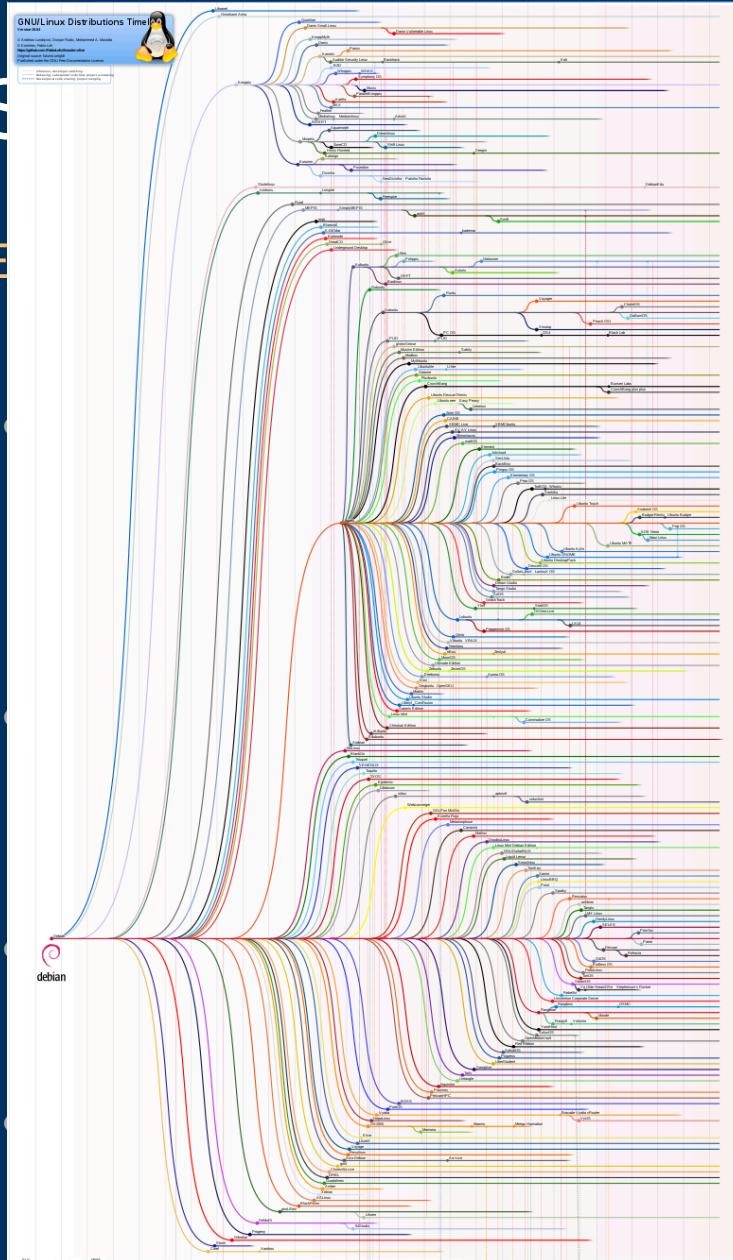


Systèmes d'exploitation (OS)

« Ensemble des programmes qui **dirige** l'utilisation des ressources d'un ordinateur »

- Multi-utilisateur
 - Conçu pour permettre à plusieurs utilisateurs d'interagir simultanément avec le système, avec lequel vient la dimension de « permissions d'accès » et de « temps partagé »
- Multitâche
 - Permet l'exécution simultanée de plusieurs programmes (en apparence), grâce à un changement de contexte très rapide
 - Préemptif vs Coopératif
- Temps-réel
 - Capacité d'un système à respecter un délai de réponse fixé
- Embarqué
 - Grande restriction de ressources (CPU, mémoire...)





tion (

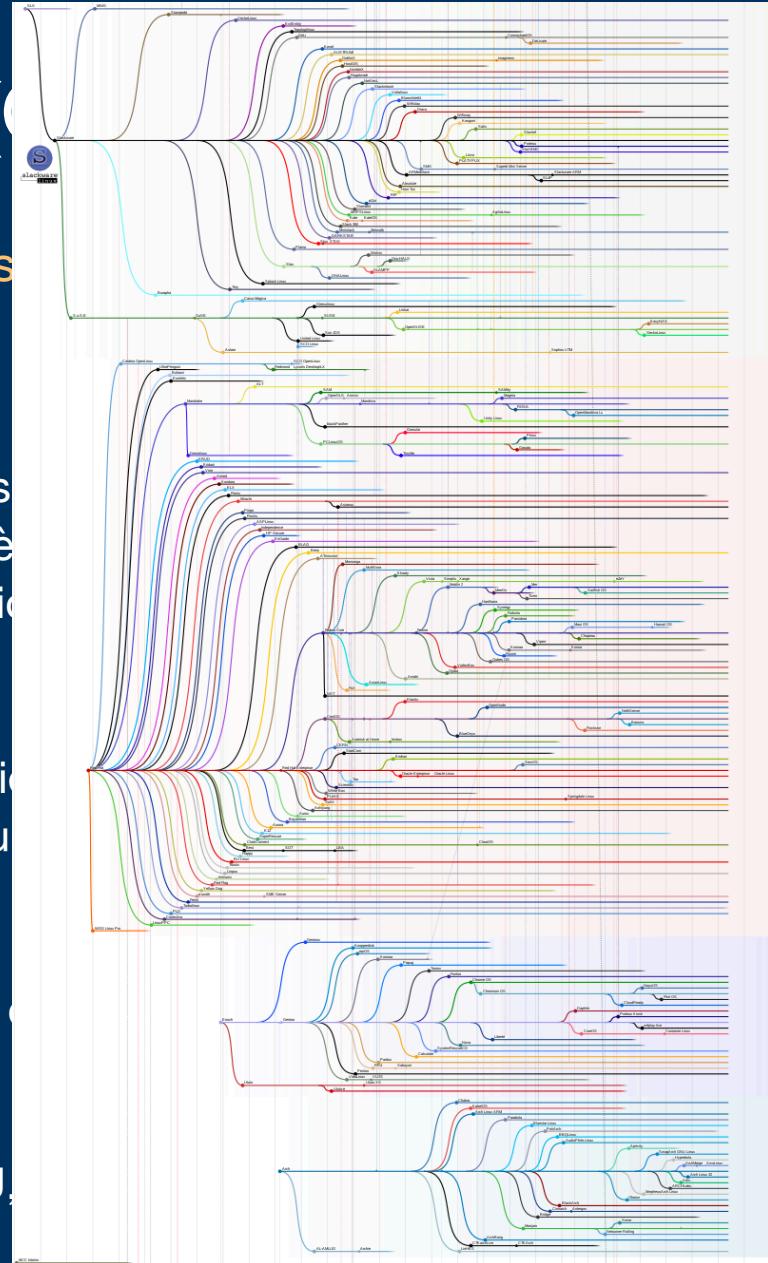
ge l'utilis

eurs utilis
c le systè
permissio
gé »

e de plusi
grâce à u
rapide

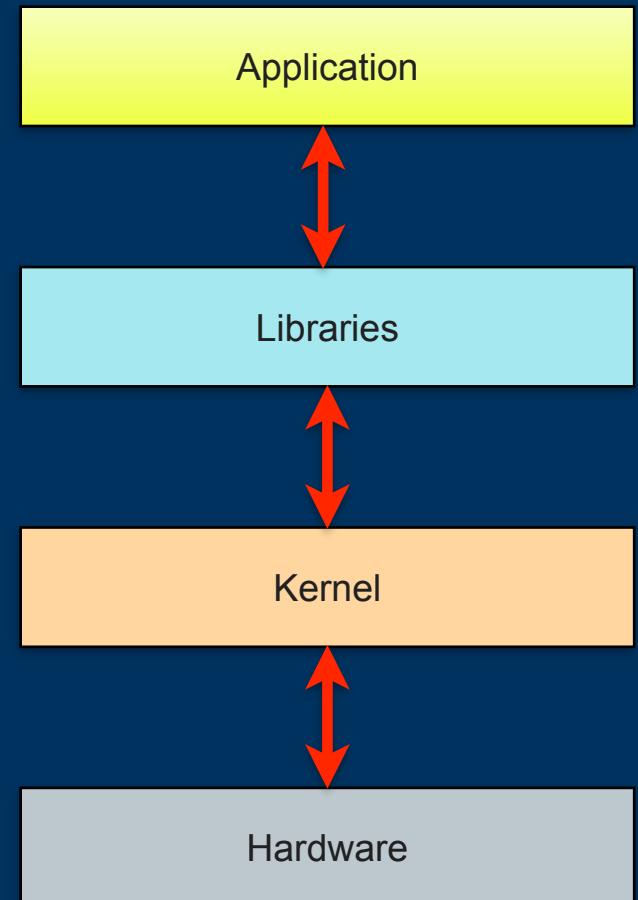
pecter un

ces (CPU)



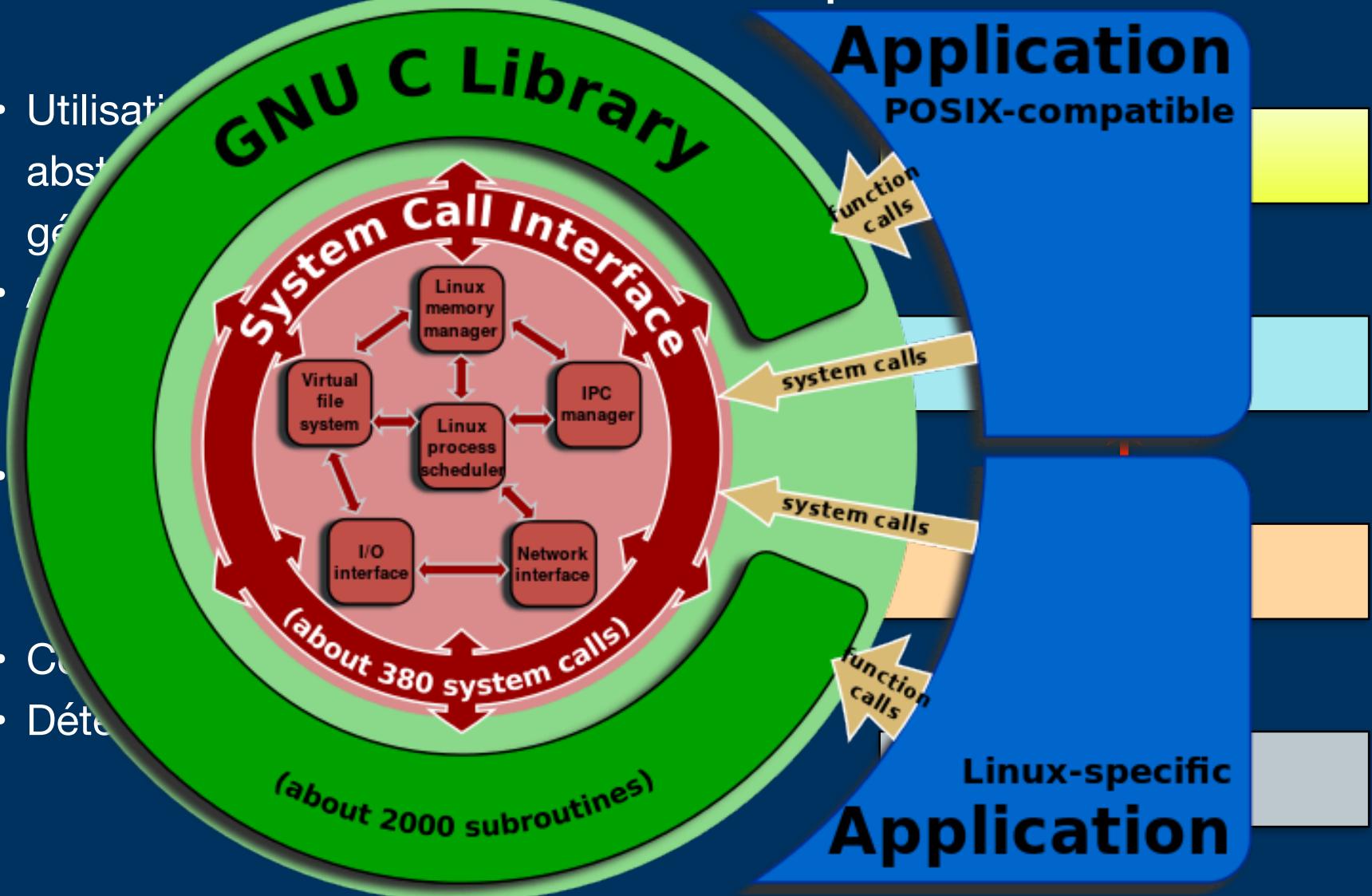
Structure d'un système d'exploitation

- Utilisation des périphériques et abstraction au travers d'interfaces génériques
- Accès aux données par isolation
 - Droits d'accès
 - Privilèges
- Protection des ressources
 - Mémoire
 - CPU
- Contrôle de maintien opérationnel
- Détection d'erreur (résilience)

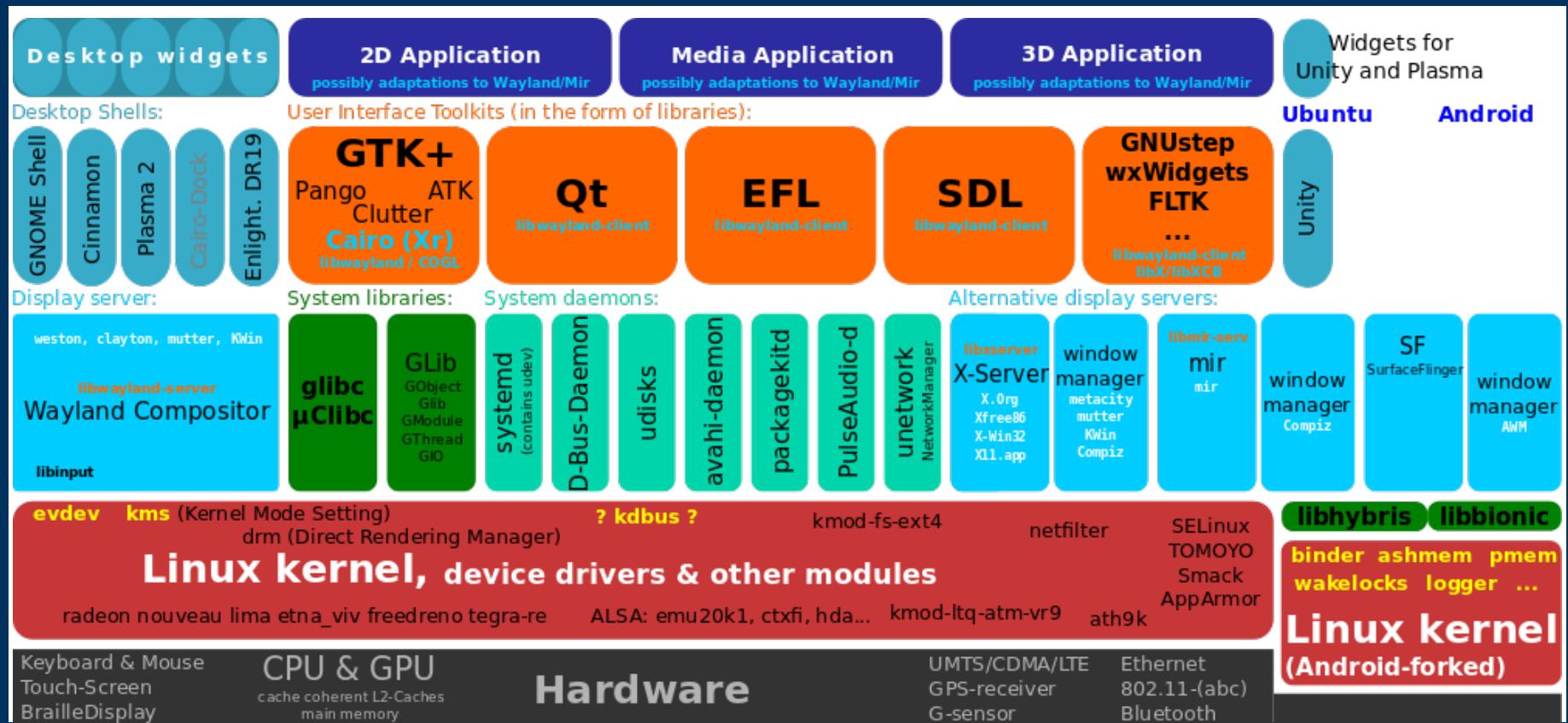


Structure d'un système d'exploitation

- Utilisation d'abstractions générales
- Fonctionnalités
- Configuration
- Détection d'erreurs



« Couches » d'un système d'exploitation sous Linux



« Sticky-bit »

- Sur Fichier
 - Fichier persistant en mémoire
 - Déprécié, voire plus implémentée
- Sur Répertoire
 - Interdit la suppression de fichiers autre que par son propriétaire
 - Utilisé pour les points de montage partagés
- Valeur: 1000 // +t
- Gestion de droits étendus : ACL & SELinux

```
Betelgeuse ~
↳ ls -l / | grep tmp
drwxrwxrwt root root 128 KB Mon Mar  7 10:08:25 2022 tmp
```

Setuid / Setgid

- Transfert de droits entre utilisateurs
- Un programme Set-UID est exécuté **au nom de** l'utilisateur du programme, même si ce n'est pas ce dernier qui l'exécute
- Tous les droits sont propagés (=déguisement)
- Utile quand aucune des autres restrictions n'est adéquate
- Ex: modifier son mot de passe personnel
 - Implique de modifier /etc/shadow, qui contient TOUS les mots de passe utilisateurs
- Cela peut-il poser un problème ?

```
↳ ls -l ./a.out
-rwsrwxr-x. 1 adamj adamj 8296 5 févr. 21:51 ./a.out
```

Et le noyau ?

- Segment privilégié permettant la communication entre logiciel et matériel
- Communication logicielle au travers de « requêtes » émises = les appels systèmes (=*syscalls*)
- Communication matérielle au travers d'un pilote de périphérique (=*driver*), spécifique à un matériel donné. Réalise la conversion s'une interface générique à une interface spécifique
- Assure consistance et cohérence dans un contexte concurrent
- Gestion du partage efficace entre les différentes instances demandeuses de temps de calcul (=ordonnancement)
- Distribution équitable de la mémoire
- **Tous les branchements d'exécution doivent être gérés !**

Depuis le tout début...

1. Le bloc d'alimentation délivre la tension
2. Amorçage de la carte-mère (+checks
 - « Anciennement » BIOS (*Basic Input/Output System*)
 - Récemment UEFI
 - Contrôle d'intégrité (« beeps »)
3. Affichage de l'initialisation

Depuis le tout début...

1. Le b
2. Amc

- <<
- R
- C

3. Affic



American
Megatrends

Version 2.17.1245. Copyright (C) 2015 American Megatrends, Inc.

BIOS Date: 07/14/2015 12:00:00 Ver: BLXSV609

Total Memory: 1048576MB

Processor Type: Intel(R) Xeon(R) CPU E7-8870 v3 @ 2.10GHz

CPU ID:306f4

CPU Cores: 18

Total CPUs: 32

Press F12 go to PXE

Press F11 go to Boot Manager(F3 on Remote Keyboard)

Press to enter setup(F4 on Remote Keyboard)

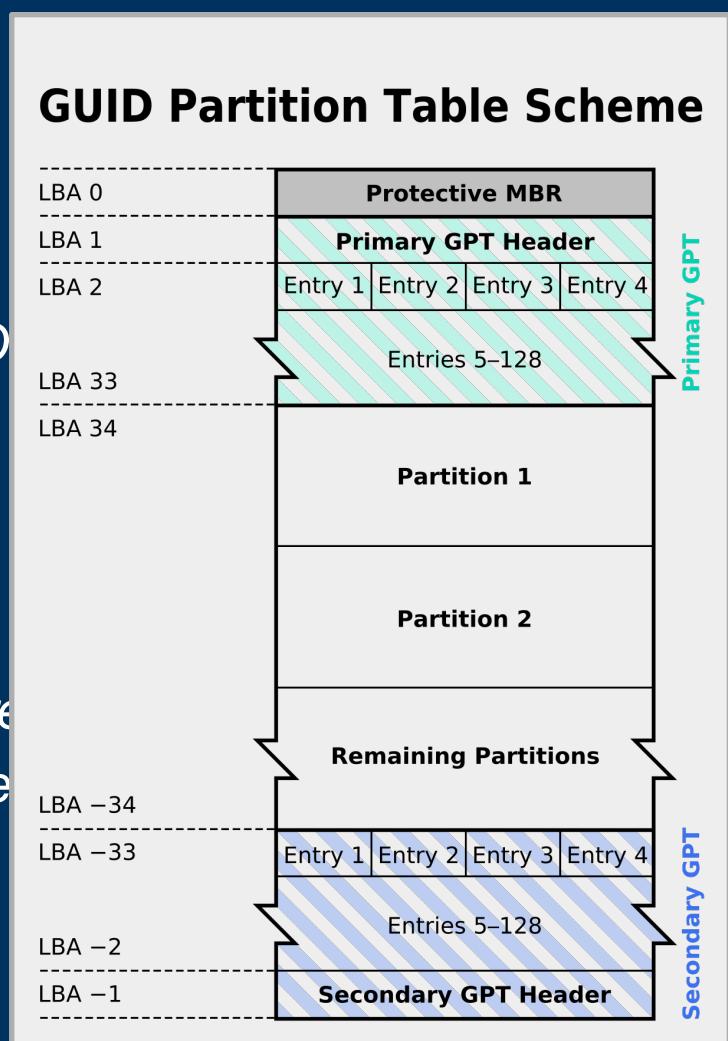
Entering Setup...

Depuis le tout début...

1. Le bloc d'alimentation délivre la tension
2. Amorçage de la carte-mère (+checks)
 - « Anciennement » BIOS (*Basic Input/Output System*)
 - Récemment UEFI
 - Contrôle d'intégrité (« beeps »)
3. Affichage de l'initialisation
4. Chargement de la « zone amorce »
 - « Anciennement » MBR (*master boot record*) »
 - Récemment GPT (*GUID partition Table*)

Depuis le tout début...

1. Le bloc d'alimentation délivre la tension
2. Amorçage de la carte-mère (+checks)
 - « Anciennement » BIOS (*Basic Input/O*)
 - Récemment UEFI
 - Contrôle d'intégrité (« beeps »)
3. Affichage de l'initialisation
4. Chargement de la « zone amorce »
 - « Anciennement » MBR (*master boot record*)
 - Récemment GPT (*GUID partition Table*)



Depuis le tout début...

1. Le bloc d'alimentation délivre la tension
2. Amorçage de la carte-mère (+checks)
 - « Anciennement » BIOS (*Basic Input/Output System*)
 - Récemment UEFI
 - Contrôle d'intégrité (« beeps »)
3. Affichage de l'initialisation
4. Chargement de la « zone amorce »
 - « Anciennement » MBR (*master boot record*) »
 - Récemment GPT (*GUID partition Table*)
5. Le micrologiciel utilise cette table pour lancer le lanceur d'amorçage
 - GRUB2, PXELinux, NTLDLR, winload.exe...

Depuis le tout début...

1. Le bloc d'alimentation délivre la tension

2. Amorçage de la

- « Anciennement
- Récemment
- Contrôle d'information

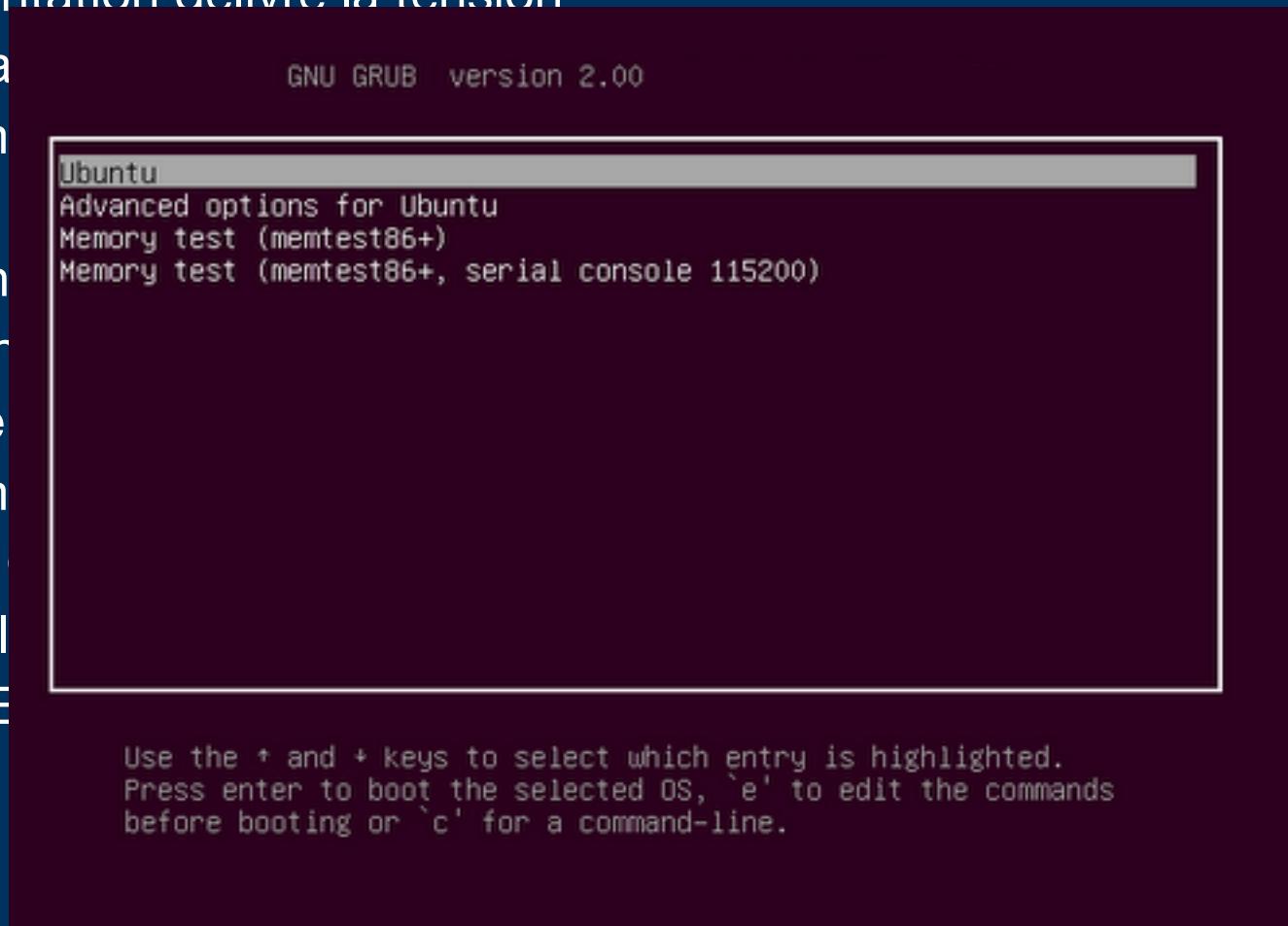
3. Affichage de l'interface

4. Chargement de

- « Anciennement
- Récemment

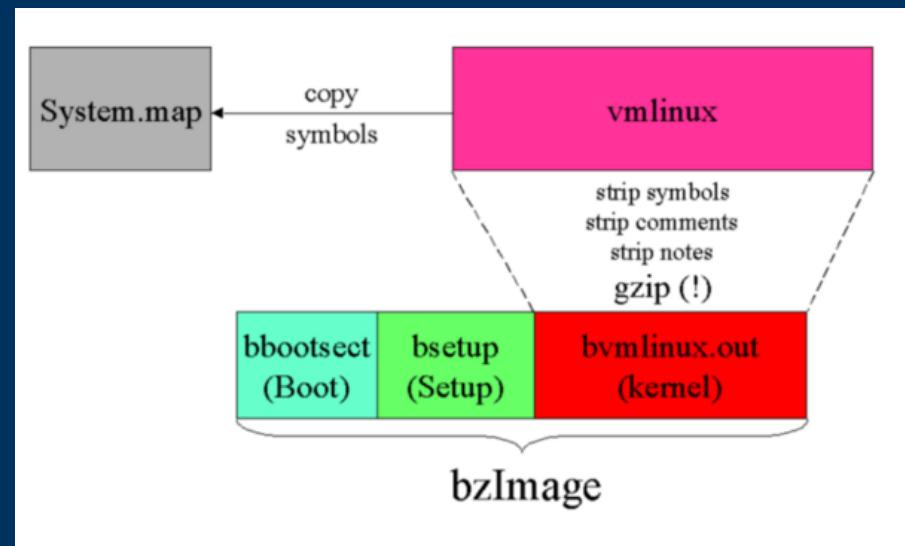
5. Le micrologiciel

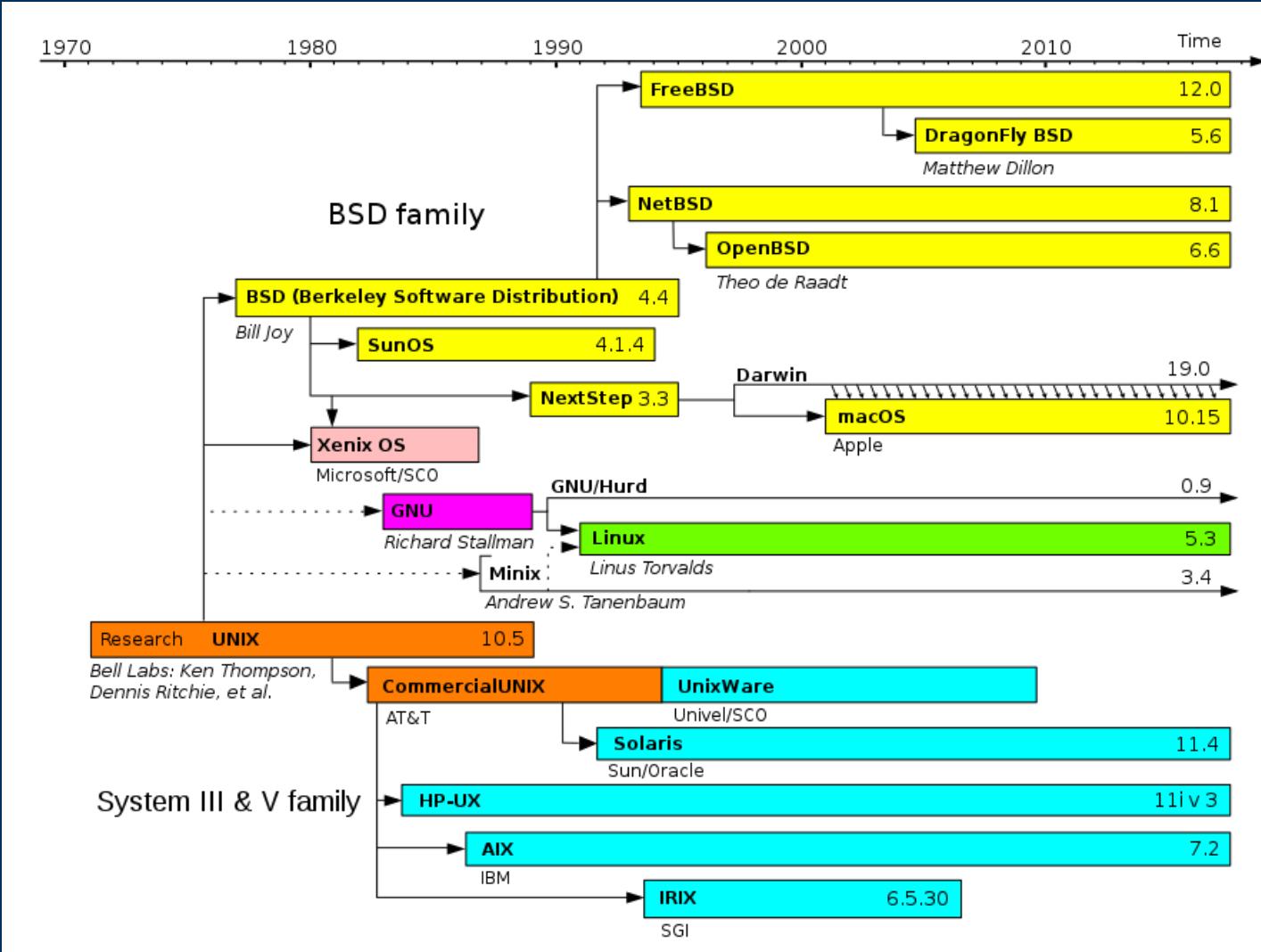
- GRUB2, PXE



Chargement du noyau

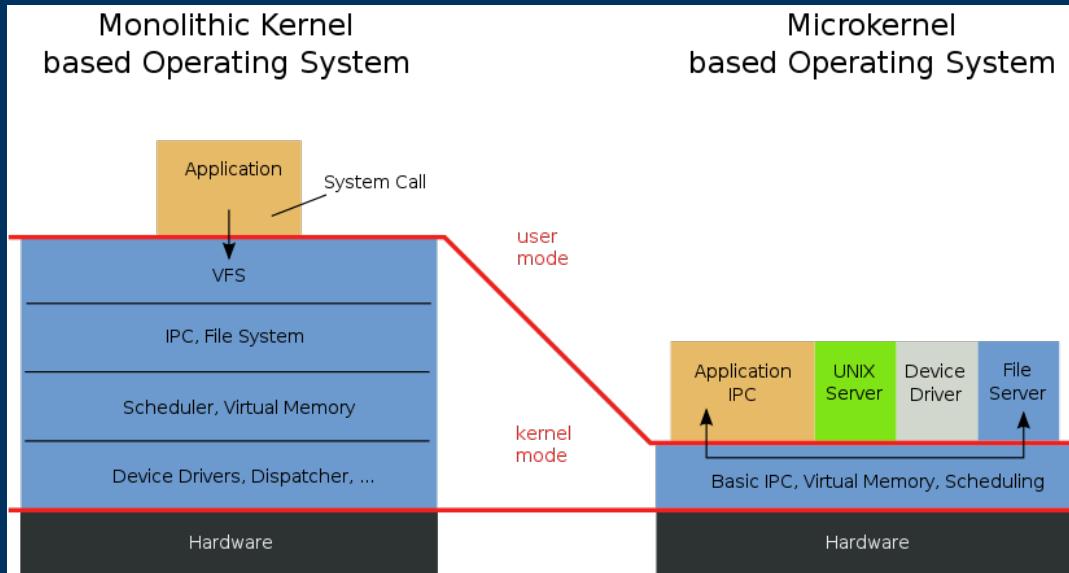
1. Le noyau est souvent stocké compressé. Il est généralement « rangé » dans /boot (c'est là que le chargeur d'amorçage cherche par défaut)
2. Décompression de vmlinu-\$ (version) avec zlib
3. Mise en place:
 1. Filesystem en RAM (initramfs)
 2. Initialisation mémoire (pagination)
 3. Initialisation du hardware
4. `start_kernel()`





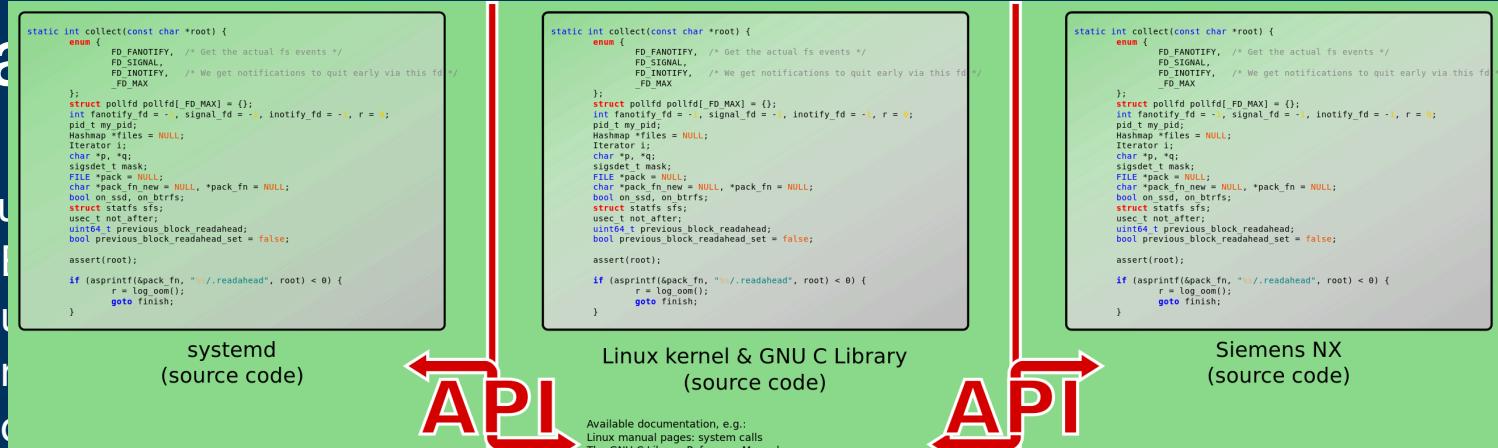
Noyaux monolithiques vs micro-noyaux

- Noyau monolithique:
 - Ensemble du code regroupé dans un seul bloc, évoluant en espace noyau et faisant office d'interface complète.
 - Faible latence
 - Modularité maintenant éprouvée
 - Intolérant aux bugs !
 - ABI moins portable
- Micro-noyau:
 - Un maximum de services du noyau sont déportés dans l'espace utilisateur
 - Le cœur du noyau est restreint au minimum
 - Tolérant aux pannes/bugs
 - Peut avoir une plus grande latence
 - Grande flexibilité / portabilité



Noya

- Noya



stable **API** is guaranteed,
source code remains portable

Compilation

compatible **ABI** can be guaranteed,
machine code becomes portable

binary compatible
(same instruction set,
same compilation environment)

Linux kernel & GNU C Library
(machine code)

binary compatible
(same instruction set,
same compilation environment)

systemd *not*
(machine code) binary compatible

ABI

not
binary compatible (machine code)

```
45 78 3C 1E A6 4F 3D 06 80 E9 D3 A6 B2 A7 A8 88 55 AB D7 34
71 D2 89 B3 98 5D 12 12 42 97 38 F9 32 79 12 93 26 4C
C8 03 F7 B3 65 EB 56 56 AE 5A 5D 6F FF 88 BE 70 01 93 27 4E
F9 D0 B2 60 5B 70 F2 A4 89 19 3D 83 F2 8A 34 5D 4C 28 4C
AA 2A 62 AC 45 CC BC 8E 98 39 83 F2 8A 04 34 5D A3 6F
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
50 9D 18 D3 D5 AA D5 AC 5D B7 8E BB 6E BF 8D 19 D3 A6 32 76
CC 68 1E FB 21 34 SD 83 FF 70 C4 9A 85 EB 08 B2 75 2B
37 5D 77 10 BD 7A F6 24 3E 8E EE 5D BB 86 FB 1F C9 FC A0
69 35 E4 71 8E 50 E8 09 18 07 B5 ED 71 87 3E 44 99 55 05
35 5C 6E 02 1F AF 0F 05 EA 31 76 85 69 6D 61 FD 29 4A E3
FB 0B 8D 71 77 31 58 53 EF 37 67 DC 48 D7 90 57 54 BA 89 EF
A9 BF B3 63 D7 6E B2 32 D2 C9 CA 48 27 2F BF 88 39 4F FE 9D
F2 CA 16 D8 4F 83 25 2B 87 BB EC D1 29 53 70 39 53 3C
FB B7 47 F9 F2 EB 6F 2B 2F E7 F6 98 C2 66 B3 35 D8 B9
46 7D 09 76 2B E8 48 0C C5 7F BB 9C 6E 39 BC F1
D0 DB 8D 60 5B 70 F2 A4 89 19 3D 83 F2 8A 04 34 5D 4C
B9 75 FA EE CE CF A7 5B 07 AE E1 F7 C0 60 30 88 CF E7 8B 87
9F 7E EA 29 6C F8 E5 11 E6 C6 D9 CF C2 A5 48 CB CA CC 64 F6
09 C7 F3 FC 8B 2F B1 76 FD 7A 96 FD FC 33 C3 87 80 25 2B B3
13 1F 7E FC 31 00 83 06 0C 08 BF D1 AF 5F 00 76 EC CC AB
53 5E B8 FD 6D ED 2B 6D AC 48 19 88 59 C7 1F 27
C1 F8 S1 E9 AB 6F BE 05 E0 CC 73 CE AB BB FD DB 0F 19 36 74
```

```
45 78 3C 1E A6 4F 3D 06 80 E9 D3 A6 B2 A7 A8 88 55 AB D7 34
71 D2 89 B3 98 5D 12 12 42 97 38 F9 32 79 12 93 26 4C
C8 03 F7 B3 65 EB 56 56 AE 5A 5D 6F FF 88 BE 70 01 93 27 4E
F9 D0 B2 60 5B 70 F2 A4 89 19 3D 83 F2 8A 04 34 5D 4C
AA 2A 62 AC 45 CC BC 8E 98 39 83 F2 8A 04 34 5D A3 6F
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
50 9D 18 D3 D5 AA D5 AC 5D B7 8E BB 6E BF 8D 19 D3 A6 32 76
CC 68 1E FB 21 34 SD 83 FF 70 C4 9A 85 EB 08 B2 75 2B
37 5D 77 10 BD 7A F6 24 3E 8E EE 5D BB 86 FB 1F C9 FC A0
69 35 E4 71 8E 50 E8 09 18 07 B5 ED 71 87 3E 44 99 55 05
35 5C 6E 02 1F AF 0F 05 EA 31 76 85 69 6D 61 FD 29 4A E3
FB 0B 8D 71 77 31 58 53 EF 37 67 DC 48 D7 90 57 54 BA 89 EF
A9 BF B3 63 D7 6E B2 32 D2 C9 CA 48 27 2F BF 88 39 4F FE 9D
F2 CA 16 D8 4F 83 25 2B 87 BB EC D1 29 53 70 39 53 3C
FB B7 47 F9 F2 EB 6F 2B 2F E7 F6 98 C2 66 B3 35 D8 B9
46 7D 09 76 2B E8 48 0C C5 7F BB 9C 6E 39 BC F1
D0 DB 8D 60 5B 70 F2 A4 89 19 3D 83 F2 8A 04 34 5D 4C
B9 75 FA EE CE CF A7 5B 07 AE E1 F7 C0 60 30 88 CF E7 8B 87
9F 7E EA 29 6C F8 E5 11 E6 C6 D9 CF C2 A5 48 CB CA CC 64 F6
09 C7 F3 FC 8B 2F B1 76 FD 7A 96 FD FC 33 C3 87 80 25 2B B3
13 1F 7E FC 31 00 83 06 0C 08 BF D1 AF 5F 00 76 EC CC AB
53 5E B8 FD 6D ED 2B 6D AC 48 19 88 59 C7 1F 27
C1 F8 S1 E9 AB 6F BE 05 E0 CC 73 CE AB BB FD DB 0F 19 36 74
```

```
45 78 3C 1E A6 4F 3D 06 80 E9 D3 A6 B2 A7 A8 88 55 AB D7 34
71 D2 89 B3 98 5D 12 12 42 97 38 F9 32 79 12 93 26 4C
C8 03 F7 B3 65 EB 56 56 AE 5A 5D 6F FF 88 BE 70 01 93 27 4E
F9 D0 B2 60 5B 70 F2 A4 89 19 3D 83 F2 8A 04 34 5D 4C
AA 2A 62 AC 45 CC BC 8E 98 39 83 F2 8A 04 34 5D A3 6F
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
B9 01 E3 4C 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
50 9D 18 D3 D5 AA D5 AC 5D B7 8E BB 6E BF 8D 19 D3 A6 32 76
CC 68 1E FB 21 34 SD 83 FF 70 C4 9A 85 EB 08 B2 75 2B
37 5D 77 10 BD 7A F6 24 3E 8E EE 5D BB 86 FB 1F C9 FC A0
69 35 E4 71 8E 50 E8 09 18 07 B5 ED 71 87 3E 44 99 55 05
35 5C 6E 02 1F AF 0F 05 EA 31 76 85 69 6D 61 FD 29 4A E3
FB 0B 8D 71 77 31 58 53 EF 37 67 DC 48 D7 90 57 54 BA 89 EF
A9 BF B3 63 D7 6E B2 32 D2 C9 CA 48 27 2F BF 88 39 4F FE 9D
F2 CA 16 D8 4F 83 25 2B 87 BB EC D1 29 53 70 39 53 3C
FB B7 47 F9 F2 EB 6F 2B 2F E7 F6 98 C2 66 B3 35 D8 B9
46 7D 09 76 2B E8 48 0C C5 7F BB 9C 6E 39 BC F1
D0 DB 8D 60 5B 70 F2 A4 89 19 3D 83 F2 8A 04 34 5D 4C
B9 75 FA EE CE CF A7 5B 07 AE E1 F7 C0 60 30 88 CF E7 8B 87
9F 7E EA 29 6C F8 E5 11 E6 C6 D9 CF C2 A5 48 CB CA CC 64 F6
09 C7 F3 FC 8B 2F B1 76 FD 7A 96 FD FC 33 C3 87 80 25 2B B3
13 1F 7E FC 31 00 83 06 0C 08 BF D1 AF 5F 00 76 EC CC AB
53 5E B8 FD 6D ED 2B 6D AC 48 19 88 59 C7 1F 27
C1 F8 S1 E9 AB 6F BE 05 E0 CC 73 CE AB BB FD DB 0F 19 36 74
```

Device Drivers, Dispatcher, ...

kernel mode

Basic IPC, Virtual Memory, Scheduling

Hardware

Hardware

Linux kernel map

functions
layers

user space
interfaces

virtual

bridges

functional

devices
control

hardware
interfaces

electronics

system

kernel/

system interfaces
linux/syscalls.h
asm-i386/uaccess.h
copy_from_user
cdev = cdev_add
register_chrdev
cdev_map
sys_reboot
sys_init_module

processing

kernel/

processes
sys_execve
fs/exec.c
sys_fork
sys_vfork
sys_clone
linux_binfmt
sys_nanosleep

memory

mm/

Virtual memory
virtually continuous memory
find_vma_prepare
vmalloc
vmlist
vm_struct
vma_to_page

storage

fs/

Virtual File System
files & directories access
sys_open
do_path_lookup
sys_read
sys_write
sys_mount
sys_sync

networking

net/

Protocol families
sockets access
sys_socketcall
sys_socket
socket_file_ops
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops

human interface

HI char devices
cdev
input_fops
console_fops
fb_fops
video_fops
snd_fops
kmmsg
sys_syslog

security/
Security

may_open
security_socket_create
security_inode_create
printk
security_ops
selinux_ops

HI subsystems

Protocols

proto
/proc/net/protocols
tcp_prot
ip_rcv
ip_queue_xmit
ip_forward
sk_buff

Virtual network device

net_device
netif_rx
dev_queue_xmit
alloc_etherdev
alloc_ieee80211
ether_setup
ieee80211_rx
ieee80211_xmit

Network devices drivers

net
e100_open
rtl8139_open
ipw2100_open
zd1201_net_open

HI peripherals device drivers

atkbd_drv
psmouse
I2O42_driver
a997_driver
drivers/hid/en

User peripherals
keyboard
mouse
cam
audio
graphics card

Device Model

kset
subsystem_register
bus_type
class
device
device_create
class_device
class_device_create
device_driver
probe
driver_register

threads

work_struct
workqueue_struct
create_workqueue
kthread_create
kernel_thread
current
thread_info
do_fork

Synchronization

wake_up
add_timer
msleep
wake_up
add_timer
msleep

system run

init/kernel/
kernel_restart
kernel_power_off
int/main.c
start_kernel
dp_initcalls_run
init_process
load_module
module
request_region
request_mem_region
ioremap
usb_hcd

generic HW access

include/asm/
arch/i386/
usb_driver
pci_driver
pd_registered_driver
usb_hcd_giveback_urbs
request_region
request_mem_region
ioremap
usb_hcd
ehci_irq
ehci_urbs
pciread
pciwrite
outw_inw
usb_hcd_irq
PCI
PCI Controller
DMA
USB Controller
controller

devices access and bus drivers

writew
readw
ehci_urbs
pciread
pciwrite
enqueue

Scheduler

kernel/sched.c
task_struct
schedule_timeout
setup_timer
process_timeout
activate_task
rq
context_switch

interrupt context

timer_list
jiffies
++
setup_irq
timer_interrupt
do_softirq
do_IRQ - irq_desc
tasklet_struct
atomic_t
switch_to
pt_regs
show_regs
trap_init
cli_sti

CPU specific

arch/i386/kernel/
start_thread
/proc/interrupts
system_call
show_regs
pt_regs
trap_init
atomic_t
switch_to
cli_sti

Memory Mapping

mm/mmap.c
do_mmap_pgoff
vma_fix
mm_struct
vm_area_struct
mmap_pgoff
vma_fix
mm_struct
vm_area_struct

logical memory

mm/slab.c
/proc/slabinfo
physically mapped memory
kfree
kmalloc
kmem_cache_alloc
kmem_cache
slab

Page Allocator

mm/page_alloc.c
/proc/buddyinfo
get_free_pages
zone
page
alloc_pages
get_page_from_freelist
try_to_free_pages

Physical memory operations

arch/i386/mm/
atomic_t
switch_to
do_page_fault
die
do_page_fault

Page Cache

do_sync
address_space
pdfflush

Logical File Systems

ext3_file_operations
ext3_get_sb
super_block
inode
file_operations
file_system_type
get_sb
rname_fs_type
ramfs_fs_type
nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport

Block devices

block/
block_device_operations
gendisk
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
Scsi_Host
libata
ahci_pci_driver
alc94xx_init
ide_disk_ops
ide_intr
ide_do_request

Disk controllers

SCSI
IDE
SATA

Network controllers

Ethernet
WIFI

user peripherals

keyboards
mouse
cam
audio
graphics card

memory

RAM
MMU

CPU

registers
APIC
interrupt controller

disk controllers

SCSI
IDE
SATA

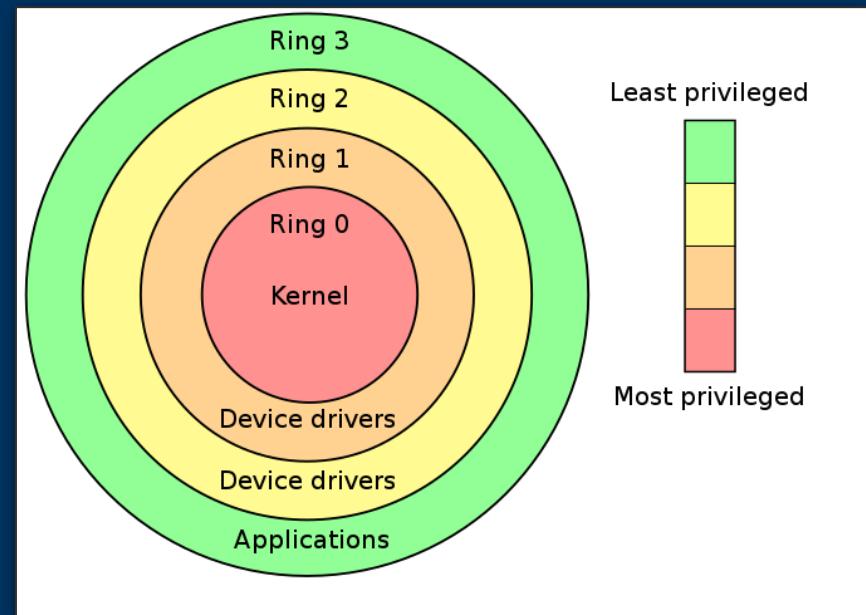
network controllers

Ethernet
WIFI

© 2007 Constantine Shalyapin
www.LinuxDriver.co.uk/kernel_map

Noyaux et anneaux de protection

- Autre nom: *Hierarchical Protection domains*
- Fournit plusieurs niveaux d'accès aux ressources, de manière hiérarchique (l'accès à un niveau privilégié permet aussi l'accès à tous les niveaux moins privilégiés)
- X86 vient avec 4 niveaux, ring0 est dédié au noyau, et ring3 à l'espace utilisateur
- User mode
- Supervisor mode
 - Mode d'exécution privilégiée
 - **Rien à voir avec root...**
- « Hypervisor » mode: les machines virtuelles d'aujourd'hui encapsulent généralement l'OS invité dans son propre set, créant plus profondément ses propres niveau



Noyaux et capacités

- Autre système de sécurité mis en place par la majorité des noyaux
- Permet d'éviter un « tout-ourien » pour les processus non-privilégiés, à grain plus fin que les anneaux de protection
- *Ex: Comment avoir la possibilité de tracer un programme sans être root/dans le noyau ?*
- Les accès sont rangés par groupes de privilèges
- Un programme peut recevoir l'autorisation d'utiliser les permissions d'un groupe => **une capacité**

```
↳ getcap `which ping`  
/usr/bin/ping = cap_net_admin,cap_net_raw+p
```

```
cap_audit_control  
cap_audit_read  
cap_audit_write  
cap_block_suspend  
cap_chown  
cap_dac_override  
cap_dac_read_search  
cap_fowner  
cap_fsetid  
cap_ipc_lock  
cap_ipc_owner  
cap_kill  
cap_lease  
cap_linux_immutable  
cap_mac_admin  
cap_mac_override  
cap_mknod  
cap_net_admin  
cap_net_bind_service  
cap_net_broadcast  
cap_net_raw  
cap_setfcap  
cap_setgid  
cap_setpcap  
cap_setuid  
cap_sys_admin  
cap_sys_boot  
cap_sys_chroot  
cap_syslog  
cap_sys_module  
cap_sys_nice  
cap_sys_pacct  
cap_sys_ptrace  
cap_sys_rawio  
cap_sys_resource  
cap_sys_time  
cap_sys_tty_config  
cap_wake_alarm
```

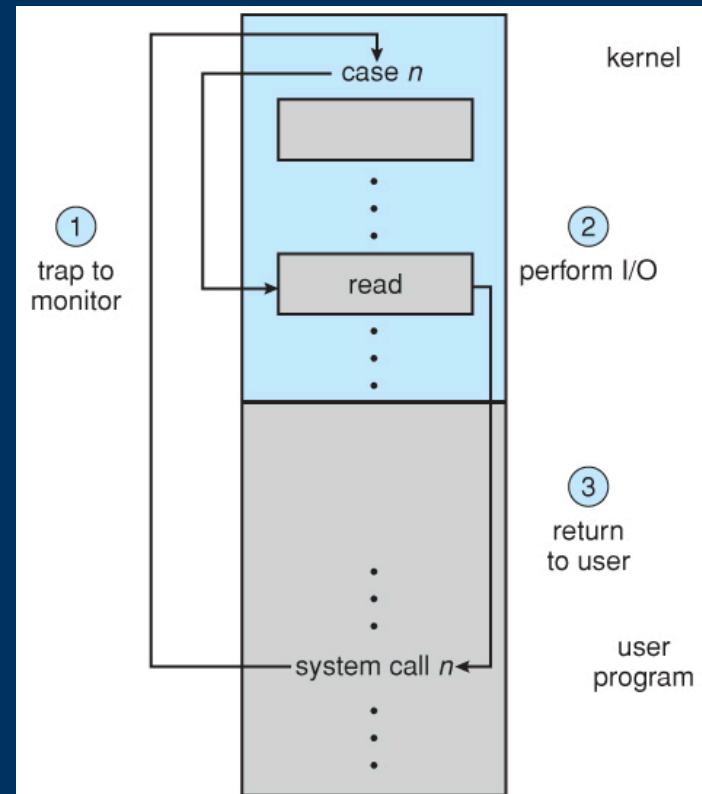
Noyaux et capacités

- En ligne de commande
 - Lister les capacités: getcap <fichier>
 - Changer les capacités: setcap <capacité> <fichier>
- Une capacité est composé de deux termes:
 - Un nom (« cap_net_raw », « cap_sys_module »...)
 - Un modificateur (=, +, -)
 - Une permission
 - p: peut être utilisé (=permitted)
 - e: effectivement utilisé (=effective)
 - i: Héritée lors d'un appel à fork() / exec() (=inherited)

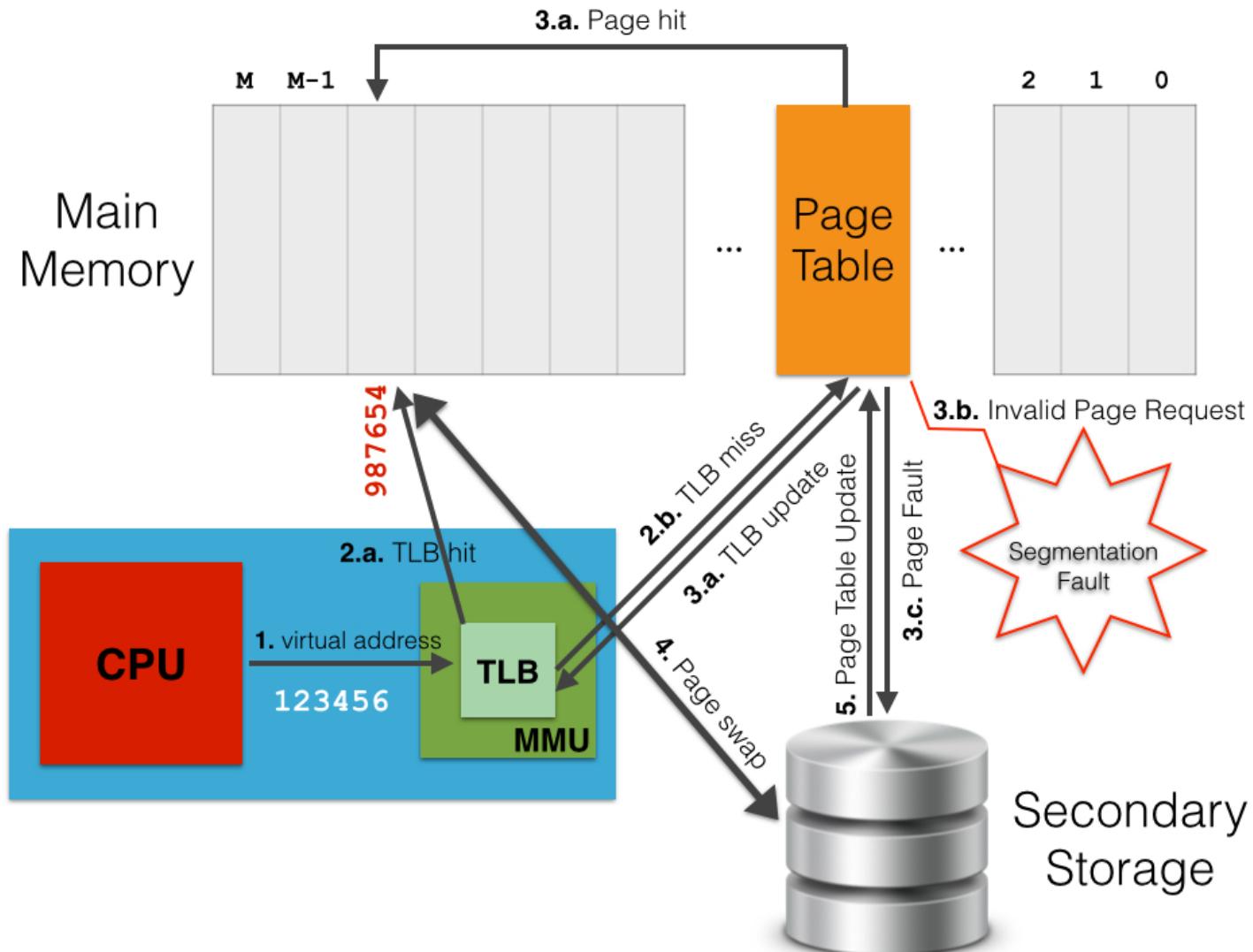
```
(root) → linux-5.5.7 setcap cap_net_raw+ep /bin/ping && ping google.com
PING google.com (216.58.213.142) 56(84) bytes of data.
64 bytes from par21s03-in-f14.1e100.net (216.58.213.142): icmp_seq=1 ttl=50 time=3.75 ms
64 bytes from par21s03-in-f14.1e100.net (216.58.213.142): icmp_seq=2 ttl=50 time=2.64 ms
64 bytes from par21s03-in-f14.1e100.net (216.58.213.142): icmp_seq=3 ttl=50 time=2.70 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 2.649/3.034/3.753/0.510 ms
```

Composants noyau

- System Call Interface (SCI)
 - Couche d'abstraction espaces utilisateur/noyau permettant aux processus réguliers d'interagir avec le noyau et ses ressources privilégiées
- Process Management (PM)
 - Crée / supprime les processus
 - Support de la communication inter-processus
 - Gestion du PCB dans l'espace noyau de l'espace d'adressage
 - Ordonnancement sur CPU

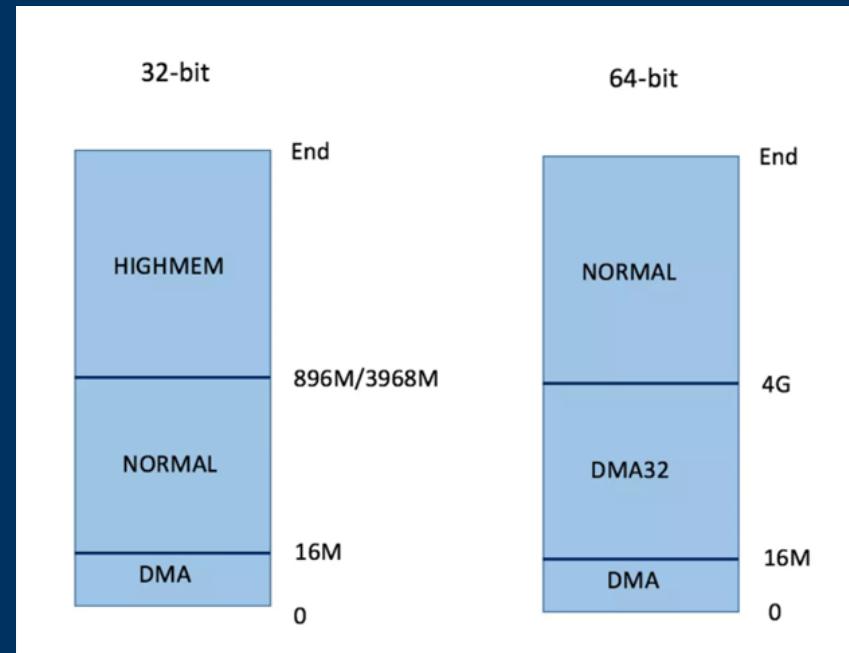


Noy



Noyau & Mémoire

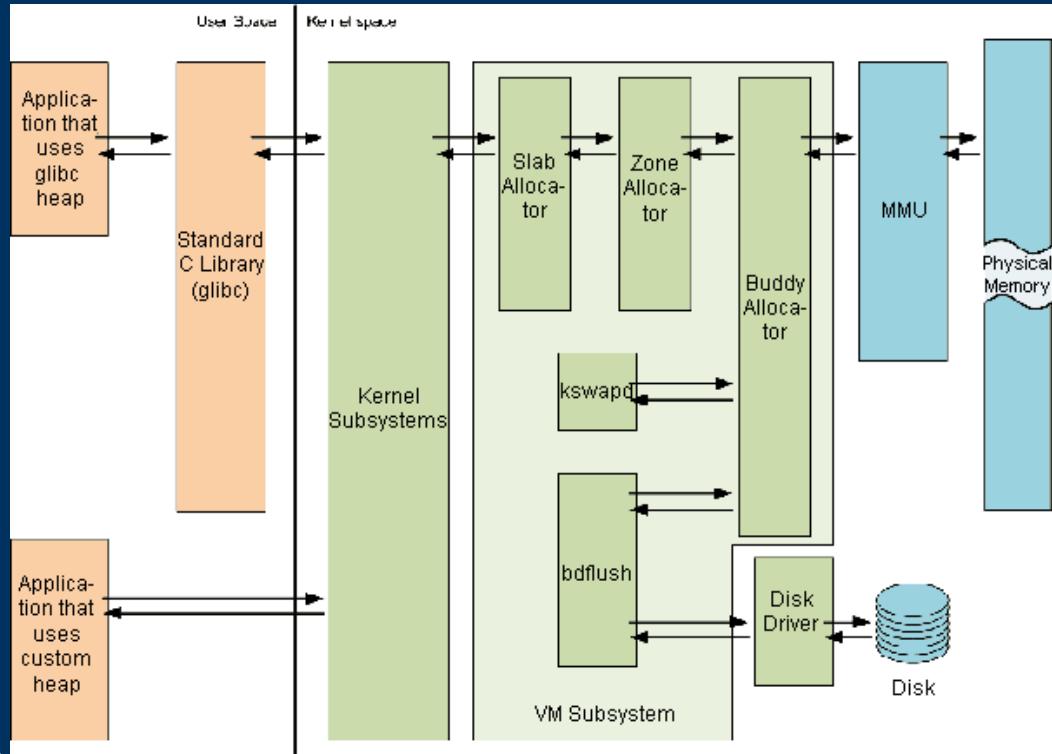
- Contrairement à la mémoire virtuelle, le noyau ne considère pas la mémoire physique comme linéaire. Découpage:
 - Noeuds
 - Zones:
 - DMA[32]: Mémoire dédiée pour accès direct (aujourd'hui à disposition des Devices)
 - HIGHMEM: Mémoire non persistante dans l'adressage du noyau (*obsolète*)
 - NORMAL: mémoire adressable



```
→ ~ cat /proc/buddyinfo
Node 0, zone      DMA      0      0      0      0      0      0      0      0      0      1      1      3
Node 0, zone    DMA32      4      2      1      1      4      4      2      3      4      3      3      864
Node 0, zone   Normal    970    412    252   186   112    41     15     14      3     19    594
```

Noyau & Mémoire

- Petites allocations:
Kmalloc/kfree (< 1 page)
- Grosses allocations:
vmalloc/vfree
- Allocation de pages:
alloc_pages/free_pages
- 3 allocateurs : SLAB, SLOB,
SLUB
- Slab allocator : méthode de
cache pour l'allocation de
petites structures
- Buddy allocator: rapide &
efficace mais beaucoup de
fragmentation



`GFP_USER` - Allocate memory on behalf of user. May sleep.

`GFP_KERNEL` - Allocate normal kernel ram. May sleep.

`GFP_ATOMIC` - Allocation will not sleep. May use emergency pools.

`GFP_HIGHUSER` - Allocate pages from high memory.

`GFP_NOIO` - Do not do any I/O at all while trying to get memory.

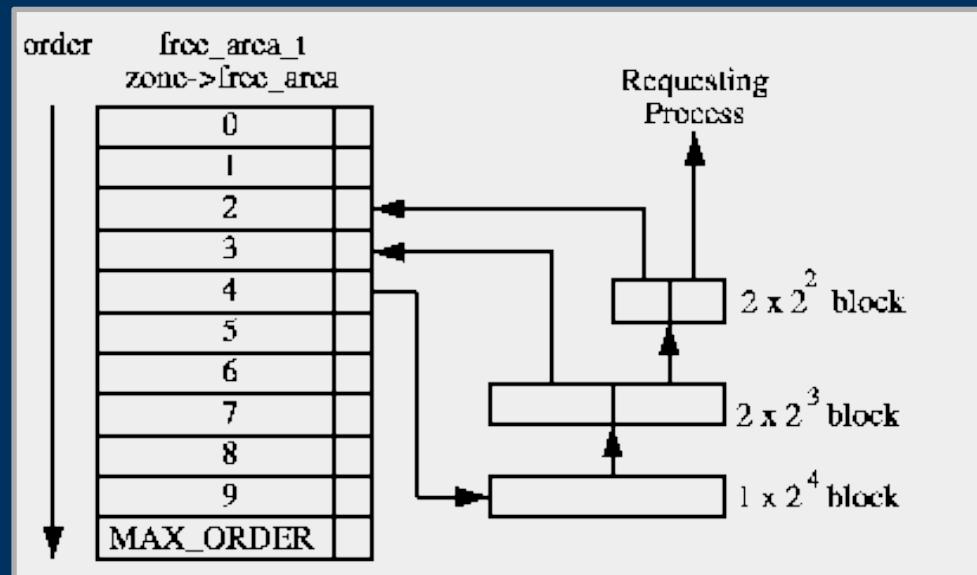
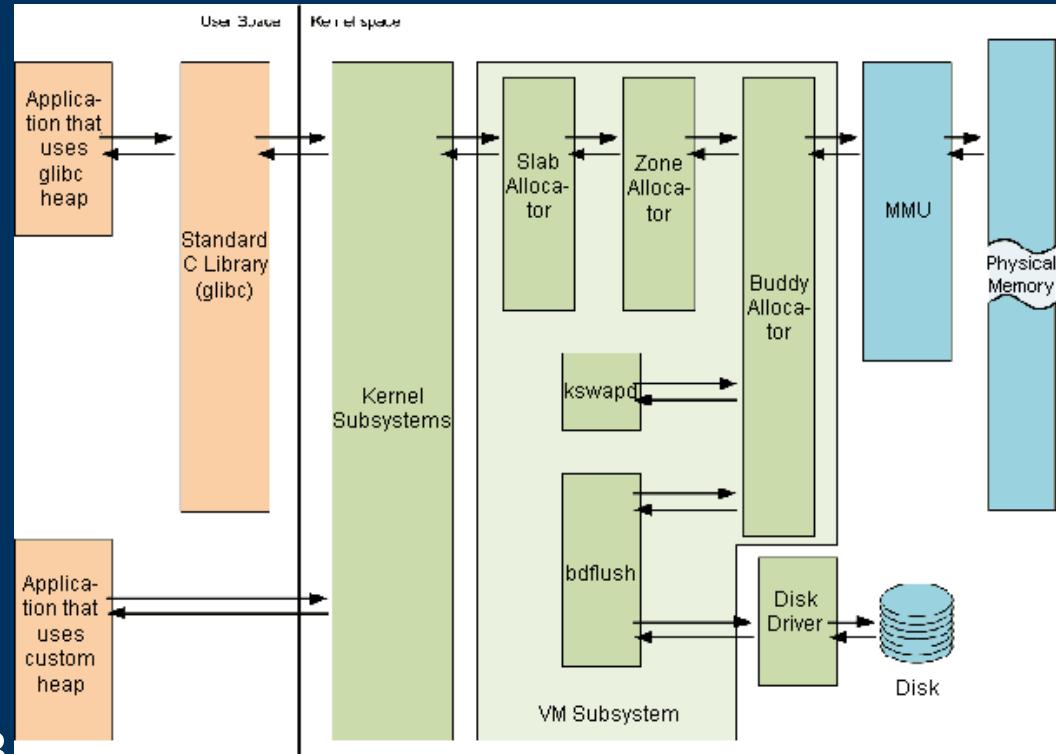
`GFP_NOFS` - Do not make any fs calls while trying to get memory.

`GFP_NOWAIT` - Allocation will not sleep.

`GFP_THISNODE` - Allocate node-local memory only.

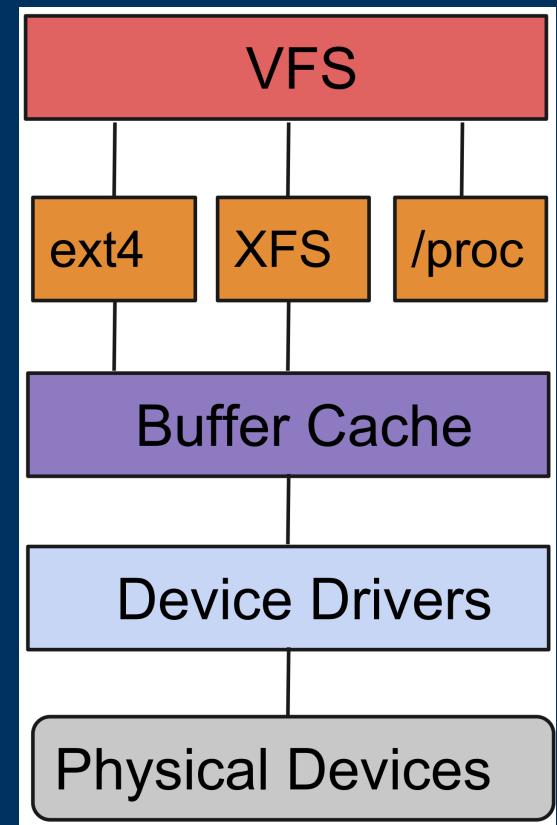
Noyau & Mémoire

- Petites allocations:
Kmalloc/kfree (< 1 page)
- Grosses allocations:
vmalloc/vfree
- Allocation de pages:
alloc_pages/free_pages
- 3 allocateurs : SLAB, SLOB, SLUB
- Slab allocator : méthode de cache pour l'allocation de petites structures
- Buddy allocator: rapide & efficace mais beaucoup de fragmentation



Composants noyau

- Memory Management (MM)
 - Allocation, gestion mémoire (virtuelle <-> physique)
 - Support du swap
 - Table des pages
 - TLB
- Network
 - Protocoles réseau
 - Couche d'abstraction entre applications et interfaces réseau.
- Device Drivers (DD)
 - Gestion des périphériques I/O



Device drivers

- Un « device driver » agit comme un opérateur permettant le contrôle d'un type donné de périphériques attachés à la machine
 - Ex: un disque, une clé USB, un lecteur CD, Terminal
- La liste des devices est disponible dans **/dev**
- Numérotation des devices
 - Majeur : valeur fixée par le noyau pour tous les devices utilisant le même driver
 - Mineur: Valeur arbitraire gérée par le driver pour différencier les différents devices sous sa responsabilité
- Création via « mknod » : `mknod /dev/mydev c 42 1`

```
Betelgeuse ~
└─ MAJOR=42; cat /usr/src/linux-headers-4.9.0-12-common/include/uapi/linux/major.h | grep "$MAJOR"
Betelgeuse ~
└─▶
```

« Device Drivers » : types

- « **Chardev** » : le flux est transmis, un octet à la fois, se comportant comme des pipes ou des ports séries. C'est le type d'interface convenant pour une communication série (une information à la fois)
 - Entrée clavier
 - Carte son
 - Carte vidéo
- « **blockdev** »: données cachées par bloc, permettant aux opérations une manipulation en groupe
 - Point de montage disque
 - Flux réseau

```
↳ ls -l /dev/ | egrep "^\b"  
brw-rw---- 1 root disk    8,   0 févr. 19 08:38 sda  
brw-rw---- 1 root disk    8,   1 févr. 19 08:38 sda1  
brw-rw---- 1 root disk    8,   2 févr. 19 08:38 sda2  
brw-rw---- 1 root disk    8,   3 févr. 19 08:38 sda3  
brw-rw---- 1 root disk    8,   4 févr. 19 08:38 sda4  
brw-rw---- 1 root disk    8,  16 févr. 19 08:38 sdb  
brw-rw---- 1 root disk    8,  17 févr. 19 08:38 sdb1  
brw-rw---- 1 root disk    8,  32 févr. 19 08:38 sdc  
brw-rw---- 1 root disk    8,  48 févr. 19 08:38 sdd  
brw-rw---- 1 root disk    8,  64 févr. 19 08:38 sde  
brw-rw---- 1 root disk    8,  80 févr. 19 08:38 sdf  
brw-rw----+ 1 root cdrom  11,   0 févr. 19 08:38 sr0
```

```
↳ ls -l /dev/ | egrep "^\c.*tty.*"  
crw-rw-rw- 1 root tty      5,   2 mars  9 17:07 ptmx  
crw-rw-rw- 1 root tty      5,   0 mars  9 16:24 tty  
crw--w---- 1 root tty     4,   0 févr. 19 08:38 tty0  
crw--w---- 1 root tty     4,   1 mars  5 11:04 tty1  
crw--w---- 1 root tty     4,  10 févr. 19 08:38 tty10  
crw--w---- 1 root tty     4,  11 févr. 19 08:38 tty11  
crw--w---- 1 root tty     4,  12 févr. 19 08:38 tty12  
crw--w---- 1 root tty     4,  13 févr. 19 08:38 tty13  
crw--w---- 1 root tty     4,  14 févr. 19 08:38 tty14  
crw--w---- 1 root tty     4,  15 févr. 19 08:38 tty15  
crw--w---- 1 root tty     4,  16 févr. 19 08:38 tty16  
crw--w---- 1 root tty     4,  17 févr. 19 08:38 tty17  
crw--w---- 1 root tty     4,  18 févr. 19 08:38 tty18  
crw--w---- 1 root tty     4,  19 févr. 19 08:38 tty19  
crw--w---- 1 root tty     4,   2 févr. 19 08:38 tty2  
crw--w---- 1 root tty     4,  20 févr. 19 08:38 tty20
```

Ordonnancement de processus

- Principe de préemption:
 - Méthode permettant de suspendre l'exécution d'un programme afin d'invoquer un ordonnanceur de tâche pour déterminer qui doit s'exécuter ensuite
 - S'oppose au principe coopératif, où le système « attend » gentiment que le processus veuille bien passer la main
- Est réalisé au moyen d'interruption, **à tout moment** (kernel ou userspace)
- Cette approche permet une distribution plus juste des ressources en fonction d'un ensemble de critères
 - Besoin de réactivité immédiate
 - Attente d'I/O (=I/O bound)
 - Besoin en calcul (=CPU bound)
 - Temps passé sur CPU vs temps passé à attendre le CPU
- La majorité des systèmes industriels reposent aujourd'hui sur un noyau préemptif.

Ordonnancement de processus

- Pour s'adapter au mieux à l'évolution de la charge, la politique d'ordonnancement (scheduling) doit être dynamique
- Le temps d'usage d'un CPU est divisé en *slices*
- L'idée est d'avoir une répartition **juste** des slices sur les processus exécutables
- Sous Linux, avec un noyau préemptif:
 - Une priorité « statique » est attribué à chaque processus lorsqu'il démarre en fonction d'un ensemble de critères (utilisateur, programme)
 - Une priorité « dynamique », qui change au cours de la vie du processus est appliquée périodiquement par le scheduler en fonction de son activité (selon l'ordonnancement choisi)

Ordonnancement de processus

man shed

- **Politiques traditionnelles:**

- **SCHED_NORMAL | SCHED_OTHER:** Politique par défaut, « Completely Fair Scheduler », chaque tache dispose d'une quantité égale de temps CPU.
 - Famine mis en défaut par un incrément à chaque fois qu'un processus est prêt mais non exécuté
- **SCHED_IDLE** (2.6): Jobs à basse priorité, lorsque le CPU « a le temps » (nice() inefficace)
- **SCHED_BATCH** (2.6): Similaire à SCHED_OTHER, mais considère les threads CPU-bound (non-interactifs) et réalise les ajustements nécessaires pour ne pas trop les pénaliser
- Attribut modifiable via:
 - `sched_setaffinity()`
 - `nice()`

chrt -p \$PID

- **Politiques temps-réel:**

- **SCHED_FIFO**: premier arrivé, premier servi, sans prendre en compte une priorité dynamique.
- **SCHED_RR**: chaque processus ne peut garder l'accès au CPU que pour N quantum, après quoi il est remis en fin de file (à priorité constante)
- **SCHED_DEADLINE** (3.14): Le processus qui doit compléter avant la prochaine période passe en premier

Compiler le noyau Linux

- Un seul site web: <https://kernel.org>
- Récupération de l'archive **et de sa signature.**
- Vérification de l'archive pour corruption !
 - gpg2 --keyserver https://pgp.mit.edu/ --search-keys gregkh@kernel.org
 - gpg2 --verify <linux-archive>.tar.sig
- Décompression de l'archive
 - tar xf <linux-archive>.tar

```
↳ tree -d -L 1
.
├── arch
├── block
├── certs
├── crypto
├── Documentation
├── drivers
├── fs
├── include
├── init
├── ipc
├── kernel
├── lib
└── LICENSES
  └── mm
  ├── net
  ├── samples
  ├── scripts
  ├── security
  ├── sound
  ├── tools
  └── usr
    └── virt
22 directories
```

22 directories

```
gpg: assuming signed data in 'linux-5.5.7.tar'
gpg: Signature made ven. 28 févr. 2020 17:24:41 CET
gpg:           using RSA key 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Good signature from "Greg Kroah-Hartman <gregkh@linuxfoundation.org>" [unknown]
gpg:           aka "Greg Kroah-Hartman <gregkh@kernel.org>" [unknown]
gpg:           aka "Greg Kroah-Hartman (Linux kernel stable release signing key)
<greg@kroah.com>" [unknown]
```

Compiler le noyau Linux

- Un seul fichier de configuration: **.config**
- Généré par « make <...>config » (ex: make menuconfig)
- Possibilité de copier celui du noyau courant: **/boot/config-***
- Compilation & installation du noyau
 - `make -j<X>`
 - `make modules_install`
 - `make install`
- Durée dépendante de votre configuration et de vos ressources
 - De quelques minutes à plusieurs heures
- C'est tout !

Importing 9349506041576182222

```
disk vdev '/dev/gptid/6ca110ab-3952-11e9-8cc1-0024ecf12fe7': best uberblock found for spa $import.
disk vdev '/dev/gptid/e5c1478c-90a8-11ea-a2bb-0024ecf12fe7': best uberblock found for spa mpool.
txg condensing: txg 9309306, msp[97] 0xfffffff80246ff9000, vdev id 1, spa mpool, smp size 54472, segments
txg condensing: txg 9309306, msp[97] 0xfffffff80261221c00, vdev id 0, spa mpool, smp size 49168, segments
txg condensing: txg 9309306, msp[51] 0xfffffff8024685c800, vdev id 5, spa mpool, smp size 33408, segments
txg condensing: txg 9309306, msp[52] 0xfffffff80246858c00, vdev id 5, spa mpool, smp size 33936, segments
txg condensing: txg 9309306, msp[122] 0xfffffff80246b43000, vdev id 3, spa mpool, smp size 44688, segments
txg condensing: txg 9309306, msp[121] 0xfffffff80246b46c00, vdev id 3, spa mpool, smp size 37472, segments
txg condensing: txg 9309306, msp[123] 0xfffffff80246d66400, vdev id 2, spa mpool, smp size 48360, segments
spa=mpool async request task=32panic: Solaris(panic): zfs: freeing free segment (offset=3435973836800 size=4096)
```

cpuid = 1

KDB: stack backtrace:

```
db_trace_self_wrapper() at db_trace_self_wrapper+0x2b/frame 0xfffffe2022481550
vpanic() at vpanic+0x177/frame 0xfffffe20224815b0
panic() at panic+0x43/frame 0xfffffe2022481610
vcmn_err() at vmn_err+0xcf/frame 0xfffffe2022481740
zfs_panic_recover() at zfs_panic_recover+0x5a/frame 0xfffffe20224817a0
range_tree_remove_impl() at range_tree_remove_impl+0x100/frame 0xfffffe2022481860
space_map_load_callback() at space_map_load_callback+0x73/frame 0xfffffe2022481890
space_map_iterate() at space_map_iterate+0x253/frame 0xfffffe2022481910
space_map_load() at space_map_load+0x91/frame 0xfffffe2022481960
metaslab_load() at metaslab_load+0x65/frame 0xfffffe2022481990
metaslab_preload() at metaslab_preload+0x89/frame 0xfffffe20224819c0
taskq_run() at taskq_run+0x10/frame 0xfffffe20224819e0
taskqueue_run_locked() at taskqueue_run_locked+0x154/frame 0xfffffe2022481a40
taskqueue_thread_loop() at taskqueue_thread_loop+0x98/frame 0xfffffe2022481a70
fork_exit() at fork_exit+0x83/frame 0xfffffe2022481ab0
fork_trampoline() at fork_trampoline+0xe/frame 0xfffffe2022481ab0
--- trap 0, rip = 0, rsp = 0, rbp = 0 ---
```

KDB: enter: panic

[thread pid 0 tid 101780]

Stopped at kdb_enter+0x3b: movq \$0,kdb_why

LKM: Loadable Kernel Module

- Chargement dynamique de modules compilés
- Extension des capacités du noyau
- Tout en gardant le cœur minimal
- Économie d'espace mémoire
- Force une pratique modulaire
- Facilité de débug (sans reboot machine obligatoire)
- Principaux usages:
 - Device drivers: extension de support de nouveaux périphériques
 - Filesystem drivers: extension du support de systèmes de fichiers
 - Syscalls : étend les capacités du noyau en introduisant de nouveaux sys calls
 - Network Drivers: Gestion dynamique de nouvelles technologies réseaux sans avoir à recompiler un noyau entier

Écrire un module noyau: les bases

- Avoir les headers du noyau cible
 - apt install linux-headers-<...>
 - yum install kernel-headers-<...>
 - ls /usr/src/linux<...>/
- Code de base -> init & fini
- Makefile
- Tests
 - modinfo mymodule.ko
 - insmod mymodule.ko
 - rmmod mymodule.ko
 - dmesg || /var/log/kern.log

```
obj-m+=mymodule.o
all:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("adamj");
MODULE_DESCRIPTION("A First module");
MODULE_VERSION("1.0");

static int __init mymod_init(void){
    printk(KERN_INFO "MyMod: Hello there !\n");
    return 0;
}

static void __exit mymod_exit(void){
    printk(KERN_INFO "MyMod: Goodbye !\n");
}
module_init(mymod_init);
module_exit(mymod_exit);
```

DKMS

- Compilation d'un module est dépendante du noyau cible.
- Comment survivre à une mise à jour noyau ?
 - => Recompile, à chaque mise à jour, la liste des modules
- DKMS (Dynamic Kernel Module Support), conçu par Dell en 2003
 - `dkms add -m mymodule -v 1.0.0 -c /dkms.conf`
 - `dkms remove -m mymodule -v 1.0.0 --all`
 - `dkms build -m mymodule -v 1.0.0 -k <kernel_version>`
 - `dkms install/uninstall`
 - `dkms status`
- Description du module dans `dkms.conf`, explicitant comment construire ce module noyau.
- 3 préfixes utilisées par DKMS (par défaut, voir `/etc/dkms/framework.conf`):
 - `/usr/src`: code sources des modules
 - `/lib/modules`: Modules compilés (souvent dans un répertoire « updates »)
 - `/var/lib/dkms`: artifacts de modules (logs, *.ko originaux...)

```
↳ sudo awk -F\' '/menuentry / {print $2}' /boot/grub2/grub.cfg
CentOS Linux (3.10.0-514.6.1.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-514.2.2.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-327.36.3.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-327.36.2.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-327.36.1.el7.x86_64) 7 (Core)
CentOS Linux (0-rescue-a6dd0bd0c8fc4cba95b0a0996a0e1078) 7 (Core)
```

DKMS: Example de configuration

Configuration d'un driver réseau Linux pour supporter des périphériques Wireless

```
PACKAGE_VERSION="1.0.4"
PACKAGE_NAME="ipw2200"
MAKE[0]="make -C ${kernel_source_dir} SUBDIRS=${dkms_tree}/${PACKAGE_NAME}/${PACKAGE_VERSION}/build modules HOSTAP_SRC=${kernel_source_dir}/3rdparty/hostap/"
CLEAN="make -C ${kernel_source_dir} SUBDIRS=${dkms_tree}/${PACKAGE_NAME}/${PACKAGE_VERSION}/build clean"
BUILT_MODULE_NAME[0]="$PACKAGE_NAME"
BUILT_MODULE_NAME[1]="ieee80211_crypt_ccmp"
BUILT_MODULE_NAME[2]="ieee80211_crypt"
BUILT_MODULE_NAME[3]="ieee80211_crypt_tkip"
BUILT_MODULE_NAME[4]="ieee80211_crypt_wep"
BUILT_MODULE_NAME[5]="ieee80211"
DEST_MODULE_LOCATION[0]="/kernel/drivers/net/wireless/ipw2200/"
DEST_MODULE_LOCATION[1]="/kernel/drivers/net/wireless/ipw2200/"
DEST_MODULE_LOCATION[2]="/kernel/drivers/net/wireless/ipw2200/"
DEST_MODULE_LOCATION[3]="/kernel/drivers/net/wireless/ipw2200/"
DEST_MODULE_LOCATION[4]="/kernel/drivers/net/wireless/ipw2200/"
DEST_MODULE_LOCATION[5]="/kernel/drivers/net/wireless/ipw2200/"
MODULES_CONF_ALIAS_TYPE[0]="eth"
REMAKE_INITRD="no"
AUTOINSTALL="yes"
```

Généralités sur les IPC System V

Les IPC System V

Apparus dans Unix en 1983 ils permettent des communication inter-inter-processus (Inter-Process Communications, IPC)

- Files de messages
- Segment de mémoire partagée
- Sémaphores

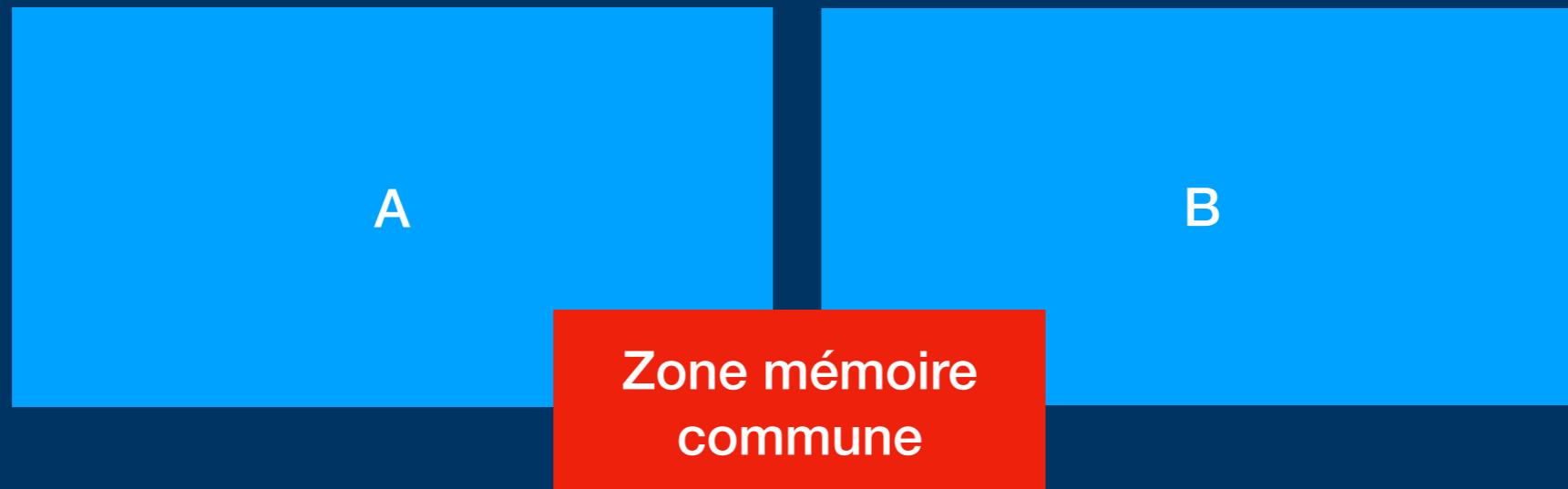
Le noyau est chargé de la gestion des ressources associées via des commandes

Files de Messages



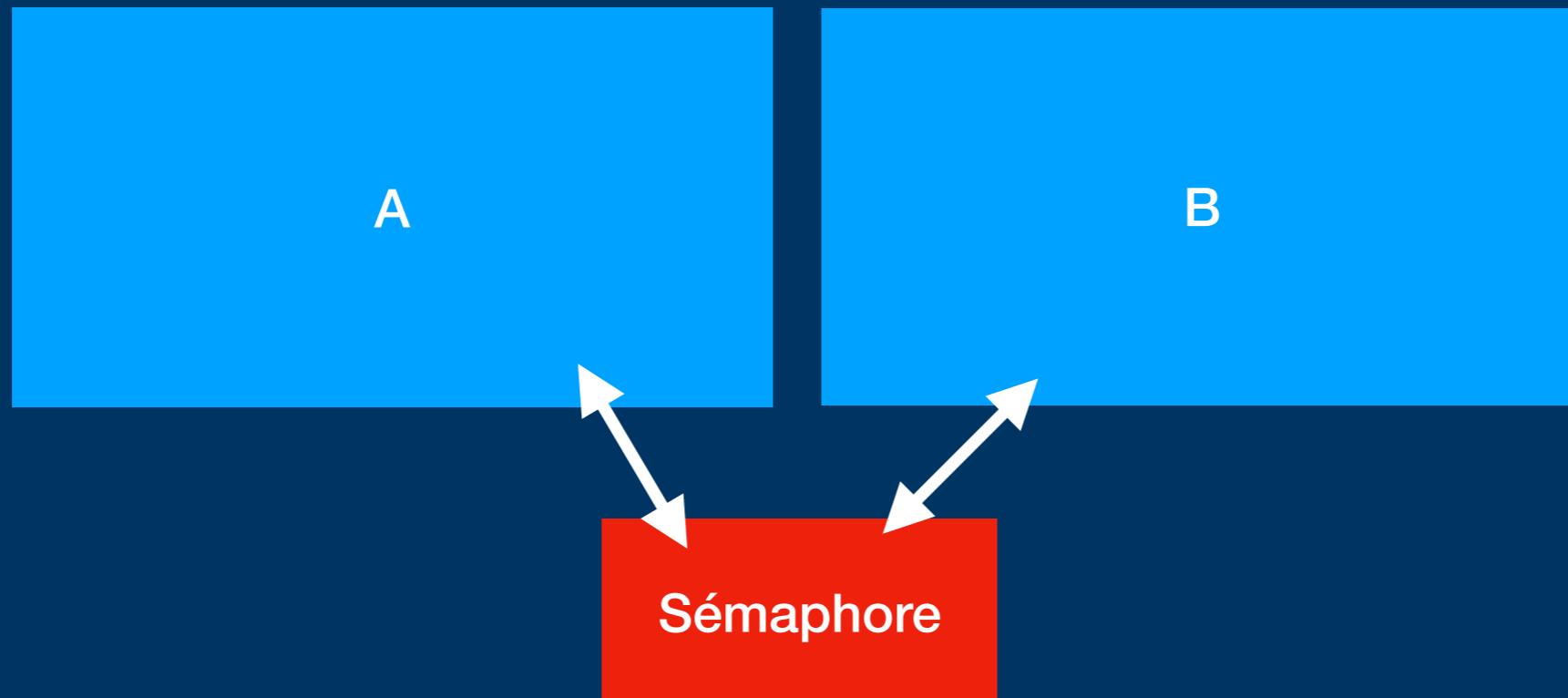
- ***ftok:* génération d'une clef IPC**
- ***msgget:* Récupère un identificateur de file de message**
- ***msgrecv:* Réception d'un message depuis une file**
- ***msgsend:* Envoi d'un message dans une file**
- ***msgctl:* Contrôle de la file de messages**

Segment de mémoire partagée



- ***ftok:*** génération d'une clef IPC
- ***shmget:*** Récupère un identificateur de segment shm
- ***shmat:*** Projection d'un segment SHM
- ***shmdt:*** Supression d'un segment SHM
- ***shmctl:*** Contrôle du segment SHM

Sémaphore IPC



- ***ftok:* génération d'une clef IPC**
- ***semget:* Récupère un identificateur de sémaphore**
- ***semop:* Fait une opération sur le sémaphore**
- ***semctl:* Contrôle du sémaphore**

Les IPC System V

```
$ ipcs
```

```
----- Files de messages -----
```

clef	msqid	propriétaire	perms	octets utilisés	messages
------	-------	--------------	-------	-----------------	----------

```
----- Segment de mémoire partagée -----
```

clef	shmid	propriétaire	perms	octets	nattch	états
0x00000000	42729472	jbbesnard	600	1048576	2	dest
0x00000000	39616513	jbbesnard	600	524288	2	dest

```
----- Tableaux de sémaphores -----
```

clef	semid	propriétaire	perms	nsems
------	-------	--------------	-------	-------

Les IPC System V

```
$ ipcrm -h
```

Utilisation :

```
ipcrm [options]
ipcrm shm|msg|sem <id> ...
```

Supprimer certaines ressources IPC.

Options :

-m, --shmem-id <ident.>	retirer le segment de mémoire partagée par ident.
-M, --shmem-key <clef>	retirer le segment de mémoire partagée par clef
-q, --queue-id <ident.>	retirer la file de messages par identifiant
-Q, --queue-key <clef>	retirer la file de messages par clef
-s, --semaphore-id <id.>	retirer le sémaphore par identifiant
-S, --semaphore-key <clef>	retirer le sémaphore par clef
-a, --all[=shm msg sem]	tout retirer (dans la catégorie indiquée)
-v, --verbose	expliquer les actions en cours
-h, --help	afficher cette aide et quitter
-V, --version	afficher les informations de version et quitter

Consultez ipcrm(1) pour obtenir des précisions complémentaires.

Les IPC System V

```
$ ipcmk -h
```

Utilisation :

```
ipcmk [options]
```

Créer diverses ressources IPC.

Options :

```
-M, --shmem <taille>    créer un segment de mémoire partagée de taille <taille>
```

```
-S, --semaphore <nsems>  créer un tableau de sémaphores à <nsems> éléments
```

```
-Q, --queue                créer une file de messages
```

```
-p, --mode <mode>         droits de la ressource (0644 par défaut)
```

```
-h, --help      afficher cette aide et quitter
```

```
-V, --version   afficher les informations de version et quitter
```

Consultez ipcmk(1) pour obtenir des précisions complémentaires.

Resources

Les resources IPC sont indépendante des processus

- Il est possible de laisser des scories si l'on ne fait pas attention
- Un processus peut se « ratacher » à un segment lors de son redémarrage par exemple
- Les processus partagent des segments avec un mécanisme de clef qui est un secret « a priori » pour la sécurité

Clefs pour les IPCs

System V

La Clef

Un IPC (de tout type) est partagé par une clef:

- C'est un entier qui doit être le même entre tous les processus partageant la resource;
- On peut la connaître a priori avec risque de conflit (un peut comme un port TCP);
- Une clef spéciale IPC_PRIVATE crée une file limité à un processus et l'ensemble de ses descendants;
- On peut la créer avec une fonction « ftok » qui repose sur un fichier et un nom de projet.

Ftok

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(const char *pathname, int proj_id);
```

DESCRIPTION

The `ftok()` function uses the identity of the file named by the given pathname (which must refer to an existing, accessible file) and the least significant 8 bits of `proj_id` (which must be nonzero) to generate a `key_t` type System V IPC key, suitable for use with `msgget(2)`, `semget(2)`, or `shmget(2)`.

The resulting value is the same for all pathnames that name the same file, when the same value of `proj_id` is used. The value returned should be different when the (simultaneously existing) files or the project IDs differ.

RETURN VALUE

On success, the generated `key_t` value is returned. On failure -1 is returned, with `errno` indicating the error as for the `stat(2)` system call.

Projet

Device

Inode

31

24 23

16 15

0

Création / Récupération de ressources

Une fois que l'on a une clef de type *key_t* on peut retrouver/créer une resource:

- File de message : *msgget*
- Segment de mémoire partagée: *shmget*
- Sémaphore: *semget*

Les Files de Messages

IPC SYSTEM V

Files de Messages pour une Communication entre Processus sur un Même Noeud.

Le message sera toujours de la forme:

```
Struct XXX {  
    long id; // Toujours > 0 !  
    ... DATA ....  
    // Taille max sans le long MSGMAX (8192 Octets)  
};
```

Lors de l'envoi et de la réception d'un message la taille et **TOUJOURS** sans le long qui définit le type de message. Cette même valeur (ici id) doit **TOUJOURS** être supérieure à 0.

En pratique on crée une struct statique sur la pile car l'allocation d'un objet avec piggybacking demande plus de code.

Créer Une File de Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

- Key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- msgflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Créer Une File de Messages

- Créer une file pour un processus et ses fils
 - ➡ file = msgget(IPC_PRIVATE, 0600);
- Créer une file pour accéder à une file potentiellement existante:
 - ➡ file = msgget(key , IPC_CREAT | 0600);
- Pour être sûr de créer une nouvelle file en lecture écriture pour soi et en lecture seule pour les autres utilisateurs:
 - ➡ file = msgget(key, IPC_CREAT | IPC_EXCL | 0622);
- Utiliser uniquement une file existante précédemment créée par un serveur:
 - ➡ file = msgget(key, 0);

Envoyer un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgflg: mode d'envoi du message
 - ➡ IPC_NOWAIT ne pas bloquer si la file est pleine (renvoie EAGAIN dans errno)
 - ➡ 0 en général

Recevoir un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

ssize_t msgrcv(int msqid,
                void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
 - ➡ 0 : prochain message de la file
 - ➡ 0 < TYP prochain message avec l'ID donné
 - ➡ TYP < 0 prochain message avec un ID inférieur ou égal à TYP, utilisé pour gérer des priorités de messages
- msgflg: mode de réception du message:
 - ➡ IPC_NOWAIT ne pas bloquer si pas de message du TYP demandé (renvoie ENOMSG dans errno)
 - ➡ MSG_EXCEPT renvoie un message d'un TYP différent de celui donné (seulement pour TYP > 0)
 - ➡ MSG_NO_ERROR permettre au message d'être tronqués à la réception (à la différence du comportement de base)

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

L'appel reste bloqué indéfiniment si un message 99 n'est jamais posté.

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

L'appel renvoie -1 et met errno à ENOMSG

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

Recevoir un Message



```
msgrecv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

Contrôler une File

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : ID de la file à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupères les informations sur la file dans la *struct msqid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct msqid_ds*
 - ➡ **IPC_RMID supprime la file toute les opérations courantes ou future échouent (avec la possibilité non gérée qu'une nouvelle file soit créée avec la même clef). La synchronisation et à la charge du programmeur.**
 - ➡ ... il existe d'autre flags voir man

PENSEZ à SUPPRIMER VOS FILES !!!

Suite ...

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <sys/wait.h>

#include <sys/time.h>

double get_time(){
    struct timeval val;
    gettimeofday(&val, NULL);
    return (double)val.tv_sec + 1e-6 * val.tv_usec;
}

#define SIZE 16

struct msg_t{
    long type;
    int data[SIZE];
};

#define NUM_MSG 65536

int main( int argc, char ** argv ){
    int file = msgget(IPC_PRIVATE, IPC_CREAT | 0600);

    if( file < 0 ){
        perror("msgget");
        return 1;
    }

    int i;
    struct msg_t m;
    m.type = 1;

    int pid = fork();

    if( pid == 0 )
    {
        int stop = 0;

        while(!stop)
        {
            msgrcv(file, &m, SIZE*sizeof(int), 0, 0);
            /* Notify end */
            if( m.data[0] == 0 )
                stop = 1;
            m.type = 1;
            msgsnd(file, &m, SIZE*sizeof(int), 0);
        }
    }
}
```

```
else
{
    double total_time = 0.0;

    for( i = 2 ; i <= NUM_MSG ; i++ )
    {
        m.data[0] = i;
        m.type = i;

        double start = get_time();
        int ret = msgsnd(file, &m, SIZE*sizeof(int), 0);

        if( ret < 0 )
        {
            perror("msgsend");
            return 1;
        }

        double end = get_time();
        total_time += end - start;

        msgrcv(file, &m, SIZE*sizeof(int), 1, 0);

        m.data[0] = 0;
        msgsnd(file, &m, SIZE*sizeof(int), 0);

        wait( NULL );
    }

    msgctl( file, IPC_RMID, NULL);

    fprintf(stderr, "Pingpong takes %g usec Bandwidth is %g MB/s\n",
            total_time/NUM_MSG*1e6,
            (double)(SIZE*NUM_MSG*sizeof(int))/(
                (total_time*1024.0*1024.0)));
}

return 0;
}
```

Les Segments SHM

IPC SYSTEM V

Partager une Zone Mémoire entre Deux Processus

SHM = SHared Memory

Les avantages:

- Communication directe sans recopie mémoire;
- Pas de passage par l'espace noyau à la différence des files messages (context switch et recopie);
- Latences plus faible (même mémoire)

Les inconvénients:

- Il faut manuellement synchroniser les communications (lock ou sémaphore)
 - ➡ Comprenez qu'il est possible de mettre un lock dans cette zone mémoire, un spin lock directement, un mutex avec le bon attribut (`PTHREAD_PROCESS_SHARED`). Ou bien un sémaphore des IPC.
- La structuration des données est à la charge du programme

Créer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- Size: taille du segment SHM en octet (arrondie à la page supérieure).
Donc mapper un int est un gros gâchis de mémoire (une page fait 4 KB).
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Projeter le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid : le descripteur du segment SHM
- shmaddr: une adresse où mapper le segment, alignée sur une frontière de page. **NULL si indifférent.**
- shmflg: options relative à la projection du segment
 - ➡ SHM_RND arrondis l'adresse passée par *shmaddr* à une frontière de page
 - ➡ SHM_RDONLY partager le segment en lecture seule
 - ➡ ... il existe d'autre flags voir man

Retirer le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

- `shmaddr`: adresse renvoyée par `shmat`

Tous les processus doivent retirer le segment de leur mémoire autrement la suppression avec `shmctl` n'est pas effective. Si un processus se termine il détache la mémoire mais cela ne marque pas le segment pour suppression.

Supprimer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmid : ID du segment à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupères les informations sur la file dans la *struct shmid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct shmid_ds*
 - ➡ **IPC_RMID marque le segment SHM pour destruction cela ne se produira que quand tout les processus l'ayant projeté se seront détachés**
 - ➡ ... il existe d'autre flags voir man particulièrement IPC_INFO et SHM_INFO utiles pour connaître les limites sur le système cible

PENSEZ à SUPPRIMER VOS Segments !!!

Totallement arbitraire

Serveur

```
#include <sys/ipc.h>
#include <sys/shm.h>

#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int),
                     IPC_CREAT | IPC_EXCL | 0600 );

    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }

    int *val = (int*) shmat(shm, NULL, 0);

    if( !val )
    {
        perror("shmat");
        return 1;
    }

    /* valeur de départ */
    val[0] = 1;
    val[1] = 0;

    while(val[0])
    {
        sleep(1);
        val[1]++;
    }

    /* Unmap segment */
    shmdt(val);
    /* Server marks the segment for deletion */
    shmctl(shm, IPC_RMID, NULL);

    return 0;
}
```

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
#include <unistd.h>
#include <stdio.h>
```

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int), 0 );
```

```
    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }
```

```
    int *val = (int*) shmat(shm, NULL, 0);
```

```
    if( !val )
    {
        perror("shmat");
        return 1;
    }
```

```
    /* valeur de départ */
    int last_val = -1;
    while(1)
    {
        if( val[1] != last_val ){
            printf("Val is %d max is 60\n", val[1]);
            last_val = val[1];
        }
    }
```

```
    /* Stop condition */
    if( 60 <= val[1] )
    {
        val[0] = 0;
        break;
    }
    else
    {
        usleep(100);
    }
}
```

```
    /* Unmap segment */
    shmdt(val);
```

```
    return 0;
}
```

Client

```
$ ./serveur &
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
```

clef	shmid	propriétaire	perms	octets	nattch	états
0x00004e1f	42827778	jbbesnard	600	8	1	

```
$ ./client
```

```
Val is 0 max is 60
```

```
Val is 1 max is 60
```

```
(...)
```

```
Val is 7 max is 60
```

```
Val is 8 max is 60
```

```
Val is 60 max is 60
```

```
[2]+ Fini
```

```
./server
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
```

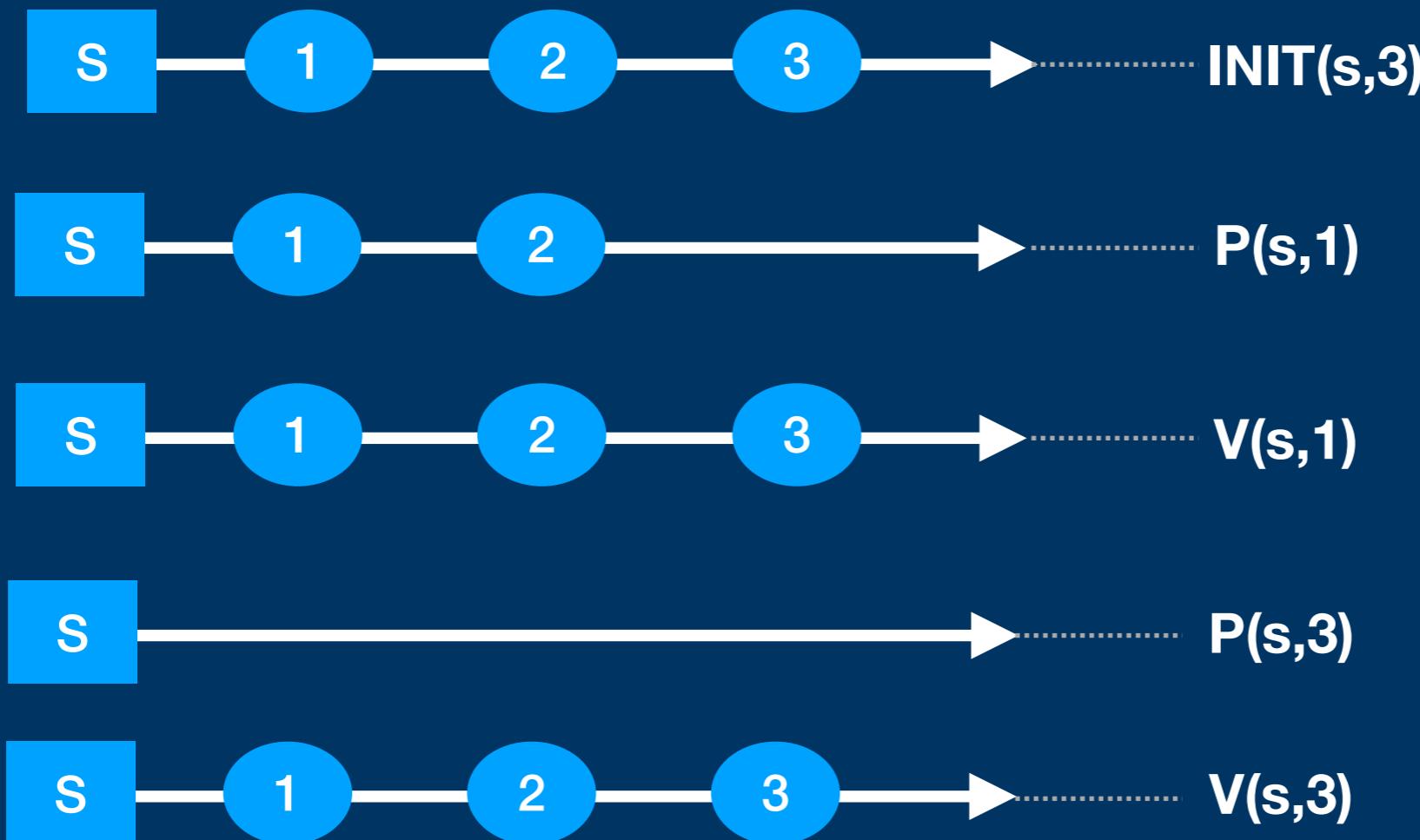
clef	shmid	propriétaire	perms	octets	nattch	états
------	-------	--------------	-------	--------	--------	-------

Les Sémaphores IPC SYSTEM V

Notion de Sémaphore

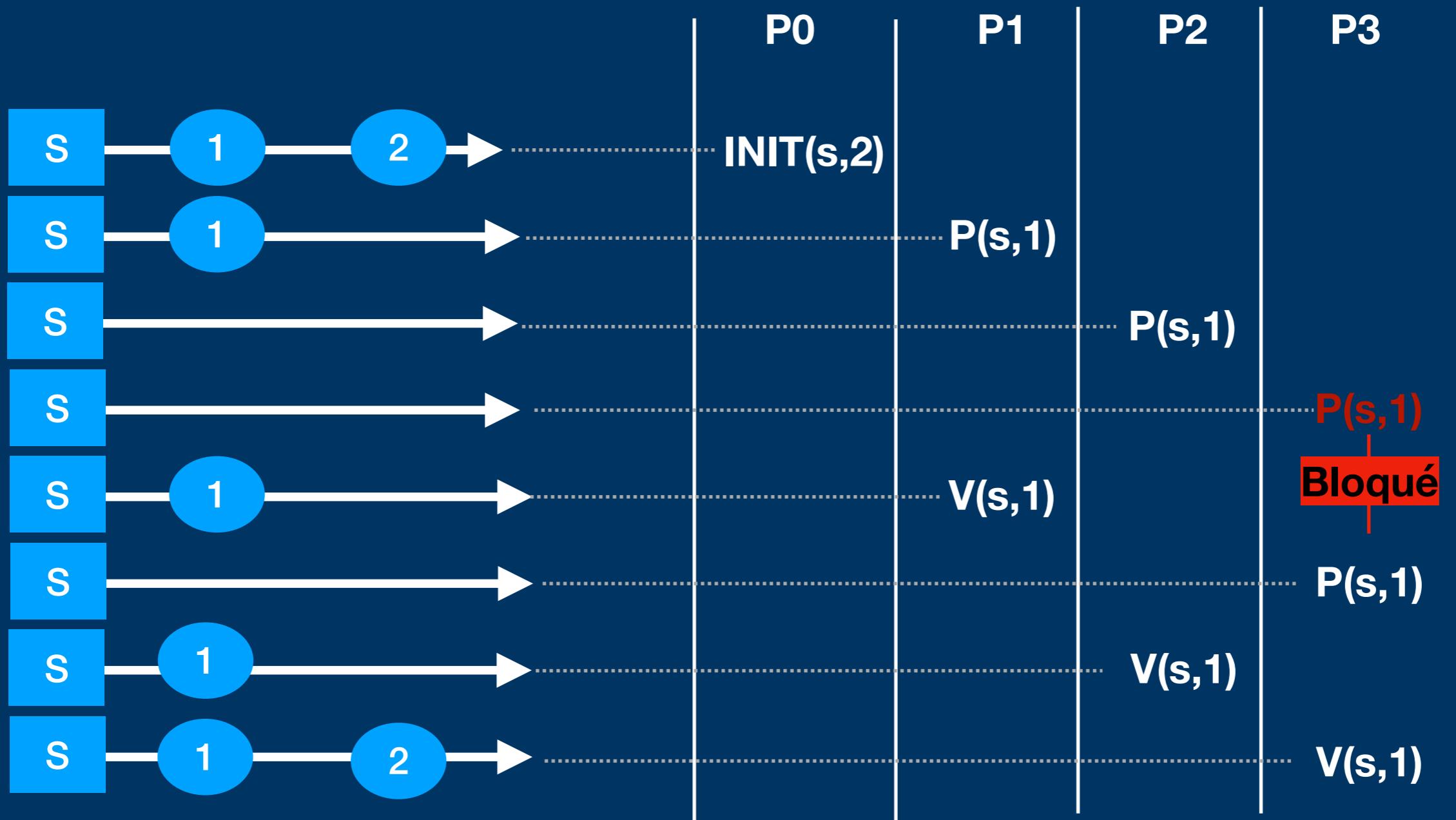
Un sémaphore est un élément de synchronisation qui permet de partager un ensemble de ressources. Il existe des sémaphores pour la programmation en mémoire partagée. Ici les sémaphore System V sont inter-processus. On définit classiquement deux opérations:

- $P(s,n)$: « Tester » (de l'allemand *passering* du fait de Dijkstra)
- $V(s,n)$: « Relâcher » (de l'allemand *vrijgave* du fait de Dijkstra)



Synchronisation avec des Sémaphores

- $P(s,n)$: « Tester »
- $V(s,n)$: « Relâcher »



Créer des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- nsem: nombre de sémaphores à créer
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Opération sur des Séma phores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
```

- *semid* : identifiant du sémaphore
- *sembuf*: opération(s) à effectuer via un tableau

```
struct sembuf {
    unsigned short sem_num; /* semaphore number */
    short      sem_op;   /* semaphore operation */
    short      sem_flg;  /* operation flags */
};
```

- ➔ *sem_num*: numéro du sémaphore
- ➔ *sem_op*: opération à effectuer
 - *sem_op > 0* : V(s)
 - *sem_op < 0* : P(s)
 - *sem_op == 0* : attente de la valeur 0 -> utile pour synchroniser les processus
- ➔ Drapeau à utiliser :
 - *IPC_NOWAIT*: non-bloquant et renvoie EAGAIN si l'opération avait dû bloquer
 - *IPC_UNDO*: demande au noyau d'annuler l'opération si le processus se termine en cas d'arrêt intempestif
- *nsops*: nombre d'opérations à effectuer (elles sont faites de manière atomique)

Contrôle du SémaPhore

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

- *semid* : identifiant du sémaPhore
- *semnum*: identifiant du sémaPhore
- *cmd*: commande à appliquer au sémaPhore

Non défini dans les headers !

```
union semun {
    int          val; /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* Array for GETALL, SETALL */
    struct seminfo *_buf; /* Buffer for IPC_INFO
                           (Linux-specific) */
};
```

➡ IPC_STAT récupères les informations sur le sémaPhore

➡ SETALL définit la valeur du sémaPhore (prend un tableau de unsigned short int en paramètre additionnel)

➡ **IPC_RMID** supprime immédiatement le sémaPhore et débloque les processus en attente

➡ ... il existe **BEAUCOUP** d'autre flags voir man

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <sys/wait.h>

int main( int argc, char ** argv ){
    int sem = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);

    if( sem < 0 ){
        perror("msgget");
        return 1;
    }

    unsigned short val = 1;
    if( semctl(sem, 0, SETALL, &val) < 0){
        perror("semctl");
        return 1;
    }

    int pid = fork();
    struct sembuf p;

    p.sem_num = 0;
    p.sem_op = -1;
    p.sem_flg = SEM_UNDO;

    struct sembuf v;

    v.sem_num = 0;
    v.sem_op = 1;
    v.sem_flg = SEM_UNDO;

    if( pid == 0 ) { /* Child */
        while(1){
            if( semop(sem, &p, 1) < 0 ){
                printf("Child: SEM deleted\n");
                return 0;
            }

            printf("CHILD holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
        }
    }
}

```

Suite ...

```

else
{
    /* Parent */
    int i = 0;
    while(i < 5)
    {
        semop(sem, &p, 1);

        printf("PARENT holding the sem\n");
        sleep(1);
        semop(sem, &v, 1);
        i++;
    }

    /* Parent delete the sem and unlock the child */
    semctl(sem, 0, IPC_RMID);

    wait( NULL );
}

return 0;
}

```

Sortie du Programme

```
$ ./a.out
PARENT holding the sem
CHILD holding the sem
Child: SEM deleted
```

IPCs POSIX

IPC POSIX

Le standard POSIX plus récent propose également les même mécanismes:

- Files de messages
 - Segment de mémoire partagée
 - Sémaphores
-
- Ils sont plus fiables en termes de libération et de partage de la ressource;
 - Enfin l'ensemble de l'interface est thread-safe;
 - Les objets sont demandés par nom et non avec une valeur donnée;
 - Ces appels sont un peu moins portables et sont à attendre plus sur des LINUX que des UNIX au sens large;
 - On les décrit généralement comme plus simples à utiliser.

Files de Message POSIX

À vous de jouer avec le man:

- mq_open
- mq_close
- mq_send
- mq_receive
- mq_unlink

Portez l'exemple SYS-V

Que pensez-vous de *mq_notify* ?

Segment SHM POSIX

À vous de jouer avec le man:

- `shm_open`
- `shm_unlink`
- `mmap`

Portez l'exemple SYS-V

Sémaphore IPC POSIX

Aussi « sémaphore nommé » à ne pas confondre avec les sémaphore « anonymes » de la NPTL (libpthread) qui sont dans le même header.

Rappel (ou pas) pour un sémaphore «anonyme »:

- `sem_init`
- `sem_destroy`
- **`sem_post`**
- **`sem_wait`**

À vous de jouer avec le man pour un sémaphore nommé:

- `sem_open`
- `sem_close`
- **`sem_post`**
- **`sem_wait`**
- `sem_unlink`

Portez l'exemple SYS-V

Peut-on l'implémenter avec un sémaphore anonyme et pourquoi ?

Préférez vous POSIX
ou SYS-V ?