mikro - About paging

Victor Apercé

viaxxx@lse.epita.fr
`http://lse.epita.fr/`

Outline I

1. Introduction
2. Hardware management on x86
3. Prerequisites
4. Paging kernel management
   - Physical allocator
   - Virtual allocator
   - Kernel malloc
   - Paging initialisation
5. Other paging features
6. Conclusion

Introduction

A little advice

Never ever work on paging!!!

Let the others do it for you

Paging for dummies

Paging

A way to isolate process memory from each other and much more.

- Kernel development can be achieved in 1 week only
- Doing it properly can last... 3 months

Virtual physics

Physical address

Real hardware address in your RAM.

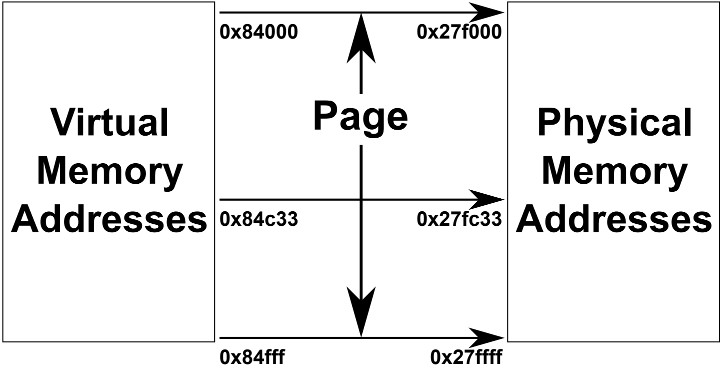
Virtual address

Address relative to an address space that are defined to point to physical ones.
The kernel is free to define what virtual address points to what physical one.

Paging unit is page

Page

Unit in memory that is the minimal memory portion mapped linearly from virtual to physical addresses.
Its typical size is 4 KB or 0x1000 B in hexadecimal.



To find your way in paging: use a map

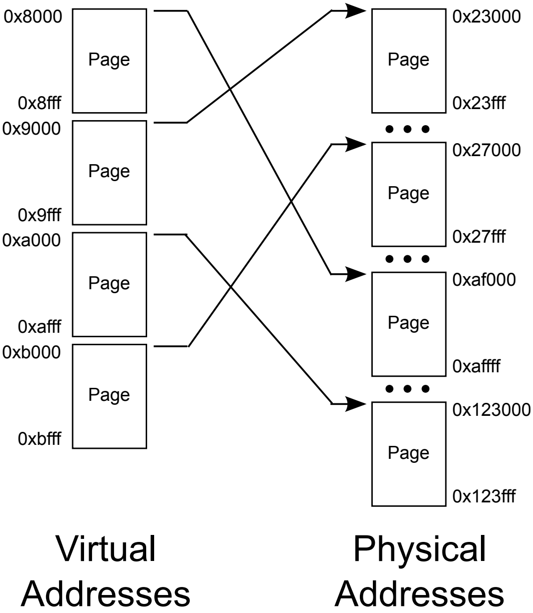
Mapping

Linear virtual addresses corresponding to their linear or not physical addresses.
This is always expressed in unit of pages.

Address space

Set of mapping that is related to one process.
Threads share the same address space.

A little drawing of a mapping



Hardware management on x86

Origin of SEGV

Paging rights

Every page have the following rights:

- Write: allow writing this page
- Userland: allow userland to access this page

Read is always authorized.

NX flag

NX flag is a right that disable execution of page.

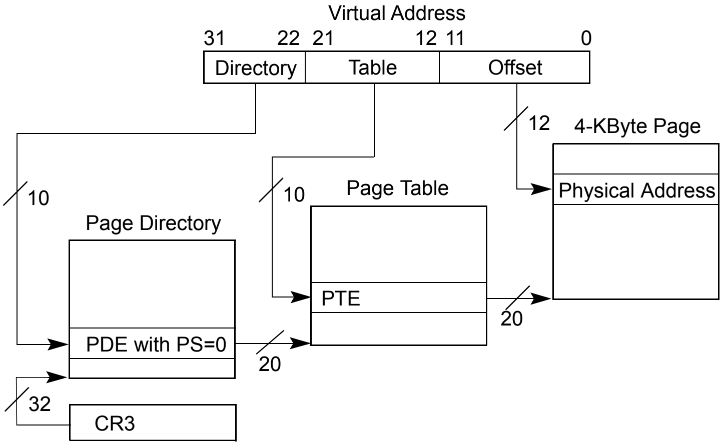
It's a big security improvement.

Paging modes

x86 processor family supports 3 paging modes:

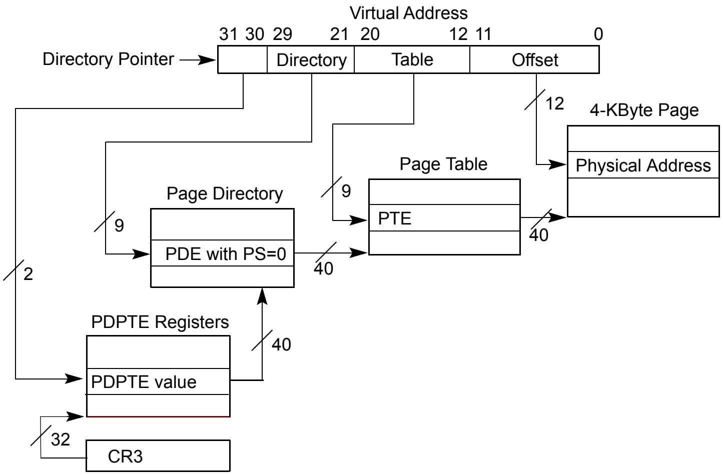
Paging Mode	**Virtual Address Width**		**Physical Address Width**		**Nx Support**
32 bit	32 bit	4 GB	32 bit	4 GB	No
PAE	32 bit	4 GB	52 bit	4096 TB	Yes
64 bit	48 bit	256 TB	52 bit	4096 TB	Yes

i386 Paging



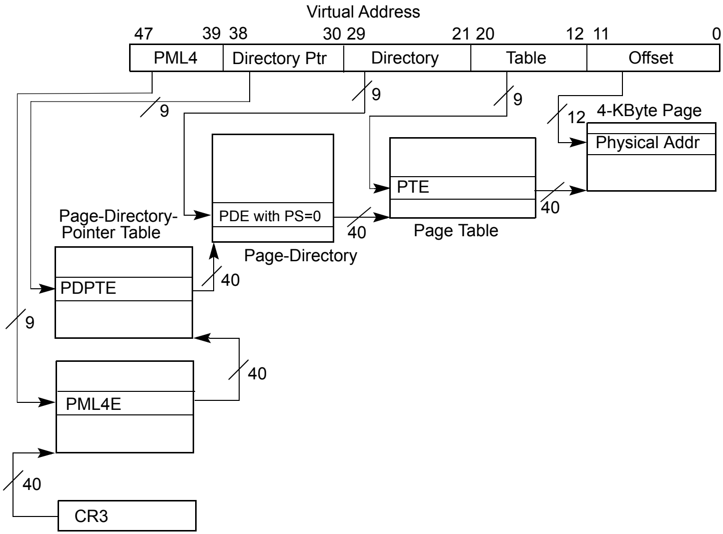
Source: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide

i386 Paging with PAE



Source: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide

amd64 Paging



Source: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide

Cache me

For better performance a cache is used: the TLB – Translation Lookaside Buffer

- It caches pairs of Virtual/Physical addresses
- It is flushed on every CR3 change
- Every removal/modification in tables must be notified

To be clear

- If you want more: RTFM Intel Architectures Software Developer's Manual Volume 3, chapter 4
- I'll focus on 32 bit mode from now because it's:
    - the simplest.
    - the only one handled by mikro for now.

Prerequisites

Put your hands up!

Let's play a game

Question: I want to use all my memory

Question

Can I use freely all my RAM?

Answer: I want to use all my memory

Answer

No, some portion are reserved by your hardware.

- BIOS can gently tell you where these portions are.
- Or maybe not...

Q: ISA

Question

Ok, so I can use all the rest of my memory as I want?

A: ISA

Answer

No, ISA drivers can access only the first 16 MB of your memory. As this area is small, you should try to use this area as less as you can to let it to these drivers.

Q: Great I have only 128 MB to handle!

Question

I have only 128 MB of memory.
So my physical addresses goes from 0x0 to 0x7FFFFFF (128 MB), don't they?

A: Great I have only 128 MB to handle!

Answer

No, some addresses above 0x7FFFFFF are hooked by the CPU to do some hardware stuffs.

- Always suppose, you can use all the possible addresses from 0x0 to 0xFFFFFFFF in 32bits.
- This is why having 4GB of RAM on 32bits computer is... useless.

Q: My precious memory

Question

Well ok, that’s just some constraints. Is that all?

A: My precious memory

Answer

No, your memory is managed by Page Tables that are too in memory.
Moreover, be careful because when you do a malloc in your memory management system it can trigger a mmap...

Where's my kernel?

Question

What about the virtual addresses of the kernel?

Answer

- Kernel binary is mapped in all your User land processes in order to do syscalls
- Its address must be the same for all processes for performance reason
- On most 32 bits UNIX system, the kernel address space is above 0xC0000000

Where's my kernel?

Question

What about the virtual addresses of the kernel?

Answer

- Kernel binary is mapped in all your User land processes in order to do syscalls
- Its address must be the same for all processes for performance reason
- On most 32 bits UNIX system, the kernel address space is above 0xC0000000

Summing up

Paging = Masochism for geeks!

Paging kernel management

Physical allocator

What's that?

Physical allocator

Handles free physical pages.

Typically, it responds to 2 main requests:

- Give me some free physical pages
- I don't use these pages anymore, take them

So it's malloc but for pages

- Corresponds to a malloc but for pages
- All malloc algorithms can work for this allocator
- As for malloc, buddy algorithm is the best: O(1)
- This is the one chosen for Linux and mikro

Buddy algorithm

- Free areas are stored as power of 2
- Every area has a buddy
- A buddy is found by xoring the address with the size of the element
- Every element needs the following metadatas:
  - Free/Used
  - Size of the element

Where are my metadatas?

- Typically metadatas are stored in the element itself reducing element storage space
- This cannot work for paging because a page:
    - starts at address + 0x0
    - ends at address + 0xFFF
- So no room for anything else

Metadatas in the cloud

Metadatas must then be stored elsewhere...

Linux solution

- Metadata and all information about pages are stored in a big array
- Finding a page metadata is O(1)
- All pages are in the table so it takes a lot of memory

mikro solution

- Metadata and all information about pages are stored in the heap
- A big array of pointers is used to find page metadata
- Still O(1) but less memory is used
- This works because mikro needs less information about pages than Linux

Virtual allocator

What's this?

Virtual allocator

Handles free virtual pages for one address space.

Does exactly the same as physical allocator but for virtual addresses in each address space.

Why not the same algorithm then?

- Allocatable areas aren't powers of 2
- It needs more flexibility than physical pages

There's always a solution

Solution

Like Linux, mikro use a classical red/black tree.

Algorithms are certainly not fully optimized but doing better is hard.

Kernel malloc

Kernel malloc

Yet an other allocator

Linux

Linux uses its slab allocator behind kmalloc.

mikro

mikro use an other buddy algorithm to allocate in the kernel.

The usage of a slab for mikro is overkill.

Solving the malloc that trigger mmap

Question

What happens if in the middle of my mmap code I trigger an other mmap through a malloc?

Answer

Kernel has a reserve of already mapped pages that are used in this particular case.

- mikro C++ **new** function can have a CRITICAL flag to ask for this.
- this reserve is checked prior to every mmap and **new**.

Solving the malloc that trigger mmap

Question

What happens if in the middle of my mmap code I trigger an other mmap through a malloc?

Answer

Kernel has a reserve of already mapped pages that are used in this particular case.

- mikro C++ **new** function can have a CRITICAL flag to ask for this.
- this reserve is checked prior to every mmap and **new**.

Paging initialisation

Paging initialisation

Kernel boot: physical vs virtual?

- Kernel is located above 0xC0000000 when paging is activated.
- But we can't suppose this address is usable on boot.
- So the kernel is loaded lower.
- How to load a kernel compiled to be above 0xC0000000?

mikro simple solution

- mikro uses a bootstrap that is relocatable.
- Relocatable means it can execute properly at any address in memory.
- This bootstrap initialize paging and sets kernel above 0xC0000000.

mikro paging boot sequence

1. Copy kernel from zone DMA to zone Normal and set a better kernel address space.
2. Initialize kernel allocator with a fixed set of pages to allow calling **new** during boot.
3. Fill computer reserved zones with what BIOS provides.
4. Add kernel private areas to these reserved zones.
5. Find room for paging metadatas and add it to reserved zones.
6. Initialize physical and virtual allocator.
7. Map kernel modules and command line.
8. Switch to real kernel allocator.
9. Do some cleanup.

Other paging features

The magic SEGV 1/2

The segfault aka page fault can be used to do funny things:

- File mapping optimisation:
  - When mapping a file into memory, instead of putting it all in memory, it's not mapped.
  - When a page fault occurs, it then reads from the disk and put it in memory.

The magic SEGV 2/2

- COW – copy on write – uses about the same mechanism:
    - When a page needs to be copied for an other process, it’s just shared with read right only.
    - When the process wants to write to this page, a page fault occurs and the copy is done then.

mikro User land

- mikro will manage only the previous features in User land.
- This isn't usual for a micro kernel to have so much in Kernel land.
- But it seems better for performance.
- If not, we will change that.

Conclusion

Paging status in mikro

- Paging in Kernel Land is almost fully featured.
- It's now relatively stable but there's certainly some bugs remaining.
- User land part has to be done.

Paging is painful

I hope I convince you to never work on paging :D.
If not, some last arguments:

- Testing is about impossible.
- Bugs are WTF.
- Every time someone working on your project has a bug, you're accused through paging to be its cause.

Contacts

Victor Apercé

- viaxxx@lse.epita.fr

The end

Thank you for your attention