Lecture Notes, 9/23/98

User-Level VM

I. Virtual Memory Primitives for User Programs

Basic idea: use VM facilities (which have hardware support for efficiency) for things other than providing large virtual address spaces.

Key insight: VM facilities let you detect memory references you're interested in if page faults can be exported to the application and the application can also control page protections.

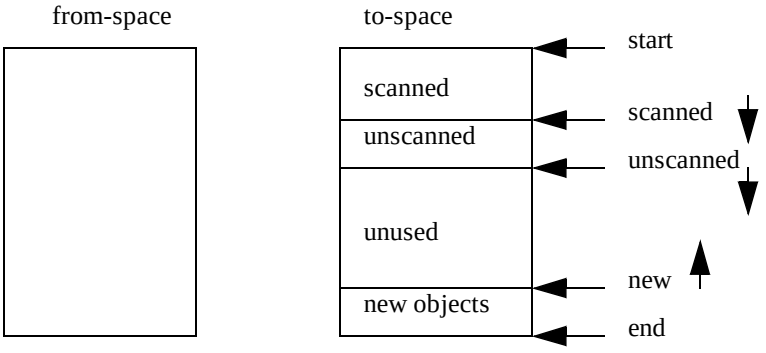
Example applications:

Concurrent GC:

Invariants maintained:

- Mutator sees only to-space refs.
- to-space objects only point into to-space
- Scanned pages in from-space only contain pointers to to-space
- Unscanned pages in from-space can point to either space.

Memory layout:



What happens when New() can't find enough free space for the requested allocation: Flip():

- Stop all the mutator threads.
- Scan any remaining unscanned objects (i.e. make sure previous GC, if still running, is finished).

- Flip the roles of the two spaces:
  - start := beginning of to-space; end := end of to-space;
  - scanned := start; unscanned := start; new := end;
- Copy the root reachable objects from from-space to the unscanned area (makes unscanned pointer grow downward in diagram). Root-reachable objects: objects referenced in registers, in global variables not stored in heap, and on stack (if stacks not stored in heap).
- Resume the mutator threads.
- Signal collector thread to start scanning again.

Collector thread:

```
loop
    while not (scanned < unscanned) do
        wait (unscannedPages c.v.);
        ScanPage(scanned);
        scanned := min(unscanned, scanned + PageSize);
    end;

ScanPage(objectAdr):
    page := PageBeginning(objectAdr);
    if Unprotected(page) then return;    (* Page already scanned *)
    while PageBeginning(objectAdr) = page and objectAdr < unscanned  do
        ScanObject(objectAdr);
            (* Forward any from-space pointer found by copying object
               from from-space if necessary (and thereby advancing
               unscanned ptr) and changing pointer to point to the new
               copy (in the unscanned area of to-space. *)
        objectAdr := objectAdr + ObjectSize(objectAdr);
    Unprotect(page);
```

When forwarding an object:

- Make sure the page in unscanned area to copy to is protected against access.
- Copy object. Can do this using either Map2 or by running in privileged mode.
- Leave a forwarding pointer in from-space for the copied object.

When a mutator thread tries to obtain a pointer from a mem. loc. in unscanned to-space:

- Page fault occurs on access to unscanned page.
- ScanPage() called to make sure only to-space pointers are left on that page.
- Resume mutator thread.

Lecture 8

Map2 is needed so the collector thread can access a page it plans to scan. Why?

- So that the page remains protected against accesses from other concurrent mutator threads.

Note: smaller page size would imply lower delay when a mutator faults on an unscanned page.

Distributed Shared Memory: see paper for details on this and other applications.

Note: the smaller the page size the less *false sharing* there will be. What is false sharing?

- False sharing: access to one variable brings over all the variables on the same page, even if not needed, or worse yet, if needed elsewhere.

Concurrent checkpointing:

Note: could copy the address space using copy-on-write instead.

Persistent store:

- Memory-mapped file used to make heap persistent.
- Lots of stuff glossed over (like efficiency of page-level locks).
- Transactional semantics => page-out to disk must be controlled.

Extended Addressing & Data Compression:

- Key idea: realize that in-memory and backing-store representations can be different.
- Keeping compressed pages in memory yields a 3-level memory hierarchy.

VM performance:

- Not an issue for regular demand paging from/to disk. Why? Disk access over-shadows everything.
- Significant differences among different OSs on the same machine. Why might Mach do so poorly?
  - Machine-independent design.
  - Support for MP
  - Didn't care
- Some machines do much better than others.

System Design Issues:

Lecture 8

- TLB shoot-down is a problem, but less so than one might think because batching can amortize the overhead. Making a page less restricted is OK, since it just causes a spurious TLB fault; making it more restricted requires TLB cleaning to avoid improper access...
- Authors advocate small page size. Trade-offs?
  - Bigger page tables.
  - Poorer TLB performance.
- Map2 support advocated (i.e. hardware shouldn't prevent it). What's an example of hardware that would prevent Map2? Hardware-defined inverted page table scheme.
- Pipelined hardware => page fault handlers can't see registers. Only a problem for GC heap overflow application.

3 key features about the paper:

- VM facilities used for applications other than providing large virtual address spaces.
- Hardware-supported VM facilities provide an efficient way to detect memory references an application is interested in knowing about; the smaller the page-size supported the better for the applications described in the paper.
- OS should be careful to provide fast implementations of VM operations such as page fault and change-of-protection.

Some flaws:

- Not much discussion of the trade-offs that OS designers might have to make with respect to efficiently supporting the features advocated by this paper (e.g. small page size) versus other concerns.

A lesson: Think about different ways that an abstraction that the OS provides could be used.