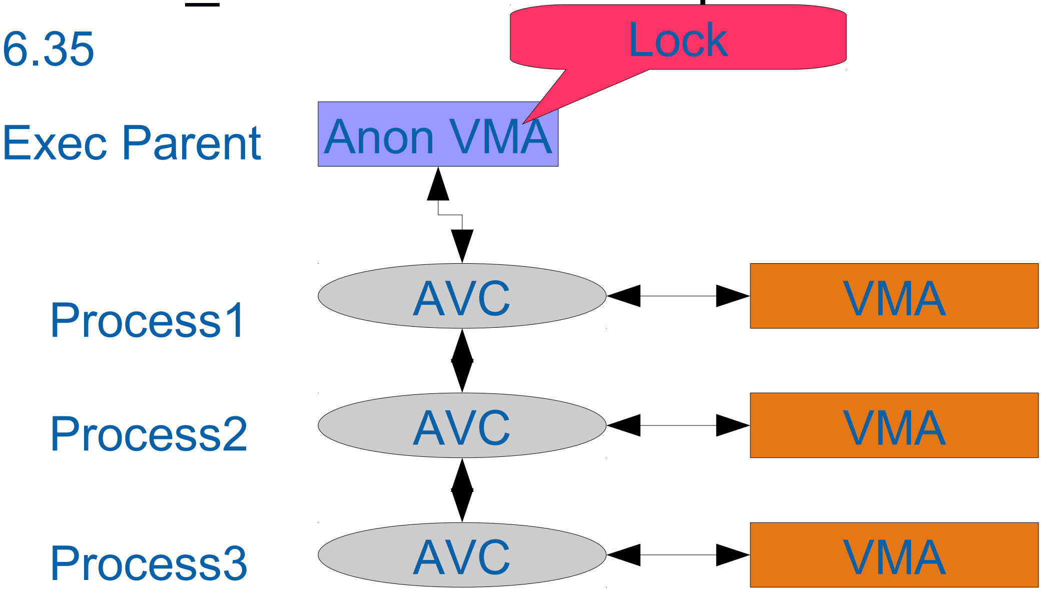
Scaling problems in Fork

Andi Kleen and Tim Chen

Sep 2011

anon_vma Chains

- Anon vma chains are used for copy-on-write and rmap
- Shared between processes
- Locking is done in the anon_vma of the “exec parent”
  - This has changed in 2.6.35
  - Severe regressions



Problem: Changing Child Address Space

- Mmap/munmap/brk try to merge/split vmas
- Requires taking the root anon vma chain lock
- Causes lock contention on the root

Problem: Locking in fork() Itself

- Fork locks the root anon_vma for every VMA
- Lots of overhead in root locking
  - With and without contention
- Mitigated by batching: reuse the lock if the previous VMA had the same one (3.0)
  - In this case making lock hold time longer increased performance
- However, spinlock->mutex change in 3.0 caused severe regression again

3.0 MM locking regression:

MOSBENCH exim workload

2.6.39(vanilla) 100.0%

2.6.39+ra-fix 166.7% (+66.7%)

Anon VMA lock change in 3.0 (spin lock -> mutex)

3.0-rc2(vanilla) 68.0% (-32%)

After a lot of tweaking from Linus and others:

3.0-rc2+fixes 140.3% (+40.3%)

(anon_vma clone + unlink + chain_alloc_tweak)

Lost 26% again compared to 2.6.39+rafix

anon_vma fragmentation of vma List

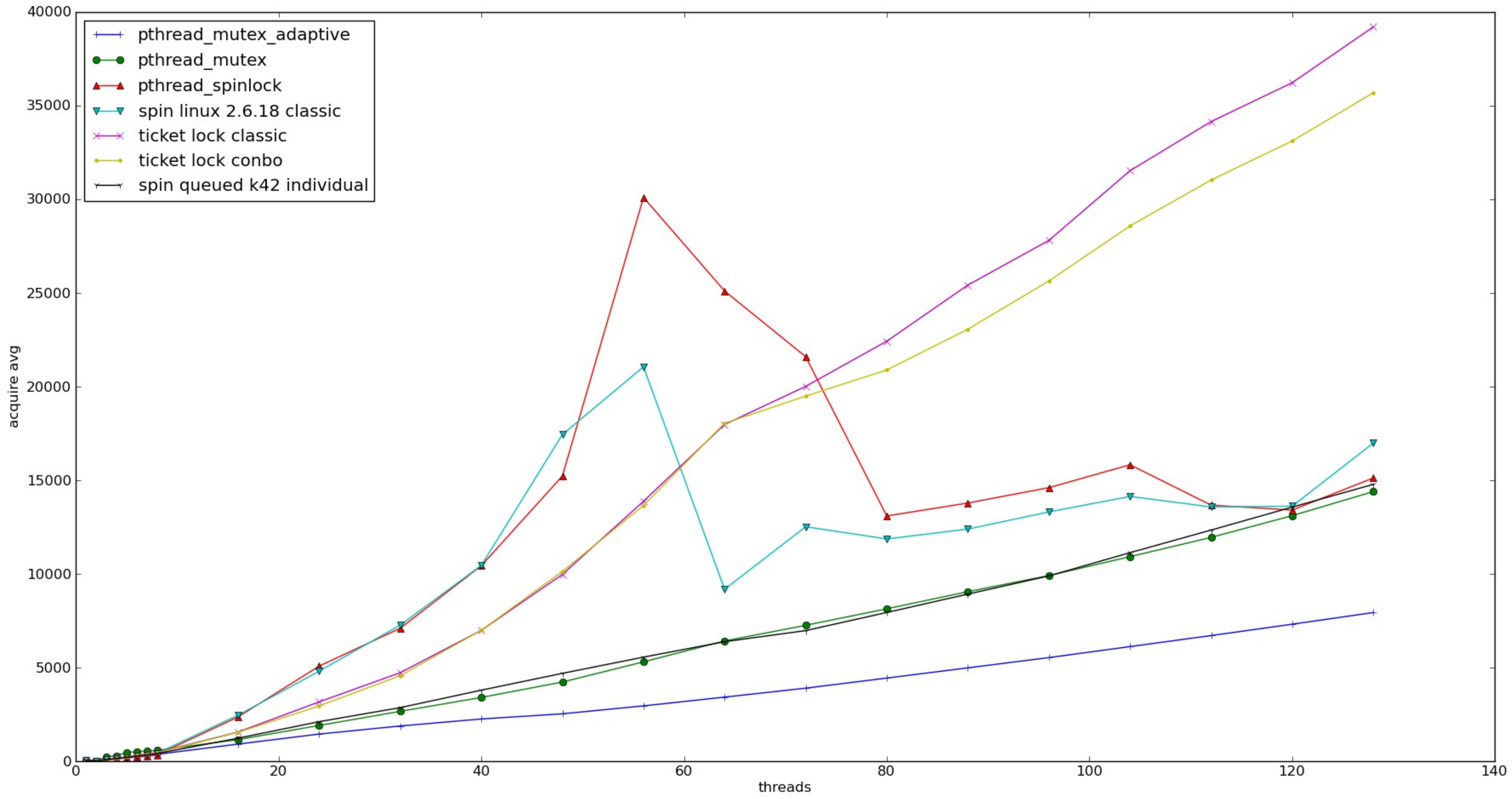
- Mmap anonymous memory space of 10 GB, create holes in this mmap region that are 4 pages wide by unmapping 4 pages of memory every 8 pages.
- We get a list of up to 327680 vmas associated with the anon_vma that we started out with!
- Traversing the same anon_vma list then becomes very expensive
  - And this holds a lock
- __split_huge_page in transparent huge page daemon goes through the entire list of vmas to locate the vma associated with the page

Fork Summary

- Anon vma chains are the main scalability problem in fork
  - Causes problems in other areas too, like Transparent Hugepages
- They can become too long
- And too coarse grained locking
- Need a new data structure?

Locking Primitives

Lock Primitives in a Micro Benchmark



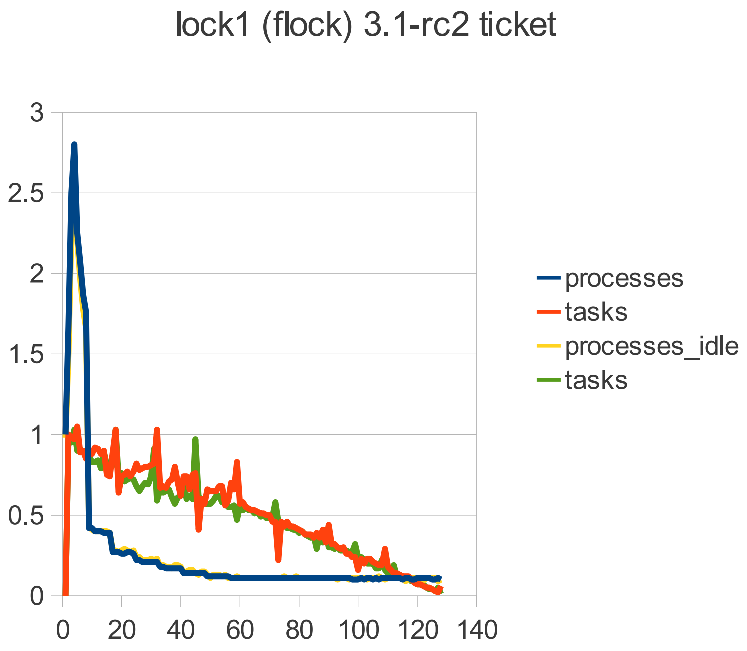
CONBO Lock

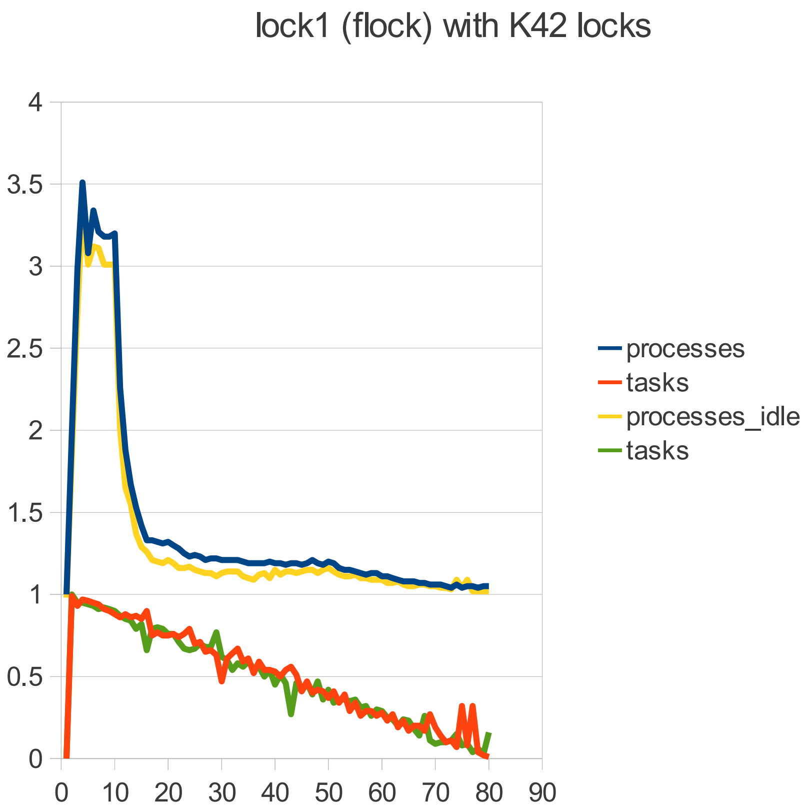
- “CONservative Backoff lOck”
- Idea: use lock backoff, but “do no harm”
- Backoff based on the ticket difference to minimize unnecessary backoffs
- Not a gigantic win over normal lock
  - But nice improvements with many threads
  - And does no harm

K42 Lock

- Queued spin lock that spins locally
- K42 variant of MCS lock
- Performs much better than ticket locks on 4-8S
  - Not much difference on 1S/2S
  - Still too fair in thread region
- Uncontended slightly slower (cmpxchg in unlock)
- Lock grows 4->8/16 bytes
  - Could be a problem for some data structures

K42: file locking micro





Lock Conclusion

- Time to reevaluate the locks?
- We can do better on 4+S at some cost

Backup

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

All dates provided are subject to change without notice.

Intel is a trademark of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.