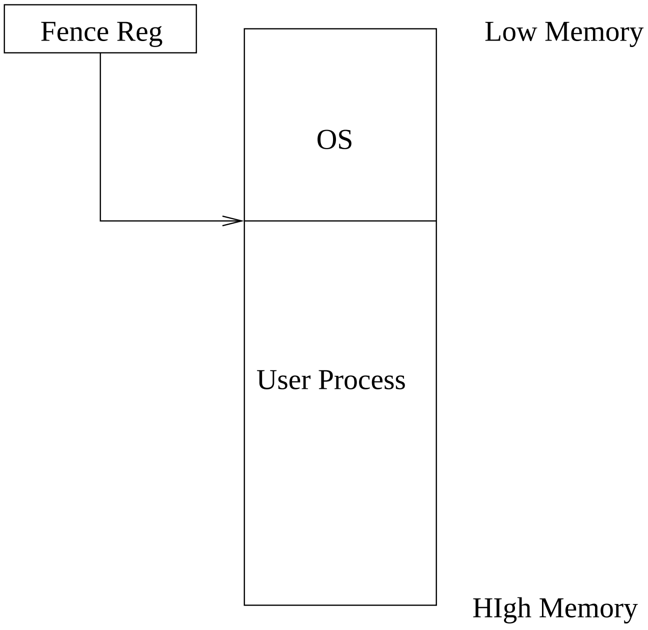
5. Share by capabilities: Each user (process) holds a list of access rights (capabilities) for objects in the system. (Similar to previous mechanism.)

6. Limit use of an object: Extend the usual read, write, execute notion to such actions as print and copy.

3 Memory Protection

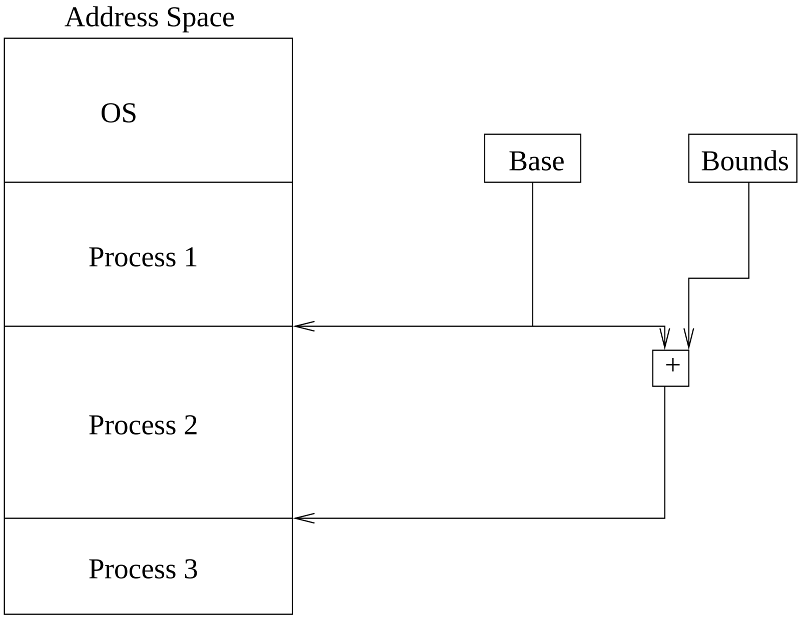
Multiple memory locations can be accessed per each instruction executed, so access limitation mechanisms must be extremely efficient.

1. Variable and fixed fence registers. Basic idea:



Access "under" the fence results in an exception.

2. Base/Bounds registers: Fence register grants access all the way to the end of memory. The bounds register specifies the size of the memory segment.

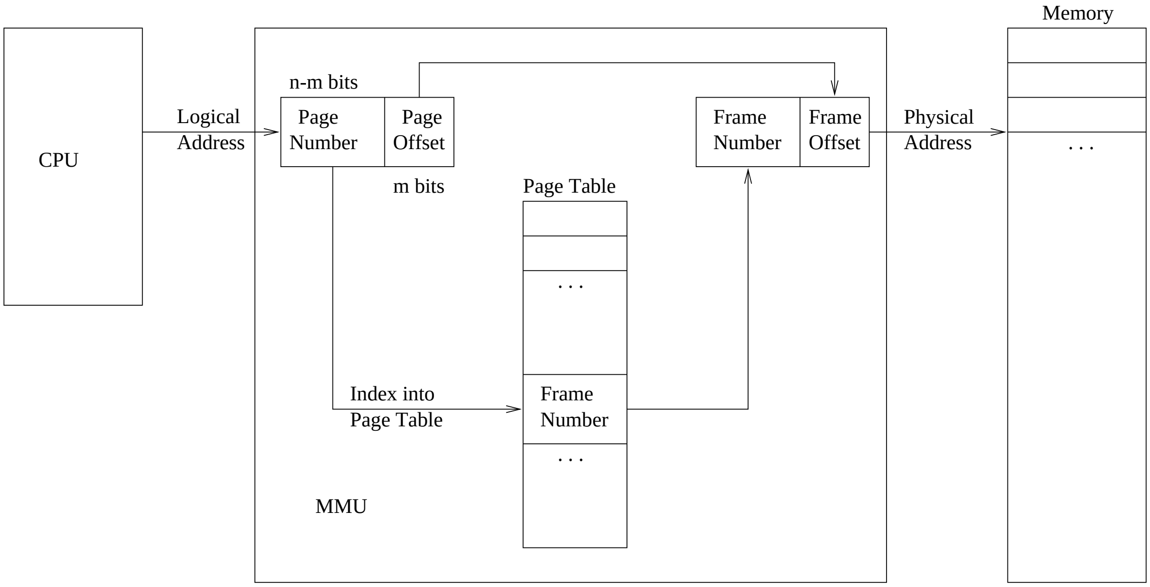


Multiple base/bounds pairs can be used to provide separate areas for the four memory segments associated with a program: code, heap, stack, and data.

3. Tagged architectures: associate tag bits with each memory word.

   Burroughs used three tag bits. Tag meanings: data, double precision data, loop index, uninitialized data, pointer, code, data block descriptor, program control word. The first four tags are for R/W words. The second four are read-only words.

4. Paging: Further partitions memory into frames. Maps a logical (page, offset) address to a physical (frame, offset) address by use of a page table.



Virtual memory takes this one step further by removing the restriction that all logical addresses be mapped to physical addresses.

4 CPU Execution Modes and Kernel Virtualization of CPU

1. Some machine instructions are "dangerous." Examples: halt, I/O access. "Privileged" instructions.

   User programs should not be able to execute these instructions.

2. Solution: CPU runs in one of two modes: supervisor or user.

3. Privileged instructions generate traps when executed in user mode.

   Generally, the trap handler terminates the process.

4. How do we get from user mode to supervisor mode to perform I/O?

   System call instruction.

5. The kernel emulates the CPU for user processes.

   The only exception are the privileged instructions.

6. Processes are isolated from one another.

5 Server Virtualization

1. A host OS, or hypervisor, runs multiple guest OSs.

   Each guest OS runs in complete isolation from other guests.

   Each guest runs as if it has its own resources (disk, network, etc.).

2. In a sense, a hypervisor is an OS that runs other OSs.

   The hypervisor emulates the CPU for guest OSs.

   In contrast, now the privileged instructions must also be emulated.

3. Virtualization requirements for an architecture:

   Any instruction which changes certain “sensitive” machine state must generate a trap if executed in user mode.

   The trap is used so as to return control to the hypervisor and it can provide the behavior required by the guest, rather than the hardware providing the behavior.

   x86 does not meet this requirement — paravirtualization requires some changes to the guest OS.

4. Both Intel and AMD have added virtualization support.

5. When done right, virtualization can be extremely efficient, as opposed to the earlier technique of emulation. For example, running Windows on a PowerPC by emulating an X86.