

Re: ten thousand small processes

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2003-06/0095.html>

From: Chuck Swiger (*cswiger_at_mac.com*)

Date: 06/26/03

Date: Wed, 25 Jun 2003 18:57:51 -0400

To: "D. J. Bernstein" <djb@cr.yp.to>

D. J. Bernstein wrote:

[...]

> *Why does the memory manager keep the stack separate from data? Suppose a program has 1000 bytes of data+bss. You could organize VM as follows:*

>

> *0x7fffac18 0x7fffb000 0x80000000*

> *<----- stack data+bss text, say 5 pages heap ----->*

>

> *As long as the stack doesn't chew up more than 3096 bytes and the heap*

> *isn't used, there's just one page per process.*

Remember that VMM hardware requires page-alignment: TEXT should be on pages marked X (or RX if the local architecture needs it), DATA+BSS should be RW, and I think FreeBSD needs the stack to be RWX. We need to consider the kernel's address space— 32-bit systems generally reserve the top 2GB, or sometimes less, exclusively for the kernel.

Besides, most programs are probably not built as PIC and could not have their starting address relocated arbitrarily, although perhaps it would be interesting to consider the following process address map:

VM Address Usage

0x0 PAGEZERO

0x4000 XXX bytes reserved per the hard limit to process stack size

XXX (+ 0x4000) TEXT segment

YYY DATA + BSS

ZZZ heap

0x80000000 KVA

> *As for page tables: Instead of allocating space for a bunch of nearly*

> *identical page tables, why not overlap page tables, with the changes*

> *copied on a process switch?*

Mach uses copy-on-write for VMO's associated with the virtual address space used by processes, which are similar abstractions to the page table entries used under classic BSD. From "man mmap":

freebsd-performance: Re: ten thousand small processes

The share mode describes whether pages are shared between processes, and what happens when pages are modified. Private pages (PRV) are pages only visible to this process. They are allocated as they are written to, and can be paged out to disk. Copy-on-write (COW) pages are shared by multiple processes (or shared by a single process in multiple locations). When the page is modified, the writing process then receives its own copy of the page. Empty (NUL) sharing implies that the page does not really exist in physical memory. Aliased (ALI) and shared (SHM) memory is shared between processes.

The share mode typically describes the general mode controlling the region. For example, as copy-on-write pages are modified, they become private to the application. Even with the private pages, the region is still COW until all pages become private. Once all pages are private, then the share mode would change to private.

The far left column names the purpose of the memory: text segment, data segment, allocated via malloc, stack, etc. For regions loaded from binaries, the far right shows the library loaded into the memory.

Some lines in vmmap's output describe submaps. A submap is a shared set of virtual memory page descriptions that the operating system can reuse between multiple processes. The memory between 0x70000000 and 0x80000000, for example, is a submap containing the most common dynamic libraries. Submaps minimize the operating system's memory usage by representing the virtual memory regions only once. Submaps can either be shared by all processes (machine-wide) or local to the process (process-only). If the contents of a machine-wide submap are changed -- for example, the debugger makes a section of memory for a dylib writable so it can insert debugging traps -- then the submap becomes local, and the kernel will allocate memory to store the extra copy.

8-cube# vmmap 252

==== Non-writable regions for process 252

```
__PAGEZERO 0 [ 4K] ---/--- SM=NUL syslogd
__TEXT 1000 [ 20K] r-x/rwx SM=COW syslogd
__LINKEDIT 7000 [ 4K] r--/rwx SM=COW syslogd
Submap 90000000-9fffffff r--/r-- machine-wide submap
__TEXT 90000000 [ 932K] r-x/r-x SM=COW ...System.B.dylib
__LINKEDIT 900e9000 [ 260K] r--/r-- SM=COW ...System.B.dylib
__TEXT 93a40000 [ 20K] r-x/r-x SM=COW ...Common.A.dylib
__LINKEDIT 93a45000 [ 4K] r--/r-- SM=COW ...Common.A.dylib
Submap a000b000-a3a3ffff r--/r-- process-only submap
Submap a3a41000-affffffff r--/r-- process-only submap
      aff80000 [ 512K] r--/r-- SM=SHM
```

==== Writable regions for process 252

```
__DATA 6000 [ 4K] rw-/rwx SM=PRV syslogd
MALLOC_USED(DefaultMallocZone_ 8000 [ 20K] rw-/rwx SM=COW
MALLOC_USED(DefaultMallocZone_ d000 [ 4K] rw-/rwx SM=ZER
MALLOC_USED(DefaultMallocZone_ e000 [ 4K] rw-/rwx SM=COW
```

Re: ten thousand small processes

freebsd-performance: Re: ten thousand small processes

```
MALLOC_FREE(DefaultMallocZone_ f000 [ 228K] rw-/rwx SM=ZER
__TEXT 8fe00000 [ 288K] rw-/rwx SM=COW /usr/lib/dyld
__DATA 8fe48000 [ 8K] rw-/rwx SM=COW /usr/lib/dyld
__DATA 8fe4a000 [ 4K] rw-/rwx SM=COW /usr/lib/dyld
__DATA 8fe4b000 [ 4K] rw-/rwx SM=ZER /usr/lib/dyld
__DATA 8fe4c000 [ 12K] rw-/rwx SM=COW /usr/lib/dyld
__DATA 8fe4f000 [ 144K] rw-/rwx SM=ZER /usr/lib/dyld
__LOCK 8fe73000 [ 4K] rw-/rwx SM=NUL /usr/lib/dyld
__LINKEDIT 8fe74000 [ 44K] rw-/rwx SM=COW /usr/lib/dyld
Submap 90000000-9ffffff r--/r-- machine-wide submap
__DATA a0000000 [ 4K] rw-/rw- SM=ZER ...System.B.dylib
__DATA a0001000 [ 4K] rw-/rw- SM=COW ...System.B.dylib
__DATA a0002000 [ 20K] rw-/rw- SM=COW ...System.B.dylib
__DATA a0007000 [ 16K] rw-/rw- SM=PRV ...System.B.dylib
Submap a000b000-a3a3ffff r--/r-- process-only submap
__DATA a3a40000 [ 4K] rw-/rw- SM=COW ...Common.A.dylib
Submap a3a41000-afffffff r--/r-- process-only submap
STACK[0] bff80000 [ 508K] rw-/rwx SM=PRV
      bffff000 [ 4K] rw-/rwx SM=PRV
```

==== Legend

SM=sharing mode:

COW=copy_on_write PRV=private NUL=empty ALI=aliased
SHM=shared ZER=zero_filled S/A=shared_alias

==== Summary for process 252

ReadOnly portion of Libraries: Total=1572KB resident=1444KB(92%) swapped_out_or_unallocated=128KB(8%)

Writable regions: Total=968KB written=40KB(4%) resident=88KB(9%) swapped_out=0KB(0%) unallocated=880KB(91%)

> *As for 39 pages of VM, mostly stack: Can the system actually allocate
> 390000 pages of VM?*

I believe 390000 4K pages is 1523 MB: if you've got the datasize resource limit set high enough and you've got the RAM or swap space available, the answer to your question should be yes.

> *I'm only mildly concerned with the memory-management
> time; what bothers me is the loss of valuable address space. I hope that
> this 128-kilobyte stack carelessness doesn't reflect a general policy of
> dishonest VM allocation ("overcommitment"); I need to be able to
> preallocate memory with proper error detection, so that I can guarantee
> the success of subsequent operations.*

Preallocate at compile time, or preallocate at process run time?

> *As for malloc()'s careless use of memory: Is it really asking so much
> that a single malloc(1) not be expanded by a factor of 16384?
>
> Here's a really easy way to improve malloc(). Apparently, right now,*

Re: ten thousand small processes

freebsd-performance: Re: ten thousand small processes

- > *there's no use of the space between the initial brk and the next page*
- > *boundary. Okay: allocate that space in the simplest possible way---*

It's easy to write a memory allocator that performs a specific case well; writing a general purpose malloc is significantly more complicated, and FreeBSD's malloc is tuned for programs which are much larger than your example.

If you know of a malloc() implementation that does better than FreeBSD's, and is suitable for SMP systems, let us know.

-Chuck

freebsd-performance@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-performance>

To unsubscribe, send any mail to "freebsd-performance-unsubscribe@freebsd.org"