Segmentation

Welcome to this lesson on Segmentation.

We have talked about paging as one way how the memory management unit of the CPU maps the logical address spaces of the various processes to the physical memory.

In this lesson we will observe that, while computer architects and some operating systems designers love paging, programmers don't like to think in terms of pages.

A more programmer-friendly way for the MMU to manage and map memory is through what is called segmentation. We will look at how segmentation works.

We will observe that segmentation has several advantages, in particular in how it simplifies the programmer's life. Unfortunately, the segmentation's management of variable-sized segments re-introduces external fragmentation.

We will see that it will be again be paging that resolves external fragmentation problem by partitioning the variable-size segments into fixed-sized pages. This is called segmented paging. We will point out that there are many other ways to combine paging and segmentation.

Let's begin!

Segmentation: Problem with Paging

This is how a portion of the logical address space is mapped to physical memory when we use a paging MMU. Memory is partitioned into pages, and pages are allocated and mapped to physical memory.

Now, ask yourself the following: When have you last allocated a page of memory? Probably never; unless you have written an OS before or you have developed some low-level code of some device driver.

That's right! Users, in this context programmers, don't think of memory as pages. They think of it as segments, which can be tables, or arrays, or some other data structure for an application programmer, and an executable, a memory stack, or a memory heap for a lower-level programmer.

Pages don't support this model at all.

So, let's try something else. Rather than managing pages, the MMU manages segments of memory, which are mapped to segments of physical memory.

A MMU that uses segmentation achieves this conceptually by relocating each segment individually.

Segmentation in Hardware

Just as for paging, this is of course not how things are done in a real system. Let's go through the steps of how a logical address gets resolved and mapped to the corresponding physical address when we use segmentation.

At the beginning the CPU issues a logical address.

This address gets split into two parts.

The first part denoting is the so-called "segment number" of the address, which says to which segment the logical address belongs to.

The second part is the "offset", which describes which location within the segment the address refers to.

Instead of a page table, we have a "segment table".

And the segment number is used to index into the segment table.

The entry in the table contains the description of the segment, with the "limit" describing the size of the segment and the "base" the base address of the segment, that is, the address where the segment starts in physical memory.

We are now ready to construct the physical address. First, we compare whether the offset exceeds the size of the segment. If it does, an exception is raised.

Otherwise, we add the base to the offset, which results in the physical address.

In this animation we see that the segment in physical memory starts at the base address.

And the offset gets simply added to the base address to form the physical address.

Segmentation: Pros and Cons

Let's look at pros and cons of segmentation, in particular relative to paging.

On the positive side, one important benefit of segmentation is that it reflects the programmers preference to think in terms of memory segments, where data is semantically related. The data in a segment can be some executable, or a stack, or some object, or a table.

Since the data is semantically related, it is easy to associated a meaningful protection policy to the data in a segment.

For example, the data in a segment may be read-only, for example if the data is executable code. Similarly, one may want to mark the segment to be non-executable, for example if it is a stack.

Also, the limit check gives us – for free – a hardware supported range check for arrays if we store them as segments.

Finally, segmentation allows for easy sharing of memory segments across address spaces. As this example shows, we can easily share segments across Process 1 and Process 2 by copying the entry in the segment table so that Process 2 refers to the same physical segment. This can be done with paging as well, but is a bit more complicated.

The main disadvantage of segmentation is that it re-introduces external fragmentation.

Let's have a few processes have a few segments allocated, in this case, two.

Now let's assume that some process wants to allocate a new segment.

Since the memory in this segment has to be contiguous, we return to having the classical external fragmentation situation where we may well have sufficient available memory to satisfy the new request, just not contiguous.

Segmented Paging

The way to resolve this is to re-introduce paging, through what is called "segmented paging". Let's see how this works.

First, we partition the memory into pages again. This means that both the segment and the physical memory are partitioned.

Now, when the CPU issues a logical address, the address is partitioned into segment and offset, just as in segmentation, but after the segment number we have a page number.

Just as in segmentation, we have a segment table, which is indexed into by the segment number.

Each segment has a page table, which is pointed to by the base of the segment table entry. The limit in the segment table entry describes the size of the page table, and we use the page number to index into the page table to look up the page table entry. The limit makes sure that we don't exceed the bounds of the page table.

The frame number in the page table entry is then concatenated with the offset, to refer to the location in physical memory.

Segmentation: Summary

Let's briefly recall what we learned in this lesson on segmentation.

We started with the observation that paging does not reflect how programmers think about memory. No programmer thinks of memory as pages. Rather, they think of it as segments that may contain data structures, or executables, or stacks, or other entites.

Therefore, a memory management mechanism reflective of this make the programmer's life easier. This is what Segmentation does: Segments in the logical address space are mapped to segments of physical memory. This allows for straightforward management – and for example protection – of semantically-related memory segments. Also, it makes it very easy to share memory segments, as now entries in segment tables of multiple address spaces can point to the same segment in physical memory.

Unfortunately, segmentation re-introduces external fragmentation because segments in physical memory have to be allocated as contiguous regions of memory.

We looked at adding paging to segmentation – resulting in segmented paging – to eliminate external fragmentation. There are many other ways you can extend segmentation, such as multi-level segmentation similarly to multi-level paging. You can also combine segmentation and paging in different ways, for example paged segmentation, where you page the segment table to save space, and many others.

Thank you for watching!