CSE451 Memory Management Continued Spring 2001

Gary Kimura
Lecture #17
May 2, 2001

Today

- How does the free page list get replenished?
- What do we do with dirty pages?
- Thrashing
- What parts of the operating system always needs to be resident in physical memory?
- What about allocating memory in smaller units than a page

Memory Management Page States and Lists

- MM maintains a list of physical pages according to the following attributes (various implementations use slightly different lists)
  - Zeroed pages
  - Free pages
  - Standby pages
  - Modified pages
  - Modified No Write pages
  - Bad pages
- MM's goal is to use these pages on these lists to supply memory to the system

Making free pages

- When a process exits it pages are freed
- When a process gets its "working set" reduced some pages are freed
- When a processes deallocates memory its pages are freed
- One special characteristic about "currently used" pages is that some are "clean" and some are "dirty"

Dirty pages

- Dirty or modified pages need to be written out before the frame can be freed
- We can write out a dirty page just before the page is freed
  - This minimizes the number of writes we need to do
  - This also means that making a free page might take a while longer
- Or we can periodically write out dirty pages
  - There can be a "modified page writer" process in the system that sweeps through writing out modified pages
  - Picking when to write a page can be a problem, because writing too often is bad

Thrashing

- Thrashing is when the system is so busy reading and writing page frames that the effective system throughput is getting close to zero.
- Overstressed systems exhibit this behavior
- Part of testing a commercial system is to load it up to capacity and fix what breaks
- Some techniques to avoid thrashing is to simply limit the number of processes that can exist at a given time.
  - Other limits are possible and useful (opened files, logged on users, etc)

Paged and non paged memory

- Some data and code must always be in memory (also called resident)
  - All of the kernel, all the time?
- Paged memory has a copy in backing store and can be discarded from main memory and brought back in with only a performance penalty
- Nonpaged memory for various reasons cannot be discarded and brought back in.
  - Sometimes the code/data must always be resident to run the system (e.g., the code that does the actual backing store support)
  - Sometimes the code/data is "pinned" in memory for a device driver to access for DMA purposes
  - Sometimes code is pinned in for performance reasons

How much to page

- There is a chicken and egg problem of making sure that the code & data necessary to page-in non-resident data is itself in memory.
- My laptop running NT has 13MB of code and 28MB of data, but only 3MB of code and 4MB of data is permanently in memory.

Sub page allocation

The basic allocation granularity for the system is a physical page. So what does a system programmer do to get smaller allocations (e.g., a malloc of 32 bytes)?

- There needs to be a sub-page memory allocator used to allocate either paged of non-paged memory in the kernel.

- This is similar to user mode heap but with some additional requirements
  - Paged versus non-paged
- Basically there needs to be an allocate and a free function.
  - In NT this is called the kernel pool allocator
- Fragmentation is still an issue

Still to come

- File systems
- I/O systems
- Software File Caching
- Distributed systems
- Security and administration