Memory Subsystem in the Linux Kernel

Merlin Koglin

Universitaet Hamburg

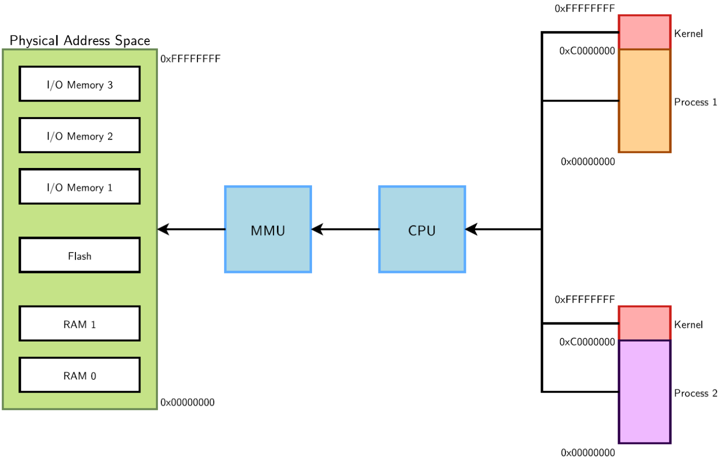
December 8, 2015

Overview

Kinds of memory

- Physical addresses
  - addresses used between the processor and the system's memory
- (Kernel) logical addresses
  - normal address space of the kernel
  - almost 1-1 mapping to physical memory
  - on most architectures logical associated physical addresses differ only by an offset
- (Kernel) virtual addresses
  - also mapping from kernel space address to physical address
  - not necessarily 1-to-1 mapping
  - able to allocate physical memory that has no logical address

Virtual Memory - Physical Memory

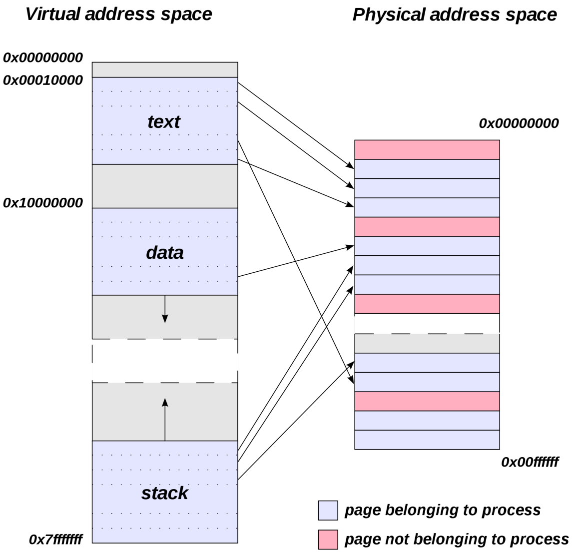


http://free-electrons.com/doc/training/linux-kernel/linux-kernel-slides.pdf

Pages

- physical memory is divided in parts of the same size called page
- basic unit of memory management
- size is architecture-dependent, but typically 4096 byte
  `$ getconf PAGE_SIZE`
- in the kernel, every page is represented as a `struct page`, this structure ist defined in `<linux/mm_types.h>`

Pages and mapping



https://upload.wikimedia.org/wikipedia/commons/3/32/Virtual_address_space_and_physical_address_space_relationship.svg

Zones (1)

- because of hardware limitations, the kernel cannot treat all pages as identical
- some hardware can perform direct memory access to only certain memory adress
- some architectures can address larger amounts of physical memory than they can virtually address, so this memory is not permanently mapped into the kernel address space
- $\rightarrow$ physical memory is divided into (more ore less) three zones

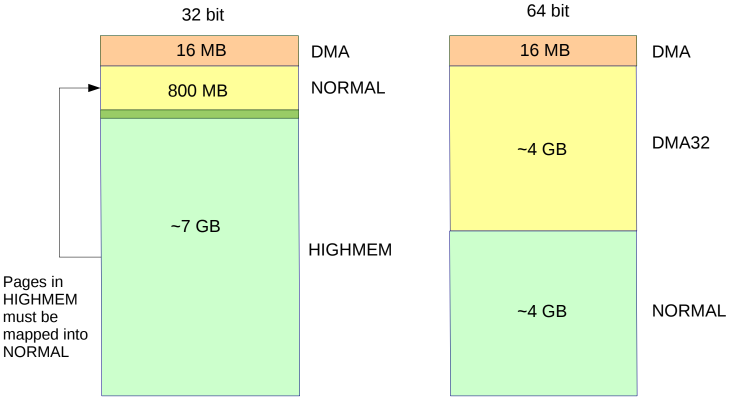
Zones (2)

- DMA
  - low 16MB of memory
  - exists for historical reasons, sometime there was hardware that could only do DMA in this area
- 32DMA
  - only in 64-bit linux
  - $\sim$low 4GBytes of memory
  - today, there is hardware that can do DMA to 4GBytes

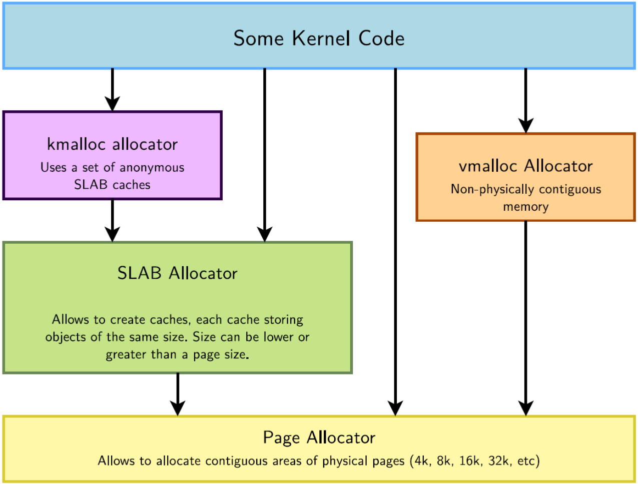
Zones (3)

- Normal
  - different on 32-bit and 64-bit machines
  - 32-bit: Memory from 16MB to 896MB
  - 64-bit: Memory above ~4GB
- HighMem
  - only on 32-bit Linux
  - all Memory above ~896 MB
  - is not permanently or automatically mapped into the kernel's address space
- `cat /proc/pagetypeinfo`

Memory zones for 8 GB RAM



Kernel Memory Allocation Overview



http://free-electrons.com/doc/training/linux-kernel/linux-kernel-slides.pdf

Buddy system

- the kernel uses a buddy allocator strategy so only allocations of power of two number of pages are possible:
  1 page, 2 pages, 4 pages, 8 pages, 16 pages, etc.
- if a small area is needed and only a larger area is available, the larger area is split into two halves (buddies), possibly repeatedly.
- when an area is freed, it is checked whether its buddy is free as well, so they can get merged
- number of free areas can be seen here `/proc/buddyinfo`

Getting a page

- `unsigned long __get_free_page(int flags)`
  - returns virtual adress of a free page
- `unsigned long get_zeroed_page(int flags)`
  - returns virtual adress of a free page, initialized to zero
- `unsigned long __get_free_pages(int flags, unsigned int order)`
  - returns the starting virtual adress of an are of contiguous free pages, with `order` $= \log_2(\text{number\_of\_pages})$

Flag categories

- The flags are broken up into three categories:
- action modifiers
  - specify how the kernel is supposed to allocate memory
- zone modifiers
  - specify where the kernel is supposed to allocate memory
- types
  - type flags specify a combination of action and zone modifiers as needed by a certain type of memory allocation
  - these are mostly used

frequently used flags

- GFP_KERNEL
  - standard kernel memory allocation
  - the allocation may block in order to find enough free memory
  - fine for most needs, except in interrupt handler context
  - this flag should be your default choice
- GFP_ATOMIC
  - the allocation is high priority and is not allowed to sleep
  - never blocks, allows to aaccess emergency pools
  - can fail if no free memory is readily available
- GFP_DMA
  - allocates memory in an area of the DMA Zone
  - device drivers that need DMA-able memory use this flag
- for all flags see `include/linux/gfp.h`

free pages

- `void free_page(unsigned long addr)`
  - frees one page
- `void free_pages(unsigned long addr, unsigned int order)`
  - frees multiple pages
  - order has to be the same as in allocation, passing the wrong order can result in corruption.

Usage

- the low-level page functions are useful when you need page-sized chunks of physically contiguous pages especially if you need exactly a single page or two
- it is also possible to use:
  `struct page * alloc_pages(int flags, unsigned int order)`
  - returns a pointer to the first pages page struct, on error it returns NULL

slab allocator

- allows to creates caches, which contains a set of objects of the same size
- it uses the page allocator
- principle aims
  - caching of commonly used objects
    → system does not waste time allocating, initialising and destroying objects
  - allocation of small blocks of memory
    → help eliminate internal fragmentation that would be otherwise caused by the buddy system

Different SLAB allocators

- there are three different implementations of a SLAB allocator in the linux kernel.
- you can choose one at configuration of the kernel
- SLAB
  - legacy
- SLUB
  - default, simpler, better scaling, less fragmentation
- SLOB
  - simpler, more space effizient but doesn't scale well.

kmalloc allocator

- kmalloc() is the normal method of allocating memory in the kernel
- for small sizes it relies on SLAB caches → `/proc/slabinfo`
- for larger sizes it relies on the page allocator
- kmalloc() guarantees that the pages are physically contiguous (and virtually contiguous)
- same flags as for the page allocator
  `GFP_KERNEL`, `GFP_ATOMIC`, `GFP_DMA`, etc

kmalloc sizes

- the maxium of space that can be allocated by kmalloc depends on the architecture
- Maximum sizes on x86 and arm
  - Per allocation: 4 MB
- Maximum sizes on 64-bit
  - We will test this later.
- For completely portable code, do not allocate anything larger than 128 KB

kmalloc api

- `#include <linux/slab.h>`
- `void *kmalloc(size_t size, int flags);`
  - allocate `size` bytes and return pointer to the area (virtual adress)
  - size: number of bytes to allocate
  - flags: same flags as the page allocator
- `void kfree(const void *addr);`
  - frees a block of memory previously allocated with kmalloc()

kmalloc API 2

- `void *kzalloc(size_t, int flags);`
  - Allocates zero-initialized memory
- `void *kmalloc_array(size_t n, size_t size_t, gfp_t flags);`
  - allocates memory for an array of `n` elements of `size` size
- `void *kcalloc(size_t n, size_t, size, int flags);`
  - allocates memory from an array of `n` elements of `size` size and the memory is set to zero,

kmalloc example

- similar to malloc()
- If not enough memory is available, kmalloc() can return NULL so check after all calls to kmalloc() and handle the error appropriately
- ```
  struct cat *p;

  p = kmalloc(sizeof(struct cat), GFP_KERNEL);
  if (!p)
      /* handle error ... */

  //free the memory
  kfree(buf);
  ```
  ```

devm_kmalloc

- devm_kmalloc is a resource-managed kmalloc
- automatically frees the allocated buffers when the corresponding device is detached
- `void *devm_kmalloc(struct device *dev, size_t size, int flags);`
  - dev → Device to allocate memory for
- less errors/memory leaks

vmalloc()

- vmalloc() allocates memory that is only virtually contiguous, but not physically contiguous
- pages obtained via vmalloc() must be mapped by their individual pages (because they are not physically contiguous)
- is used only when absolutely necessary
- typically, to obtain large regions of memory

vmalloc api

- `#include <linux/vmalloc.h>`
- `void *vmalloc(unsigned long size);`
  - returns a pointer to at least `size` bytes
- `void vfree(const void *addr);`
  - frees an allocation obtained via vmalloc()

large buffers

- what if you want to allocate a lot of (physically contiguous) memory?
- $\rightarrow$ allocate at boot time
- only drivers directly linked to the kernel can do that
- to install, rebuild kernel and reboot
- freed memory ist possibly not reuseable!

bootmem

- bootmem for allocating memory at boot time
- #include <linux/bootmem.h>
- void *alloc_bootmem_pages(unsigned long size);
  void *alloc_bootmem_low_pages(unsigned long size);
  - allocated memory may be high memory unless _low is used
  - unsigned long size size of memory
  - page-aligned memory areas
-
  void free_bootmem(unsigned long addr, unsigned long siz
  - but not all pages are returned to the system

Picking an allocation

- kmalloc()
  - general purpose memory allocator for the kernel
  - contiguous physical pages
  - should be used as the primary allocator
  - can allocate DMA memory
- vmalloc()
  - only virtual contiguous
  - slower than kmalloc()
  - allocations of fairly large areas are possible

Summary

Thank you :)

Any questions?

References

- makelinux.net - Chapter 15 - Constantine Shulyupin
- Linux Device Drivers, 3rd Edition - O'Reilly
- The Linux Kernel - Chapter 3 - David A Rusling
- Linux Kernel Development - Robert Love (pdf)
- Linux Kernel and Driver Development Training - free electrons (pdf)
- Memory Subsystem and Data Types in the Linux Kernel - Bjoern Broenmstrup and Alexander Koglin (pdf)