User Level Management of L4 Kernel Memory

Andreas Haeberlen
University of Karlsruhe
haeberlen@ira.uka.de

Abstract

In the Hazelnut microkernel, kernel memory is a limited resource. This allows malicious applications to run denial-of-service attacks against the kernel, because there is no way to restrict the amount of kernel memory they may use. I suggest to integrate kernel memory into the normal memory management concept, which permits to control its allocation entirely from user level.

1 Introduction

In an L4 system, the microkernel is responsible for maintaining TCBs and page tables; additionally, it keeps a mapping database to permit asynchronous revocation of mappings. In Hazelnut, the memory for these structures comes from a central page pool and is allocated on demand. If a malicious application executes certain sequences of kernel operations (e.g. a Ping-Pong map attack), it can exhaust this pool, forcing the kernel to deny service to other applications.

A solution has already been proposed by Liedtke et al. [1], which allows applications to remedy this situation by donating some of their own memory to the kernel. This is done by granting the memory to a pager; the request then travels along the pager hierarchy until it arrives at σ_0, which grants it to the kernel. This approach has some important disadvantages:

- Kernel operations may fail due to lack of memory. This forces applications to add recovery code around most kernel invocations.
- The donation process is inverse to the normal pager hierarchy
- Logically, it is not the *kernel* who needs the memory, but rather the *application* (to store its mappings in)

For these reasons, I think that kernel memory should be controlled transparently by pagers, just like ordinary memory.

2 Suggestion: K-pages

Every page has access bits to control read and write permission. I introduce a new bit, the *K-bit*, to indicate whether the page can be used by the kernel. Only σ_0 is allowed to modify this bit. When active, it overrides the read and write bits, and the page is inaccessible from user level. A page with the K-bit set is called a *K-page*.

When a kernel operation (e.g. a map IPC) needs more memory, the kernel generates a *K-pagefault* on behalf of the application. This is sent to the pager exactly like a normal pagefault, but accepts only mappings of K-pages.

K-pages may be unmapped just like ordinary pages; the effect depends on its contents. If a page table, page directory or mapping database entry is hit, the memory area it covered will be flushed. If TCBs are hit, the threads disappear. In either case, the contents of the K-page are invalidated.

3 Security Considerations

Of course, care must be taken not to change existing L4 semantics or to create security risks. Some of the issues are:

- Internal design decisions of the kernel must not be exposed. This can be done by virtualizing K-pagefaults so that they seem to come from fixed locations inside the kernel area.
- The implementation must be deadlock free. When a K-pagefault is raised by a response to a normal pagefault, the faulting map operation must be aborted or its results discarded.
- Pagers should not gain more power over applications than they had before. This is not critical since they can map the initial code and thereby control the application's behaviour.
- It must not be possible to compromise the kernel's security, e.g. by mapping page tables to a page directory or vice versa. This can be achieved by marking the pages internally.

4 Conclusion

If user-level pagers are allowed to control the allocation of kernel memory, related denial-of-service attacks can be effectively prevented. This requires small changes to the kernel interface, but otherwise fits well into the pager concept and can be completely transparent to applications.

References

[1] J. Liedtke, N. Islam, and T. Jaeger. Preventing Denial-of-Service Attacks on a μ-Kernel for WebOSes. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS)*, May 1997.