Flexible-Sized Page-Objects

H. Härtig, J. Wolter
Dresden University of Technology
Computer Science Department
{Hermann.Haertig,Jean.Wolter}@inf.tu-dresden.de

J. Liedtke
IBM T.J. Watson Research Center
GMD
Jochen@watson.ibm.com

Abstract

Demand-paging memory-management systems usually work with pages of fixed size. This is a limitation in systems relying on hierarchies of pagers to build layers of abstract machines. A generalized scheme is presented that allows for pages of flexible sizes and for multiple pages to be mapped within a single page-fault operation. Performance measurements (microbenchmarks) of a prototype implementation are presented.

1. Introduction

The invention of external pagers in the Mach system has been a major step in the development of μ-kernels. Here, so called memory objects are mapped to regions of an address space. A page fault is transformed by the kernel into a message to another task (a pager) which implements the memory object. The pager loads a page frame and returns it to the kernel which in turn inserts it into the address space of the faulting task. The kernel maintains a memory cache object for bookkeeping purposes. Also, the kernel implements page replacement policies. Once it selected a page frame to be replaced, it sends a message to the external pager. In detail, this results in a rather complex and inherently inefficient protocol. More recent μ-kernels[3] overcome most of these limitations and inefficiencies.

Another important invention, generally attributed to the Mach system, are *out-of-line* messages. Messages are not explicitely but lazily copied using memory mapping techniques (copy on write). For such out-of-line messages the kernel chooses a contiguous location in the receiver's address space. In OSF's version of Mach, the receiver may specify a sequence of areas which are filled contiguously with the incoming message.

Both mechanisms are important for *Object Orientation in Operating Systems*. They allow a server to present — potentially very large — objects to a client in the client's address space, either as a mapped structure using page faults or as a copy.

However, both mechanisms in their current form are pretty limited. They do not allow to preserve the composite structure of an object and to make use of that structure for efficiency and other purposes. Rather, the kernel enforces a break down of the structure to contiguous messages or to single page faults.

In addition, message passing as caused by page faults and as used for out-of-line messages are subtly different. Page faults result in temporary mapping while out-of-line messages result in (virtual) copies. I.e., an explicit send/receive pair may not replace a send/receive pair as executed as the result of a page fault.

This paper describes an alternate scheme. It allows a receiver to specify an arbitrary area (*a flexpage*) in its address space and it allows a sender to place several object partitions of arbitrary size (*flexpages*) into the area specified by the receiver. Hence, a page fault is nothing but a kernel generated send operation with the faulting address followed by a receive operation specifying the complete address space as placement area. It may be replaced on the faulting side by an explicit rpc operation. A page fault handler may return multiple pieces to be inserted into the faulting address space.

This paper discusses the questions that immediately arise. Is the scheme useful at all? How is the the receiving address space protected? How do sender and receiver determine where the kernel maps flexpages? Is that scheme efficiently implementable?

2. Application scenarios

The first simple application is informed prefetching. E.g., once a blocked process restarts and starts faulting, the external pager can reload the working set and place it into the faulting address space within one page fault operation. This, to our knowledge, is not possible with current external pagers, but quite common in monolithic systems (e.g. VMS). A scheme that enforces page faults per non-mapped page may cause a severe bottleneck in a system that heav-

1063-5351/96 $5.00 © 1996 IEEE
Proceedings of IWOOOS '96

ily relies on hierarchies of pagers to build levels of abstract machines [3].

More generally speaking, when an address space has very large areas with mapped objects that are managed by their pagers, a single receive operation for a whole area (e.g. a single page fault) allows for the insertion of all page frames currently available at the side of the object manager. No knowledge can be expected on the sender's side of the available area in the receiver's address space. No knowledge can be expected at the receiver's side about the availability of page frames on the manager's side.

If everything is broken down to fixed-size single page-faults, structural information that is available at higher levels of abstraction is lost. Hence, flexpages permit to handle larger regions as composed objects and thus support optimization at the user- and μ-kernel-level. For example, mapping and unmapping of 4 MB regions (even if the hardware does not support 4 MB pages) or complete address spaces can be substantially improved on some processors.

Some speculations may be allowed for the areas where the described scheme may become very useful:

- An external pager may maintain a large database which is partially mapped to a client's address space. When a page-fault occurs, the pager may want to map either a very small fraction of the memory object containing just one record or a very large one to enable the client to do extensive searches.

- An external pager may provide its clients with an abstraction of a variable bandwidth stream. For example, depending on the compression ratio, the next page of a stream may be either moderately small or very large.

- An external pager providing code images may maintain code files in chunks of variable size based on prior knowledge of working set behavior. For example, a page-fault in a certain region of the program may result in the mapping of either a very large portion of the address space or just a page corresponding to a page as supported by the underlying architecture.

These examples show that the use of pages of variable size is not restricted to hardware architectures supporting several sizes but also to software determined sizes.

3. Passing flexible pages

This section introduces a mechanism for passing flexible pages in a stepwise manner.

3.1. Message passing and basic page-fault handling in L4

L4 components of messsages that are transferred using mapping techniques are called flexpages due to their flexible size properties. However, since even for fixed-sized pages, L4's message passing differs significantly from Mach and similar systems, its semantics is explained here by contrasting it to Mach's message passing.

In Mach, a send/receive pair of operations causes the kernel the produce a virtual copy, i.e. the sent partition of a memory object is virtually copied. A new virtual memory region is created, that is backed up by the original external pager as long as it is not written to. The kernel uses its knowledge about memory objects, especially the memory object cache.

In L4, sending a flexpage means only mapping it. The operations merely establishes a temporary mapping into the receiver's address space. It is neither permanent nor it is a copy. At any given time, the sender may flush the mapping again.

When a page fault occurs, the faulting thread traps into the kernel (see Figure 1). The kernel generates a blocking RPC to the appropriate pager (which is a thread attribute set by the user). Basically, the virtual fault address and access type (read/write) are sent to the user-level pager. The pager makes the page available (allocates page frames, loads them from disk etc.) and then passes the appropriate flexpage to the faulting thread. The receive part of the μ-kernel-generated RPC maps the received flexpage into the address space of the faulting thread. (Note that there is nothing special on a kernel-generated page-fault RPC. The user can execute the same RPC explicitly and will get the same flexpage mapped.)

Here is another notable difference to Mach: the region structure of an address space as maintained by the Mach kernel is in L4 systems generally maintained at user level (by the involved pagers), not in the μ-kernel. This situation is illustrated in Figure 2. A kernel generated pagefault is sent first to a pager acting as region manager, i.e. maintaining information which objects are mapped to which regions of an address space. The region manager uses an arbitrary protocol to communicate with an object manager. E.g., the region manager sends an address indicating at which object offset the pagefault occurred. The receive operation executed by the region manager uses at most the involved region as a flexpage, since it trusts the object manager only with respect to that region.

One advantage of using message passing for mapping is that the principle of independence is not violated: The pager determines whether it will send a page, which page and page size it is willing to supply. The recipient determines, whether it accepts a page, at which region in its own

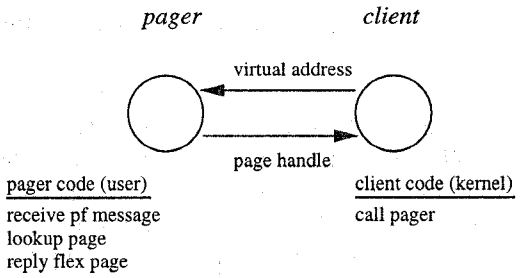

Figure 1. Basic Page-fault Handling.

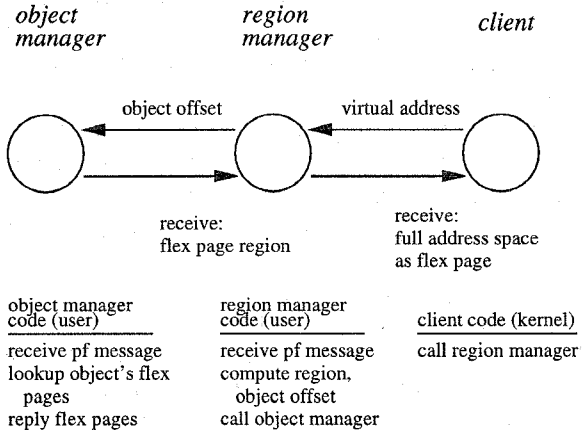

Figure 2. Generalized Page-fault Handling.

address space and up to which size.

Similar to other types of messages, these messages can be intercepted by chiefs. In particular, a chief may substitute another page mapping to forward to the faulting thread.

The next subsection discusses flexpages in more detail using examples as they arise in the context as shown in Figure 2.

3.2. Flexible pages

A flexpage is an address space interval of the size $2^n \times$ HW-pagesize. Locations of flexpages are aligned to multiples of their size. The binary representation as shown in Figure 3 is chosen to achieve very high efficiency for the implementation.

A receive operation specifies a flexpage in the *receiver's* address space, a send operation a flexpage in the *sender's* address space and an offset. These flexpages can be of arbitrary size. The parameters suffice to determine the map address without any further negotiation protocol. That is described in the following using page-fault handling as an example.

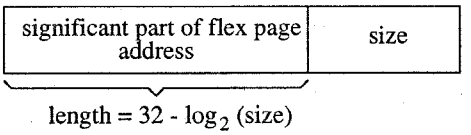

Figure 3. Binary Representation of Flex Pages.

First, let us assume that a region in an address space consists of just one flexpage and a pager maintains a memory object in multiples of flexpages of the same size as the region (Figure 4). Then, a page-fault message containing the memory object offset is used by the pager to identify the flexpage to be sent back to the faulting thread. Since both flexpages are the same size, the mapping to be performed by the implementation of the receive operation is obvious.

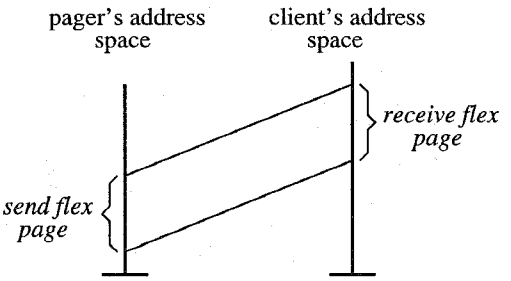

Figure 4. Sender's and receiver's flexpages are the same size

However, the assumption that a client uses regions of memory objects only in such sizes as used by a pager implementing the memory object is overly restrictive. In general, the following page-fault situations are common:

- the flexpage identified by the pager is larger than the region (Figure 5)
- the flexpage is smaller (Figure 6)
- the pager uses some small scattered flexpages to fill a receiver's flexpage (Figure 7).

These three cases and their treatment are discussed below.

In the case shown in Figure 5, only a fraction of flexpage as specified by the pager is mapped. The address of the fraction to be mapped is derived from the base address of the receiver's flexpage as indicated in Figure 5.

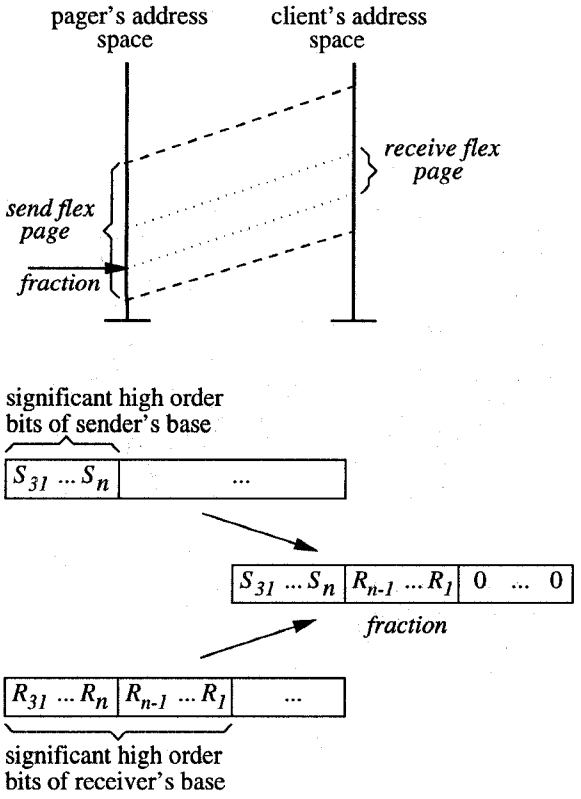

Figure 5. Large Send Flexpage

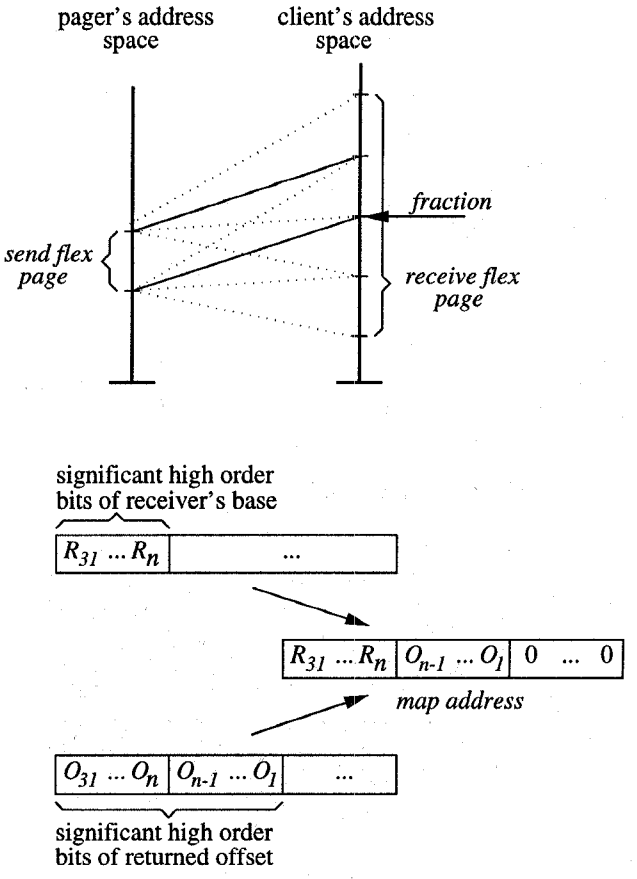

Figure 6. Small Send Flexpage

In the case shown in Figure 6, the fraction of the receiving flexpage cannot be chosen without prior knowledge about its position. To that purpose the sender returns a positioning address, which is used by the kernel to derive the position of the sent page within the receiving flexpage as indicated in Figure 6.

The scheme can also be used to handle case 3 as described above. If a thread specifies a flexpage in a receive operation then a pager thread can send arbitrary many flexpages, e.g. all pages currently present in main memory. For each flexpage sent by the pager, its positioning address is sent along leading to the situation shown in figure 7. It also makes sense to reply no flexpages. E.g., in the situation as illustrated in Figure 2, the region manager may be a thread sharing the client's address space. Then, all the all necessary mapping has been done as effect of the region manager's receive operation.

4. Performance measurements

This section gives performance figures as obtained from the initial implementation on a 33 Mhz i486 based PC. The measurement scenario is as follows. Flex pages are sent from a sender to a receiver and returned back. Figure 8 shows the time needed to send and receive a flexpage of size between 4 and 1024 Kbytes. Figure 9compares sending and receiving flexpages when send and receive operations specifies flexpages of different size.

Figure 10 shows the effect of multiple flexpage parameters. In the measurement, the sender specified 1, 2, 4, 15 flexpages of 4 Kbyte size each.

5. Summary

A new message passing primitive has been presented, which has two distinguishing properties:

- it establishes a temporary mapping of the sent message into the receiver's address space rather than providing a copy to the receiver
- pages of arbitrary size can be sent and received without prior knowledge on either receiver's or sender's part.

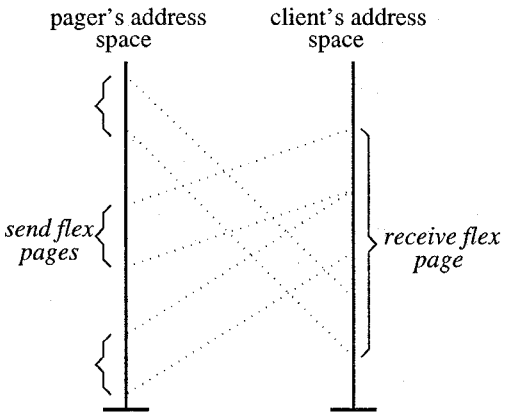

Figure 7. Multiple Send Flexpages

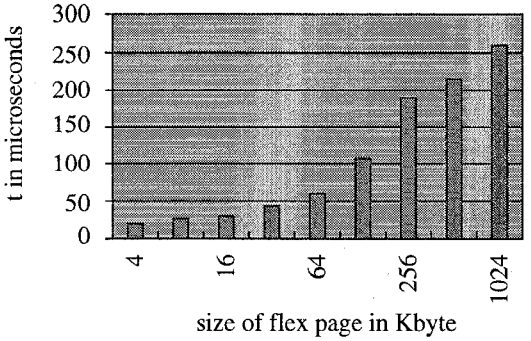

Figure 8. Sending and Receiving Single Flex Pages

The new primitive is especially useful for external pagers that handle pages of varying size. Furthermore, it permits to handle larger regions as composed objects and thus supports optimizations at the user and μ-kernel level.

References

[1] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy. Lightweight remote procedure call. In *12th ACM Symposium on Operating System Principles*, pages 102–113, Lichfield Park, December 1989.

[2] K. Harty and D. Cheriton. Application-controlled physical memory using external page cache management. In *ASPLOS V*, pages 187–197, Boston MA, October 1992.

[3] J.Liedtke. Towards real micro-kernels. *to appear in CACM*, September 1996.

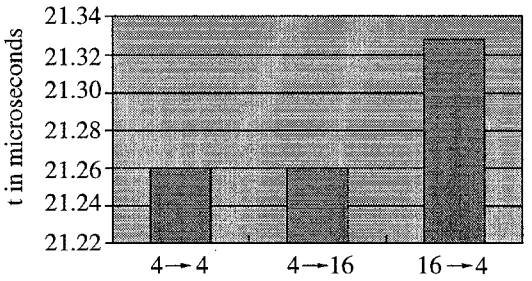

Figure 9. Sending and Receiving Single Flex Pages of different Sizes

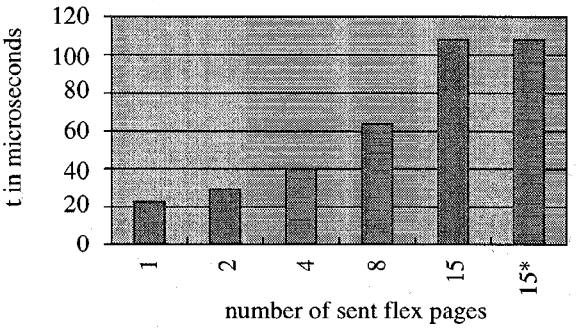

Figure 10. Multiple flexpage components