Fork à la carte für In-Memory-Datenbanken

Mario Mintel*, Ralf Ramsauer*, Daniel Lohmann†, Stefanie Scherzinger‡, Wolfgang Mauerer*§





* Ostbayerische Technische Hochschule Regensburg
† Gottfried Wilhelm Leibniz Universität Hannover
‡ Universität Passau
§ Siemens AG, Corporate Research and Technology, München

Frühjahrstreffen der Fachgruppen Betriebssysteme, Hamburg

17. März 2022

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

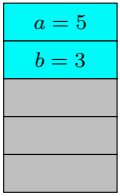
Leibniz
Universität
Hannover

UNIVERSITÄT
PASSAU

```
:(){ :|:& };:
```

System Call `fork()`

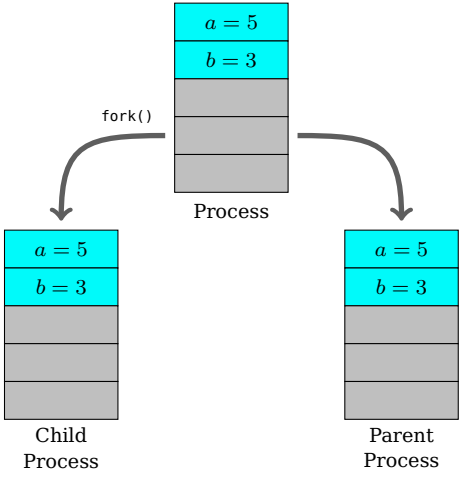
- Process duplication
- Has own virtual address space
- Typically uses Copy-on-Write



Process

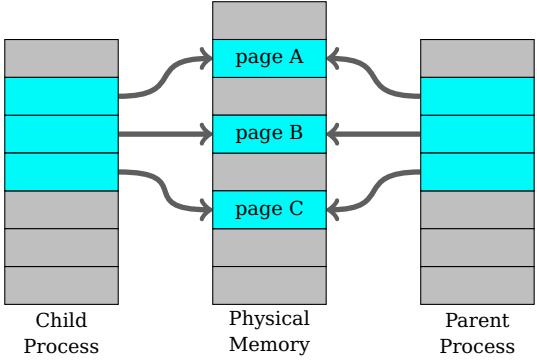
System Call `fork()`

- Process duplication
- Has own virtual address space
- Typically uses Copy-on-Write



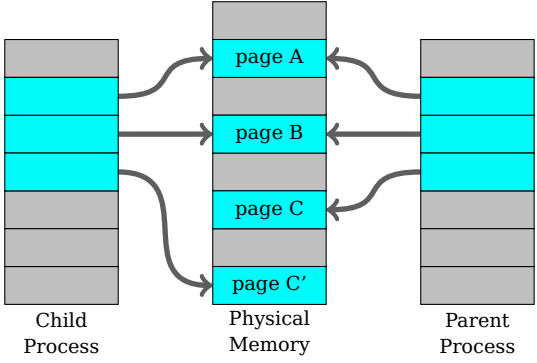
Copy-on-Write Mechanism

- Efficient copy operation on modifiable resources
- Reduces resource consumption of unmodified copies
- Adds an overhead to modifying operations



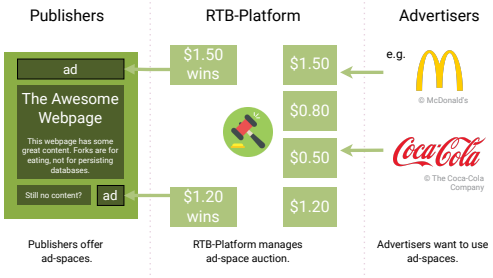
Copy-on-Write Mechanism

- Efficient copy operation on modifiable resources
- Reduces resource consumption of unmodified copies
- Adds an overhead to modifying operations



In-Memory Databases:

- Real-Time Bidding
- Caching
  - Results of database queries
  - Session data
  - High-frequency data like images, files, metadata
- In general: Latency-critical applications [1]
- Problem: Persistence



[1] Jeffrey Dean and Luiz André Barroso. “The Tail at Scale”. In: *Communications of the ACM* 56 (2013), pp. 74–80. URL: http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

Redis [2]: Persistence-Options

- AOF (Append-Only File):
  - Every second (fast but unsafe)
  - Every write (safe but slow)
- Snapshot:
  - Saves memory content
  - Exploits `fork()`

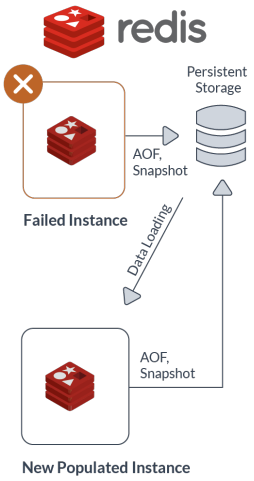


Image © Redis

[2] Redis Ltd. *Redis*. URL: `https://redis.io` (visited on 03/15/2022)

HyPer [3]: Hybrid OLTP & OLAP

- OLTP (Online Transactional Processing)
- OLAP (Online Analytical Processing)
- Simultaneous use of OLTP + OLAP exploiting `fork()`
- Persistence ensured exploiting `fork()`

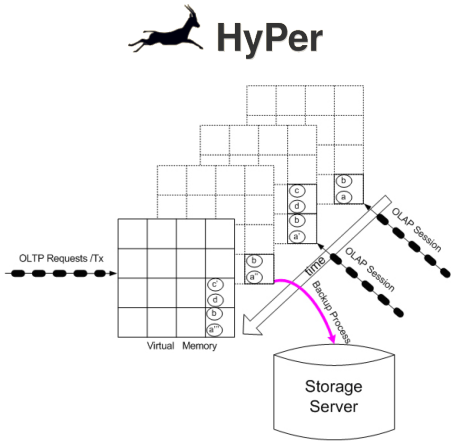
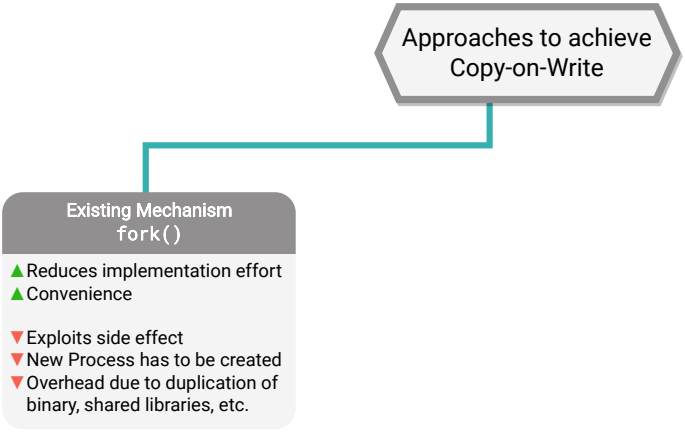
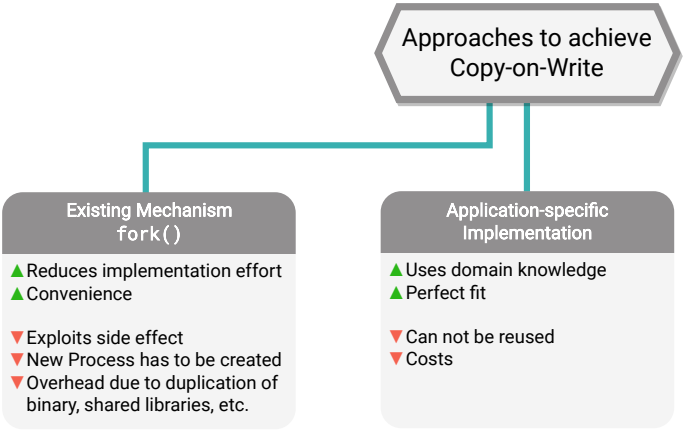


Image © Kemper and Neumann

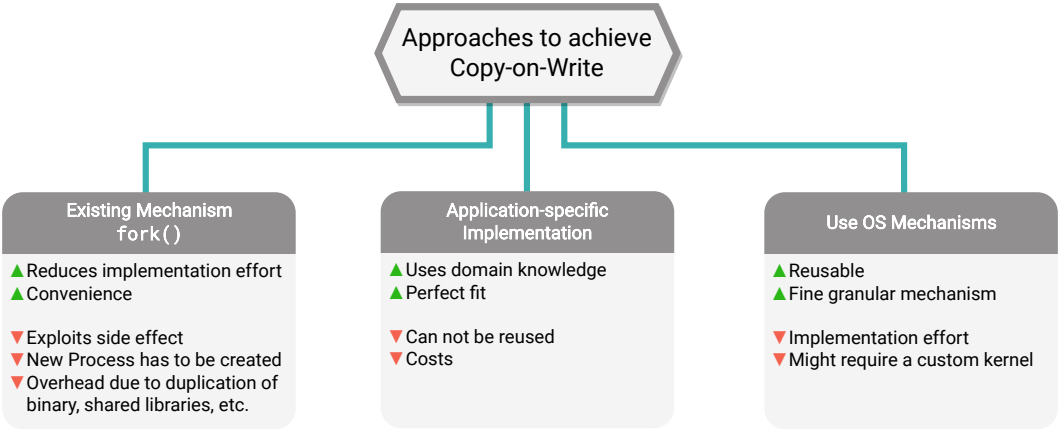
[3] Alfons Kemper and Thomas Neumann. “HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots”. In: *2011 IEEE 27th International Conference on Data Engineering*. IEEE. 2011, pp. 195–206

Approaches to achieve Copy-on-Write

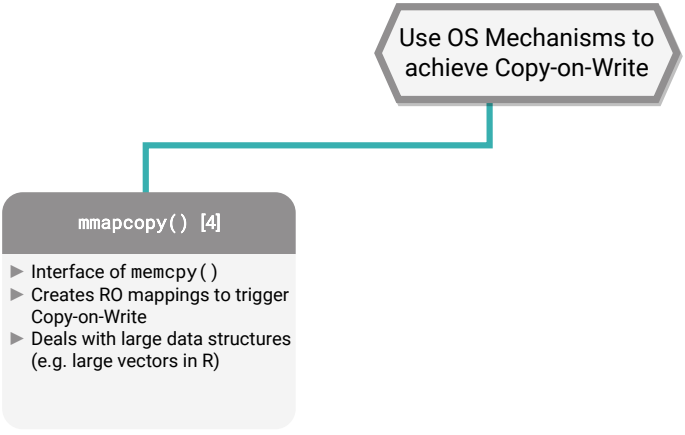

Approaches to achieve Copy-on-Write
Existing Mechanism
fork()
▲Reduces implementation effort
▲Convenience
▼Exploits side effect
▼New Process has to be created
▼Overhead due to duplication of binary, shared libraries, etc.


Approaches to achieve Copy-on-Write
Existing Mechanism
fork()
▲Reduces implementation effort
▲Convenience
▼Exploits side effect
▼New Process has to be created
▼Overhead due to duplication of binary, shared libraries, etc.
Application-specific Implementation
▲Uses domain knowledge
▲Perfect fit
▼Can not be reused
▼Costs

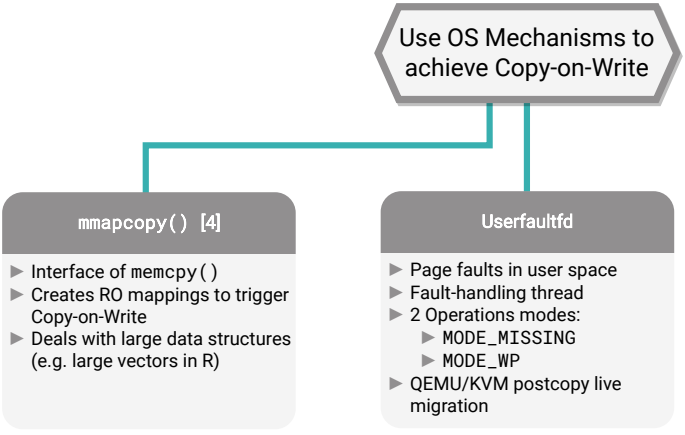
Is `fork()` the best choice?



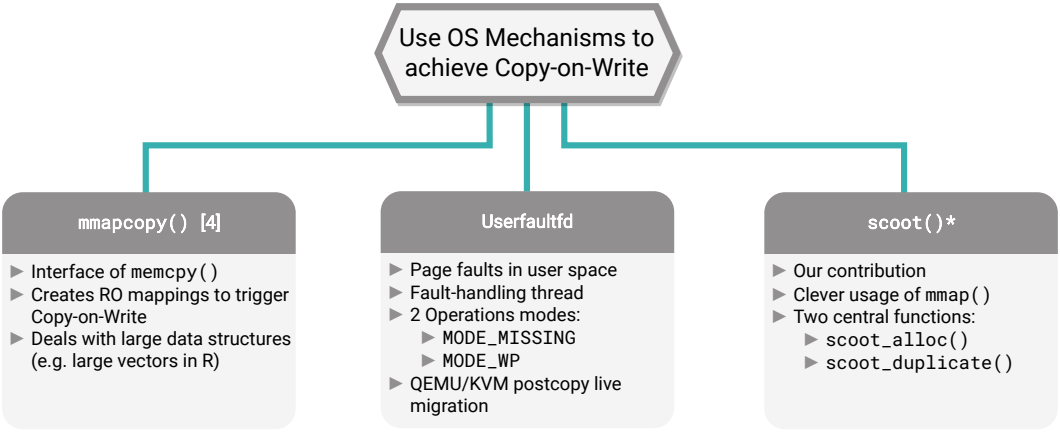
Use OS Mechanisms to achieve Copy-on-Write



[4] Ingo Korb, Helena Kotthaus, and Peter Marwedel. “mmapcopy: efficient memory footprint reduction using application knowledge”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, pp. 1832–1837

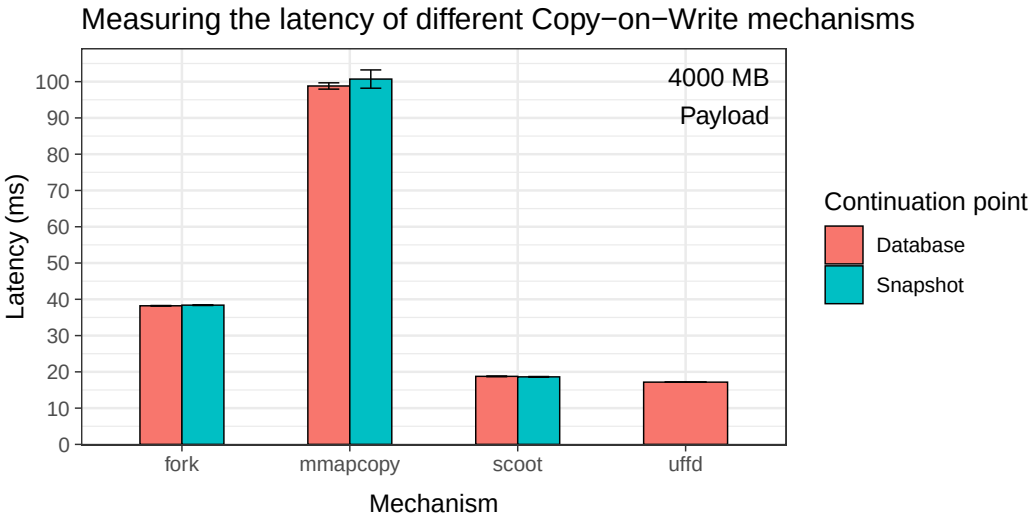


[4] Ingo Korb, Helena Kotthaus, and Peter Marwedel. “mmapcopy: efficient memory footprint reduction using application knowledge”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, pp. 1832–1837



* Editor's Choice

[4] Ingo Korb, Helena Kotthaus, and Peter Marwedel. "mmapcopy: efficient memory footprint reduction using application knowledge". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, pp. 1832–1837


Measuring the latency of different Copy−on−Write mechanisms
4000 MB
Payload
Latency (ms)
0
10
20
30
40
50
60
70
80
90
100
fork
mmapcopy
scoot
uffd
Mechanism
Continuation point
Database
Snapshot

Conclusion

- Fork is often exploited for its Copy-on-Write memory duplication
- Need for dedicated Copy-on-Write mechanism
- Scoot has potential to reduce latency significantly

Conclusion

- Fork is often exploited for its Copy-on-Write memory duplication
- Need for dedicated Copy-on-Write mechanism
- Scoot has potential to reduce latency significantly

Future Work

- Introduce scoot to redis
- Measure net gain in real environment
- Develop ScooterDB

Thank you!

<mario.mintel@st.oth-regensburg.de>

Measuring Environment

- Linux Kernel v5.14
- Intel(R) Xeon(R) Gold 5118 CPU @2.30GHz
- 32GB Single Channel DDR4 RAM @2666Mhz

Measuring Tool

```
Usage: ./measure -m {scoot|fork|mmapcopy|uffd} [-s size_in_mb] [-p]
                [-t] [-x db/snap] [-v]

Options:
    -m: Allocation method {scoot|fork|mmapcopy|uffd}
    -s: Allocation size in megabytes [default: 1]
    -p: Populate pages when allocating the VMA via mmap
    -t: Single shot run with verbose output to test duplication
    -x: Measurement point; either snapshot or db continuation point
    -v: Verbose output
```

Measuring Environment

- Linux Kernel v5.14
- Intel(R) Xeon(R) Gold 5118 CPU @2.30GHz
- 32GB Single Channel DDR4 RAM @2666Mhz

Measuring Tool

```
$ ./measure -m scoot -s 2048 -p -x db -t
```

```
My PID: 2204
Working on 2048MiB (524288 pages @4096B)
Allocated 524288 pages at 0x600000000000
Possible duplicate location: 0x600080000000
Moved memory from 0x600000000000 -> 0x600080000000 to create duplicate
Newly created origin is behind our old origin confirmed
Origin virtual address: 0x600000000000
Duplicate virtual address: 0x600080000000
9566657ns (9.57ms)
Origin:
    0x600000000000 -> 0x00000001242a3000
Duplicate:
    0x600080000000 -> 0x00000001242a3000
Testing deduplication by writing to origin
Origin first page: 0x55
Copy first page: 0x0
Origin:
    0x600000000000 -> 0x00000001b2d87000
Duplicate:
    0x600080000000 -> 0x00000001242a3000
```