Benchmarking memory allocators

Julien Voisin — dustri.org

BLACK ALPS

Story time

- My infra at home is running on Alpine Linux, using musl libc.
- Musl libc has its own memory allocator.
- Musl old allocator used to suck apparently.
- Fortunately, musl libc's malloc-ng is better™.
- But what does better™ actually means?
  - Speed?
  - Security?
  - Simplicity?
  - RAM consumption?
  - CPU consumption?
  - Locks contention/scalability?
- Are there more better™ generic userland allocators around?

Measuring performances

I’m sure someone already did this…

- Daan Leijen from Microsoft published mimalloc-bench
- Written in bash, but surprisingly nice and clean.
- Had some benchmarks, and a couple of allocators.
- I simply added **moar moar moar!**

Benchmarks

- Real life and real-life-ish workloads:
  - redis, ghostscript, z3, lean, rocksdb, gcc(lua), espresso, barnes, ….
- Tons of academic ones used in various papers:
  - cfrac, espresso, larsonN, sh6bench/sh8bench, rbstress, mstress, …
- Running on every commit via github actions on:
  - ubuntu, fedora, alpine and osx.

Benched allocators

- dieharder: error-resistant memory allocator
- ffmalloc: from the Usenix Security 21 paper
- freeguard: a Faster Secure Heap Allocator
- guarder: tunable secure allocator by the UTSA.
- hoard: one of the first multi-thread scalable allocators.
- hardened_malloc: security-focused, from GrapheneOS
- isoalloc: isolation-based aiming at providing a reasonable level of security without sacrificing too much the performances.
- jemalloc: by Jason Evans, now developed at Facebook and widely used eg. FreeBSD and Firefox
- libpas: used by WebKit since 2022
- lockfree-malloc: the world's first Web-scale memory allocator
- ltalloc: LightweighT Almost Lock-Less Oriented for C++ programs memory allocator
- musl: musl's memory allocator since 2020
- mesh/nomesh: allocator that automatically reduces the memory footprint of applications
- mimalloc/smimalloc: compact general purpose allocator with excellent performance, used by UnrealEngine, Azure, Bing, …
- rpmalloc: 16-byte aligned allocations by Mattias Jansson at Epic Games, used by Haiku
- scalloc: fast, multicore-scalable, low-fragmentation memory allocator
- scudo: used by Fuschia and Android.
- slimguadr: secure and memory-efficient.
- supermalloc: uses hardware transactional memory to speed up parallel operations.
- snmalloc: concurrent message passing allocator
- Intel TBB: from the Thread Building Blocks (TBB) library
- tcmalloc: maintained by the community
- tcmalloc: maintained and used by Google.
- native: uses the system allocator, usually glibc.

Results and shiny graphs

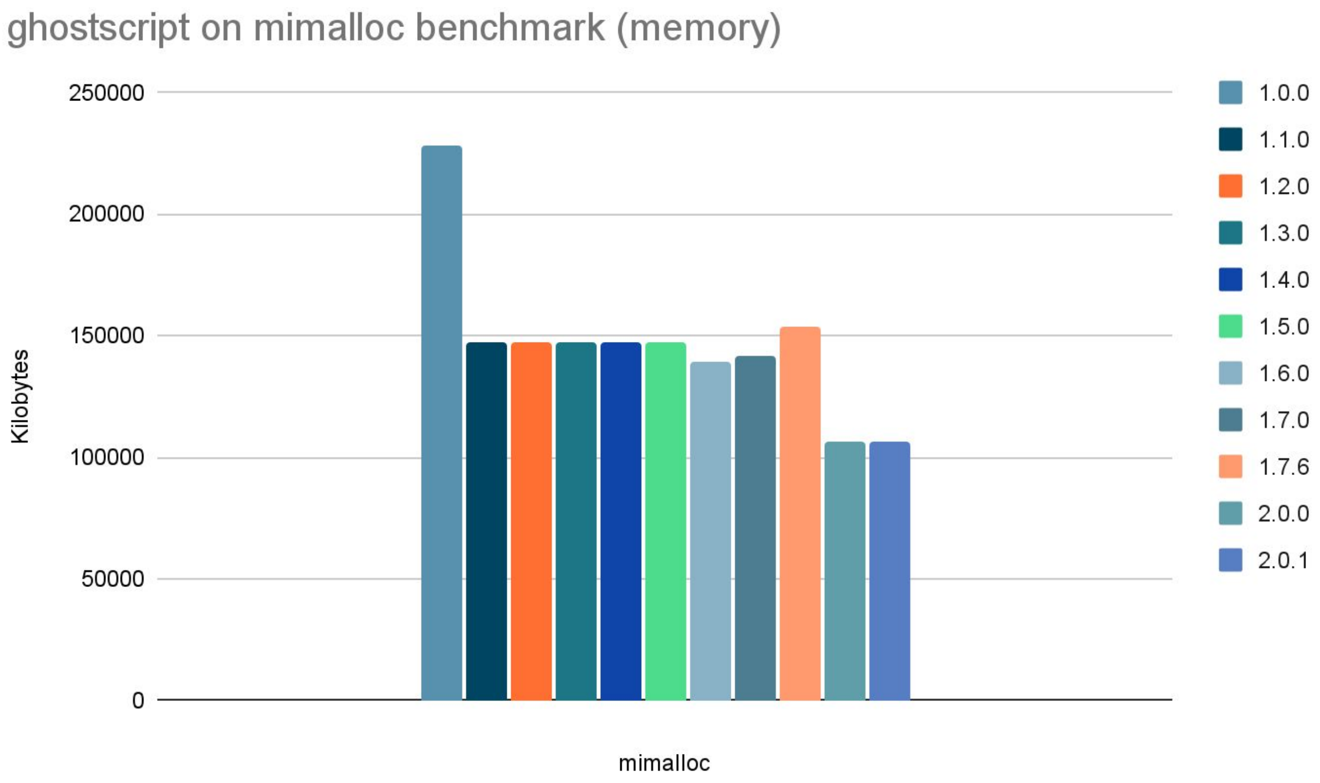
Some sad results

- Most allocators are Linux-specific.
- Some allocators are glibc-specific.
- Some are *conferenceware* and don’t even compile.
- Some were too slow to be included in the CI.
  - Some allocators explicitly don’t care about performances.
- Some are crashing when running benchmarks.

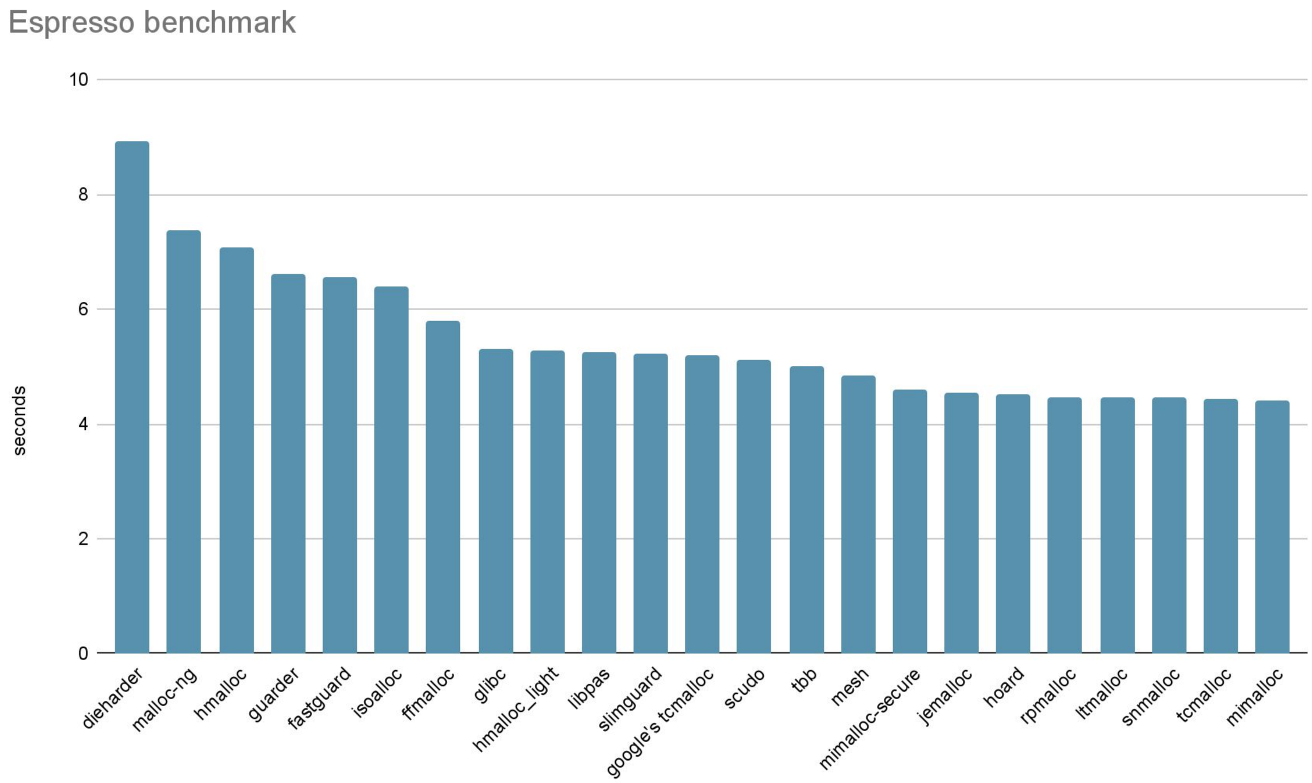
Side-effect improvements

- Caught a crash in isoalloc
- Security improvements in snmalloc
- Portability improvement in Intel TBB
- Caught a compilation issue in rpmalloc
- Minor performances improvement in isoalloc
- Portability improvements in Google's tcmalloc
- Added parallel compilation support in DieHarder

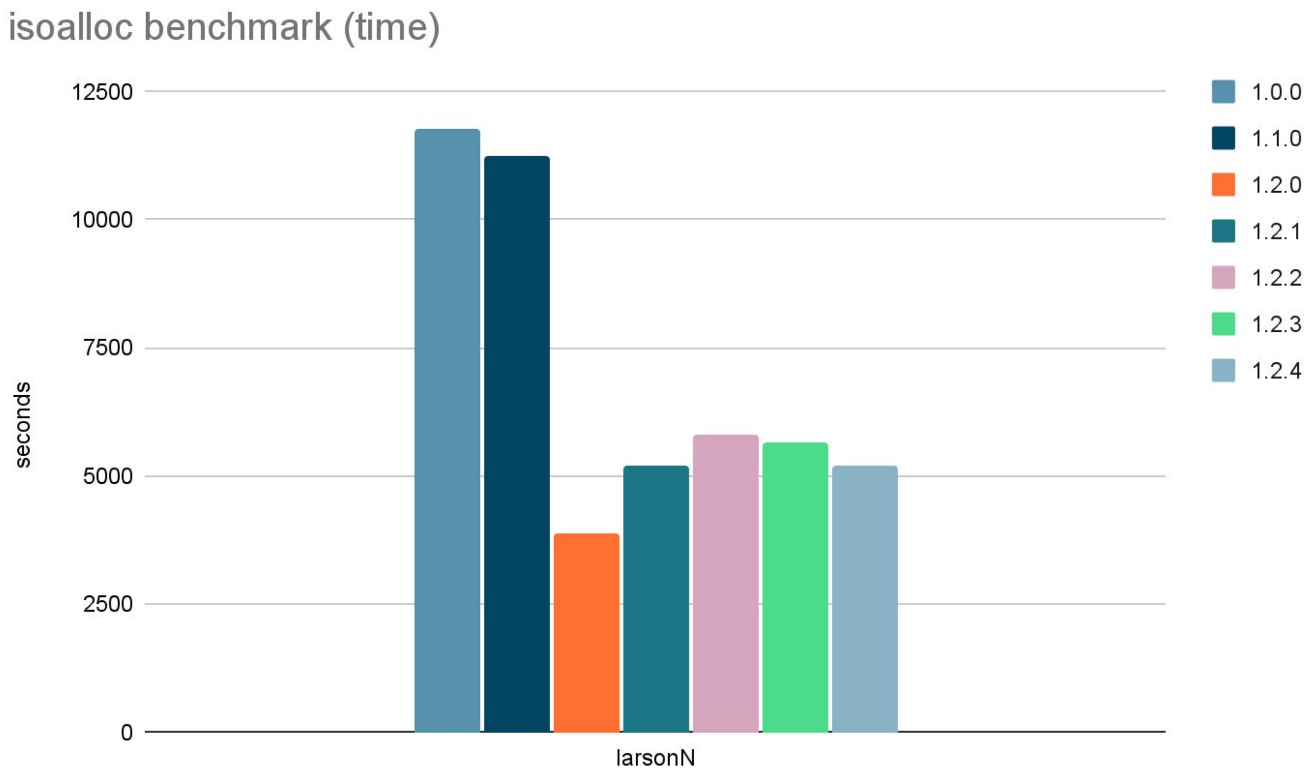
Example: Memory used (lower is better)



Example: Pretty graphs (lower is better)



Example: Time taken (lower is better)



Measuring security

Spatial and type safety

- chunks alignment
- elastic objects isolations
- checksums for inline metadata
- (permanent) size/type based partitioning
- randomization: makes everything harder
- invalid free detection: overlapping chunks
- guard pages: catches large linear-overflows
- elastic objects isolations: complicates/mitigates UAF
- chunks alignment: mitigates some overlapping chunks
- size/type based partitioning: complicates/mitigates UAF
- (non global) canaries/cookies: catches some linear overflows
- (read-only) OOB metadata: kills all the house-of-… techniques
- …

Temporal safety

- double-free detection: kills… double-free
- sanitization on free: mitigates some infoleaks/UAF
- sanitization on allocation: mitigates some infoleaks/UAF
- delayed-free: makes UAF exploitation/heap-spraying harder
- multi-queues free: makes UAF exploitation/heap-spraying harder
- quarantines: makes UAF exploitation/heap-spraying harder
- …

Memory tagging

- Software
  - Fat pointers
- Hardware
  - Complicated topic, out-of-scope for this talk

Exotic stuff and specific mitigations

- gigacages
- safe-unlink
- CPU pinning
- lack of free-list
- permanent frees
- guarded `memcpy`
- PAX_MPROTECT-like
- elastic-objects isolation
- GWP-ASAN-like sampling
- reference-counting: BackupRefPtr
- zero-sized allocations special handling
- pointer obfuscation/encryption/mangling
- dangling-pointers detection: DCScan/PCScan/…
- …

It's almost as if benchmarking security was nontrivial.

- Ticking a lot of boxes ≠ a lot of security.
- Tight integration allows powerful pervasive mitigations
- With arbitrary r/w, ~all bets are off without hardware assistance.
- Beware of detection vs. neutering design choices.
- The security/performance function is roughly x^2:
  - Diminishing returns are plenty.
  - ~~Waste~~ spend your budget wisely.
  - Designing mitigations is hard:
    - Beware of the MitiGator!
    - Follow halvar's rule

Now what?

- Add more allocators
  - Is your favourite one missing?
- Add more (relevant) benchmarks
  - Ideas and suggestions are welcome.
- Publish more shiny graphs and data
  - What kind of metrics are interesting/relevant?
- Drive adoption of systematic benchmarks forward
  - CS papers without code shouldn’t be a thing.

Heavily subjective and biased conclusion

- mimalloc is great
- hardened_malloc or isoalloc if you want “security”
- The default allocator is usually good enough™

~All big software and interpreted languages have their own allocator anyway:

  - apache2, nginx, python, java, php, go, firefox, thunderbird, chrome, exim, …

Thanks!

Sources and cool things to check out

- https://github.com/daanx/mimalloc-bench
- https://github.com/struct/isoalloc/blob/master/SECURITY_COMPARISON.MD
- https://downloads.immunityinc.com/infiltrate-archives/webkit_heap.pdf
- https://security.apple.com/blog/towards-the-next-generation-of-xnu-memory-safety/