16 Shared Virtual Memory, COW, DSM

Readings for this topic: Siberschatz/Galvin, Chapter 9

We've seen that dynamic relocation provides flexibility in managing physical memory and allows us to virtualize main memory. Next, we'll study additional uses of the dynamic relocation hardware.

Shared virtual memory: Map virtual address ranges in *different* processes to the *same* physical memory range. This works with all dynamic relocation hardware. What is it good for?

1) Shared code or shared read-only data segments: when an OS runs several instances of the same program, can save physical memory by mapping the code segments of each process to one copy of the code segment in physical memory. Since code segments are normally read-only, it is safe to do this. The same can be done for initialized data segments that are read-only.

2) Shared memory segments: when two cooperating processes want to share a region of their virtual address space to coordinate or exchange data. Note that such processes need to synchronize, just like threads in a single process.

Copy-on-write: Suppose we would like to save even more physical memory by using only a single copy in physical memory for multiple instance of the same program whenever possible. Note that this sharing must be transparent to the processes, i.e., when a process writes a page, these changes must not be visible to the other processes.

Idea: create multiple physical copies of a page lazily and only when needed. Initially, share a single physical page for each of the program's pages among all instances, but mark the pages as read-only. When a process writes into a page, a page fault happens. The OS then allocates another physical page, initializes the page with a copy of the original page's data, enables writes and then returns from the page fault.

If a process writes into all of its pages, then the OS must eventually allocate a private copy of each page and nothing is gained. However, most programs only modify some of their memory.

Modern UNIX/Lnux implementations use COW to implement the `fork()` system call efficiently. (`fork()` creates a child process that is initially an exact copy of the parent process). Initially, the child process all of its pages mapped to the same pages as the parent process. If either of the processes subsequently writes a page, a phycial copy is created.

Distributed shared memory: Suppose we'd like to write a multi-threaded program that can take advantage of the CPUs in multiple machines connected by a fast network. Create a virtual address space that spans multiple machines with separate physical memories.

In the simplest implementation, each page has a single physical memory page that is located on one of the computers at any given time. If the page is accessed on a different machine, a page fault happens, upon which the OS fetches the page across the network. As an optimization, copies of the same page can exist on multiple machines; only when the page is written do we have to make sure that the changes are send to all copies in a manner that preserves the consistency of the shared memory.