

## Re: hot path optimizations in uma\_zalloc() & uma\_zfree()

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2005-06/0392.html>

---

**From:** John Baldwin (*jhb\_at\_FreeBSD.org*)

**Date:** 06/30/05

To: freebsd-hackers@freebsd.org

Date: Thu, 30 Jun 2005 09:55:09 -0400

On Wednesday 29 June 2005 06:08 pm, ant wrote:

> *Hi folks.*  
>  
> *I just tried to make buckets management in perCPU cache like in*  
> *Solaris (see paper of Jeff Bonwick – Magazines and Vmem)*  
> *and got performance gain around 10% in my test program.*  
> *Then i made another minor code optimization and got another 10%.*  
> *The program just creates and destroys sockets in loop.*  
>  
> *I suppose the reason of first gain lies in increasing of cpu cache hits.*  
> *In current fbsd code allocations and freeings deal with*  
> *separate buckets. Buckets are changed when one of them*  
> *became full or empty first. In Solaris this work is pure LIFO:*  
> *i.e. alloc() and free() work with one bucket – the current bucket*  
> *(it is called magazine there), that's why cache hit rate is bigger.*  
>  
> *Another optimization is very trivial, for example:*  
> *– bucket->ub\_cnt--;*  
> *– item = bucket->ub\_bucket[bucket->ub\_cnt];*  
> *+ item = bucket->ub\_bucket[--bucket->ub\_cnt];*  
> *(see the patch)*

This produces no change in the object code though as gcc is smart enough to implement both cases the same.

I ran ministat against your tests with 1000 sockets loop and there isn't a lot of difference in the user times:

```
x one.u
+ two.u
* three.u
```

```
+-----+
|* * * x x + x ++ |
|_____A_M_____||_____M_A_____||_____A_M_____||
+-----+
```

freebsd-hackers: Re: hot path optimizations in uma\_zalloc() & uma\_zfree()

```
N Min Max Median Avg Stddev
x 3 0.085 0.117 0.093 0.098333333 0.016653328
+ 3 0.101 0.125 0.124 0.11666667 0.013576941
No difference proven at 95.0% confidence
* 3 0.054 0.077 0.07 0.067 0.011789826
No difference proven at 95.0% confidence
```

There is some difference in the system times though:

```
> /usr/src/tools/tools/ministat/ministat one.s two.s three.s
```

```
x one.s
```

```
+ two.s
```

```
* three.s
```

```
+-----+
|* * * ++ + x x x|
||_A_| |MA_| |_____A_____||
+-----+
```

```
N Min Max Median Avg Stddev
x 3 2.689 2.797 2.739 2.7416667 0.05404936
+ 3 2.373 2.403 2.381 2.3856667 0.015534907
Difference at 95.0% confidence
  -0.356 +/- 0.0901334
  -12.9848% +/- 3.28754%
  (Student's t, pooled s = 0.039766)
* 3 2.298 2.34 2.32 2.3193333 0.021007935
Difference at 95.0% confidence
  -0.422333 +/- 0.0929396
  -15.4043% +/- 3.38989%
  (Student's t, pooled s = 0.0410041)
```

There is also some difference between the second and third runs though not huge. Given that your ++ and -- optimizations don't produce any actual changes in the compiled code I think that is more due to your testing environment. A much better test might be to reboot into single user mode and run the socket loop program with 10000 or 100000 sockets and run that in a loop 20 times for each case and log the output. That should give you some better benchmarks with less noise from your environment. If you haven't used ministat before (/usr/src/tools/tools/ministat), it's a neat little tool. You just hand it data files that contain one number per line and it does the work for you. Also, given that here are SMP implications here, it might be fruitful to have another benchmark which spawned several threads (one per CPU maybe?) and had each of them sit in the same loop.

--

John Baldwin <jhb@FreeBSD.org> <>< <http://www.FreeBSD.org/~jhb/>  
"Power Users Use the Power to Serve" = <http://www.FreeBSD.org>

---

freebsd-hackers@freebsd.org mailing list

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

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