Memory Management in FreeBSD 12.0

Mark Johnston

BSDTW 2017

Introduction

- Mark Johnston, markj@FreeBSD.org
- OS Team at Dell EMC Isilon
- FreeBSD user since 2010, committer since December 2012
- sys/vm neophyte since 2015

Responsibilities

- Implementing the virtual memory abstraction
- Syscalls: `mmap`, `madvise`, `mprotect`, `minherit`, `mincore`, `msync`, `mlock`, `mlockall`, `munmap`, `munlock`, `munlockall`
- More syscalls: `fork`, `vfork`, `execve`, `read`, `write`, `sendfile`, `ptrace`, `...`
- Page fault handling
  - Enforcing memory protection
  - Copy-on-write
  - Tracking page dirty state
  - Page-in from swap, or a filesystem, or device memory

Responsibilities, cont'd

- Handling memory pressure
  - Reclaiming pages, approximating LRU
  - Page queue maintenance
  - Laundering dirty pages
- Kernel memory allocation
  - `malloc(9)`, `free(9)`
  - Slab allocation (`uma(9)`)
  - KVA allocation (`vmem(9)`)
  - Physically contiguous allocations
  - Physical address constraints (e.g., ISA DMA)
- Miscellaneous tightly coupled subsystems
  - FS buffer cache
  - `pmap(9)`
  - `tmpfs(5)`, `md(4)`
  - `vmm(4)`, Linux drm drivers
  - SYSV and POSIX shared memory
  - Scalability w.r.t. CPU count and RAM size
  - Good single-threaded performance

VM Factoids

- The VM is not a virtual machine
- Originated from Mach, since 4.4BSD
- Heavily reworked in the 1990s, `dg@`, `dyson@`, `dillon@`, `alc@`

Kernel	Path	Comments	Code
OpenBSD	`sys/uvm/`	7,716	13,853
FreeBSD	`sys/vm/`	9,246	21,468
illumos	`uts/common/vm/`	14,352	34,100
XNU	`osfmk/vm/`	17,007	52,400
Linux	`mm/`	28,476	78,260

Implications

- The VM is complicated
- Old foundations
- Consistency and conceptual integrity are important
- Some workloads are more important - can't catch 'em all
  - Think carefully about tradeoffs
  - Good "Pareto optimizations" are nice
  - Pathological behaviour is not OK
- It's easy to write code that seems to work
  - Simple tests aren't going to find your race conditions, but your users probably will
  - CVE-2013-2171, CVE-2016-5195
  - It's easy to silently break optimizations
- Think twice, commit once

vm_page_t

- One per page of physical RAM
- vm_page_array
- vm_phys.c buddy allocator
- Wire a page to make it unreclaimable
  - Removes page from paging queue
  - Unwire queues page at the tail of a paging queue
  - Used to implement mlock(2), buffer cache, ZFS ARC, etc.
- Managed pages keep track of their mappings (PV entries)
- 104 bytes(!) on amd64
- Locking
  - Physical address locks (mtx)
  - Per-page busy lock
  - Object lock

`vm_map_t`, `vm_map_entry_t`

- Organize memory layout for userland processes
- Contiguous regions described by `vm_map_entry_t`'s
- Provide $O(\log(n))$ address space allocation
- Organize map entries in a sorted list and splay tree
- Special handling for userland stacks
- Locking
  - Per-map reader-writer lock (`sx`)

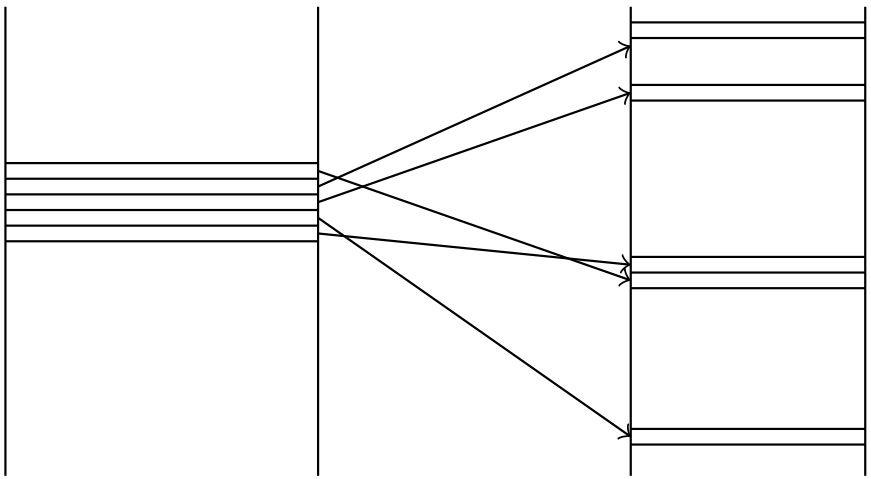
```
# procstat -v 1
  PID              START                END PRT  RES PRES REF SHD FLAG TP PATH
    1            0x400000           0x51a000 r-x  210  224   2   1 CN-- vn /sbin/init
    1            0x71a000           0x71f000 rw-    5    0   1   0 CN-- vn /sbin/init
    1            0x71f000           0x952000 rw-   16   16   1   0 CN-- df
    1        0x80071a000        0x80091a000 rw-    7    7   1   0 CN-- df
    1        0x80091a000        0x80091b000 r--    1    1  43   0 ---- dv
    1        0x80091b000        0x800d2d000 rw-   14   14   1   0 CN-- df
    1        0x800d2d000        0x800d55000 rw-   17   17   1   0 C--- df
    1        0x800d55000        0x800d58000 rw-    1    1   1   0 C--- df
    1     0x7fffdffff000     0x7ffffffdf000 ---    0    0   0   0 ---- --
    1     0x7ffffffdf000     0x7fffffff000 rw-    2    2   1   0 C--D df
    1     0x7fffffff000     0x800000000000 r-x    1    1  44   0 ---- ph
```

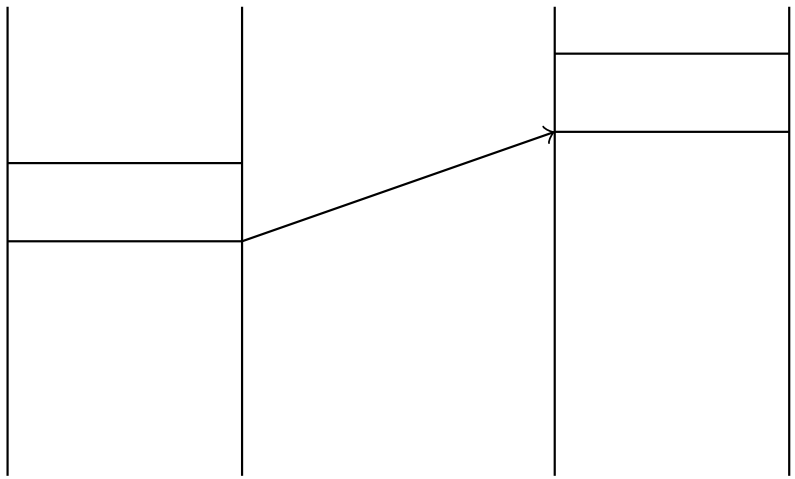
vm_object_t

- Acts as a generic "source" of pages
- Integrated with pager methods
- 7 different object types (`OBJT_*`)
  - `OBJT_DEFAULT` converted to `OBJT_SWAP` upon first pageout
  - Object type selects pager methods
- Resident pages stored in a sorted queue and a radix tree
  - Addressed by 64-bit virtual `pindex` (`vm_pindex_t`)
  - A `vm_page_t` belongs to at most one object
- Objects don't contain information about their mappings
  - Often mapped into multiple address spaces
  - Object hierarchy used to implement COW
- Locking
  - VM object lock (`rwlock`)
  - vnode lock, for `OBJT_VNODE` objects

vm_reserv_t

- Support speculative allocation of physically contiguous pages
- Integrated with `pmap(9)` to allow transparent creation of large mappings (e.g., 2MB instead of 4KB on x86)
- Locking
  - Free page queue lock (`mtx`)





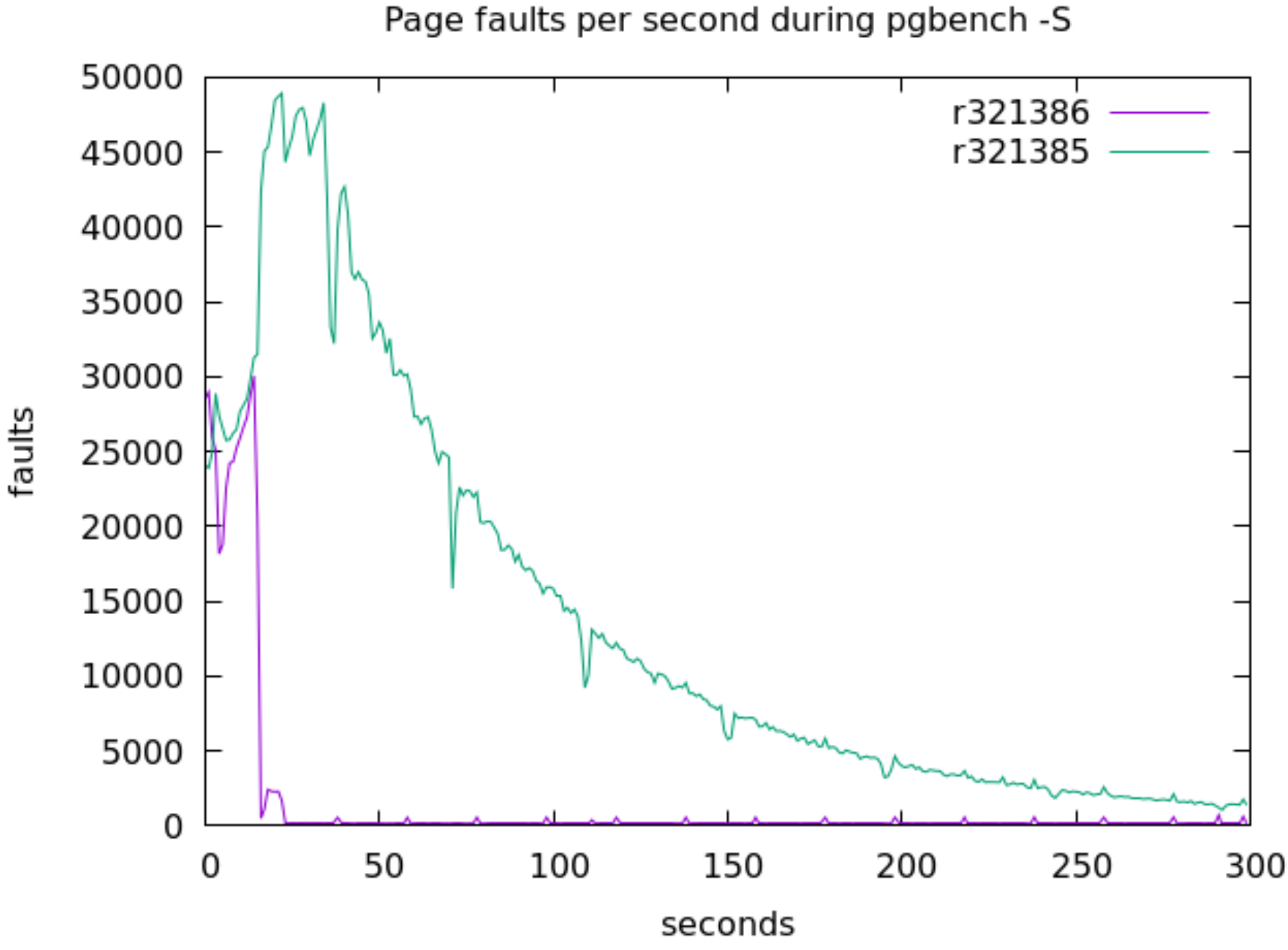
pmap_enter(psind=1)

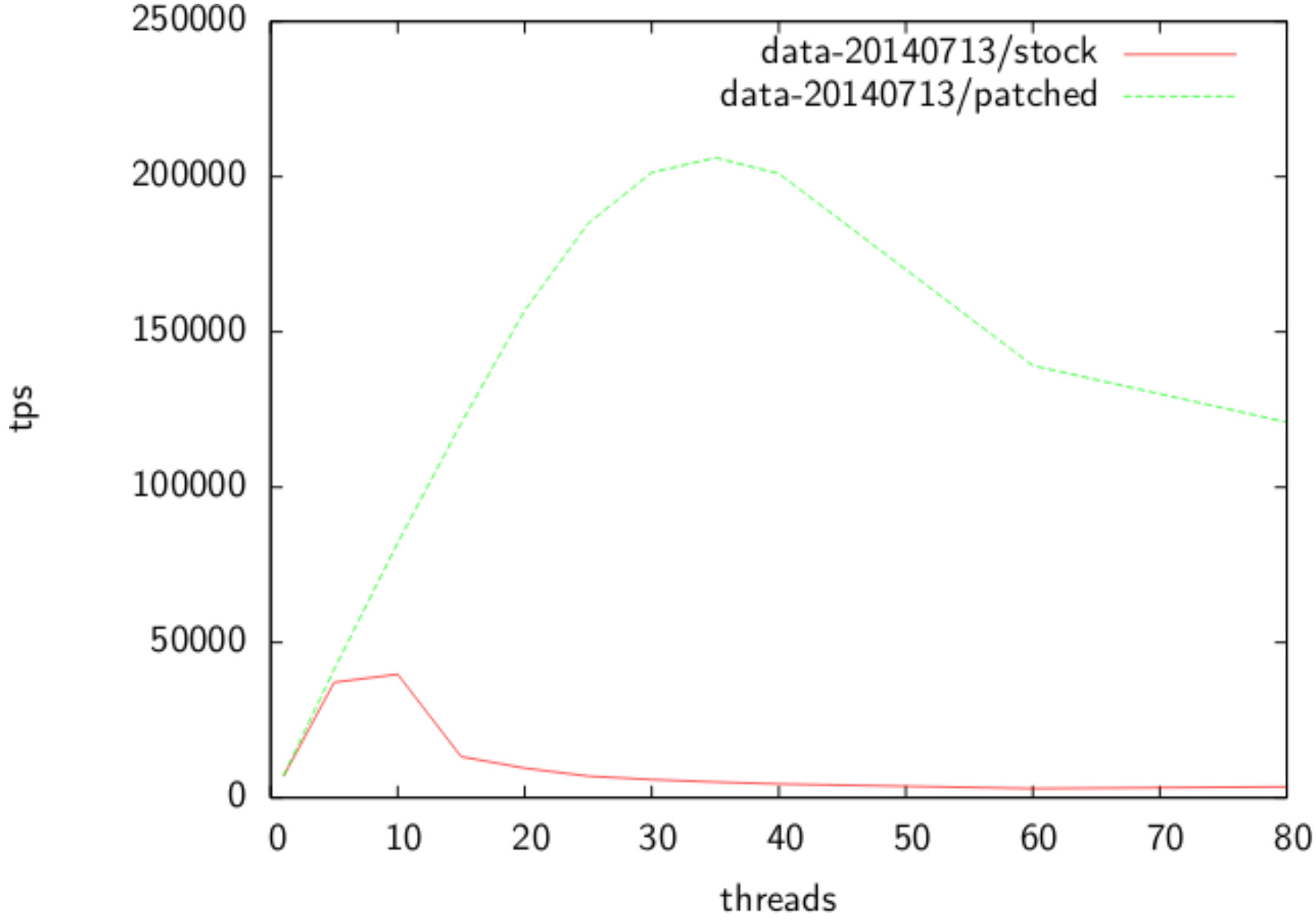
- r321386 by alc, review D11556
- Immediately promote mapping if reservation is fully populated
- Eliminates many page faults on vnodes and shmem

```
dtrace -n
'fbt::pmap_enter:entry /args[5]/{printf("%s", execname);}'
```

pmap_enter(psind=1), cont'd

```
if ((m->flags & PG_FICTITIOUS) == 0 &&
    (m_super = vm_reserv_to_superpage(m)) != NULL &&
    rounddown2(vaddr, pagesizes[m_super->psind]) >= fs->entry->start &&
    roundup2(vaddr + 1, pagesizes[m_super->psind]) <= fs->entry->end &&
    (vaddr & (pagesizes[m_super->psind] - 1)) == (VM_PAGE_TO_PHYS(m) &
    (pagesizes[m_super->psind] - 1)) && pmap_ps_enabled(fs->map->pmap)) {
        flags = PS_ALL_VALID;
        if ((prot & VM_PROT_WRITE) != 0) {
                /*
                 * Create a superpage mapping allowing write access
                 * only if none of the constituent pages are busy and
                 * all of them are already dirty (except possibly for
                 * the page that was faulted on).
                 */
                flags |= PS_NONE_BUSY;
                if ((fs->first_object->flags & OBJ_UNMANAGED) == 0)
                        flags |= PS_ALL_DIRTY;
        }
        if (vm_page_ps_test(m_super, flags, m)) {
                m_map = m_super;
                psind = m_super->psind;
                vaddr = rounddown2(vaddr, pagesizes[psind]);
                /* Preset the modified bit for dirty superpages. */
                if ((flags & PS_ALL_DIRTY) != 0)
                        fault_type |= VM_PROT_WRITE;
        }
}
rv = pmap_enter(fs->map->pmap, vaddr, m_map, prot, fault_type |
    PMAP_ENTER_NOSLEEP | (wired ? PMAP_ENTER_WIRED : 0), psind);
```


Page faults per second during pgbench -S
faults
seconds
r321386
r321385
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
0
50
100
150
200
250
300



From *PostgreSQL/FreeBSD performance and scalability on a 40-core machine*

PQ_LAUNDRY

- r308474 by alc
- When and how much do we swap?
- Dirty pages must be laundered before they can be freed
- Old algorithm: scan PQ_INACTIVE for clean pages, and launder a a few dirty ones. If we don't find enough clean pages, scan again and launder as many pages as possible.
- New algorithm: PQ_INACTIVE scans move dirty pages to the laundry (PQ_LAUNDRY). A dedicated thread launders pages depending on:
  - how quickly clean pages are being freed
  - ℓ(PQ_LAUNDRY)/ℓ(PQ_INACTIVE)
- Dedicated thread makes it harder to hit low-memory deadlocks

Avoiding TLB shootdows in `execve(2)`

- r311346 and r313756 by markj, D8921, D9586
- Massive overhead observed on 128-vCPU EC2 instances
  - `make -j128 buildkernel`
- Silly KVA management caused excessive IPIs
- Solution: cache `execve(2)` argument KVA and use `madvise(MADV_FREE)` to release backing pages when under memory pressure

```
commit 18b3f4573dd73f98b9b9716883eda65014196d59
Author: Matthew Dillon <dillon@dragonflybsd.org>
Date:   Thu Jun 7 23:14:29 2007 +0000

    Entirely remove exec_map from the kernel. ...
```

Global Lock Removals

- r299788 by kib, removed `pvh_global_lock` on amd64
- Replaced with "delayed invalidation" blocks
- Improves `pmap(9)` scalability (on amd64)

- r322913 by kib, removed `swhash_mtx`
- Global hash table for (`obj`, `pindex`) $\rightarrow$ `swblk` mappings
- Replaced with per-object trie, protected by the object lock
- Many VM operations required swap hash lookups

MAP_GUARD

- r320317 by kib, plus followups
- New `mmap(2)` flag added partly in response to Stack Clash
- Allows the creation of reservations in the virtual address space
- Access of a guard region raises `SIGSEGV`
- `mmap(2)` will not return a mapping in the region unless `MAP_FIXED` is used

Planned changes

- D11943 by markj: Avoid dequeuing pages in `vm_page_wire()`
- Frequent wiring and unwiring causes page queue lock contention
- Partial solution: lazily dequeue wired pages to reduce number of queue ops

- D13014 by jeff: NUMA awareness in the page allocator
- Takes us closer to fine-grained locking in the page allocator

Acknowledgements

Alan Cox (alc@FreeBSD.org)
Konstantin Belousov (kib@FreeBSD.org)