High Memory Handling

By: Peter Zijlstra <a.p.zijlstra@chello.nl>

What Is High Memory?

High memory (highmem) is used when the size of physical memory approaches or exceeds the maximum size of virtual memory. At that point it becomes impossible for the kernel to keep all of the available physical memory mapped at all times. This means the kernel needs to start using temporary mappings of the pieces of physical memory that it wants to access.

The part of (physical) memory not covered by a permanent mapping is what we refer to as 'highmem'. There are various architecture dependent constraints on where exactly that border lies.

In the i386 arch, for example, we choose to map the kernel into every process's VM space so that we don't have to pay the full TLB invalidation costs for kernel entry/exit. This means the available virtual memory space (4GiB on i386) has to be divided between user and kernel space.

The traditional split for architectures using this approach is 3:1, 3GiB for userspace and the top 1GiB for kernel space:

```
+--------+ 0xffffffff
| Kernel |
+--------+ 0xc0000000
|        |
| User   |
|        |
+--------+ 0x00000000
```

This means that the kernel can at most map 1GiB of physical memory at any one time, but because we need virtual address space for other things - including temporary maps to access the rest of the physical memory - the actual direct map will typically be less (usually around ~896MiB).

Other architectures that have mm context tagged TLBs can have separate kernel and user maps. Some hardware (like some ARMs), however, have limited virtual space when they use mm context tags.

Temporary Virtual Mappings

The kernel contains several ways of creating temporary mappings:

- vmap(). This can be used to make a long duration mapping of multiple physical pages into a contiguous virtual space. It needs global synchronization to unmap.
- kmap(). This permits a short duration mapping of a single page. It needs global synchronization, but is amortized somewhat. It is also prone to deadlocks when using in a nested fashion, and so it is not recommended for new code.
- kmap_atomic(). This permits a very short duration mapping of a single page. Since the mapping is restricted to the CPU that issued it, it performs well, but the issuing task is therefore required to stay on that CPU until it has finished, lest some other task displace its mappings.

  kmap_atomic() may also be used by interrupt contexts, since it is does not sleep and the caller may not sleep until after kunmap_atomic() is called.

  It may be assumed that k[un]map_atomic() won't fail.

Using kmap_atomic

When and where to use kmap_atomic() is straightforward. It is used when code wants to access the contents of a page that might be allocated from high memory (see __GFP_HIGHMEM), for example a page in the pagecache. The API has two functions, and they can be used in a manner similar to the following:

```
/* Find the page of interest. */
struct page *page = find_get_page(mapping, offset);

/* Gain access to the contents of that page. */
void *vaddr = kmap_atomic(page);

/* Do something to the contents of that page. */
memset(vaddr, 0, PAGE_SIZE);
```

```
/* Unmap that page. */
kunmap_atomic(vaddr);
```

Note that the kunmap_atomic() call takes the result of the kmap_atomic() call not the argument.

If you need to map two pages because you want to copy from one page to another you need to keep the kmap_atomic calls strictly nested, like:

```
vaddr1 = kmap_atomic(page1);
vaddr2 = kmap_atomic(page2);

memcpy(vaddr1, vaddr2, PAGE_SIZE);

kunmap_atomic(vaddr2);
kunmap_atomic(vaddr1);
```

Cost of Temporary Mappings

The cost of creating temporary mappings can be quite high. The arch has to manipulate the kernel's page tables, the data TLB and/or the MMU's registers.

If CONFIG_HIGHMEM is not set, then the kernel will try and create a mapping simply with a bit of arithmetic that will convert the page struct address into a pointer to the page contents rather than juggling mappings about. In such a case, the unmap operation may be a null operation.

If CONFIG_MMU is not set, then there can be no temporary mappings and no highmem. In such a case, the arithmetic approach will also be used.

i386 PAE

The i386 arch, under some circumstances, will permit you to stick up to 64GiB of RAM into your 32-bit machine. This has a number of consequences:

- Linux needs a page-frame structure for each page in the system and the pageframes need to live in the permanent mapping, which means:
- you can have 896M/sizeof(struct page) page-frames at most; with struct page being 32-bytes that would end up being something in the order of 112G worth of pages; the kernel, however, needs to store more than just page-frames in that memory...
- PAE makes your page tables larger - which slows the system down as more data has to be accessed to traverse in TLB fills and the like. One advantage is that PAE has more PTE bits and can provide advanced features like NX and PAT.

The general recommendation is that you don't use more than 8GiB on a 32-bit machine - although more might work for you and your workload, you're pretty much on your own - don't expect kernel developers to really care much if things come apart.