Dynamic Context-Based Code Elimination

Florian Rommel   Daniel Lohmann

Leibniz Universität Hannover

2021-09-22

Binary Tailoring and Debloating

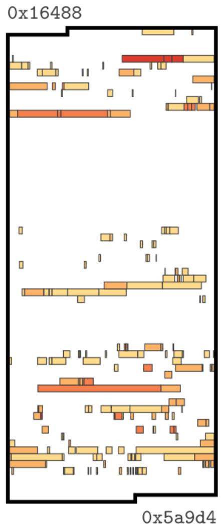
Remove unnecessary code from binaries

Why?

→ Smaller binaries
→ Reduced attack surface

Well researched.

Can we go further?



Executions

⬜	unused
🟨	1 – 10
🟧	$\leq$ 100
🟧	$\leq$ 1 000
🟥	$\leq$ 10 000

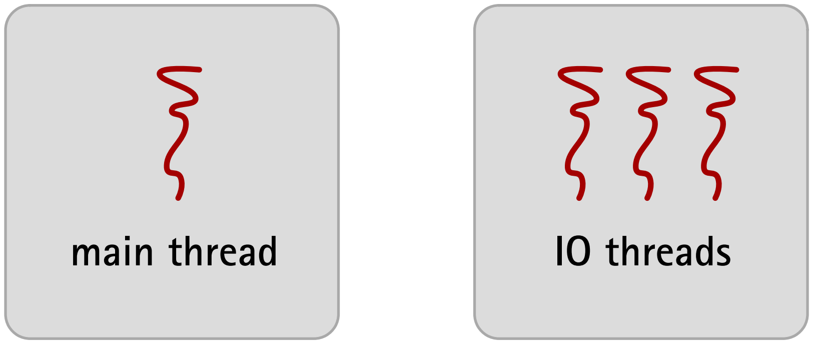
max 7 068

Fig. 1. Use of MUSL libc [16] functions by VSFTPD [15].

A. Ziegler et al. 2019
ACM Transactions on Embedded Computing Systems

Context-Based Code Elimination

Example: Redis (In-Memory Database)



...

Communicate with
Connected clients

~82 % of the functions are not needed in IO threads (excl. libraries)

→ Idea:
Elimination of these functions
But only in IO threads

Context-Based Code Elimination

How can we eliminate code on the basis of threads?

Dynamic Context-based Code Eliminiation via Address-Space Views

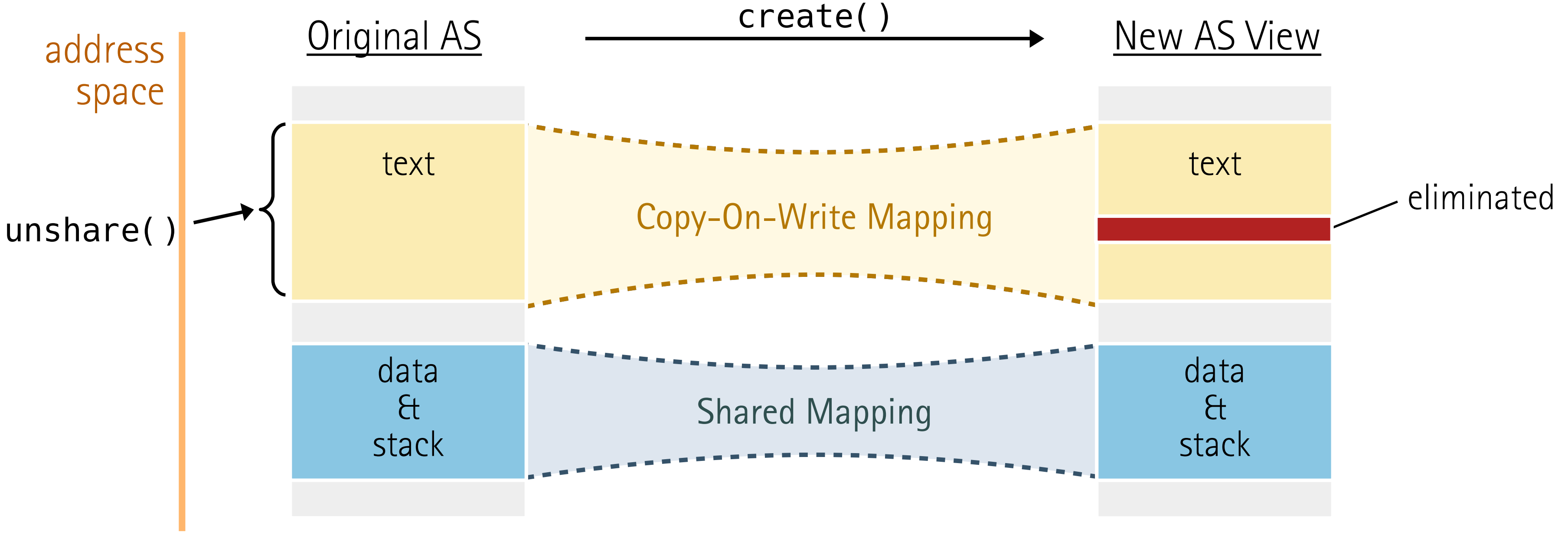
Address-space views:

→ Synchronized clones of the process's address space that differ can differ areas

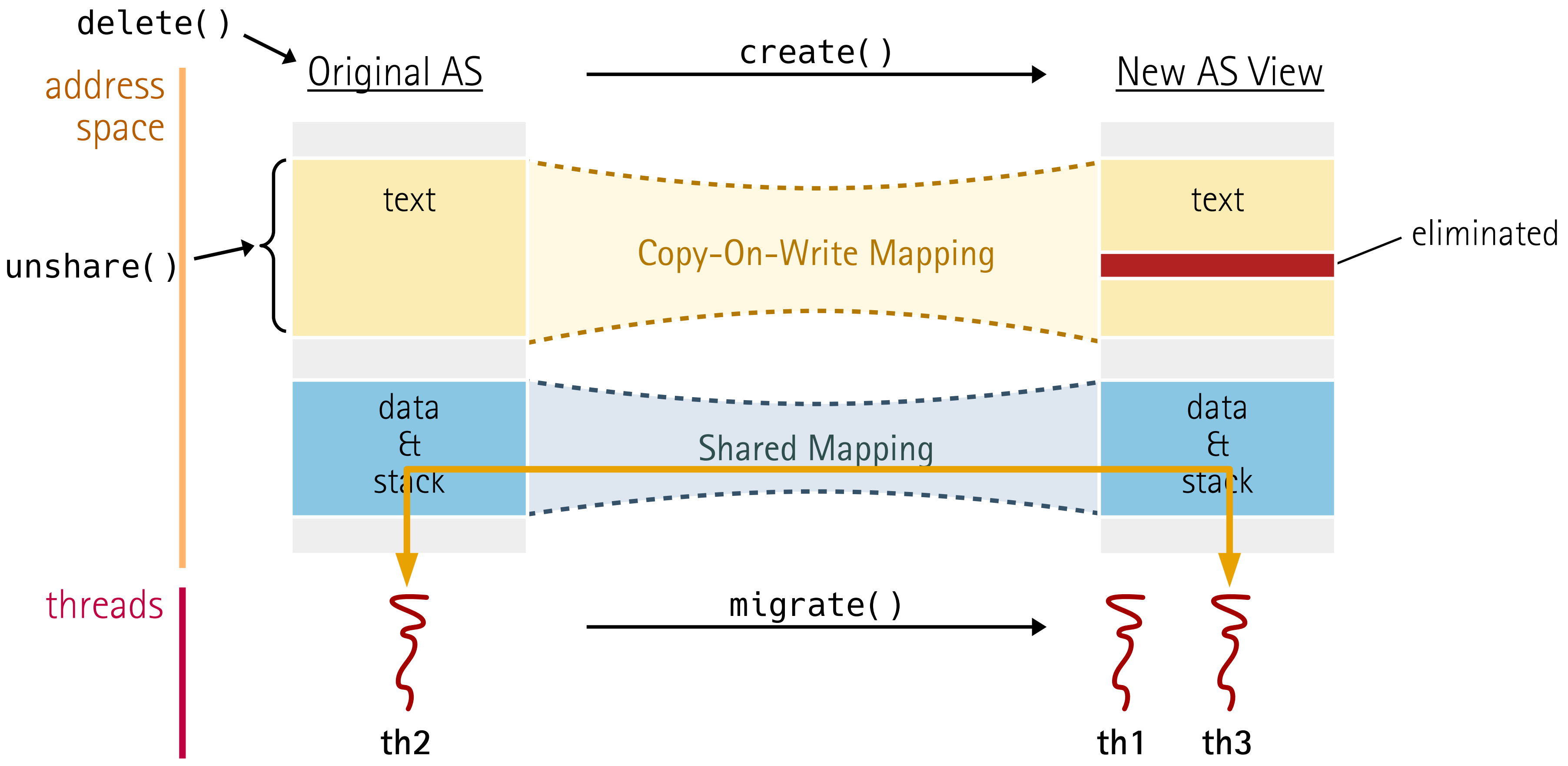
→ Threads can move between Views

→ Implemented in the Linux Kernel

Address-Space Views



Address-Space Views



Implementation

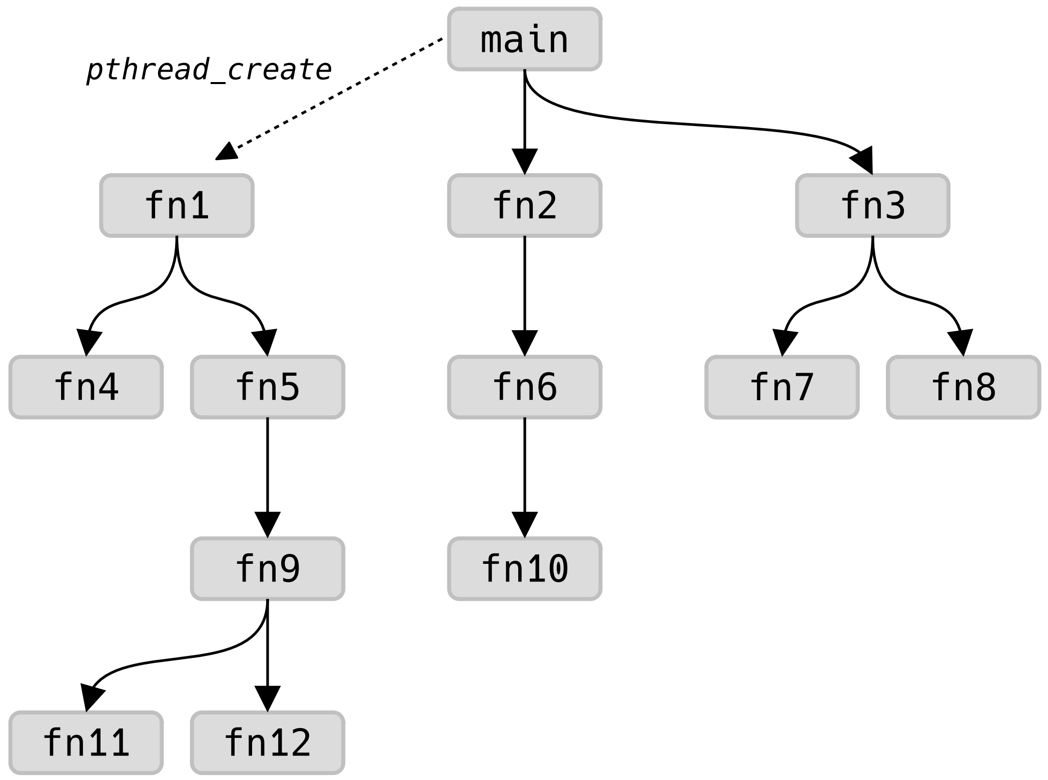
- Compiler Plugin (GCC)
  - Captures static call-graph information
  - Embeds information into the binary object (metadata section)

- Runtime Library
  - Consolidates metadata
  - Allows elimination of unused functions $\rightarrow$ replaces code with Invalid Opcodes

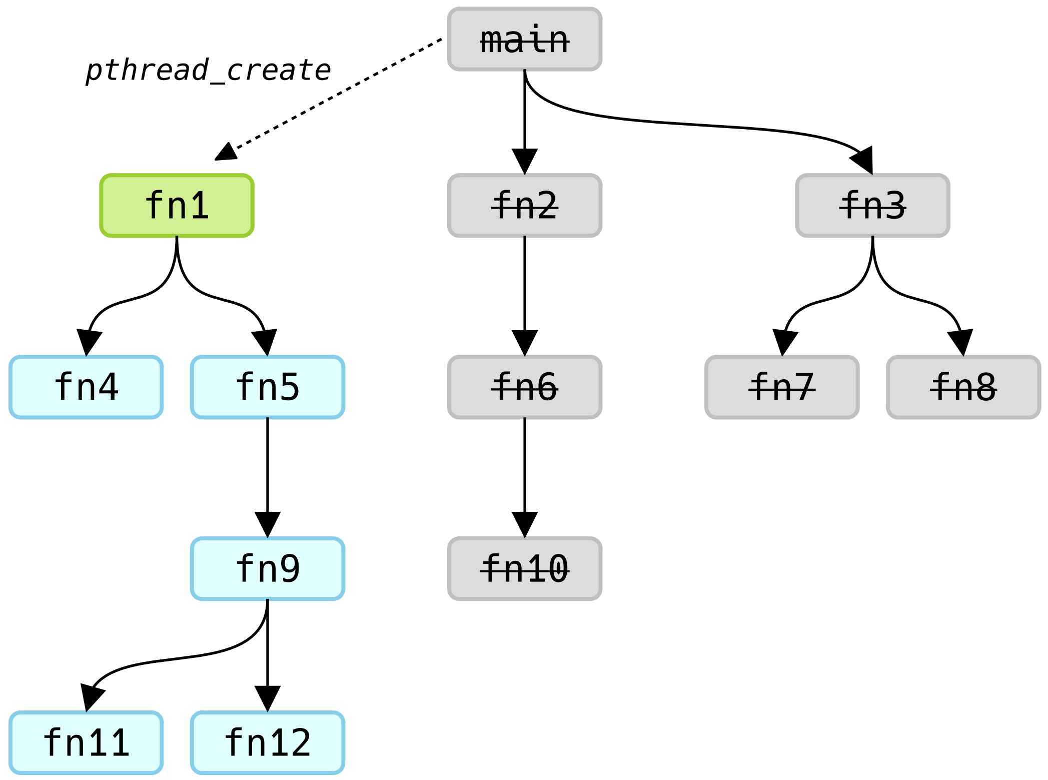
API:

```
void cte_init(void);
void cte_eliminate(void *keep[], long keepc, void *nokeep[], long nokeepc);
void cte_eliminate_self(void);
```

Implementation



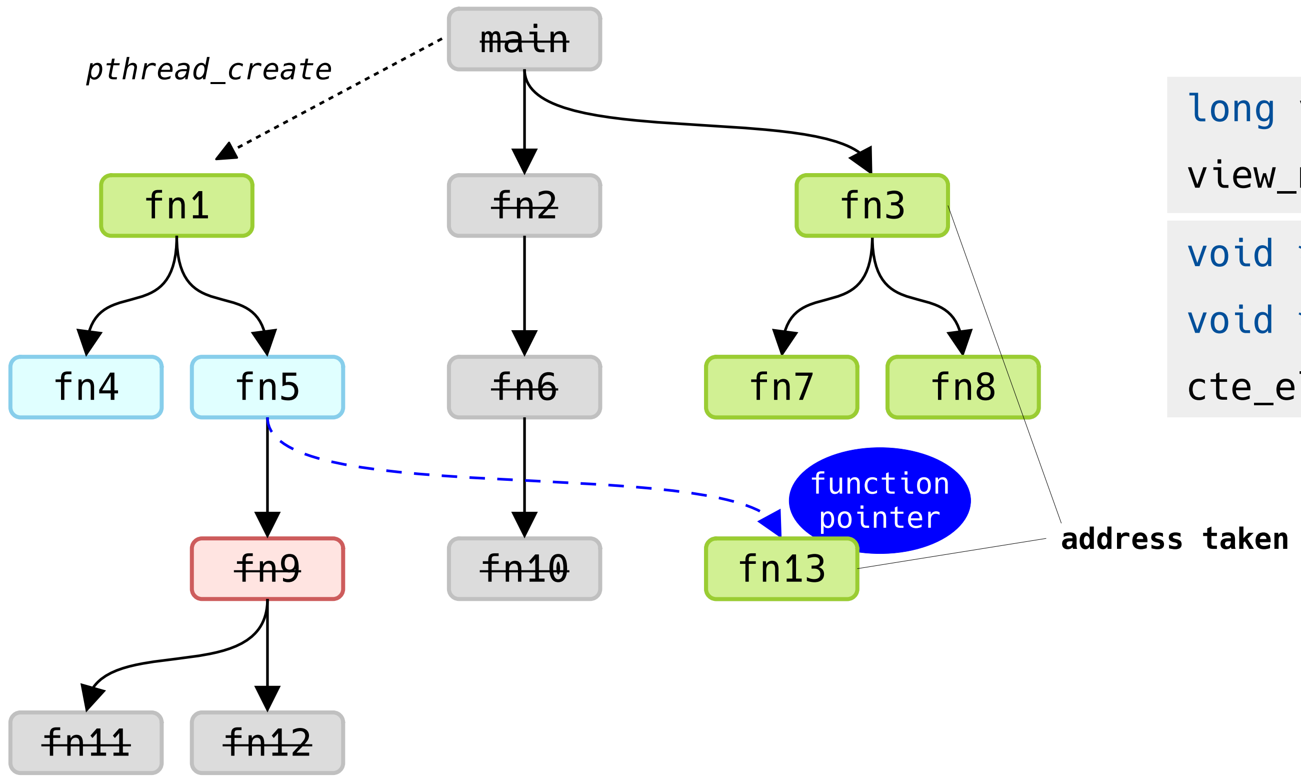
Implementation



```
long view = view_create();
view_migrate(view);
```

```
void *keep[] = { fn1 };
cte_eliminate(keep, 1, NULL, 0);
```

Implementation



```
long view = view_create();
view_migrate(view);
```

```
void *keep[] = { fn1 };
void *nokeep[] = { fn9 };
cte_eliminate(keep, 1, nokeep, 1);
```

Example: Redis IO Threads

→ Eliminate unnecessary functions for IO threads

- Functions eliminated
  - while preserving all "address-taken" functions:
    Removed **1738** of **2717** functions (~72 % code size [bytes])
  - While preserved only hand-selected "address-taken" functions:
    Removed **2227** of **2717** functions (~82 % code size [bytes])
- No measurable performance impact

Alternatives

- Separate processes / fork
  - Address spaces diverge
  - No more thread-like communication between contexts
  - Switching between contexts is not possible
- Intel Protection Keys
  - Available since Skylake in server CPUs
  - 16 protection domains per process
  - Restricted to page granularity

Future Work

- Improve call-graph analysis
- Context-based elimination for data
- More areas of application (browser, web services)

Summary

Dynamic Context-based Code Eliminiation

- Goal:
  Dynamically eliminate unreachable code on the basis of user-defined contexts.

- Approach:
  Use address space views to give each context its own view of the text segment.

Thank you for your attention.