# Bounded Resource Reclamation

Viktor Reusch
Barkhausen Institut
Dresden, SN, Germany

Till Miemietz
Barkhausen Institut
Dresden, SN, Germany

## 1 Introduction

During their lifetime, processes frequently allocate and release resources provided by the operating system. Obviously, the performance of resource allocations within the OS is critical to the overall performance of applications. This includes tail latency as well as the worst case execution time of allocation procedures. Consequently, existing work often focuses on improving allocation performance.

In contrast, few works address the resource reclamation performance of operating systems. However, there are prominent use cases that benefit from a swift and predictable reclamation of OS resources. First, bounded resource reclamation is of great use in resource-constrained multi-user systems as required by software-defined radio solutions like Open-RAN. These approaches implement tasks like signal processing in software rather than in ASICs. However, as the hardware deployed in the field cannot be arbitrarily large for reasons like power constraints, applications need to share resources such as memory. Therefore, in order to provide a smooth user experience, these scenarios profit from bounded resource reclamation as it avoids long application boot times due to cleanup of terminated processes.

Second, bounded resource reclamation is also important in cloud settings like function-as-a-service environments. Cloud functions typically have fixed main memory allocations and hard time limits. As the cloud provider aims at maximizing system utilization (i.e., the time during that customer applications perform actual work), resource reclamation phases of terminated functions should be as short as possible.

## 2 Study of Reclamation Latency

For the scenarios described in the introduction, reclamation latency is primarily relevant in the event of process termination. We thus analyze the reclamation latency observed during process termination on several OS platforms, including Linux, L4Re, and M³ [1]. We evaluated L4Re as it supports real-time applications but also has emerging support for function as a service [3]. M³ is a hardware-software co-design that focuses on security-critical use cases, such as telecommunication infrastructure [2]. The results for M³ were obtained using the gem5 system simulator. Due to the comparatively slow speed of simulation, we limited our M³ scenario to thousands of kernel structures.

Figure 1 shows that the amount of kernel structures (e.g., semaphores or memory mappings) that a process allocates strongly influences the time required to terminate said process. As expected, on complex kernels like Linux, the termination time of a process significantly increases with the number of kernel structures it allocated. But also the microkernel approaches of L4Re and M³ need a considerable amount of time to clean up terminated tasks. Thus, the CPU is still occupied and kernel memory is still claimed during process termination, rendering those resosources inaccessible to other processes. We argue that the termination latency on all these systems can be unexpected and hard to account for. Thus, we present a solution to speed up reclamation and ensure bounded termination of processes in the following.

## 3 Towards Bounded Resource Reclamation

The results of the previous section motivate the design of a system that puts a bound on the time required to reclaim resources. To make a concrete example, the operating system should be able to guarantee that a given process terminates in a time of, e.g., one second. This implies that all resources allocated to such a process have to be fully reclaimed one second after termination was initiated. This reclamation can be subdivided into individual steps, like freeing a single page of process memory or closing and removing a single file descriptor. As such, the combined execution time of these individual steps must not exceed the supposed one second limit. To enforce a bound on reclamation time, our design introduces accounting for the individual steps of reclamation. The approach is as follows: The operating system assigns a reclamation quota of one second to each process on startup. Whenever the process allocates a resource through the kernel, the expected reclamation time of this resource is subtracted from the quota. Thus, the operating system can enforce the total reclamation time to stay under one second.

However, reclamation time of, e.g., page table entries, heap memory, and semaphores can quickly add up as shown in the previous section. Thus, we propose to additionally group resource allocations — especially kernel memory — belonging to the same process to speed up reclamation on termination. Grouped resources can be swiftly reclaimed because they are close together and do not require the kernel to trace for all the individual pieces of memory allocated to the terminating process. With this, we intend to minimize runtime of the — now bounded — reclamation procedure with little loss in flexibility and low overhead.

## 4 Preliminary Results

We implemented a prototype for solving the problem of reclamation latency based on the M³ kernel. As M³ is a hardware/operating system co-design with a tiled architecture,

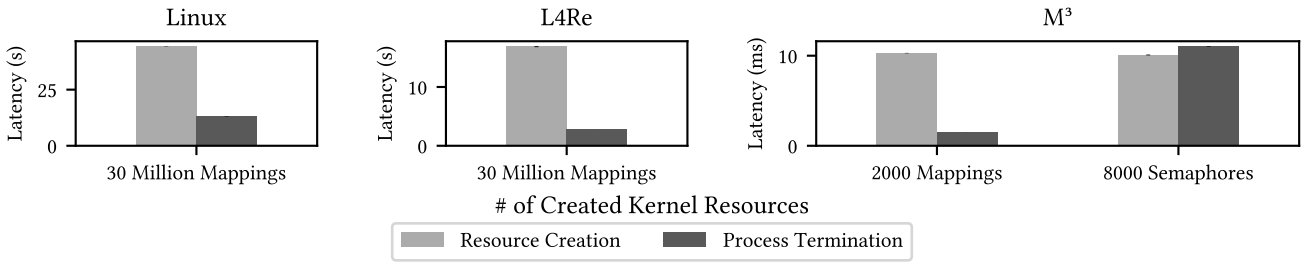


**Figure 1.** Latency of creating a large quantity of kernel structures and then terminating the allocating process forcefully on different operating systems.

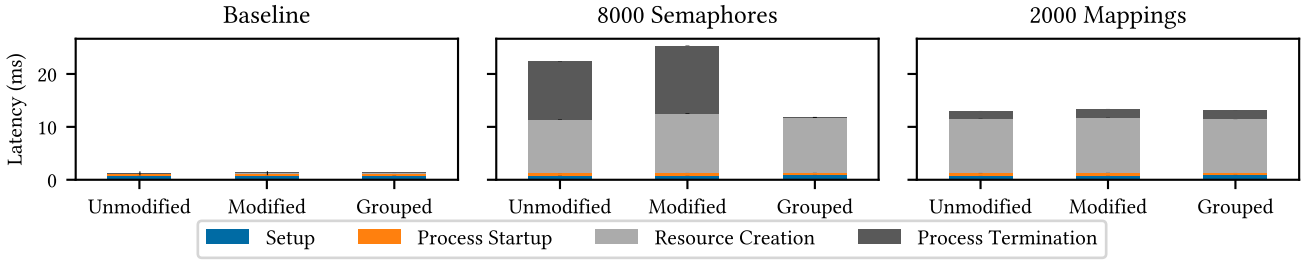


**Figure 2.** Latency of experiment setup, process startup, resource creation, and process termination of a single process under M³. This figure compares the unmodified M³ kernel with the modified kernel that enables the grouping feature. The process is terminated the usual way for "Unmodified" and "Modified". For "Grouped", the grouping feature is actually used to terminate the task and quickly reclaim associated resources.

it has a greater variety of hardware resources than conventional systems. For example, the M³ system supports direct, hardware-based communication channels between individual compute tiles. Such channels need hardware resources and, thus, need to be reclaimed once the communication partners terminate. This makes M³ particularly interesting for our prototype on resource reclamation. Moreover, there is previous work that also discusses how to efficiently reclaim memory in the M³ kernel [4]. Our prototype currently focuses on semaphore kernel objects. It implements the proposed grouping of resources to accelerate reclamation and allows attributing resources to actors in the system, thus demonstrating the feasibility of our approach.

Figure 2 shows the impact of the proposed reclamation optimizations on the termination time of M³ processes. The second panel highlights the effectiveness of resource grouping regarding a reduction of termination latency. The comparison of "Unmodified" and "Modified" shows that the grouping feature incurs only minimal overhead to kernel operations. As our reclamation optimizations are not yet implemented for memory mappings, there is no improvement for processes having many mappings as shown in the rightmost subfigure. However, we conclude that our approach works in the right direction to reduce reclamation latency of OS resources. We thus believe that future work on our system will fully enable bounded resource reclamation.

## References

[1] Nils Asmussen, Marcus Völp, Benedikt Nöthen, Hermann Härtig, and Gerhard P. Fettweis. 2016. M3: A hardware/operating-system co-design to tame heterogeneous manycores. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*. Tom Conte and Yuanyuan Zhou, (Eds.) ACM, 189–203. DOI: 10.1145/2872362.2872371.

[2] Sebastian Haas, Mattis Hasler, Friedrich Pauls, Stefan Köpsell, Nils Asmussen, Michael Roitzsch, and Gerhard P. Fettweis. 2022. Trustworthy computing for O-RAN: security in a latency-sensitive environment. In *IEEE Globecom 2022 Workshops, Rio de Janeiro, Brazil, December 4-8, 2022*. IEEE, 826–831. DOI: 10.1109/GCWKSHPS56602.2022.10008543.

[3] Till Miemietz, Viktor Reusch, Matthias Hille, Max Kurze, Adam Lackorzynski, Michael Roitzsch, and Hermann Härtig. 2024. A perfect fit? - towards containers on microkernels. In *Proceedings of the 10th International Workshop on Container Technologies and Container Clouds, WOC 2024, Hong Kong, Hong Kong, December 2-6, 2024*. ACM, 1–6. DOI: 10.1145/3702637.3702957.

[4] Viktor Reusch, Nils Asmussen, and Michael Roitzsch. 2024. Robust and immediate resource reclamation with M3. In *Proceedings of the 2nd Workshop on Kernel Isolation, Safety and Verification, KISV 2024, Austin, TX, USA, November 3-6, 2024*. ACM, 1–7. DOI: 10.1145/3698576.3698763.