Advanced Topics in Operating Systems, CS262

Prof. Eric A. Brewer

Lecture Notes, 9/25/98

Application-Controlled Virtual Memory

I. Application-Controlled Physical Memory

Basic idea: export all VM policy decisions.

Why is this useful?

- o Applications that don't fit into physical memory and have nonstandard page access patterns.
- o Applications that can trade space for time.
- o Techniques like page coloring
- o Obtain a very small kernel; most of the VM system can be punted if not needed (e.g. for embedded systems)

Key concept:

- o Treat hardware as a page frame cache at the kernel level.
- o Pieces of the cache are exported to user-level managers.

V++ implementation looks similar to Mach's: segments ~ memory objects.

Page fault handled by a separate segment manager process:

- o Kernel reflects fault to segment manager (as an IPC message).
- o Segment manager obtains desired data from somewhere (e.g. backing storage file server).

Data goes into a free page frame in a segment of the segment manager's address space.

- o Data is migrated from segment in segment manager's address space to segment in the address space in which the page fault originally occurred (using MigratePages kernel call).
- o Segment manager returns from page fault (by replying to the IPC message).

Page fault handled by faulting process (note: process = thread in V):

- o Segment manager code run as a ``signal handler'' procedure by the faulting thread. Why? Can avoid almost all the context switching overhead with the right hardware (e.g. MIPS R3000)
- o Segment manager procedures run on a separate stack that is pinned in memory to avoid infinite recursive page faults.

UIO protocol: block streams (of bytes) on top of IPC.

Cached files done by means of memory-mapping them. File cache management done by user-level segment manager that communicates with file servers.

Things that application-specific segment management allows:

- Steal free page frames from temporary index data structures (so don't need a dedicated free segment).
- Have multiple free segments and multiple segment managers.
- Can implement any VM policy (including standard ones) you want. What are some examples?
    - Clock algorithm: sweep through memory periodically, recording and resetting ref bits.
    - Delete whole segments of dirty data after application is done with them.
    - Avoid replacing critical pages.
    - Switch from LRU to MRU when appropriate.

Segment manager initialization (if managing its own code and data segments):

- Bring pages of segment manager's segment into memory. Why is this necessary?

Because the other segment manager won't be responsible for page faults later on.

- Assume ownership of the segment.
- Touch all the segment pages to verify that they're actually still in memory.

Start over if not. Why is this necessary? Because we can't retrieve the missing pages anymore.

- Don't have to worry about page fault in initialization code because this code is always in a segment backed by the default segment manager.

System Page Cache Manager:

- Controls how much of the physical memory is available for caching each segment's page frames.
- Free market allocation model.
    - Goal: let applications decide how to ``spend'' their dram resources. Can have lots of memory for brief periods of time or less memory for longer periods of time.
    - Unclear how this approach will work in practice.

Performance:

- Minimal fault time given is miss-leading - measured on R3000 processor.
- Bottom line: relatively small degradation for ``normal'' applications, potentially big wins for memory-intensive applications.
- How could you speed up the ``normal'' case? Put default segment manager into the kernel.

3 key features of the paper:

- Export all VM policy decisions to the user level; either to default segment managers or to application-specific segment manager
  - which may run in the same address space as the application itself.
- Application-specific VM control allows large, memory-intensive applications to potentially obtain major performance improvements by implementing nonstandard VM

management policies and by being aware of how much physical memory they actually have available at any given time to compute and use things like temporary database indices, etc.

- Performance is slightly degraded for normal applications in exchange for offering potentially big improvements for nonstandard, big applications.

Some flaws:

- Paper ``waves its hands" about how physical page frames are allocated to different applications. No evidence is give concerning how well their proposed free market approach will work in practice.
- Paper presents minimal fault time number for a processor that allows extremely efficient returns from a page fault handled on the thread that incurred the page fault. Should have given the number for another standard architecture that doesn't support this feature as well.

A lesson: Exporting policy decisions to the application can potentially provide big performance wins. The challenge is to do it in a fashion that still provides good performance to ``normal" applications that don't need this extra level of control.