

## Re: ten thousand small processes

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2003-06/0092.html>

---

**From:** Jon Mini ([mini\\_at\\_freebsd.org](mailto:mini_at_freebsd.org))

**Date:** 06/25/03

Date: Wed, 25 Jun 2003 03:29:30 -0700

To: "D. J. Bernstein" <[djb@cr.yp.to](mailto:djb@cr.yp.to)>

D. J. Bernstein [[djb@cr.yp.to](mailto:djb@cr.yp.to)] wrote :

- > *In case my subject line isn't clear enough: I'm not talking about*
- > *browsers. I'm not talking about programs that ``use up stack space," or*
- > *that would feel even slightly ``restricted" by a 1GB limit on the heap,*
- > *or that use threads, or that allocate many file descriptors per process.*

However, you *are* talking about modifying the behaviour of a system that normally runs processes like this. This is a general-purpose operating system. We have to consider the impact of any changes we make on all types of loads, but primarily the most common ones. Especially the most common performance-critical ones. That means processes which do not behave at all like what you are talking about.

If FreeBSD primarily ran very small processes that only consumed a few kilobytes of state, then it would be designed very differently, and the problem you are describing would not exist. However, that is not the case.

- > *I'm talking about `_small_` processes. I'm talking about programs that*
- > *might have quite a bit of code, and might read and write quite a lot of*
- > *data, but that don't use much memory per process. The problem is that*
- > *I'm talking about ten thousand of these processes running at once.*

You are going to run in a far larger list of problems when trying to run orders of tens of thousands of processes on FreeBSD. The fact of the matter is not that FreeBSD might have poorly designed implementation details, but rather that a large-scale architectural decision was made among the UNIX community at large to, simply put, not support that.

The fact of the matter is that a process is a heavy-weight entity. The resource consumption you are describing in userland is least of your worries. In the kernel many more resources are consumed on a per-process bases and, unfortunately, there are a handful of algorithms that are going to have poor growth characteristics with

## freebsd-performance: Re: ten thousand small processes

that many processes active on the system. The impact on the scheduling subsystem alone is probably worth several months of research.

You are not going to find much of a positive response here for your use case. What you suggest would take many hours of development time in order to provide gain only on use cases that are (a) not common, and (b) not helpful. Simply put, we don't want users to make tens of thousands of small processes. This, among other things, is what threads are for.

> *In this situation, if malloc() fritters away 24K of RAM in one process,  
> it's actually frittering away more than 234 \_megabytes\_ of RAM. This  
> memory is \_not\_ going to be ``used later." The behavior of malloc()  
> here is not ``quite good." For the program that prompted this thread,  
> malloc() achieved 4.3% fill, 95.7% external fragmentation.*

Congratulations! You've successfully shown that there is no "One True Memory Allocator" that allocates perfectly in all cases, and that memory fragmentation can become terribly wasteful!

Our malloc(3) exists in order to solve a very different sort of problem than the one you are trying to solve: the common case for FreeBSD, which is large processes that perform many allocations of wildly varying sizes. I suggest you read Paul's paper on his malloc(3) implementation and the problems he was trying to solve. You can find a copy in /usr/share/doc/papers/malloc.ascii.gz.

> *In this situation, squeezing variables into a single page---rather than  
> spreading them among the first few bytes of several separate pages---  
> often produces huge improvements in CPU cache effectiveness. Yes, I know  
> that doesn't make a difference for your browser; I'm not talking about  
> your browser.*

You have an economy of scale problem here. One moment, you argue that you want to create a system where you have tens of thousands of processes, and that wasting a few pages per process is unacceptable because there are so many of them. Then, in the next moment, you say that this is also bad because of cache behaviour?

I'm sorry, but you are way off here. First of all, caches are \*much larger\* than the size of the processes you are talking about. Second of all, every time you perform a context switch between processes, you must flush your cache. Given that you are talking about "tens of thousands" of processes, one would imagine that you would be context-switching constantly. Cache behaviour would definately be a problem in that system, but it's not because of malloc(3).

It sounds very much like you are trying to use the wrong solution here, and that what you want to do will never work well. Perhaps a better approach would be to try using (in order of weight):

Re: ten thousand small processes

