

Généralités

Bases du Shell

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

Jean-Baptiste Besnard
<jean-baptiste.besnard@paratools.com>



Julien Adam
<julien.adam@paratools.com>

Organisation

- Chaque session est découpée en deux parties. Un cours théorique le matin et une mise en pratique l'après-midi (TD) portant sur les connaissances vues le matin.
- Des QCMs sur les bases **importantes** au fil des semaines et portant sur un cours précédent. Le QCM aura toujours lieu durant les 15 premières minutes de cours.
- Un projet sur la durée du semestre, date de rendu au 10/03/2020
- Un Examen final (1h30 - 2h) le 17/03/2020
- Une présentation + démo des projets le même jour
- L'accent est mis sur la **pratique**. C'est pourquoi le projet est une part importante de la note finale. Nous ne sommes pas là pour vous piéger, à vous de nous surprendre !

Programme du Semestre

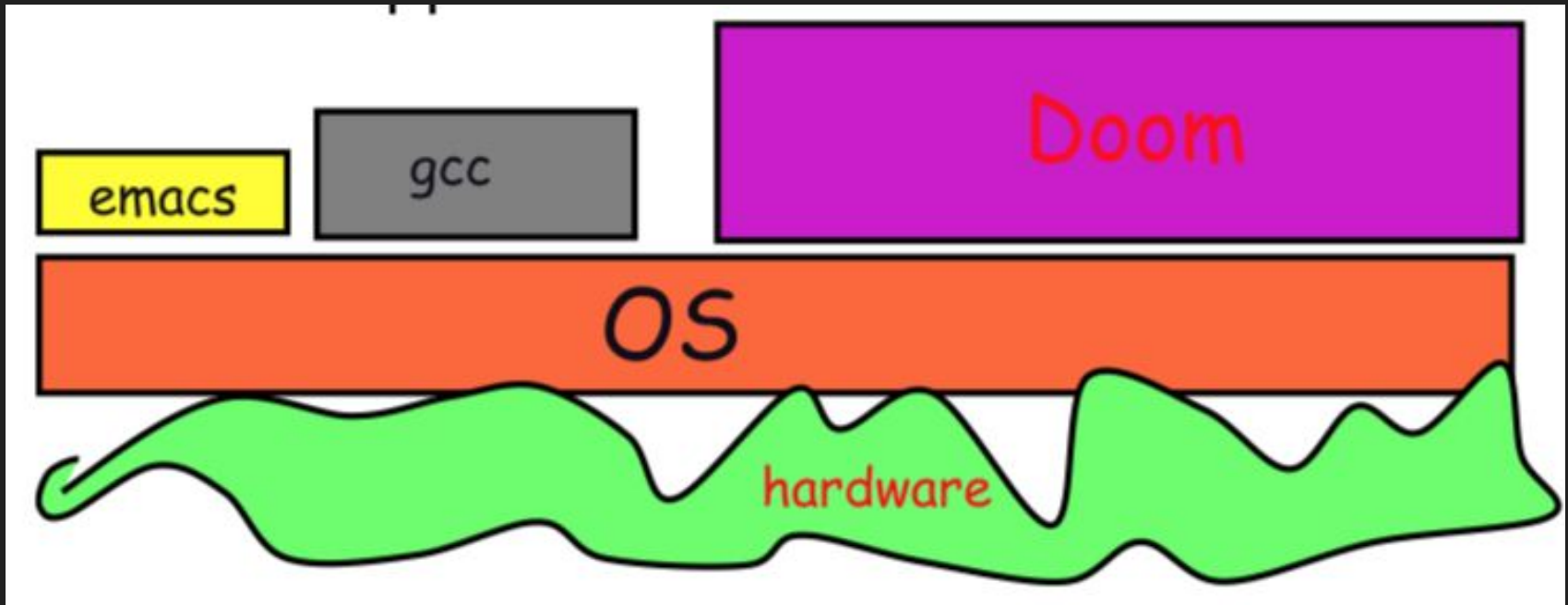
- ◆ 1 - Généralités sur les OS et Utilisation de base
- ◆ 2 - Processus, Threads & Synchronisation
- ◆ 3 - Compilation et représentation Binaire
- ◆ 4 - Architecture Mémoire d'un processus
- ◆ 5 - Programmation réseau et entrées/sorties avancées
- ◆ 6 - Virtualisation et Conteneurs
- ◆ 7 - Noyau Linux et bases d'ordonnancement
- ◆ Examen + Démo de projets

Type d'Examen	Coefficient
QCMs	15 %
PROJET	45 %
EXAMEN	45 %

Programme du Semestre

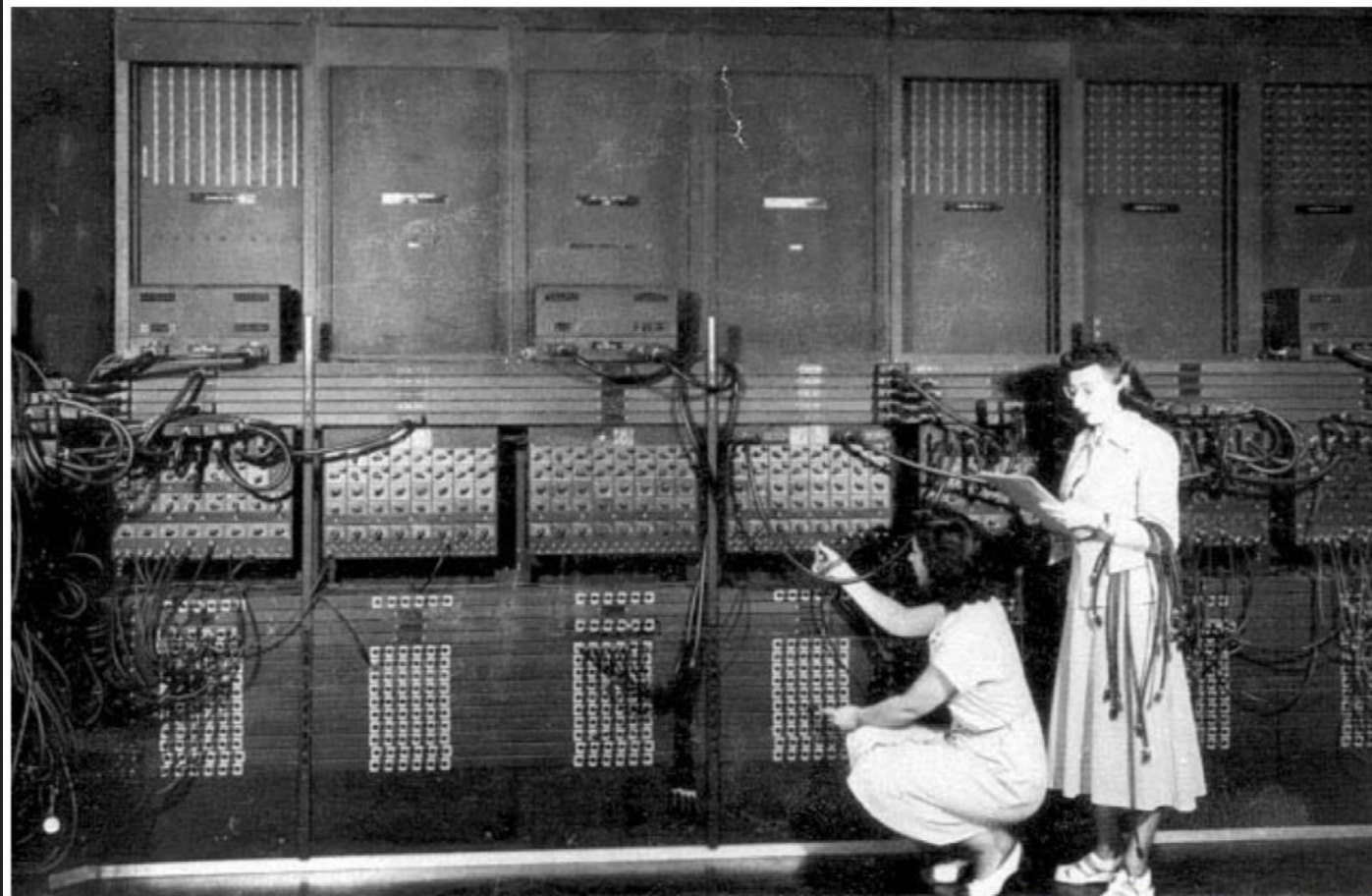
- ◆ 1 - Généralités sur les OS et Utilisation de base
- ◆ 2 - Processus, Threads & Synchronisation
- ◆ 3 - Compilation et représentation Binaire
- ◆ 4 - Architecture Mémoire d'un processus
- ◆ 5 - Programmation réseau et entrées/sorties avancées
- ◆ 6 - Virtualisation et Conteneurs
- ◆ 7 - Noyau Linux et bases d'ordonnancement
- ◆ Examen + Démo de projets

Systeme d'Exploitation



- Interface entre le matériel et les logiciels:
 - Il rend les programmes portables (standards);
 - Abstrait le matériel (même code sous Android et sur x86);
 - Partage les ressources entre plusieurs applications et utilisateurs;
 - Assure la sécurité et la résilience de la machine;

Les Ancêtres



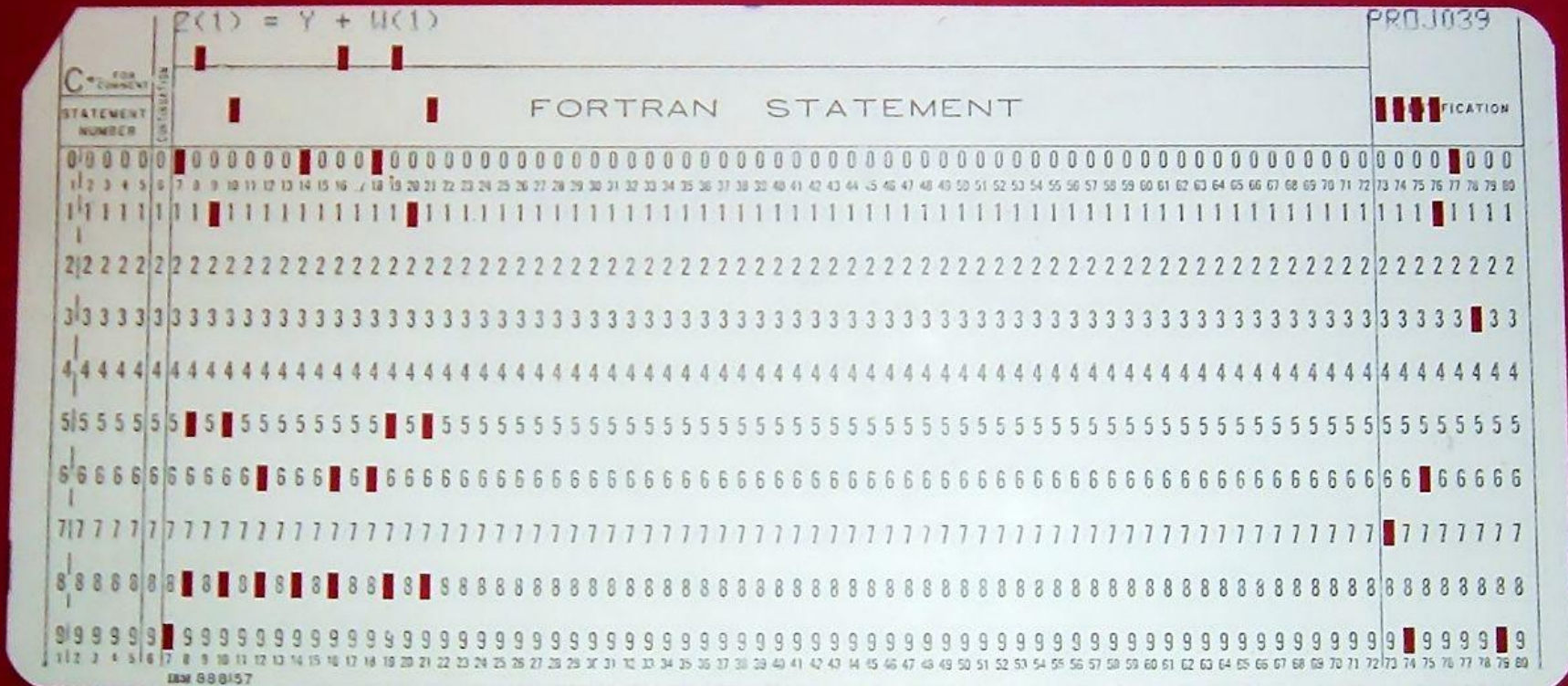
Années 40-50: Calculateur ENIAC -> Pas d'OS la machine est câblée pour un programme donné.

Les Ancêtres 2/3



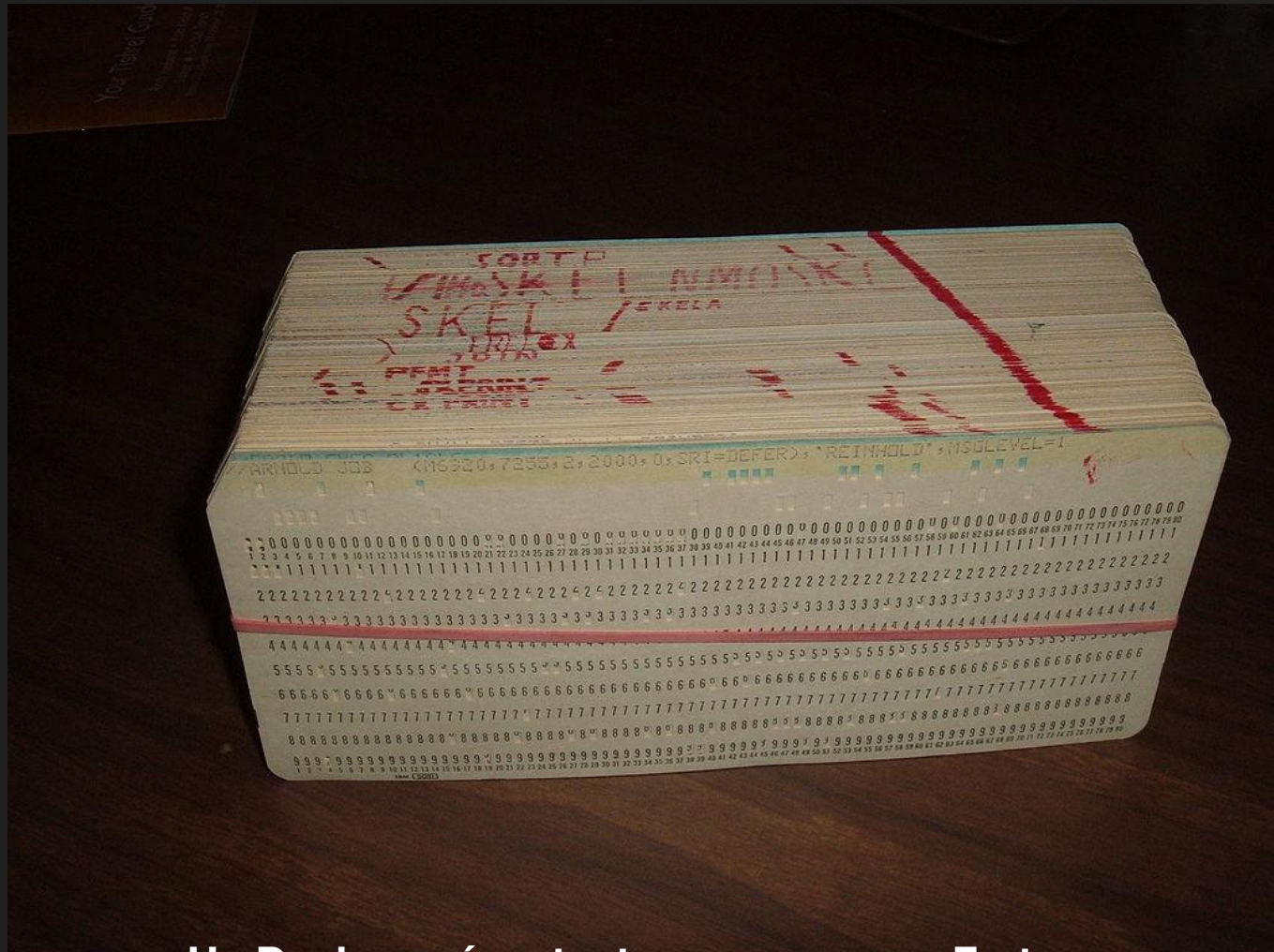
Années 55-65: IBM 701, on charge des cartes perforées. On a donc un traitement par lot et on abstrait la reconfiguration. On charge manuellement des cartes perforées.

Les Ancêtres 2/3



Une carte perforée « Fortran » (punch card) admirez la limite des 72 caractères
(source wikipedia)

Les Ancêtres 2/3



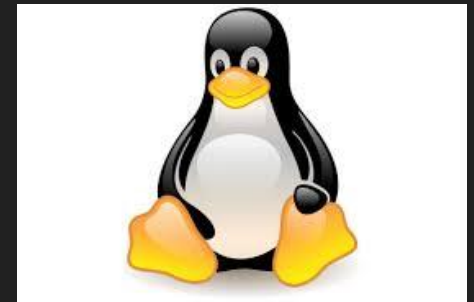
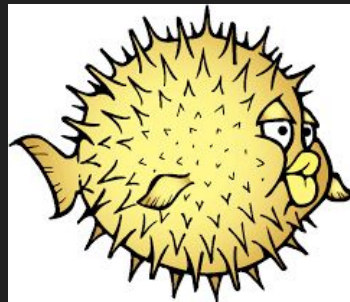
Un Deck représentant un programme Fortran
complet
(source wikipedia)

Les Ancêtres 3/3

Dans les années 50-70, premières bribes de système d'exploitation avec le moniteur résident qui est capable de charger des programmes automatiquement depuis un support de stockage tel qu'une bande magnétique.

Un programme dont le rôle est de charger puis gérer d'autre programme afin de maximiser l'utilisation de la machine ... le début d'un Système d'Exploitation (ES) ou Operating System (OS) en Anglais.

Les Familles d'OS



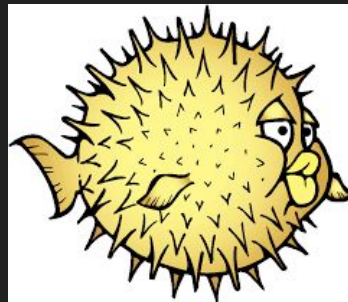
Les Familles d'OS



MAC OS/X



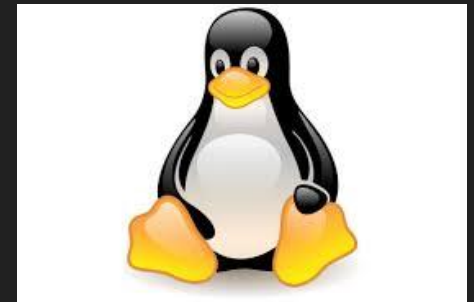
FreeBSD



OpenBSD

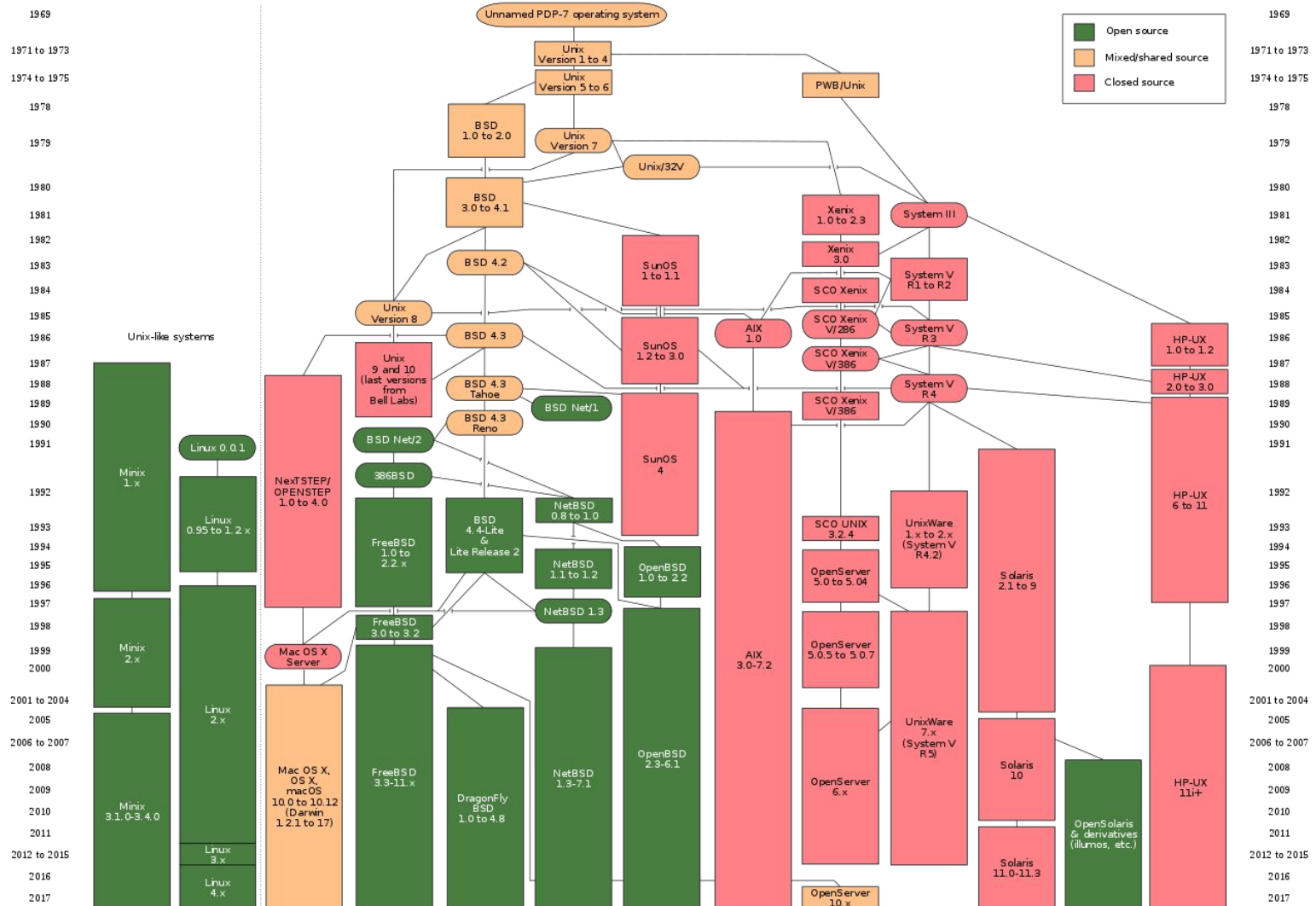


Solaris



GNU/LINUX

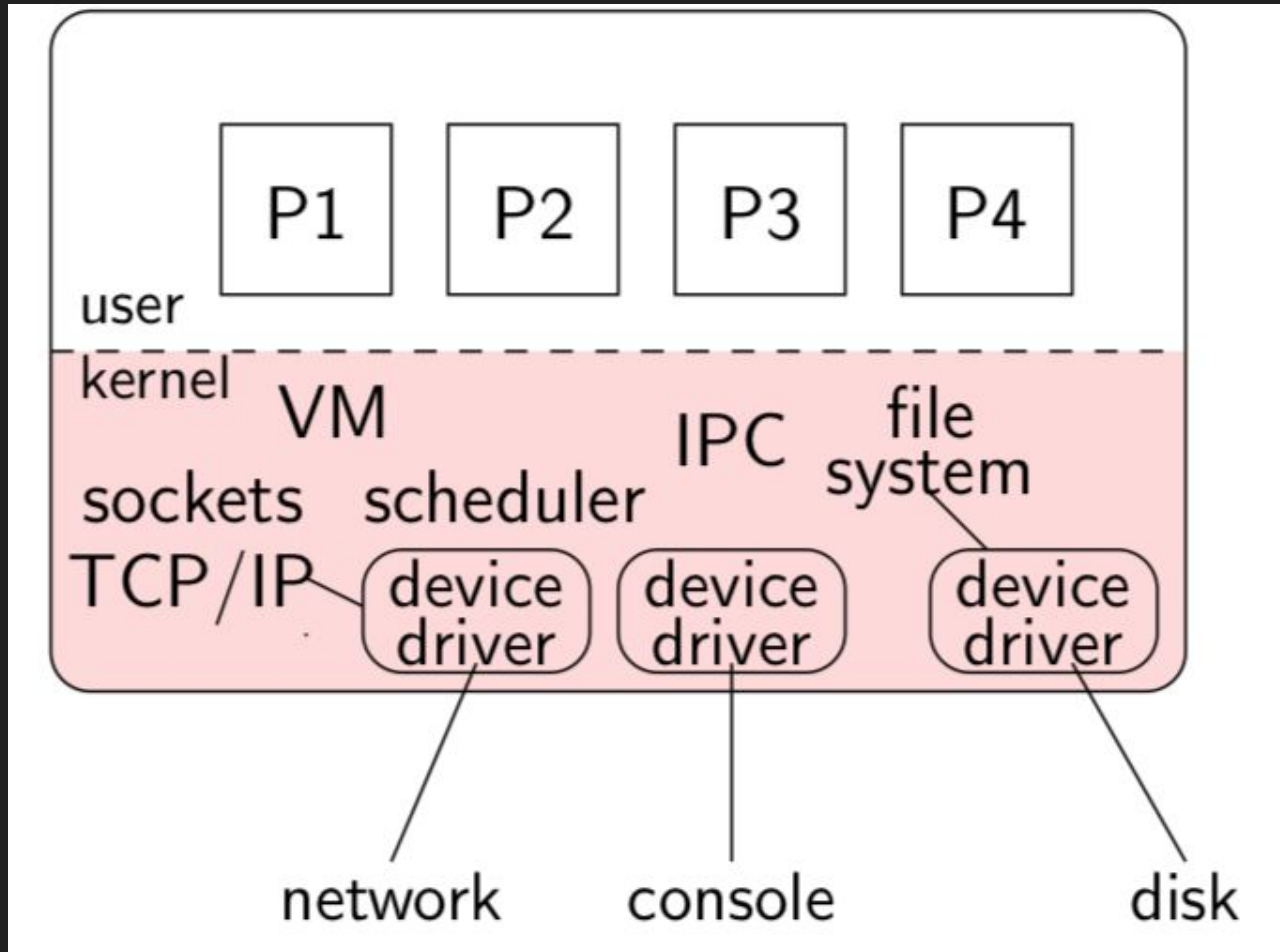
Les Familles d'OS



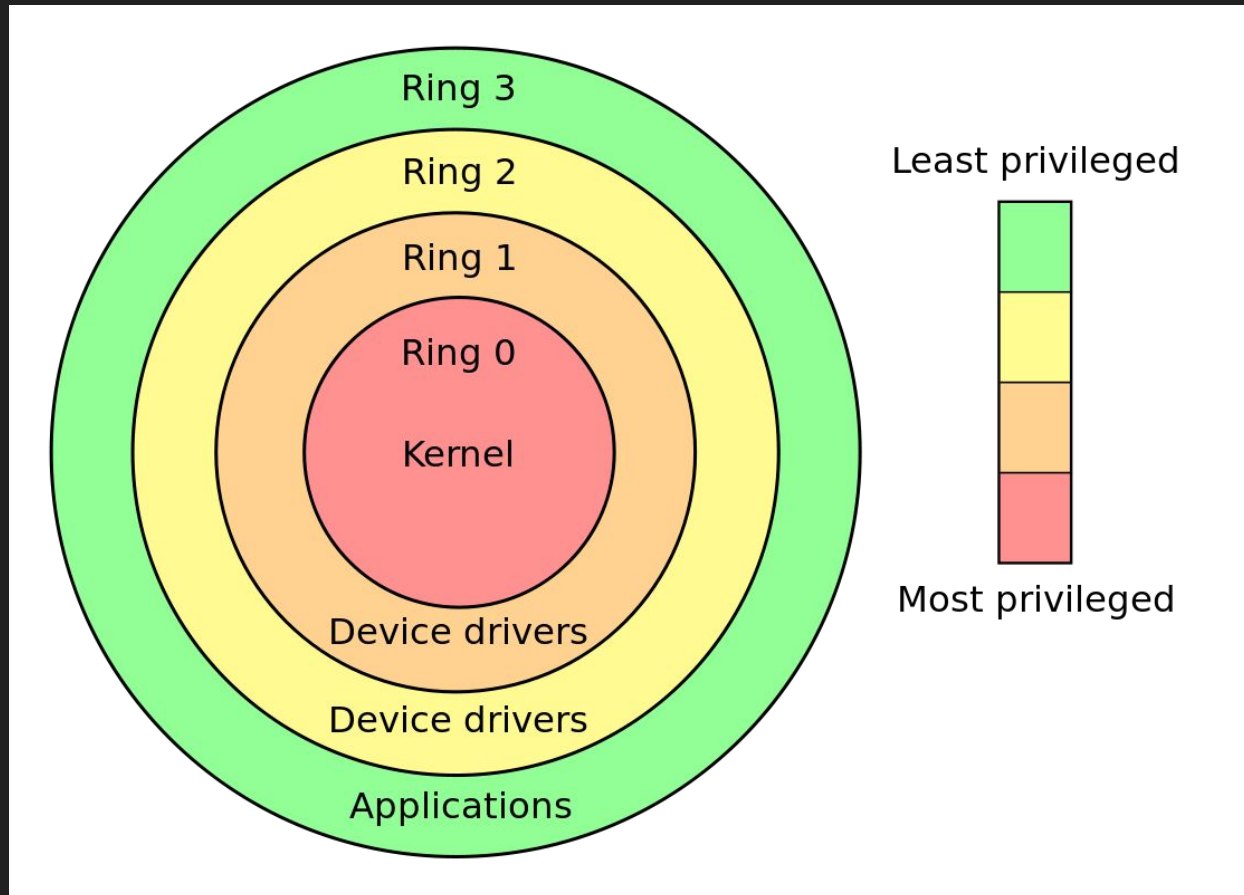
Objet du Cours

- Nous allons nous concentrer sur les OS utilisés dans le calcul haute-performance qui sont dans une majorité écrasante basés sur Linux et donc une base UNIX
- Il existe de forte similitudes entre tous les Système UNIX:
 - La présence du SHELL et la structure des processus
 - Des « commandes » partagées (yes, grep, ...)
 - Le standard de programmation système POSIX
 - La structure des répertoires principaux (racine, notion de chemins relatifs et absolus)
 - La notion d'utilisateur et de droits sur les fichiers

Architecture du Kernel

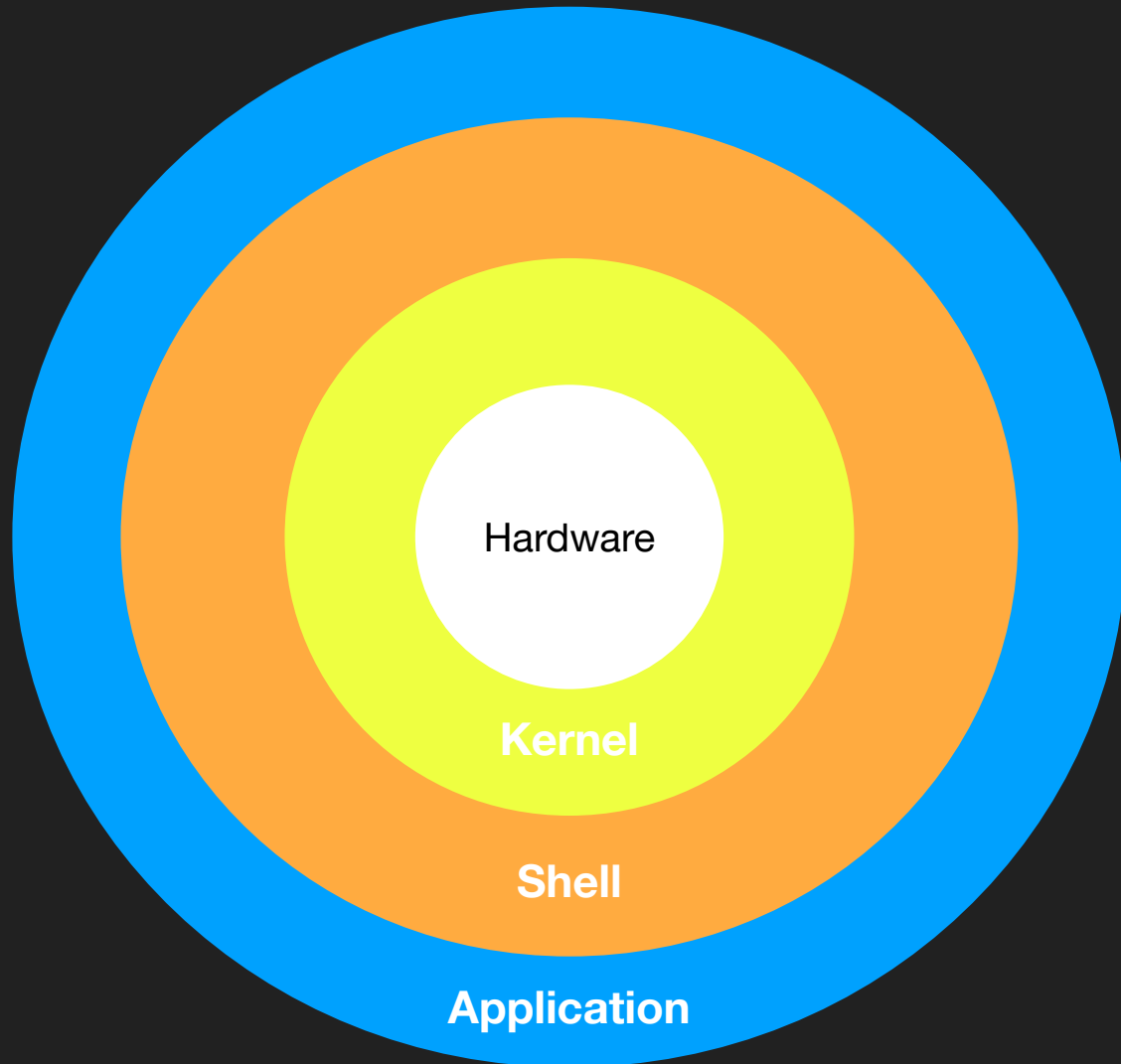


Les Rings CPU



[https://fr.wikipedia.org/wiki/Anneau de protectio](https://fr.wikipedia.org/wiki/Anneau_de_protection)

Architecture d'un OS



Shell et Port Série

A l'origine pour programmer une machine il suffisait d'une simple interface textuelle, connue comme la console. Il faut noter à quel point cette interface est toujours proéminente !



Ceci est un port série fonctionnant selon le protocole RS232, on pouvait s'y connecter pour avoir une console textuelle.

Shell et (Pseudo-)Terminaux

- Linux généralement expose de multiples TTY (teletype terminals) accessibles via les touches CTRL+ALT+FX (en étant sur le système);
- Tout programme peut créer un PTY (pseudo-teletype) qui est une instance virtuelle d'un port série dont le rôle est de regrouper la sortie d'un ou plusieurs programmes

Tout comme notre port série, le TTY prend une entrée texte et produit des données sur sa sortie. C'est un simple flux bidirectionnel. De plus un TTY peut avoir de nombreux paramètres tels que:

- Le comportement sur séquence d'échappement (envoi de signaux);
- Le débit en bauds;
- Une largeur et une hauteur;

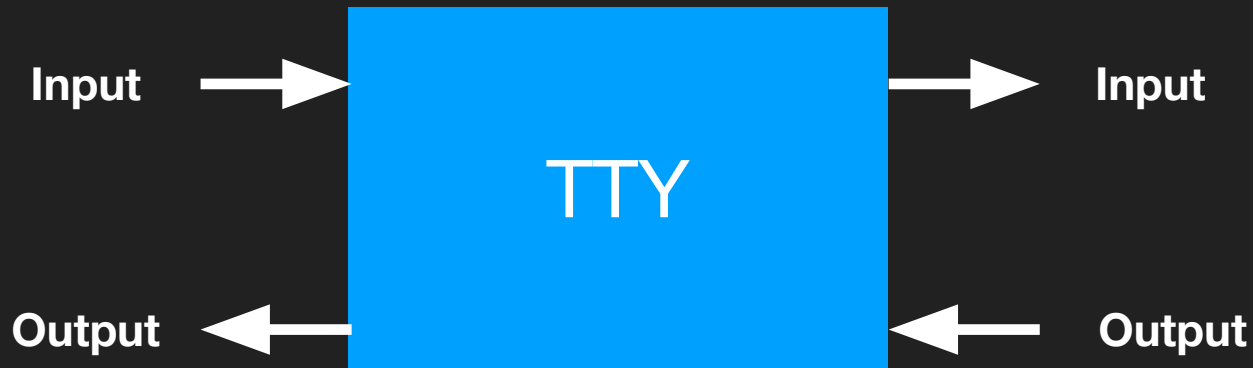
<http://www.linux-france.org/article/man-fr/man3/tcflow-3.htm>

!

<http://man7.org/linux/man-pages/man7/pty.7.htm>

!

Pseudo-Teletype



Descripteur de Fichier A

Descripteur de Fichier B

Mais il y a plus ...

Pseudo-Teletype



**Teletype Model 33 (1963) (source
wikipedia)**

Un TTY possède une taille

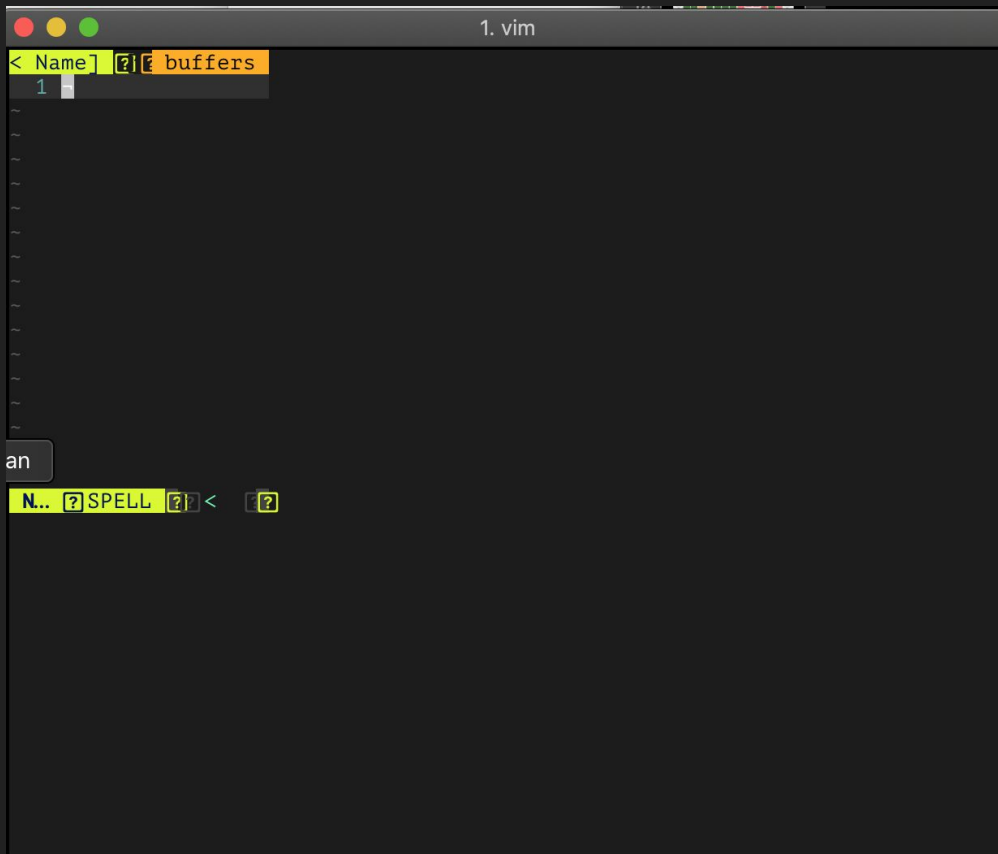
```
$ stty -a | grep row  
speed 38400 baud; 25 rows; 80 columns;
```



**Par défaut VIM occupe
toute la fenêtre.**

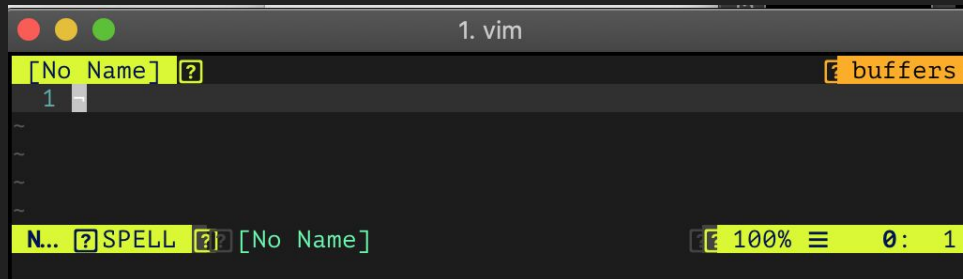
Un TTY possède une taille

```
$ stty rows 20 cols 20
```



**L'attribut du TTY a altéré
les dimension du terminal.**

Un TTY possède une taille



Tout redevient normal si l'on redimensionne la fenêtre mais pourquoi ???

TTY et signaux

Shell A

```
$ tty  
/dev/ttys001  
$ vim
```




Shell B

```
$ stty -f /dev/ttys001 rows 10
```

```
1. bash  
Last login: Wed Jan 9 07:30:42 on ttys001  
Isis:~ jbbesnard$ stty -F /dev/tty  
Display all 132 possibilities? (y or n)  
Isis:~ jbbesnard$ stty -F /dev/tty  
Display all 132 possibilities? (y or n)  
Isis:~ jbbesnard$ stty -F /dev/ttys001 rows 10  
stty: illegal option -- -F  
usage: stty [-a|-e|-g] [-f file] [options]  
Isis:~ jbbesnard$ stty -f /dev/ttys001 rows 10  
Isis:~ jbbesnard$ reset  
^[[A^[[AIsis:~ jbbreset  
Isis:~ jbbesnard$ stty -f /dev/ttys001 rows 10  
Isis:~ jbbesnard$ stty -f /dev/ttys001 rows 10  
Isis:~ jbbesnard$
```

TTY et signaux

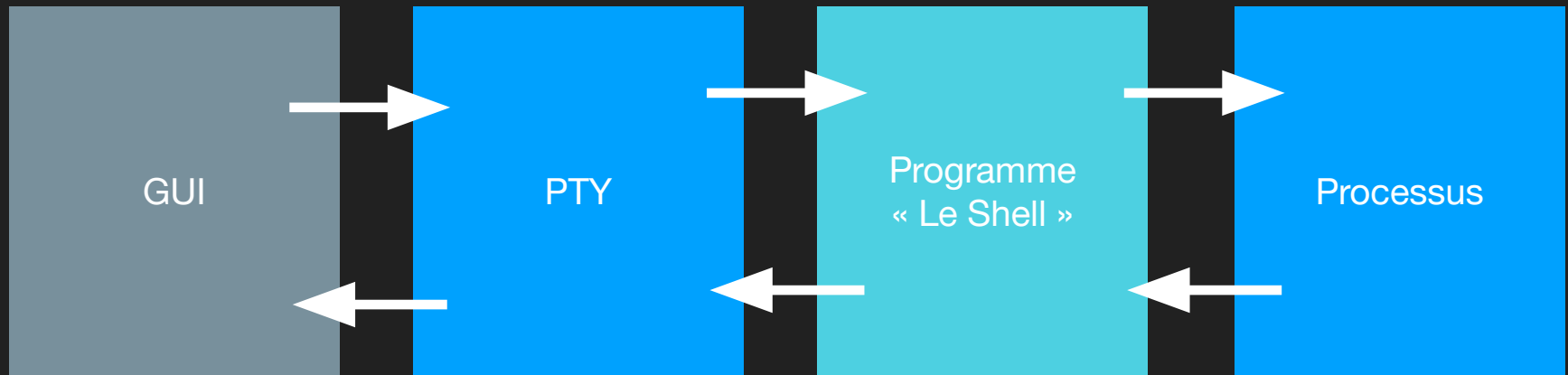
Le TTY gère de multiple signaux (et configurations) a destination des processus qui lui sont attachés.

Paramètre « stty »	Description	Touche par Défaut (voir stty -a)	SIGNAL
 intr	Interruption du programme	CTRL + C	SIGINT
 EOF	Fin de l'entrée	CTRL + D	
stop	Interrompre la sortie	CTRL + S	
Start	Reprendre la sortie	CTRL + Q	
 susp	Interrompt le programme cible	CTRL + Z	SIGSTOP
Rows,cols	Changement de taille		SIGWHINCH

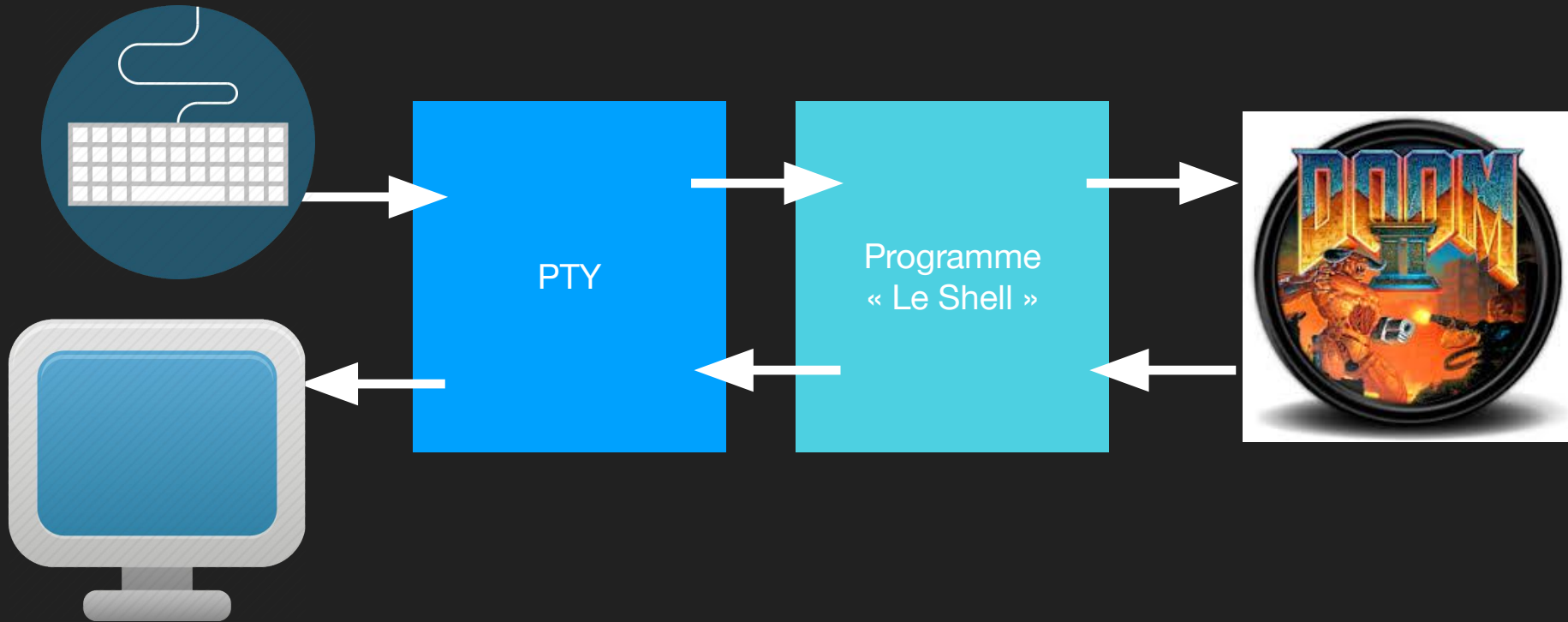
Ces paramètres sont tous dans la structure
termios:

<http://man7.org/linux/man-pages/man3/termios.3.htm>

Le Shell



Le Shell



Le Shell



ZSH



CSH








Le Shell



- Possibilité de lancer des commandes et de voir leur sortie
- Possibilité de chaîner des commandes entre elles
 - > pipe
- Gestion des entrées et sorties standards
 - > redirections, here-documents
- Langage de script complet et portable
 - > /bin/sh est sur tout UNIX;
- Gestion de jobs
 - > backgrounding, foregrounding, ..
- Parsing des arguments de la ligne de commande;
- Mise à disposition de built-ins
 - > ls, cd, setenv, ...

Les Built-ins

Commande	Description
 ls	Afficher le contenu d'un répertoire
 cd	Changer de dossier
 export	Définir une variable d'environnement
 echo	Afficher (y compris une variable) sur stdout
 pwd	Affiche le répertoire courant
ulimit	Change les limites de ressources des processus du shell
alias	Permet de définir des « raccourcis » de commande

Liste des Commandes a Connaitre

MINIMUM VITAL CRITIQUE !

ls, tree, cd, ln, export, grep, cat, pwd, test, find, tar, kill, ssh, df, du, rm, mkdir, cp, mv, clear, reset, alias, scp, touch, chmod, chown, printf, date, bc, sed, git, make, su, sudo, whoami, id, groups, last, who, ps, top/htop, less/more, wc, tee, diff, sort, xargs, time, tmux/screen,

man





fork, wait, waitpid,
getpid, getppid

Shell et Gestion de Processus

La commande ps permet de voir les processus à l'instant T

```
$ ps
  PID TTY          TIME CMD
 7898 pts/2    00:00:00 bash
 8622 pts/2    00:00:00 ps
```



```
$ ps -u
```

USER		PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND		
jbbes	na+	3615	0.0	0.0	201296	5564	tty2	Ssl+	janv.03	0:0	0 /usr/lib/gdm3/gdm-x-session	--run-script	i3
jbbes	na+	3624	0.0	0.0	114420	11180	tty2	S+	janv.03	0:0	8 i3 -a --restart /run/user/10	00/i3/restart	-state.3624
jbbes	na+	7898	0.0	0.0	22940	6696	pts/2	Ss	09:26	0:00	-bash		
jbbes	na+	8058	0.0	0.0	23040	7128	pts/4	Ss+	10:03	0:00	-bash		
jbbes	na+	8130	0.0	0.0	23040	6932	pts/5	Ss+	10:06	0:00	-bash		
jbbes	na+	8638	0.0	0.0	38308	3132	pts/2	R+	11:27	0:00	ps u		
jbbes	na+	10182	0.0	0.0	20516	4508	pts/0	Ss+	janv.07	0:0	0 /bin/bash		
jbbes	na+	18102	0.0	0.0	20536	4524	pts/1	Ss+	janv.07	0:0	0 /bin/bash		

Shell et Gestion de Processus

\$ ps au

USER		PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND			
Debia	n+	891	0.0	0.0	201296	5608	tty1	Ssl+	janv.02	0:0	0 /usr/lib/gdm3/gdm-x-session	gnome-sessio n	--autostart	/usr/share/gdm/greeter/autostart
root		893	0.0	0.2	263308	49308	tty1	Sl+	janv.02	7:3	8 /usr/lib/xorg/Xorg vt1 -disp	layfd 3 -auth	/run/user/11	7/gdm/Xauthority -background none
Debia	n+	907	0.0	0.0	545540	13068	tty1	Sl+	janv.02	0:0	0 /usr/lib/gnome-session/gnome	-session-bina	ry --autostar	t /usr/share/gdm/greeter/autostart
Debia	n+	927	0.0	1.3	243803 2	33116	4 tty1	Sl+	janv.02	1:0	1 /usr/bin/gnome-shell			
Debia	n+	951	0.0	0.1	102763 2	28528	tty1	Sl+	janv.02	0:0	5 /usr/lib/gnome-settings-daem	on/gnome-sett	ings-daemo n	
jbbes	na+	3615	0.0	0.0	201296	5564	tty2	Ssl+	janv.03	0:0	0 /usr/lib/gdm3/gdm-x-session	--run-script	i3	
root		3617	3.8	0.4	342240	11889 6	tty2	Sl+	janv.03	338: 4	8 /usr/lib/xorg/Xorg vt2 -disp	layfd 3 -auth	/run/user/10	00/gdm/Xauthority -background none
jbbes	na+	3624	0.0	0.0	114420	11180	tty2	S+	janv.03	0:0	8 i3 -a --restart /run/user/10	00/i3/restart	-state.3624	
jbbes	na+	7898	0.0	0.0	22940	6696	pts/2	Ss	09:26	0:00	-bash			
jbbes	na+	8058	0.0	0.0	23040	7128	pts/4	Ss+	10:03	0:00	-bash			
jbbes	na+	8130	0.0	0.0	23040	6932	pts/5	Ss+	10:06	0:00	-bash			
jbbes	na+	8669	0.0	0.0	38308	3336	pts/2	R+	11:31	0:00	ps au			
root		8703	0.0	0.0	73992	3356	tty3	Ss	janv.03	0:0	0 /bin/login --			
root		8738	0.0	0.0	21604	5588	tty3	S	janv.03	0:0	0 -bash			
jbbes	na+	1018 2	0.0	0.0	20516	4508	pts/0	Ss+	janv.07	0:0	0 /bin/bash			
jbbes	na+	1810 2	0.0	0.0	20536	4524	pts/1	Ss+	janv.07	0:0	0 /bin/bash			
root		2463 8	0.0	0.0	50764	5748	tty3	S+	janv.03	0:0	0 ssh orion			

Shell et Gestion de Processus

\$ ps aux

- **a**: tous les processus
(autrement session courante)
- **u**: tous les utilisateurs
- **x**: processus sans terminal attaché

```
an
ps aux
```

Shell et Gestion de Processus

`CODES D'ÉTAT DE PROCESSUS`

Voici les différentes valeurs que les indicateurs de sortie `s`, `stat` et `state` (en-tête « `STAT` » ou « `S` ») afficheront pour décrire l'état d'un processus :

<code>D</code>	en sommeil non interruptible (normalement entrées et sorties) ;
<code>R</code>	s'exécutant ou pouvant s'exécuter (dans la file d'exécution) ;
<code>S</code>	en sommeil interruptible (en attente d'un événement pour finir) ;
<code>T</code>	arrêté, par un signal de contrôle des tâches ou parce qu'il a été tracé ;
<code>W</code>	pagination (non valable depuis le noyau 2.6.xx) ;
<code>X</code>	tué (ne devrait jamais être vu) ;
<code>Z</code>	processus zombie (<defunct>), terminé mais pas détruit par son parent.

Pour les formats BSD et quand le mot-clé `stat` est utilisé, les caractères supplémentaires suivants peuvent être affichés :

<code><</code>	haute priorité (non poli pour les autres utilisateurs) ;
<code>N</code>	basse priorité (poli pour les autres utilisateurs) ;
<code>L</code>	avec ses pages verrouillées en mémoire (pour temps réel et entrées et sorties personnalisées) ;
<code>s</code>	meneur de session ;
<code>l</code>	possède plusieurs processus légers (« multi-thread », utilisant <code>CLONE_THREAD</code> comme NPTL pthreads le fait) ;
<code>+</code>	dans le groupe de processus au premier plan.

Shell et Gestion de Processus

Avoir la liste des signaux et leur identifiant numérique:

```
$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42)
SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52)
SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57)
SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62)
SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Envoyer un signal à un processus:

```
$ kill PID
```

```
# CTRL +C pour la tâche
courante
```



Shell et Gestion de Processus

Tuer un processus par son nom:

```
$ killall bash
```

```
$ pkill bash
```

Envoyer un signal à un processus:

```
$ kill -9 PID
```

```
# CTRL +C pour la tâche  
courante
```



Shell et Gestion de Processus

Lancer un processus en arrière plan:

```
$ sleep 5 &
```

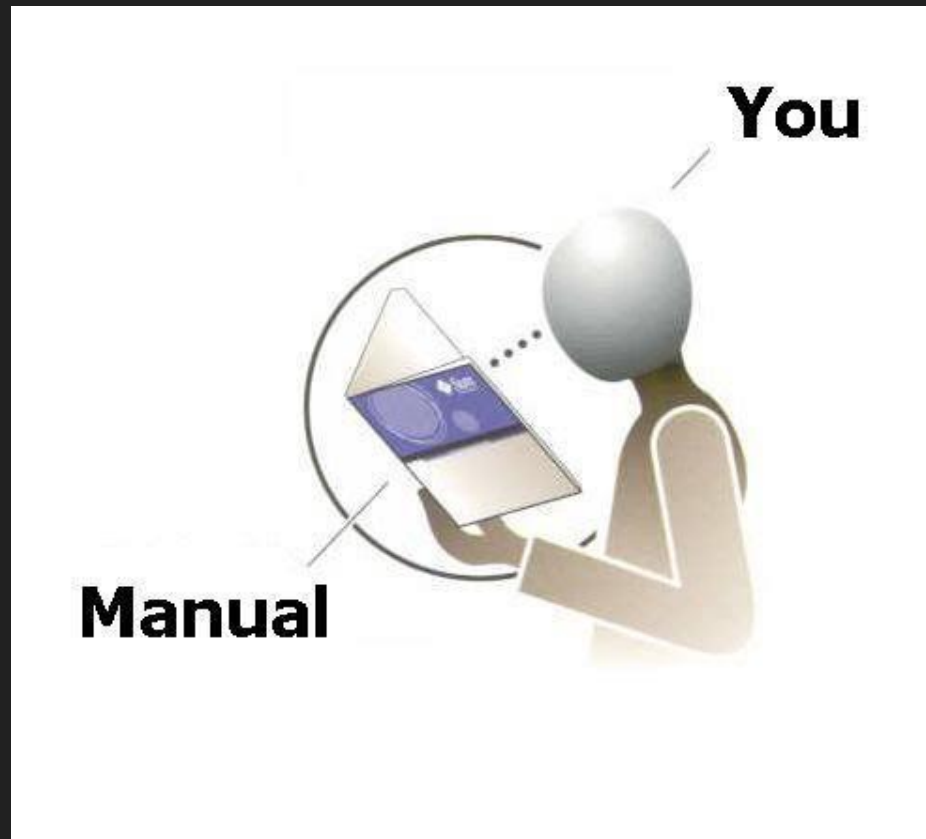
Attendre un processus en arrière plan:

```
$ wait
```

Lister les processus en arrière plan:

```
$ jobs
```





**kill, ps, killall/pkill,
pgrep**

man man

Le tableau ci-dessous indique le numéro des sections de manuel ainsi que le type de pages qu'elles contiennent.

```
1 Programmes exécutables ou commandes de l'interpréteur de commandes (shell)
2 Appels système (fonctions fournies par le noyau)
3 Appels de bibliothèque (fonctions fournies par les bibliothèques des programmes)
4 Fichiers spéciaux (situés généralement dans /dev)
5 Formats des fichiers et conventions. Par exemple /etc/passwd
6 Jeux
7 Divers (y compris les macropaquets et les conventions), par exemple man(7),
  groff(7)
8 Commandes de gestion du système (généralement réservées au superutilisateur)
9 Sous-programmes du noyau [hors standard]
```

Il existe des commandes complémentaires:

- apropos (recherche globale)
- whatis (description pour une entrée)

```
$ apropos "current working directory"
```

```
get_current_dir_name (3) - Get current working directory
```

```
getcwd (3) - Get current working directory
```

```
getwd (3) - Get current working directory
```



man man

```
$ man -k kill
kill (1) - Envoyer un signal à un processus
killall5 (8) - Envoyer un signal à tous les processus
pkill (1) - Rechercher ou envoyer un signal à des processus en fonction de leur...
skill (1) - Envoyer un signal ou rendre compte de l'état d'un processus
kill (2) - send signal to a process
killall (1) - kill processes by name
killpg (2) - send signal to a process group
killpg (3) - send signal to a process group
pthread_kill (3) - send a signal to a thread
pthread_kill_other_threads_np (3) - terminate all other threads in process
systemd-rfkill (8) - Load and save the RF kill switch state at boot and change
systemd-rfkill.service (8) - Load and save the RF kill switch state at boot and change
systemd-rfkill.socket (8) - Load and save the RF kill switch state at boot and change
systemd.kill (5) - Process killing procedure configuration
tkill (2) - send a signal to a thread
tkill (2) - send a signal to a thread
xkill (1) - kill a client by its X resource
XKillClient (3) - control clients
yes (1) - output a string repeatedly until killed
```

\$ man 1 kill

\$ man 2 kill

Pas pareil !



Commandes sur les Répertoires

Répertoire	Description
cd	Changer de répertoire
mkdir	Créer un répertoire
rm	Supprimer un fichier ou dossier
find	Trouver des fichiers
ls	Lister les fichiers
pwd	Afficher le répertoire courant
du	Afficher la taille d'un répertoire
df	Afficher l'espace libre sur les points de montage
grep	Chercher à l'intérieur des fichiers
cat	Afficher le contenu d'un fichier sur la sortie standard

Notion de chemin absolu et relatif:

- un chemin absolu commence par la racine /
- Un chemin relatif l'est par rapport au répertoire courant (../lib, ./lib/)



Anatomie d'un Os Tree

```
$ tree -L 1 /
```

```
/
├── bin
├── boot
├── dev
├── etc
├── home
├── lib
├── lib64
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── sys
├── tmp
├── usr
└── var
```

**Voici les répertoires à la racine / d'un
Linux
(Des variantes existent !!)**

Anatomie d'un Os Tree

Répertoire	Description
/bin	Binaires de base
/boot	Contient les image du noyau (kernel) et les images de boot ainsi que la configuration du bootloader
/dev	Contient les devices matérialisés sous forme de fichiers (sous UNIX tout est fichier)
/etc	Contient les fichiers de configurations pour le système et les services (apache, nfs, slurm ...)
/home	Contient les répertoires utilisateurs
/lib	Contient les bibliothèques partagées principales (libc)
/mnt	C'est ici que l'on monte temporairement des systèmes de fichier (par exemple une clef USB)
/opt	Emplacement générique des logiciels commerciaux
/proc	Système de fichier virtuel contenant de nombreuses informations sur les processus et l'état de la machine
/root	Home du super-utilisateur
/sbin	Binaires destinés au super-utilisateur uniquement
/sys	Configuration du kernel et matérielle (également via des fichiers)
/tmp	Répertoire temporaire généralement attaché à un ramfs
/usr	Répertoire ou la majorité des programmes sont installé (aussi connu comme préfixe)
/var	Répertoire où se situent les logs (/var/log) et les données des serveurs (par exemple /var/nginx/html)



Anatomie d'un préfixe

```
$ tree -L 1 /usr/  
/usr/  
├── bin  
├── include  
├── lib  
├── local  
├── share  
└── src
```

(Des variantes existent !!)

Anatomie d'un préfixe

Répertoire	Description
<code>/usr/bin</code>	Binaires des programmes installés
<code>/usr/include</code>	Headers des bibliothèques installées
<code>/usr/lib</code>	Bibliothèques (partagées et statiques installées)
<code>/usr/share</code>	Dépendances complémentaires (images, ressources, scripts) des programmes
<code>/usr/src</code>	Sources des programmes installés
<code>/usr/local/bin</code>	Binaires des programmes installés (à partir des sources et non de paquets)
<code>/usr/local/include</code>	Headers des bibliothèques installées (à partir des sources et non de paquets)
<code>/usr/local/lib</code>	Bibliothèques (partagées et statiques installées) (à partir des sources et non de paquets)
<code>/usr/local/share</code>	Dépendances complémentaires (images, ressources, scripts) des programmes (à partir des sources et non de paquets)
<code>/usr/local/src</code>	Sources des programmes installés (à partir des sources et non de paquets)



Les Différents Types de Fichiers

- : regular file
- d : directory
- c : character device file
- b : block device file
- s : local socket file
- p : named pipe
- l : symbolic link

```
$ ls -l `tty`
```

```
crw--w---- 1 jbbesnard tty 16, 1 9 jan 14:09 /dev/ttys001
```

Propriété d'un Fichier

```
$ ls -l /tmp/xauth-1000-_1
-rw----- 1 jbbesnard jbbesnard 98 janv.  3 11:44
/tmp/foo
```

Tout fichier a un groupe et un propriétaire (uid, gid)

```
$ ls -ln /tmp/xauth-1000-_1
-rw----- 1 1000 1000 98 janv.  3 11:44
/tmp/xauth-1000-_1
```

`$ chown toto ./foo` **# Changer le propriétaire du fichier**

`$ chgrp toto ./foo` **# Changer le groupe du fichier**

`$ chown toto:toto ./foo` **# Changer les deux à la fois**

Liste des groupes : /etc/group

Liste des utilisateurs : /etc/passwd



Droits des Fichiers

Correspondances de représentation des droits		
Droit	Valeur alphanumérique	Valeur octale
aucun droit	---	0
exécution seulement	--x	1
écriture seulement	-w-	2
écriture et exécution	-wx	3
lecture seulement	r--	4
lecture et exécution	rx-	5
lecture et écriture	rw-	6
tous les droits (lecture, écriture et exécution)	rwX	7

```
$ ls -l
$ chmod g-rwx ./toto.txt
$ chmod 400 ./toto.txt
```



Anatomie d'un préfixe

- Le préfixe est défini par des variables d'environnement qui déterminent où les binaires sont localisés:

PATH -> Chemin vers les binaires

LD_LIBRARY_PATH -> Chemin vers les bibliothèques

- Which et ldd permettent de comprendre comment ces variables sont exploitées:

```
$ which ls  
/bin/ls
```

```
$ ldd /bin/ls  
linux-vdso.so.1 (0x00007ffc0a26f000)  
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1  
(0x00007f009b0f7000)  
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f009ad58000)  
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f009aae5000)  
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f009a8e1000)  
/lib64/ld-linux-x86-64.so.2 (0x00007f009b540000)  
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0  
(0x00007f009a6c4000)
```

- La variables d'environnement LD_DEBUG permet de déboguer la résolution des symboles.

Points de Montage

Afficher les disques locaux et les partitions (disques durs)

```
$ lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPPOINT
sda	8:0	0	1,8T	0	disk	
—sda1	8:1	0	500M	0	part	
—sda2	8:2	0	40M	0	part	
—sda3	8:3	0	128M	0	part	
—sda4	8:4	0	2G	0	part	
—sda5	8:5	0	926,5G	0	part	
—sda6	8:6	0	10,2G	0	part	
—sda7	8:7	0	954M	0	part	/boot
—sda8	8:8	0	922,8G	0	part	/
sdb	8:16	0	29,8G	0	disk	
—sdb1	8:17	0	1,9G	0	part	[SWAP]
—sdb2	8:18	0	28G	0	part	/ssd
sr0	11:0	1	1024M	0	rom	

Points de Montage

Afficher l'ensemble des points de montage et l'espace libre:

```
$ findmnt -D
```

SOURCE	FSTYPE	SIZE	USED	AVAIL	USE%	TARGET
udev	devtmpfs	11,8G	0	11,8G	0%	/dev
tmpfs	tmpfs	2,4G	55,5M	2,3G	2%	/run
/dev/sda8	ext4	907,3G	9,5G	851,7G	1%	/
tmpfs	tmpfs	11,8G	73,1M	11,7G	1%	/dev/shm
tmpfs	tmpfs	5M	4K	5M	0%	/run/lock
tmpfs	tmpfs	11,8G	0	11,8G	0%	/sys/fs/cgroup
/dev/sdb2	ext4	27,4G	14G	12G	51%	/ssd
/dev/sda7	ext4	923M	43M	816,4M	5%	/boot
orion.france.paratools.com:/media/raid	nfs4	10,8T	1,5T	8,8T	13%	/media/raid
tmpfs	tmpfs	2,4G	16K	2,4G	0%	/run/user/117
tmpfs	tmpfs	2,4G	56K	2,4G	0%	/run/user/1000
gvfsd-fuse	fuse.gvfsd-fuse	0	0	0	-	/run/user/1000/gvfs
tmpfs	tmpfs	2,4G	0	2,4G	0%	/run/user/0

Liste des points de montage et leur options (plus verbeux):

```
$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=12307376k,nr_inodes=3076844,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=2464432k,mode=755)
/dev/sda8 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
(rw,relatime,fd=32,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=619)
mqueue on /dev/mqueue type mqueue (rw,relatime)
(...)
/dev/sdb2 on /ssd type ext4 (rw,relatime,data=ordered)
/dev/sda7 on /boot type ext4 (rw,relatime,data=ordered)
orion.france.paratools.com:/media/raid on /media/raid type nfs4
(rw,relatime,vers=4.2,rsiz=1048576,wsiz=1048576,namlen=255,hard,proto=tcp,port=0,timeo=600,retrans=2
,sec=sys,clientaddr=192.168.201.42,local_lock=none,addr=192.168.201.127)
```

Le fichier /etc/fstab

Ce fichier de configuration définit les points de montage:

```
$ cat /etc/fstab
# <file system> <mount point>    <type>  <options>. <dump>  <pass>
UUID=0dd1272e-15a3-4386-b3e2-8f8b2e050e35 /          ext4      errors=remount-ro 0      1
UUID=ca5adb7f-568b-4ac4-a3f0-63213234af1b /boot      ext4      defaults          0      2
UUID=b999d670-0ab2-4011-92f2-65df1266678c /ssd       ext4      defaults          0      2
UUID=254a9813-ca63-4578-a9ad-d58dd9af1f48 none        swap      sw                0      0
/dev/sr0          /media/cdrom0  udf,iso9660 user,noauto      0      0
orion.france.paratools.com:/media/raid/  /media/raid  nfs        rw,async,vers=4  0
0
```

Lancement d'un Programme

```
$ ls -la .
```

```
int main(int argc, char  
**argv)  
{  
    return 0;  
}
```

argc = 3

argv = {« ls », « -la », « . » , NULL }



Lancement d'un Programme

```
$ ls -la .
```

```
int main(int argc, char **argv, char
**envp)
{
    return 0;
}
```

argc = 3

argv = {« ls », « -la », « . » , NULL }

envp = {« PATH=XXX », « XX », ... }

Executer un Programme

```
int execvp(const char *command, char *const argv[]);
```

- **command:** nom du binaire
- **argv:** tableau argv

Il en existe d'autres:

```
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...
          /* (char *) NULL */);
int execlp(const char *file, const char *arg, ...
          /* (char *) NULL */);
int execlx(const char *path, const char *arg, ...
          /*, (char *) NULL, char * const envp[] */);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
            char *const envp[]);
```



```
#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

int main(int argc, char ** argv )
{
    pid_t child = fork();

    if( child == 0 )
    {
        char * const eargv[] = { "ls", "-la", NULL };
        execvp(eargv[0], eargv);
    } else {
        wait(NULL);
        fprintf(stderr, "child done\n");
    }

    return 0;
}
```

Executer un Programme



Les Signaux

On le définit comme une interruption logicielle asynchrone envoyée à un programme. Celle ci déclenche (ou non) un traitement ou l'interruption du programme. Il est possible à un programme d'ignorer les signaux et/ou de déclencher un traitement associé. Dans ce dernier cas on exécute un sighandler (gestionnaire de signal).

Les signaux se manipulent avec des ensemble de signaux:

```
#include <signal.h>

int sigemptyset(sigset_t *set);    // vider un semble de signaux

int sigfillset(sigset_t *set);    // Remplir un ensemble de signaux

int sigaddset(sigset_t *set, int signum); // Ajouter un signal à l'ensemble

int sigdelset(sigset_t *set, int signum); // Retirer un signal de l'ensemble

int sigismember(const sigset_t *set, int signum); // Vérifier l'appartenance
```

Bloquer des Signaux

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Il est possible de retarder certains signaux dans des sections critiques de programmes.
Il n'est jamais possible de bloquer SIGKILL et SIGSTOP

```
sigset_t blk;  
sigset_t sigsv;  
  
sigemptyset(&blk);  
sigaddset(&blk, SIGINT);  
sigaddset(&blk, SIGPIPE);  
sigprocmask(SIG_BLOCK, &blk, &sigsv);  
  
/* section critique */  
  
sigprocmask(SIG_SETMASK, &sigsv, NULL);
```

Récupérer les signaux en attente

```
int sigpending(const sigset_t *set);
```

Lorsque l'on bloque des signaux ils ne sont pas perdus !

```
sigset_t pending;
```

```
sigpending(&pending);
```

```
if(sigismember(&pending, SIGINT))  
{  
    puts("SIGINT is pending");  
}
```

Déclencher un signal en Attente

Lorsque l'on bloque des signaux ils ne sont pas perdus !

```
int sigsuspend(const sigset_t *set);

sigset_t pending;
sigset_t notpipe;

sigfillset(&notpipe);
sigdelset(&notpipe, SIGPIPE);

sigpending(&pending);

if(sigismember(&pending, SIGPIPE))
{
    sigsuspend(&notpipe);
}

// SIGPIPE déclenché masque précédent rétabli
```

Définir des Actions de Signaux

```
int sigaction(int signum, const struct sigaction *act,  
              struct sigaction *oldact);
```

```
struct sigaction {  
    void (*sa_handler)(int signal);  
    void (*sa_sigaction)(int signal, siginfo_t *, void *);  
    sigset_t sa_mask;  
    int sa_flags;  
};
```

Membre de struct	Description
sa_handler	Callback à appeler pour faire l'action
sa_sigaction	Callback à appeler pour faire l'action (étendu si flag SA_SIGINFO)
sa_mask	Signaux bloqués lors du traitement
sa_flags	Drapeaux décrivant le traitement du signal

Définir des Actions de Signaux

```
#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>


static void hdl (int sig, siginfo_t *siginfo, void *context)
{
    printf ("Sending PID: %ld, UID: %ld\n",
            (long)siginfo->si_pid, (long)siginfo->si_uid);
}


int main (int argc, char *argv[])
{
    struct sigaction act;

    memset (&act, '\0', sizeof(act));

    /* Use the sa_sigaction field because the handles has two additional parameters */
    act.sa_sigaction = &hdl;

    /* The SA_SIGINFO flag tells sigaction() to use the sa_sigaction field, not sa_handler. */
    act.sa_flags = SA_SIGINFO;

    if (sigaction(SIGTERM, &act, NULL) < 0) {
        perror ("sigaction");
        return 1;
    }

    while (1)
        sleep (10);

    return 0;
}
```



Précautions pour les Signaux

Il faut s'assurer que les fonctions appelées dans un gestionnaire de signal sont régénérantes, cela signifie qu'elles peuvent être interrompue durant leur exécution sans laisser le programme dans un état incohérent. Il ne faut par exemple pas altérer de variables globales.

Le cas de la variable `errno` est intéressant car la majorité des appels de la libc le modifie en cas d'erreur. Il faut donc le sauvegarder et le restaurer.

\$ man signal-safety



Mise en Place de Timers

Il existe un appel pour lever un signal au bout d'un certain temps:

Le signal levé est SIGALRM. Passer la valeur « 0 » annule un précédent timer.

```
unsigned int alarm(unsigned int seconds);
```

```
#include <stdio.h>

#include <signal.h>

#include <time.h>

#include <unistd.h>

#include <stdlib.h>

void alarmPrint(int signum){

    time_t curtime;

    time(&curtime);

    printf("current time is %s", ctime(&curtime));

    alarm(1);

}

int main(){

    signal(SIGALRM, alarmPrint);

    alarm(1);

    while(1) {

    }

    return 0;

}
```

Lever un Signal

On peut envoyer un signal à tout processus en utilisant la fonction:

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

```
#include <signal.h>
```

```
int kill(pid_t pid, int sig);
```



Ancienne Interface

Il existe une interface moins portable mais plus simple pour faire des tests:

```
#include <signal.h>
```

```
typedef void (*sighandler_t) (int) ;
```

De plus, elle ne fournis pas le sighandler étendu.

```
sighandler_t signal(int signum, sighandler_t handler) ;
```

