In short: just say NO TO DRUGS and maybe you won’t end up like the Hurd people.

-- Linus Torvald

Table of Contents

Introduction

- What's an operating system ?
  - Historical reasons
    - Batch processing
    - Unix and time-sharing
  - Hardware abstraction layer
  - Resource sharing
  - Security infrastructure

- The GNU project
  - Free Software
    - Freedom 0: freedom to use
    - Freedom 1: freedom to modify
    - Freedom 2: freedom to distribute
    - Freedom 3: freedom to distribute modified versions
  - Birth of the GNU project
  - Goals of the GNU project

What is a kernel ?

- Many programs don’t need to access hardware directly.

- To increase security, and allow programs to share resources, protections are needed at the hardware level.

- Hardware must at least provide two execution levels :
  - Kernel mode, where everything is allowed
  - User mode, where some instructions are forbidden or restricted

- So, we define two spaces at the software level:
  - Kernel space : code running in kernel mode
  - User space : other programs

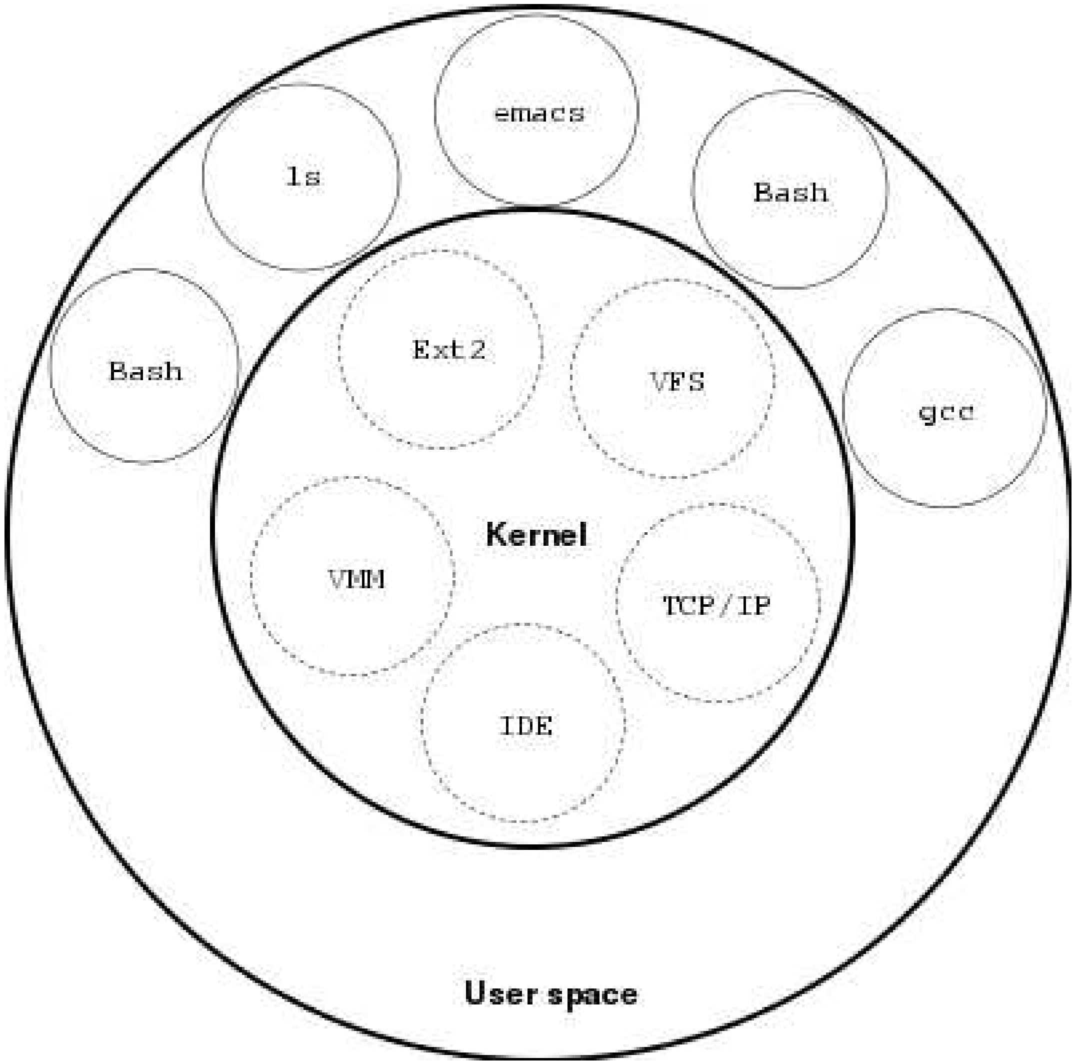
Monolithic kernel based systems

- Traditional design of Unix systems

- Every part common to each program is in kernel space :
  - File systems
  - Scheduler
  - Memory handling
  - Device drivers
  - Network stacks

- Many system calls provided

- Problems :
  - Coding in kernel space is hard
  - Bugs can have strong side-effects
  - Modularity problems



Micro-kernel based systems

- Only parts which really require to be in a privileged mode are in kernel space :
  - IPC (Inter-Process Communication)
  - Basic scheduler
  - Basic memory handling
  - Basic I/O primitives

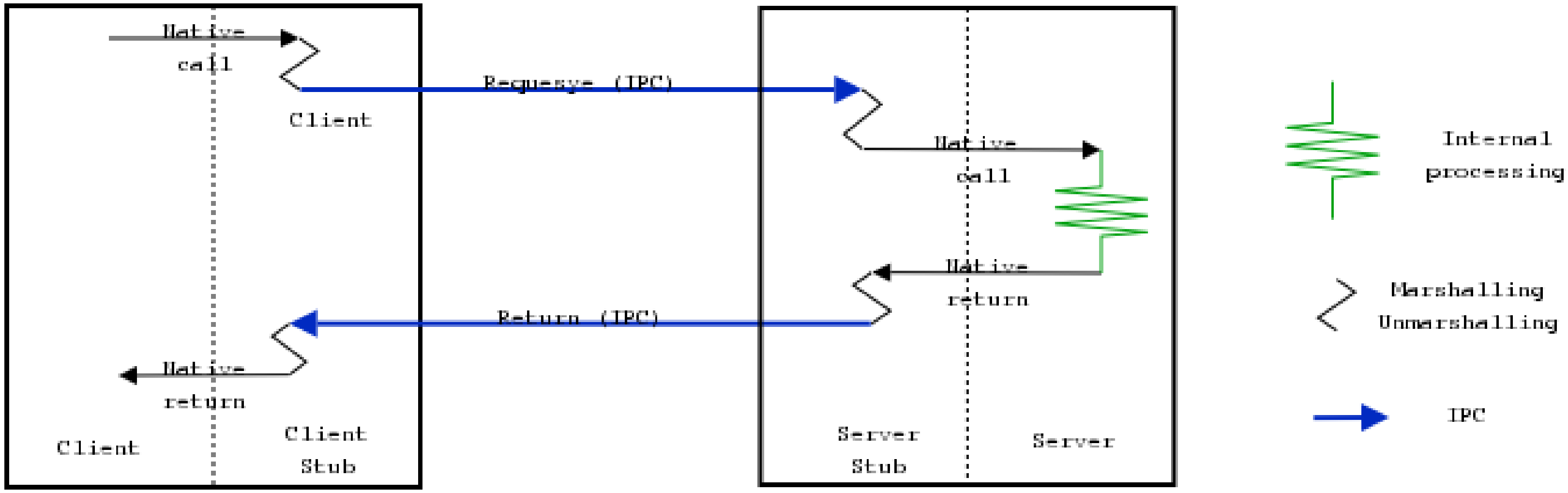
- Many critical parts are now running in user space :
  - Scheduler
  - Memory handling
  - File systems
  - Network stacks

Interlude : Remote Procedure Calls

- Definitions
  - RPC = Remote Procedure Call
  - Two processes are involved: a client, and a server

- How it works
  - The clients sends a message to the server (IPC), specifying which function it wants to call, and with which parameters
  - The server does his own computations
  - The server then does another IPC to convey the result to the client if needed

- Stubs
  - Stubs allow to call RPCs like normal function calls
  - They encode and decode parameters and results (marshalling)
  - Stubs are generated by programs like MiG, IDL4



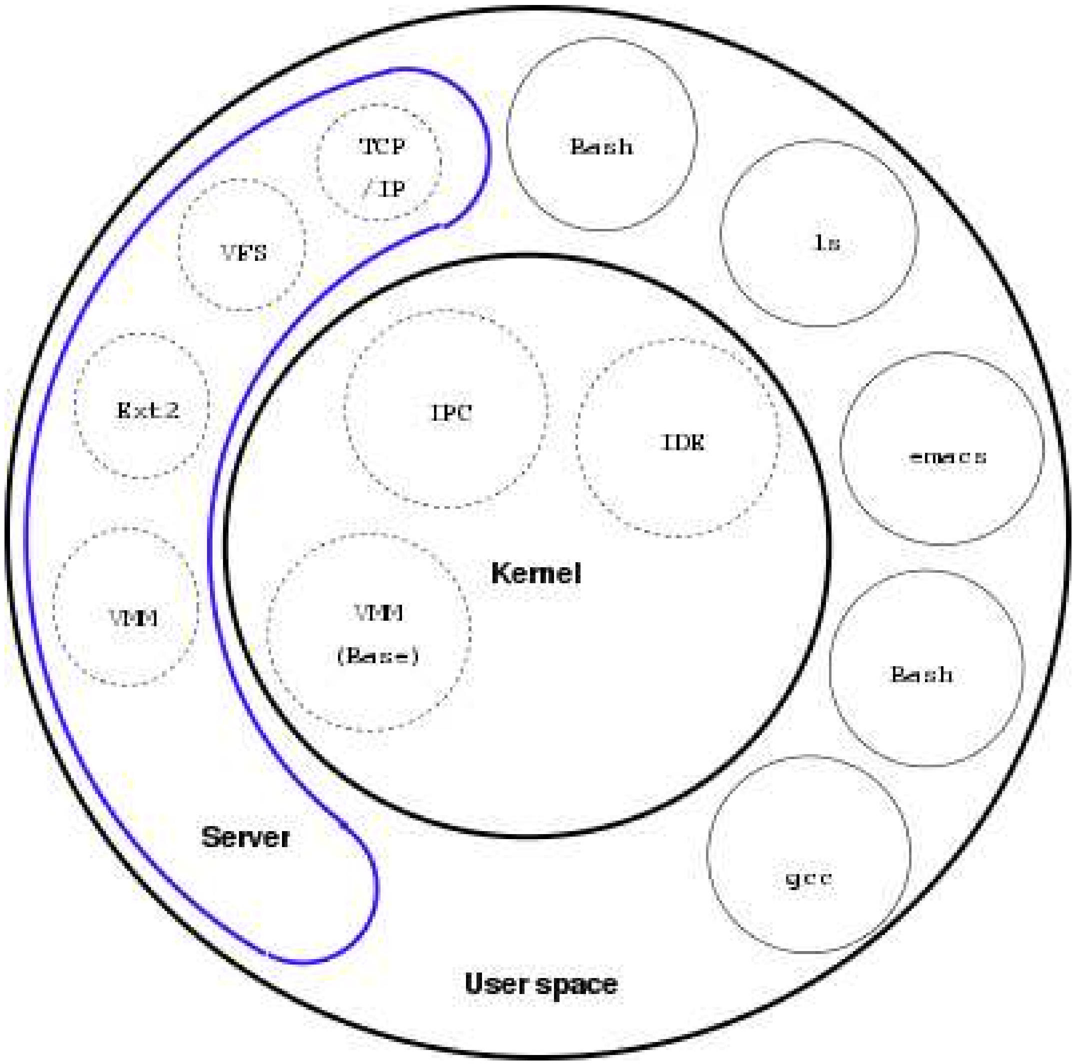
Monoserver systems

- One user-space program (server) handles everything that belonged to the kernel

- Examples :
  - MachOS
  - L4Linux

- Gains :
  - More hardware independence
  - Easier development
  - Security slightly increased

- Problems :
  - Still not modular
  - Side effects still present

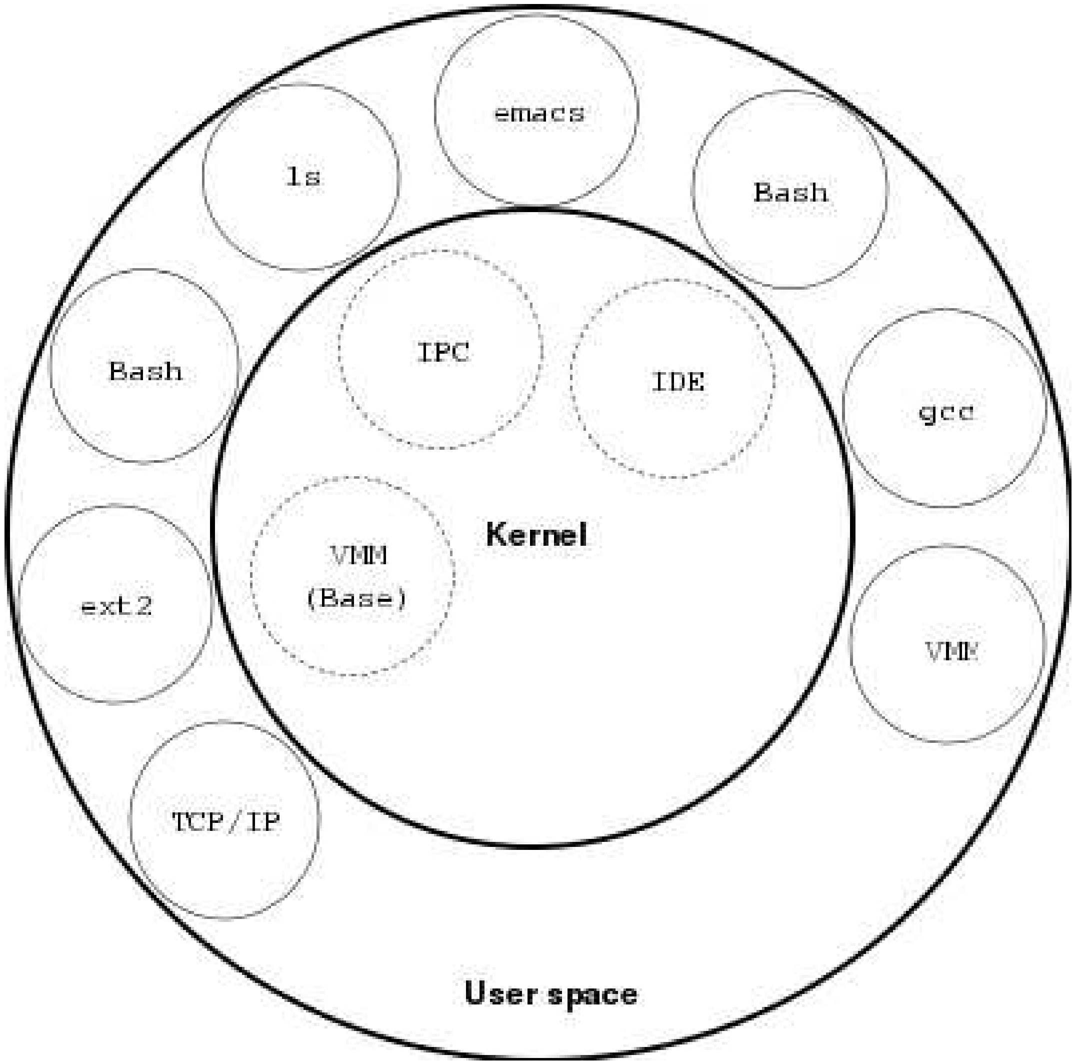


Multi-servers systems

- All features are now split into a set of communicating processes

- Gains :
  - Far more modular
  - Far better fault-tolerance
  - Development really easier

- Problems :
  - IPCs between servers can be slow
  - Well-defined interfaces needed



The GNU Hurd

- Definition
  - A set of servers, libraries and interfaces
  - Vocabulary :
    - The Hurd : the set of servers, neither an OS nor a kernel
    - GNU or GNU/Hurd: the complete operating system

- The Hurd’s goals :
  - Core of the GNU project
  - The GNU manifesto
    - URL : http://www.gnu.org/gnu/manifesto.html
    - Give back freedom to users
    - Stay compatible, but overthrow limits
  - Interfaces clearly defined, and fixed
    - Allow to replace parts of the system
    - Suppress compatibility problems
    - Examples: anonymous file creation, notification
    - It is possible : we have the experience of Unices

History

- 1983 : Richard Stallman starts the GNU project
- 1988 : Mach 3 is chosen as micro-kernel
- 1991 : Mach 3 is released under a Free license
- 1991 : Thomas Bushnell, BSG, founds the Hurd
- 1994 : GNU/Hurd boots for the first time
- 1997 : The Hurd version 0.2 is released
- 1998 : Marcus Brinkmann creates Debian GNU/Hurd
- 2002 : Debian GNU/Hurd is now 4 CDs
- 2002 : Port of the Hurd to L4 is started
- 2002 : POSIX threads are now supported
- 2003 : L4Ka:;Pistachio 0.1 is released
- 2004 : Ext2fs without the 2gb limit reach release candidate
- 2005 : Ext2fs without the 2gb in Debian GNU/Hurd
- 2005 : First program running on L4Hurd
- 2005 : Initial Gnome port

The Mach micro-kernel : history

- One of the first micro-kernel
  - Project of Carnegie-Mellon to implement a relatively new theory
  - A lot of new concepts
    - IPC
    - Designed for multiprocessor systems, and even clusters
    - External pagers
  - First system to clearly define tasks, threads, ...
  - First micro-kernel to be successful, taken and improved by OSF/1 and other research groups

- MachOS base
  - Mono-server (UX being BSD-compatible)
  - Still developed with xMach, but going farther from µkernels

The Mach micro-kernel

- What Mach does
  - Tasks being containers
  - Complex IPC
  - VMM: LRU decision algorithm
  - Basic scheduler
  - Device drivers

- GNU Mach
  - GNU Mach 1.3
  - GNU Mach 2.0
    - OSKit based
  - Slow, and buggy

- Ports
  - Asynchronous IPC
  - Receive right, send right

Translators

- Problems : how to get a port ?
  - Usually : naming services
  - Problems
    - Permissions handling
    - Servers need to register
    - Not so flexible

- Idea of the Hurd :
  - We use the VFS as the naming service
  - file_name_lookup() function
  - Usage in ‘crash’

- Properties of a translator
  - Program running like any other, with the identity and rights of the user launching it
  - Highly multi-threaded to answer different requests simultaneously
  - Answer RPCs :
    - File handling RPCs: io_*, dir_*, ...
    - Others if needed: proc_*, ...

Translator example

```
(mmenal@drizzt, 42) ~ $ id
uid=1004(mmenal) gid=1004(mmenal) groups=1004(mmenal),40(src),50(staff),100(users),518(friends),642(hurdfr)

(mmenal@drizzt, 43) ~ $ settrans -cgap ftp /hurd/hostmux /hurd/ftpfs /
(mmenal@drizzt, 44) ~ $ cd ftp
(mmenal@drizzt, 45) ~/ftp $ ls
(mmenal@drizzt, 46) ~/ftp $ cd ftp.fr.debian.org
(mmenal@drizzt, 47) ~/ftp/ftp.fr.debian.org $ ls
debian  debian-cd  debian-non-US

(mmenal@drizzt, 48) ~/ftp/ftp.fr.debian.org $ ls debian/
README              README.mirrors.html  README.non-US  dists  indices  ls-lR.gz       pool     tools    README.CD-manufacture
README.mirrors.txt   README.pgp     doc    ls-lR    ls-lR.patch.gz  project

(mmenal@drizzt, 49) ~/ftp/ftp.fr.debian.org $ head -n 2 debian/README
See http://www.debian.org/ for information about Debian GNU/Linux.
Three Debian releases are available on the main site:

(mmenal@drizzt, 50) ~/ftp/ftp.fr.debian.org $ cd ..
(mmenal@drizzt, 51) ~/ftp $ ls
ftp.fr.debian.org

(mmenal@drizzt, 52) ~/ftp $
```

Security

- Authentication tokens
  - What’s a token ?
  - The ‘auth’ server
  - Gift, destruction and creation of tokens

- POSIX compatibility
  - UIDs : one kind of tokens
  - Possibility to have many UIDs
  - Possibility to gain and lose UIDs
  - The ‘addauth’ program
  - Suid programs and non-root translators

- Password server
  - Principle
  - Application : ssh or ftp server

- ’noauth’ programs
  - The login shell
  - Useful for untrusted content: gs, browser, ...

Interlude: virtual memory (1)

- Virtual address space
  - Allow kernels to implement protection
  - Allow code to be loaded at arbitrary positions
  - Allow to easily share memory between applications
  - Translation done by hardware (MMU)

- Segmentation
  - A memory address is now an SEG:OFFS couple
  - A segment has a base, a size, and a protection mode
  - Standard segments can be made transparent (code, data, stack)
  - Allow to "swap" a full segment when there is a lack of memory

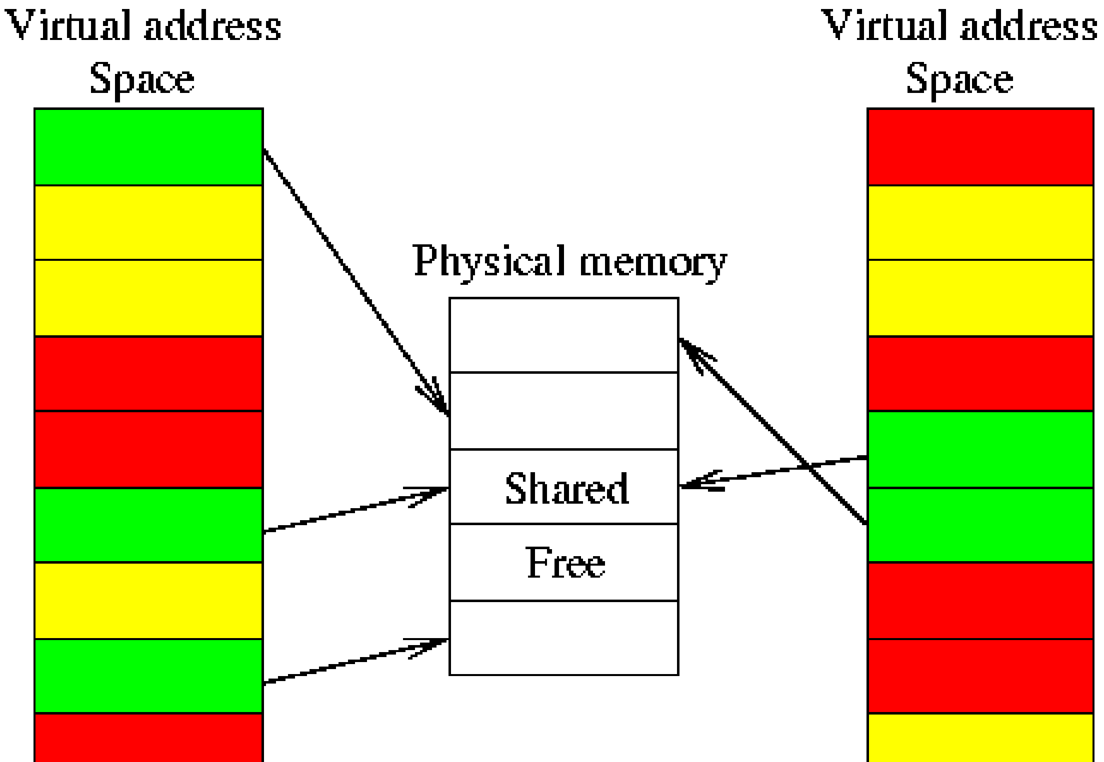
- Problems
  - Granularity not small enough
  - Physical memory fragmentation

; Programs must be aware of segments

Interlude: virtual memory (2) - paging

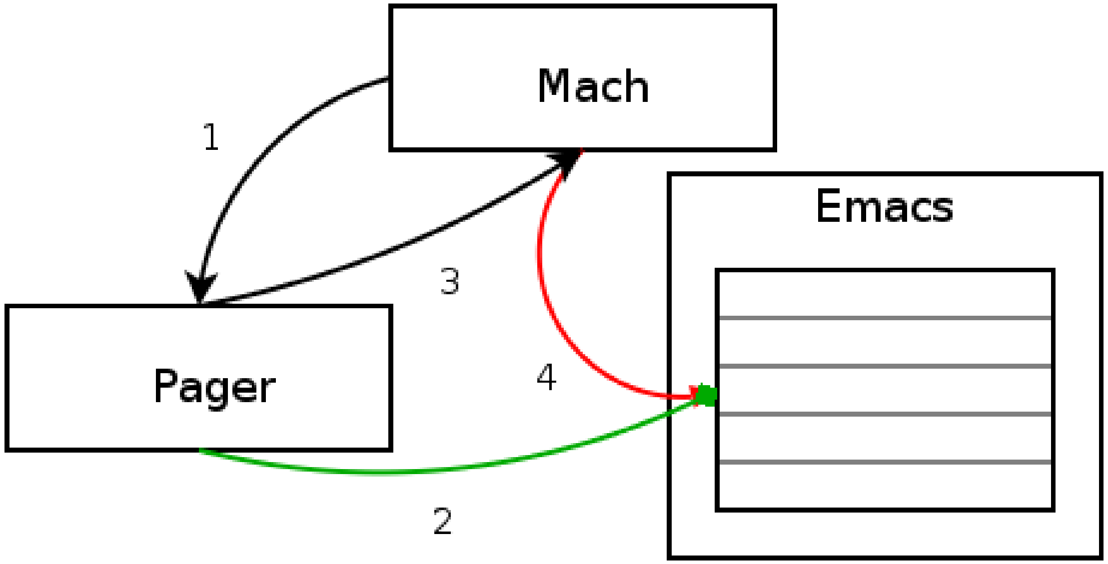
- Paging
  - The virtual address space is linear and contiguous
  - Memory is divided in small pages (4K on ia-32)
  - Allows a far better granularity
  - Transparent for programs

- A page can be :
  - Active and mapped into a physical location (green)
  - Disabled, and transferred to side-storage (yellow)
  - Invalid (red)
  - When a program tries to use a non-mapped page, it triggers a ‘page fault’

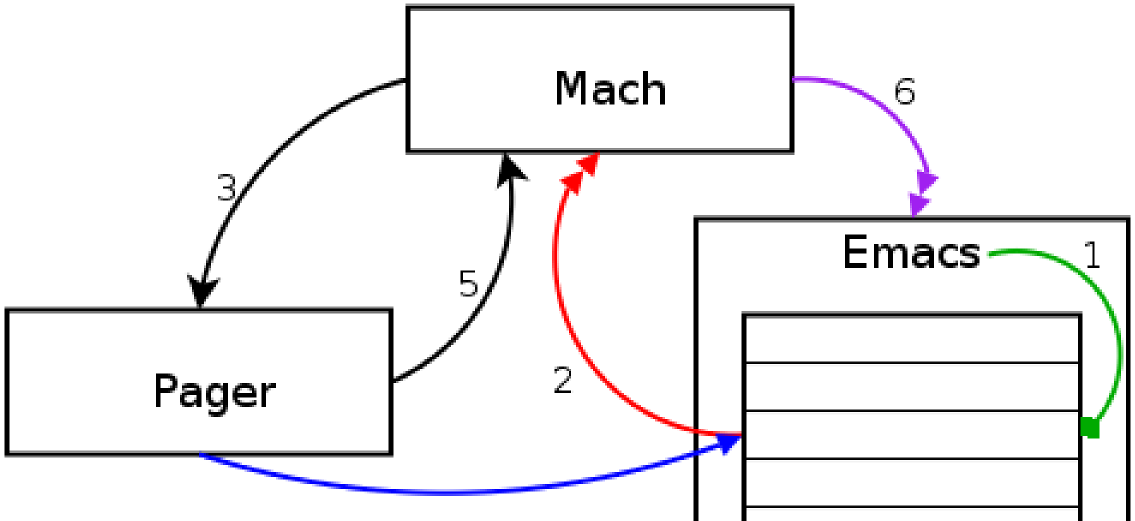


Pagination with Mach (1)

- Principles
  - Mach chooses which page to keep or discard
  - The ‘pagers’ are in user-space
  - When a program faults an IPC is sent to its ‘pager’



1. Mach sends an IPC to the pager
2. Pager reads and store the page
3. Pager answers to Mach IPC
4. Mach removes the page



1. Emacs touch a missing page
2. CPU sends a pagefault message to Mach
3. Mach sends an IPC to the pager
4. The pager restores the page contents
5. The pager answers to Mach's IPC

Pagination with Mach (2)

- Usages
  - Using different backing stores
  - Transparent shared memory inside a cluster of computers

- Main usage in the Hurd: diskfs
  - Principle of diskfs and ext2fs
  - The famous 2GB limit
  - Possible solutions
    - Map all metadata through a tree of ‘smart’ pagers
    - To have a mapping cache
  - Ognyan solution: a mapping cache, with a static mapping of fixed metadata

Current state

- It works...
  - Those slides were displayed using GNU/Hurd
  - Debian GNU/Hurd now fills 9 CDs
  - We have POSIX threads, ...

- ... but it’s still being developed
  - Many features are missing, and some limitations are still present
  - There are still bugs: we need more testers
  - Mach brings many limitations

- It’s slow
  - Mostly because of Mach
  - The code is still far from being optimized

The future: L4

- L4Ka philosophy
  - Defining basic and orthogonal concepts
  - Only provide the most basic mechanisms
  - Make the smallest possible micro-kernel (nano-kernel)
    - Hazelnut: 12K once booted !
  - Always keep performances issues in mind

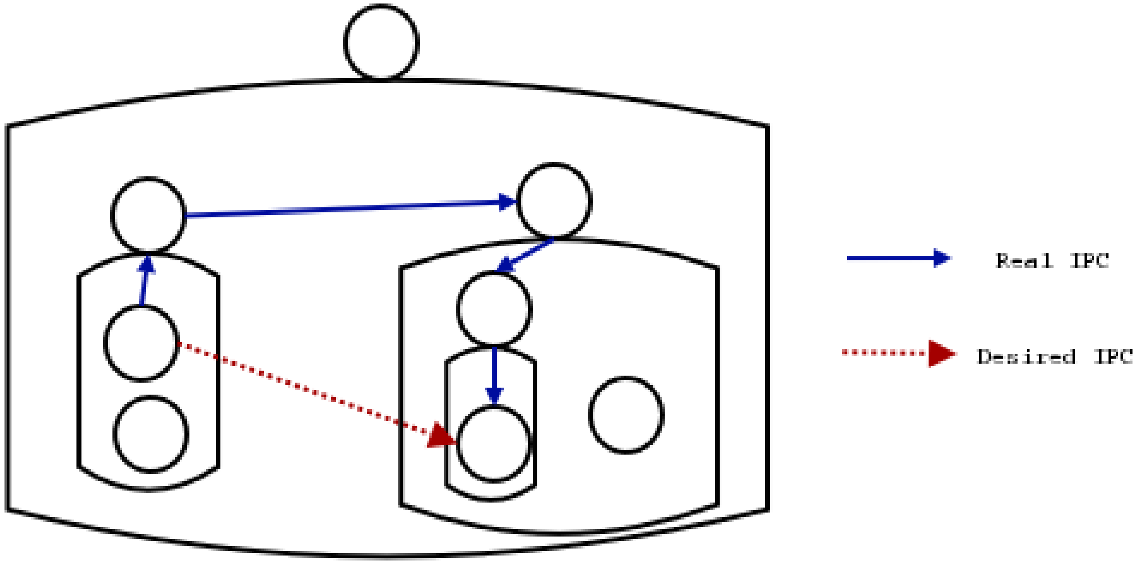
- Very fast IPCs
  - Synchronous IPCs but asynchronous RPCs
  - Smaller code: less pollution of cache lines
  - Optimization technics (address space multiplexing, ...)

- Very few things inside the kernel
  - IPC primitives
  - Scheduling primitives
  - Memory handling primitives
  - I/O primitives
  - The whole Hazelnut: around 11 system calls

L4 security (1)

- Principles
  - With L4, every sensitive operation is done using IPCs
  - Controlling IPCs allow to control the application

- Clans & Chiefs
  - One clan = all the tasks created by the same one task
  - The task creating a clan is called the chief
  - A process can only speak directly to :
    - a member of his own clan (a brother)
    - his chief (his father)
    - a member of the clan he created (a son)
  - Others IPCs must travel through a chain of chiefs, who can drop or alter the message



L4 security (2)

- The new system : IPC redirect
  - Principle
    - Each thread can have a redirector controlling incoming and/or outgoing IPCs
    - Redirectors can be changed at run-time
  - Why a new system ?
    - Clans & Chiefs was too complex and slow
    - It was a decision upon OS policy, and thus shouldn't be inside the kernel
    - It is possible to implement Clans & Chiefs on top of it

- Possibilities
  - Application monitoring (debugging, security, ...)
  - Sand-boxing
    - Allow to execute untrusted native code directly
    - Applications: web, interactive content

Conclusion

- References
  - GNU
    - http://www.gnu.org
  - The GNU Hurd
    - http://hurd.gnu.org
  - Debian GNU/Hurd
    - http://www.debian.org/ports/hurd
  - HurdFr
    - http://hurdfr.org
    - #hurdfr on irc.freenode.net
  - L4
    - http://www.l4ka.org

- Thanks

- Questions ?