Memory Sharing Revisited
(Work in Progress)

Jorrit N. Herder, Herbert Bos, Arun Thomas, Ben Gras, and Andrew S. Tanenbaum
VU University Amsterdam, FEW/CS, De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands
E-mail: {jnherder, herbertb, arun, beng, ast}@cs.vu.nl, Phone: +31-20-5987780

Abstract

Although several UNIX techniques for sharing memory exist, such as System V IPC Shared Memory and POSIX Shared Memory, these schemes are not suitable for use internal to the operating system and cannot protect against all threats posed by memory sharing. Therefore, we propose a novel, flexible memory-protection scheme for UNIX-like systems based on fine-grained, delegatable memory grants. We have already applied our ideas to safe memory copies and are currently investigating extensions for direct memory access (DMA) and memory mapping. Although memory grants originated in the MINIX 3 operating system, they is generally applicable and can be easily ported to other platforms.

1 Introduction

This work presents a novel, flexible memory-protection scheme for UNIX-like systems based on memory grants. A memory grant is a mechanism for sharing memory, similar to a capability in that it can be transferred to a another party in order to grant fine-grained access. Once a memory grant has been received, the access rights may be delegated to third parties at the grantee's discretion. The proposed structure also supports immediate revocation of memory grants at the grantor's discretion.

We propose a generalized memory-protection model based on memory grants. Grants originate from the MINIX 3 operating system where they were used to enable safe memory copies from and to device drivers [2]. On top of this, we propose two new extensions that bring memory grant-based protection for memory mappings and direct memory access (DMA).

Memory grants are not tied to device drivers or MINIX 3, but, in fact, represent a generally applicable model. For example, memory grants also could be used to achieve fine-grained protection for ordinary applications. In addition, memory grants could be ported to other operating systems including Linux and Windows.

2 Contribution of Grants

While several memory-sharing schemes exist, none of the existing models provides the same flexibility and degree of protection as memory grants do. Current UNIX techniques such as System V IPC and POSIX Shared Memory have several problems that are addressed by memory grants:

- Protection is based on group ID and/or user ID and cannot grant access to individual processes.
- Course granularity based on page size, even if only small data structures are to be shared.
- Delegation of rights to access a given piece of memory is not supported.
- Access rights are not automatically invalidated if a process sharing memory crashes.
- Not suitable for low-level drivers that cannot interface with the high-level POSIX servers.
- Cannot be used for safe direct memory access (DMA) because I/O-MMU integration is lacking.

In addition, a number of more specialized approaches exists, including seL4's frame capabilities [1]. Although several parallels can be drawn in terms of functionality offered, the naming scheme, usage, and implementation techniques are different from memory grants.

3 Structure of Memory Grants

The structure of a memory grant is given in Fig. 1. The flags field indicates whether the grant is in use, the grant's type, and the kind of access allowed. A *direct grant* means that a process A grants another process B limited access to a memory area in its own address space. The receiver of a direct grant, say, B, is allowed to transfer its access rights to a third process C by means of an *indirect grant*. The memory area covered by an indirect grant is always relative to the previous grant, which can either be a direct grant or indirect grant. Finally, the R/W flags define the access type that is granted: read, write or both.

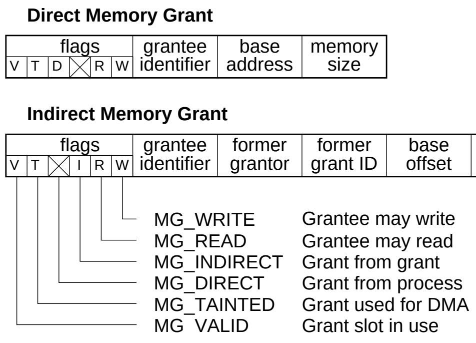

Figure 1: Structure of memory grants and grant flags.

A process that wants to grant another process access to its address space must create a grant table to store the memory grants. Memory grants can be made available to another process by sending the grant's index into the table, known as a grant ID. This can be done using the system's normal (platform-dependent) IPC mechanisms. The grant is uniquely identified by the process identifier of the grantor and grant ID: (ID, #). Delegation of memory grants is supported via indirect and results in a hierarchical structure as shown in Fig. 2. Since indirect grants contain the grantor's identifier, the kernel can follow to chain back to the root to determine the precise access rights.

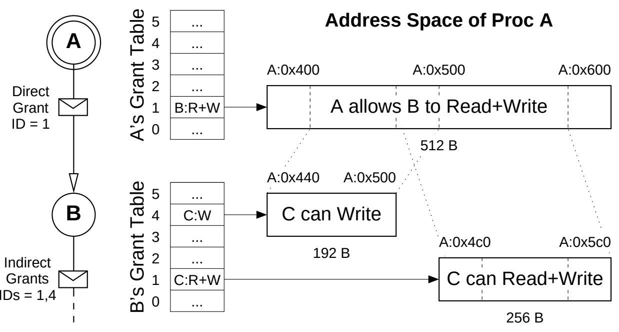

Figure 2: Using memory grants. A directly grants B access to part of its memory; B derives indirect grants for C.

4 Memory-grant Operations

When a process wants to access a memory area it has been granted, it needs to call the kernel, which verifies the grant's validity and performs the operation requested. It is important to realize that only the recipient of a grant—the grantee—can use the grant to perform grant operations. Forwarding a grant is useless since the grantee's identifier is listed in the grant. In addition, if indirect grants are used, it is not possible to grant additional powers, since each request is checked against the minimal access rights found in the path to the direct grant. Any violations will be detected by the kernel when a privileged operation is requested.

Although we are still extending the memory grant model and have not yet finalized the API, we envision memory-grant operations in the following categories:

- Grant table management
- Grant creation and revocation
- Grant permission lookup
- Memory copying
- Memory mapping
- Direct memory access

These operations will be implemented in a system library. The library code performs sanity checks on the grant table, grant ID and memory grant and then determines which operation is requested.

5 Performance Considerations

Memory grants do not come with an inherent performance overhead. Most grant table management, such as grant creation and revocation, can be done local to the caller. Privileged operations will have to be mediated by the kernel in order to enforce the protection, but setting up memory mappings and allowing DMA access can be done once during initialization and are not in the critical path. While this is the case for memory copies, the costs are limited to a single context switch, since the kernel can directly access all physical memory to read from the grant tables; no context switching is needed to follow the chain. Moreover, resolving a memory grant will be limited to one or two table lookups in typical usage scenarios.

6 Work in Progress

A prototype of the full memory-grant model is currently a work in progress. Safe memory copies could be taken over from our previous work [2], but memory mapping and direct memory are still to be done. Since these operations separate the time of checking from the time when the memory access takes place, several problems with respect to grant revocation need to be solved. We have outlined a possible solution and intend to implement it in the MINIX 3 operating system.

REFERENCES

[1] D. Elkaduwe, P. Derrin, and K. Elphinstone. Kernel Design for Isolation and Assurance of Physical Memory. In *1st Workshop on Isolation and Integration in Embedded Systems*, pages 35–40, Apr 2008.

[2] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Isolating Faulty Device Drivers. In *Accepted for publication at 39th DSN-DCCS*, 2009.