Environnement POSIX
Bases, I/O et Réseau

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

Jean-Baptiste Besnard
<jbbesnard@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 la matinée
- Un projet visant à implémenter un allocated mémoire
- Un examen le 16/12/2024

Type d'Examen	Coefficient
QCMs	10 %
Projet	40 %
EXAMEN	50 %

Cours et Corrections

https://github.com/besnardjb/AISE_25

Programme du Jour

Généralités sur les OS et Entrées-Sorties

Généralités sur les OS

Système 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_protection

Architecture d’un OS

Le TTY / Le SHELL

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.html

http://man7.org/linux/man-pages/man7/pty.7.html

Pseudo-Teletype

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
```

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.html

Le Shell

Le Shell

Le Shell

ZSH

CSH

SH

Le Shell

- Possibilité de lancer des commandes et de voir leur sortie
- Possibilité de chainer 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, …

Processus et Threads

Notion de Processus

Une machine regroupe de nombreux processus qui sont chacun une instance d'un programme donné. Ces processus se partagent les resources du système et possèdent différents attributs permis lesquels:

- 99,9% des processus ont un parent;
- Tout les processus ont un ID unique à l'instant T c'est le ProcessID (PID);
- Un processus appartient à un utilisateur et à un groupe (moyennant setuid);
- Un processus a sa pile et son tas et sa vision unique de l'ensemble de la mémoire du système (notion de mémoire virtuelle);
- Un processus peut ouvrir des fichiers;
- Il a ses propre limites de resources;
- Il est dans un état donné;
- Un processus peut contenir plusieurs threads;
- Un processus possède ses handler de signaux et son masque de signaux;

```
systemd─┬─agetty
        ├─auditd───{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─cachefilesd
        ├─celery───celery
        ├─crond
        ├─dbus-daemon───{dbus-daemon}
        ├─dhclient
        ├─dnsmasq───dnsmasq
        ├─gitlab-runner───10*[{gitlab-runner}]
        ├─gssproxy───5*[{gssproxy}]
        ├─gunicorn───gunicorn
        ├─irqbalance
        ├─ksmtuned───sleep
        ├─libvirtd───16*[{libvirtd}]
        ├─lvmetad
        ├─mdadm
        ├─mongod───10*[{mongod}]
        ├─monitor───ovsdb-server
        ├─monitor───ovs-vswitchd
        ├─munged───3*[{munged}]
        ├─mysqld_safe───mysqld───20*[{mysqld}]
        ├─ntpd
        ├─opensm-launch───sleep
        ├─polkitd───5*[{polkitd}]
        ├─redis-server───2*[{redis-server}]
        ├─rpc.idmapd
        ├─rpc.mountd
        ├─rpc.statd
        ├─rpcbind
        ├─rsyslogd───2*[{rsyslogd}]
        ├─slurmctld───11*[{slurmctld}]
        ├─slurmdbd───5*[{slurmdbd}]
        ├─smartd
        ├─sshd───sshd───sshd───zsh───pstree
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-udevd
        ├─tuned───4*[{tuned}]
        └─vsftpd
```

Notion de Thread ❗

Un thread s'exécute dans un processus, le « flot principal » d'un processus est lui même un thread et en possède tous les attributs:

- Possède leur propre id unique à l'instant T (le Thread ID (TID));
- Il a sa propre pile mais il partage la « vision » mémoire du processus parent (sont tas) et la vaste majorité des segments binaires (à part les TLS);
- Il hérite directement des fichiers ouverts de son parent;
- Il peut altérer la mémoire des autres threads de son processus;
- Un thread utilise des resources systèmes (mémoire, temps CPU) et peut s'exécuter de manière concurrente avec un autres sur un système multi-coeurs;

```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
25346 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  4:01.58 ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
29071 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:43.72 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
28698 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:31.70 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25363 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.59 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25358 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  1:10.51 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
an    jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.05 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25356 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:01.03 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25355 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:16.17 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25354 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.04 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25353 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.05 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25352 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.04 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25351 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.05 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25350 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.05 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25349 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.03 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25348 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.05 │  ├─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
25347 jbbesnard  20   0 1731M  231M 10416 S  0.0  1.0  0:00.04 │  └─ /media/raid/phomes/jbbesnard/.vscode/extensions/ms-vscode.cpptools-0.20.1/bin/Microsoft.VSCode.CPP.Extension.linux
21168 root       20   0  367M  8728  6168 S  0.0  0.0  1:07.99 ├─ /usr/lib/udisks2/udisksd --no-debug
21173 root       20   0  367M  8728  6168 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/udisks2/udisksd --no-debug
21172 root       20   0  367M  8728  6168 S  0.0  0.0  0:00.00 │  ├─ /usr/lib/udisks2/udisksd --no-debug
21171 root       20   0  367M  8728  6168 S  0.0  0.0  0:00.17 │  ├─ /usr/lib/udisks2/udisksd --no-debug
21169 root       20   0  367M  8728  6168 S  0.0  0.0  0:00.00 │  └─ /usr/lib/udisks2/udisksd --no-debug
```

Vision Canonique d'un Process

- Environnement
- Répertoire courant (CWD)
- UID, GID
- Image programme donnée (binaire)

Vision Canonique d’un Process

Tout processus contient a minima un thread lors de son lancement, ce thread peut être vu comme ayant tous les attributs du process. On l’appelle le thread principal.

Les Etats d'un Processus

Processus et Threads

Création de Processus

Tous les processus sont des descendants de « init ». Pour créer un processus on « fork ».

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

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

   if( child == 0 )
   {
      fprintf(stderr, "CHILD\n");
      sleep(5);
      fprintf(stderr, "CHILD: done waiting\n");
   } else {
      fprintf(stderr, "PARENT\n");
      wait(NULL);
      fprintf(stderr, "PARENT: child done\n");
   }

   return 0;
}
```

Création de Processus

```
$ gcc fork.c
$ ./a.out
PARENT
CHILD
CHILD: done waiting
PARENT: child done
```

Cette sortie est-elle déterministe ?

Création de Processus

```
$ gcc fork.c
$ ./a.out
PARENT
CHILD
CHILD: done waiting
PARENT: child done
```

Cette sortie est-elle déterministe ?

OYEN car « Il existe a peu près un seul ordre possible »

Notion de Copy on Write (COW)

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
   int val = 2;
   pid_t child = fork();

   if( child == 0 )
   {
      val++;
   } else {
      val+=2;
      wait(NULL);
   }

   fprintf(stderr, "PID %d PPID %d VAL is %d\n",
         getpid(),
         getppid(),
         val);

   return 0;
}
```

Sortie ?

Déterministe ?

Notion de Copy on Write (COW)

```
$ ./a.out
PID 8305 PPID 8304 VAL is 3
PID 8304 PPID 8058 VAL is 4
```

Toute la mémoire du parent est reproduite « instantanément » lors du fork et différenciée par page lors d'une écriture.
—> **Copy ON Write**

Déterministe en ordre PID et PPID non prévisibles.

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	/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	/usr/lib/gdm3/gdm-x-sessio	nome-sessio	--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	gdm/Xauthority -background nor
Debia	n-+	907	0.0	0.0	545540	13068	tty1	Sl+	janv.02	0:0	/usr/lib/gnome-session/gnon	-session-binar	y --autostar	/usr/share/gdm/greeter/autostar
Debia	n-+	927	0.0	1.3	2438032	33116	4 tty1	Sl+	janv.02	1:0	1 /usr/bin/gnome-shell			
Debia	n-+	951	0.0	0.1	1027632	28528	tty1	Sl+	janv.02	0:0	/usr/lib/gnome-settings-daeo	n/gnome-set	ngs-daemon	
jbbes	na+	3615	0.0	0.0	201296	5564	tty2	Ssl+	janv.03	0:0	/usr/lib/gdm3/gdm-x-sessio	--run-script	i3	
root		3617	3.8	0.4	342240	118896	tty2	Sl+	janv.03	338:48	/usr/lib/xorg/Xorg vt2 -disp	layfd 3 -auth	/run/user/10	gdm/Xauthority -background no
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+	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			
root		24638	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

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 :

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

               <    haute priorité (non poli pour les autres utilisateurs) ;
               N    basse priorité (poli pour les autres utilisateurs) ;
               L    avec ses pages verrouillées en mémoire
                    (pour temps réel et entrées et sorties personnalisées) ;
               s    meneur de session ;
               l    possède plusieurs processus légers
                    (« multi-thread », utilisant CLONE_THREAD comme NPTL pthreads le fait) ;
               +    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) SIGALRM15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG24) SIGXCPU25) 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 tache 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 tache 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

Commandes Basiques

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 resources des processus du shell
alias	Permet de définir des « racourcis » de commande

Liste des Commandes à Connaitre

Rouge = 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

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
tgkill (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 de l’OS Tree LINUX

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

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 débogguer la résolution des symboles.

Fichiers et Droits

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	r-x	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
```

Points de Montage

Points de Montage

Afficher les disques locaux et les partitions (disques durs)

```
$ lsblk
NAME    MAJ:MIN RM    SIZE RO TYPE MOUNTPOINT
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,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,port=0,timeo=600,retran
s=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 du Main

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 execle(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[]);
```

Executer un Programme

```c
#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;
}
```

Notion de Signaux

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 certain signaux dans des sections critiques de programmes.
Il n'est jamais possible de bloquer SIGKILL

```
    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

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

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

```
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 a appeler pour faire l'action
sa_sigaction	Callback a 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

```c
#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 dans 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 peu envoyer un signal à tout processus en utilisant la commande:

```
#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);

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

De plus, elle ne fournis pas le sighandler étendu.

I/Os bas-niveau

Les Fichiers Bas-Niveau

L’état d’un descripteur de fichier:
- Droits (lecture ou écriture)
- Bidirectionnel (en fonction des droits)
- Peut correspondre à un flux ou bien un fichier sur le disque
- Possède un offset courant (cas d’un fichier)
- On peut y lire et écrire (Read/Write)

Des descripteurs spéciaux:
- Stdin (0) (ou STDIN_FILENO de unistd.h) -> Entrée standard
- Stdout (1) (ou STOUT_FILENO de unistd.h) -> Sortie standard
- Stderr (2) (ou STERR_FILENO de unistd.h) -> Sortie d’erreur standard

On peut créer un descripteur avec:
- Open (sur fichier)
- Pipe (sur flux)
- socket (pour une connection réseau)

Ouvrir un Fichier

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

int open(const char *path, int oflag, … /* mode_t mode */);
```

Soit deux empreintes pour ouvrir:

int open(const char *path, int oflag);

int open(const char *path, int oflag, mode_t mode);

Arguments:

- **path:** chemin vers le fichier
- **oflag:** ou binaire entre les options (O_RDWR, O_CREAT, O_APPEND, …)
- **(si O_CREAT) mode:** ou binaire définissant les droits du fichier

Retour < 0 si erreur

Fermer un Fichier

```
#include <unistd.h>

int close(int fildes);
```

Arguments:

- **fildes:** descripteur de fichier ouvert

Retour < 0 si erreur

Créer un fichier vide

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
   int fd = open("./toto", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );

   if( fd < 0)
   {
      perror("open");
   }


   close(fd);

   return 0;
}
```

Créer un fichier vide

```
jbbesnard@deneb | <0> | lun. janv. 14 12:17:03
~/AISE_2
$ls -la toto
-rw------- 1 jbbesnard jbbesnard 0 janv. 14 12:16 toto
```

Créer un fichier vide (non existant)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
   int fd = open("./toto", O_RDWR | O_CREAT | O_EXCL,
                           S_IRUSR | S_IWUSR );


   if( fd < 0)
   {
      perror("open");
   }


   close(fd);

   return 0;
}
```

Créer un fichier vide (non existant)

```
jbbesnard@deneb | <0> | lun. janv. 14 12:19:29
~/AISE_2
$./a.out
open: File exists
```

Lire dans un Fichier

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Descripteur **Buffer** **Taille max**

On lit un fichier « bout par bout »:

Lire dans un Fichier

(possible EAGAIN sur socket)

```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char ** argv){

     if( argc != 2 )
     {
          fprintf(stderr, "Usage %s PATH\n", argv[0]);
          return 1;
     }

     int fd = open(argv[1], O_RDONLY);

     if( fd < 0 )
     {
          perror("open");
          return 1;
     }

     ssize_t cnt;
     char buff[500];

     while( (cnt = read(fd, buff, 500)) != 0)
     {
          if( cnt < 0)
          {
               perror("read");
               return 1;
          }

     }


     close(fd);

     return 0;
}
```

Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff,  size_t size)
{
   size_t written = 0;
   while( (size - written) != 0 )
   {
      errno = 0;
      ssize_t ret = write(fd, buff + written, size-written);

      if( ret < 0 )
      {
         if(errno == EINTR)
         {
            continue;
         }

         perror("write");
         return ret;
      }

      written += ret;
   }

}
```

Symétrie avec Read !

Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff,  size_t size)
{
   size_t written = 0;
   while( (size - written) != 0 )
   {
      errno = 0;
      ssize_t ret = write(fd, buff + written, size);

      if( ret < 0 )
      {
         if( (errno == EINTR) )
         {
            continue;
         }

         perror("write");
         return ret;
      }

      written += ret;
   }

}
```

On peut aussi vouloir gérer un signal entrant EINTR

Liste des Cas

Les cas à gérer pour read bas niveau:

- **EOF :** la fin du fichier (retour 0)
- **>0** : N bytes on été lus **(peut être moins que SIZE!!)**
- **<0 :** Une erreur s'est produite
  - ➡errno == EINTR l'appel a été interrompu par un signal
  - ➡errno == EAGAIN si on a marqué le fd O_NONBLOCK

Les cas à gérer pour write bas niveau:

- **>0** : N bytes on été écrits **(peut être moins que SIZE!!)**
- **<0 :** Une erreur s'est produite
  - ➡errno == EINTR l'appel a été interrompu par un signal
  - ➡errno == EAGAIN si on a marqué le fd O_NONBLOCK

Changer d’Offset

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

off_t lseek(int fd, off_t offset, int whence);
```

Whence	Description
SEEK_SET	Règle l’offset à « offset »
SEEK_CUR	Retourne l’offset courant plus le paramètre offset
SEEK_END	Retourne l’offset de fin plus le paramètre offset

Se déplacer à la Fin d’un Fichier

lseek(fd, 0, SEEK_END);

Un équivalent ?

Un équivalent ?

Ouvir le FD avec le paramètre 0_APPEND dans le Open !

R.T.M.

open(2), close(2), stat(2), read(2), write(2), pread(2), pwrite(2), fsync(2)

Avancé : fcntl(2)

UMASK

Notion de UMASK

Il est possible régler un masque de droit par défaut:

- Pour automatiquement masquer certain droits sur les nouveaux fichiers
- Ces droits s'appliquent à tout fichier nouvellement créé et s'écrivent en octal

```
jbbesnard@deneb | <0> | lun. janv. 14 12:28:38
~/AISE_2
$umask
0022
```

Quels droits sont masqués ?

Valeur	R	W	X
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Notion de UMASK

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777 );

    if( fd < 0)
    {
        perror("open");
    }


    close(fd);

    return 0;
}
```

Umask 0022 quels sont les droits de toto_um ?

Notion de UMASK

```
jbbesnard@deneb | <0> | lun. janv. 14 12:35:22
~/AISE_2
$ls -lah toto_um
-rwxr-xr-x 1 jbbesnard jbbesnard 0 janv. 14 12:34 toto_um
```

Notion de UMASK

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    umask(0777);
     int fd = open("./toto_um", O_RDWR | O_CREAT, 0777 );

    if( fd < 0)
    {
        perror("open");
    }


    close(fd);

    return 0;
}
```

Quels sont les droits de toto_um ?

Notion de UMASK

```
jbbesnard@deneb | <0> | lun. janv. 14 12:38:00
~/AISE_2
$ls -lah toto_um
---------- 1 jbbesnard jbbesnard 0 janv. 14 12:37 toto_um
```

Notion de UMASK

Comment régler les droits totalement dans OPEN ?

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int oldm = umask(0000);
    int fd = open("./toto_um", O_RDWR | O_CREAT, 07777 );
    umask(oldm);

    if( fd < 0)
    {
        perror("open");
    }


    close(fd);

    return 0;
}
```

Calcul du UMASK

D = PERM & (~ UMASK)

Exemple:

D = 0777 & (~ 0022)

D = 111 111 111 & (~ 000 010 010)

D = 111 111 111 & 111 101 101

D = 111 101 101

Droits en Détail

set-UID: execution possible avec l'utilisateur du binaire

set-GID: execution possible avec le groupe du binaire

Sticky bit (sur repertoire): seul le propriétaire du répertoire et du fichier peuvent le supprimer utilisé pour /tmp

I/Os Haut Niveau

Ouvrir un Fichier (FILE *)

```
#include <stdio.h>

FILE *fopen(const char *pathname, const char *mode);

FILE *fdopen(int fd, const char *mode);
```

Le FILE * est une sur-couche au FD précédemment vu.

Fermer un Fichier (FILE *)

```
#include <stdio.h>

int fclose(FILE *stream);
```

Ouvrir un Fichier (FILE *)

```
#include <stdio.h>


int main(int argc, char ** argv){

   FILE * fd = fopen(argv[1], "r");

   if(!fd){
      perror("fopen");
      return 1;
   }

   fclose(fd);

   return 0;
}
```

Lire un Fichier

```c
#include <stdio.h>

int main(int argc, char ** argv){
    FILE * fd = fopen(argv[1], "r");
    if(!fd){
        perror("fopen");
        return 1;
    }
    char buff[500];
    size_t cnt;

    while( 1 )
    {
        cnt = fread(buff, sizeof(char), 500, fd);
        if( cnt == 0)
        {
            if( feof(fd) )
            {
                break;
            }
            else
            {
                perror("fread");
                return 1;
            }
        }

        /* USE your buff here */
    }
    fclose(fd);
    return 0;
}
```

Lire ligne par ligne

```c
#include <stdio.h>

int main(int argc, char ** argv){

    if( argc != 2 )
        return 1;

    FILE * fd = fopen(argv[1], "r");

    if(!fd){
        perror("fopen");
        return 1;
    }

    char buff[500];
    char * ret;

    while(1)
    {
        ret = fgets(buff, 500, fd);

        if(!ret)
        {
            if( feof(fd) )
            {
                /* EOF all OK*/
                break;
            }
            else
            {
                /* Error */
                perror("fgets");
                return 1;
            }
        }

        /* USE your buff here */
        fprintf(stdout, "%s", ret );
    }
    fclose(fd);
    return 0;
}
```

Ecrire dans un Fichier

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char ** argv){
   if(argc != 2 )
      return 1;

   FILE * fd = fopen(argv[1], "w");

   if(!fd){
      perror("fopen");
      return 1;
   }

   char data[] = "Hello I/Os\n";
   size_t cnt;

   cnt = fwrite(data, sizeof(char),
                strlen(data), fd);

   if( cnt == 0)
   {
      perror("fread");
      return 1;
   }

   fclose(fd);

   return 0;
}
```

fopen, fclose, fread, fwrite, fgets, ftell, fseek, feof, fileno, fdopen

Redirection de Flux

Redirection de Flux

```
#include <unistd.h>

int dup2(int oldfd, int newfd);
```

Dup2 remplace « newfd » par « oldfd » et se charge de fermer « newfd ».

Exemple

Redirection de sortie dans un fichier

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

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

    if( child == 0)
    {
        int out = open("./out.dat", O_CREAT | O_WRONLY ,
                                        0600);
        /* Replace stdout with the file */
        dup2(out, STDOUT_FILENO);
        close(out);
        char * argv[] = {"ls","-la", NULL};
        execvp( argv[0], argv);
    }
    else
    {
        /* Parent closes out */
        wait(NULL);
    }


        return 0;
}
```

Création de Pipe

```
#include <unistd.h>

int pipe(int pipefd[2]);
```

Crée un « tuyau » == PIPE en anglais.

pipefd[2] = { READ_END, WRITE_END };

Un pipe est UNIDIRECTIONNEL

Chainer deux Commandes

echo "Salut Tout Le Monde " | tac -s " "

```
$echo "Salut Tout Le Monde " | tac -s " "

Monde Le Tout Salut
```

Chainer deux Commandes

echo "Salut Tout Le Monde " | tac -s " "

```
$./a.out
Monde Le Tout Salut
```

```c
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
	int pp[2];
	pipe(pp);

	pid_t child1 = fork();

	if( child1 == 0)
	{
		/* Replace stdout with the write end of the pipe */
		dup2(pp[1], STDOUT_FILENO);
		/* Close read end of the pipe */
		close(pp[0]);
		/* Run command */
		char * argv[] = {« printf","Salut Tout Le Monde « , NULL};
		execvp( argv[0], argv);
	}
	else
	{
		pid_t child2 = fork();

		if(child2 == 0)
		{
			/* Replace stdin with the read end of the pipe */
			dup2(pp[0], STDIN_FILENO);
			/* Close write end of the pipe */
			close(pp[1]);
			/* Run command */
			char * argv[] = {"tac","-s", " ", NULL};
			execvp( argv[0], argv);
		}
		else
		{
			/* Close both end of the pipe */
			close(pp[0]);
			close(pp[1]);
			/* wait for two child */
			wait(NULL);
			wait(NULL);
		}
	}

	return 0;
}
```

Chainer deux Commandes

**Juste après le fork,
les descripteurs sont dans
tous les fichiers.**

Chainer deux Commandes

Ensuite on insère le PIPE entre les deux commandes.

Introduction au Réseau

Programmation Réseau

En HPC le réseau est un aspect crucial car les machines sont:

- Distribuées (milliers de noeuds)
- La mémoire par noeud est limitée

Il faut donc pouvoir interconnecter des milliers de processus UNIX pour former un calcul cohérent. MPI permet généralement cela
Mais il repose sur des réseaux et techniques plus bas niveau dont nous avons déjà vu certaines:

- Segment SHM
- Réseaux TCP (rare)
- Réseaux rapides (voir aperçu slide suivant)

Aujourd'hui nous allons implémenter des appels que vous faites plusieurs milliers de fois par jour à chaque fois que vous allez sur internet, consultez vos mail, parlez à votre assistant vocal …

Réseaux HPC

Infiniband (libverbs)

Bull Exascale Interconnect (Portals 4)

- Tofu interconnect
- Aries Interconnect
- Cray Slighshot
- TH Express

Base du Réseau

Très rapidement le but d’un réseau est de connecter des machines entre elles on peut citer comme support de réseau:

- Cable Ethernet (RJ45)
- Wifi (802.11a/b/g/n/ac)
- Bluetooth (802.15.1)
- Zigbee (802.15.4) (domotique)

D’un point de vue système ces différents réseaux reposent sur des interfaces relativement similaires nous allons voir aujourd’hui principalement du TCP mais sachez que cela se transpose assez facilement à d’autre types de réseaux.

Modèle en Couche (Rappel)

Version simplifiée (à 5 couche) du modèle OSI (Open System Interconnection)

- **Couche ethernet -> adresses MAC**
- **Couche IP -> addresses IP**
- **Couche TCP -> Modèle d'encapsulation des données**
- **Couche applicative -> Ce qui est transmis**

TCP = Transmission Control Protocol

TCP/IP

Couche IP

- **Chaque machine possède une adresse IP**
  ➡ sur 4 Octets (32 bits) pour IPV4
  ➡ Sur 16 octets (128 bits) IPV6

- **Elle communique sur un réseau identité par un masque:**
  ➡ 255.255.255.0 (IPV4) -> 254 machines (la valeur 255 est réservée)

```
Address:   192.168.0.1          11000000.10101000.00000000 .00000001
Netmask:   255.255.255.0 = 24   11111111.11111111.11111111 .00000000
Wildcard:  0.0.0.255            00000000.00000000.00000000 .11111111
=>
Network:   192.168.0.0/24       11000000.10101000.00000000 .00000000 (Class C)
Broadcast: 192.168.0.255        11000000.10101000.00000000 .11111111
HostMin:   192.168.0.1          11000000.10101000.00000000 .00000001
HostMax:   192.168.0.254        11000000.10101000.00000000 .11111110
Hosts/Net: 254                  (Private Internet)
```

Ce réseau est 192.168.0.0/24 (selon les bits de masquage)

Couche IP

```
Address:   192.168.0.1          11000000.10101000 .00000000.00000001
Netmask:   255.255.0.0 = 16     11111111.11111111 .00000000.00000000
Wildcard:  0.0.255.255          00000000.00000000 .11111111.11111111
=>
Network:   192.168.0.0/16       11000000.10101000 .00000000.00000000 (Class C)
Broadcast: 192.168.255.255      11000000.10101000 .11111111.11111111
HostMin:   192.168.0.1          11000000.10101000 .00000000.00000001
HostMax:   192.168.255.254      11000000.10101000 .11111111.11111110
Hosts/Net: 65534                (Private Internet)
```

65534 machines pour un réseau de masque /16
= 256 x 256 - 2

Exemple d’adresses non routable sur Internet

RFC1918 name	IP address range	Count	largest subnet mask)	host id size	mask bits	classful description[Note 1]
24-bit block	10.0.0.0 – 10.255.255.255	16777216	10.0.0.0/8 (255.0.0.0)	24 bits	8 bits	single class A network
20-bit block	172.16.0.0 – 172.31.255.255	1048576	172.16.0.0/12 (255.240.0.0)	20 bits	12 bits	16 contiguous class B networks
16-bit block	192.168.0.0 – 192.168.255.255	65536	192.168.0.0/16 (255.255.0.0)	16 bits	16 bits	256 contiguous class C networks

Afficher l’adresse IP

```
$ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
       valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
       valid_lft forever preferred_lft forever
3: wlp4s0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 8a:8d:54:80:7b:23 brd ff:ff:ff:ff:ff:ff
```

Afficher l’adresse IP

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
       valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
       valid_lft forever preferred_lft forever
```

- Interface -> enp3s0
- Connectée -> oui (UP)
- Adresse IPV4 -> 192.168.201.42/24
- Adresse IPV6 -> fe80::9a90:96ff:fed2:650e/64

Afficher les Routes

```
$ip route
default via 192.168.201.1 dev enp3s0 proto static metric 100
192.168.201.0/24 dev enp3s0 proto kernel scope link src 192.168.201.42 metric 100
```

- 192.168.201.0/24 est sur enp3s0
- « default » tout le reste est sur la machine 192.168.201.1

Couche Physique

Cable RJ45

Cable RJ45

Cable RJ45

192.168.201.2

« Julius »

192.168.201.3

« Tommy »

192.168.201.4

« Bob »

Couche Physique

WAN = Wide Area Network

LAN = Local Area Network

La Passerelle

- La machine qui est connectée à la fois à internet et au réseau local est appelée la « passerelle ».
- Chez les particulier c’est souvent une « Box » qui opère comme routeur entre ces deux réseau. Plus généralement c’est une machine avec au moins deux interface réseau.

La Passerelle en Entreprise

Exemples CISCO de routeurs configurables.

Network Address Translation

Network Address Translation

Network Address Translation

Google ne voit que votre passerelle qui se fait passer pour vous et vous renvoie les données

Transmission Control Protocol

- TCP permet d'établir des connections « fiables » entre machines en permettant:
  - ➡ Une validation de l'intégrité des données
  - ➡ Un mode connecté (A parle à B)
  - ➡ Un contrôle de flux (l'envoi bloque si la source ne lit pas assez vite par exemple)

- TCP connecte deux **IP** et deux **PORTS**
  - ➡ Un port est une « porte » vers votre machine
  - ➡ Il peut être « ouvert » ou « sortant »
  - ➡ Toute connection TCP est bidirectionnelle (deux FD)
  - ➡ Un port peut être en écoute (en attente de nouvelle connexions)
  - ➡ Plusieurs connexions peuvent utiliser le même port en écoute

192.168.201.2 : 3689 **192.168.201.9 : 80**

Services Communs

Protocol	Port	Name	Description
FTP	tcp/20, tcp21	File Transfer Protocol	Sends and receives files between systems
SSH	tcp/22	Secure Shell	Encrypted console access
Telnet	tcp/23	Telecommunication Network	Insecure console access
SMTP	tcp/25	Simple Mail Transfer Protocol	Transfer email between mail servers
DNS	udp/53, tcp/53	Domain Name System	Convert domain names to IP addresses
HTTP	tcp/80	Hypertext Transfer Protocol	Web server communication
POP3	tcp/110	Post Office Protocol version 3	Receive email into a email client
IMAP4	tcp/143	Internet Message Access Protocol v4	A newer email client protocol
HTTPS	tcp/443	Hypertext Transfer Protocol Secure	Web server communication with encryption
RDP	tcp/3389	Remote Desktop Protocol	Graphical display of remote devices
NetBIOS	udp/137	NetBIOS name service	Register, remove, and find Windows services by name
NetBIOS	udp/138	NetBIOS datagram service	Windows connectionless data transfer
NetBIOS	tcp/139	NetBIOS session service	Windows connection-oriented data transfer
SLP	tcp/427, udp/427	Service Location Protocol	Find Mac OS services by name
SMB	tcp/445	Server Message Block	Windows file transfers and printer sharing
AFP	tcp/548	Apple Filing Protocol	Mac OS file transfers

Résolution de Noms

Serveur DNS

Traduire un nom de domaine (Domain Name) en une adresse IP.

Par exemple:

```
$ dig google.com

; <<>> DiG 9.10.3-P4-Debian <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35279
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             270     IN      A       216.58.204.110

;; Query time: 9 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:09 CET 2020
;; MSG SIZE  rcvd: 55
```

Serveur DNS

On peut faire cette traduction en IPV6 (record AAAA)

Par exemple:

```
$ dig AAAA google.com

; <<>> DiG 9.10.3-P4-Debian <<>> AAAA google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44141
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                    IN      AAAA

;; ANSWER SECTION:
google.com.             299     IN      AAAA    2a00:1450:4007:80f::200e

;; Query time: 10 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:39 CET 2020
;; MSG SIZE  rcvd: 67
```

Serveur DNS

13 Serveur racine (source wikipedia)

Lettre	adresse IPv4[9]	adresse IPv6[9]	Ancien nom	Société	Localisation	Logiciel
A[10]	198.41.0.4	2001:503:ba3e::2:30	ns.internic.net	VeriSign	trafic distribué par anycast	BIND
B[11]	199.9.14.201Notes 1	2001:500:200::b	ns1.isi.edu	Université de Californie du Sud	Marina Del Rey, Californie, États-Unis	BIND
C[13]	192.33.4.12[14]	2001:500:2::c	c.psi.net	Cogent Communications	trafic distribué par anycast	BIND
D[15]	199.7.91.13[16],Notes 2	2001:500:2d::d	terp.umd.edu	Université du Maryland	College Park, Maryland, États-Unis	BIND
E[17]	192.203.230.10[17]	2001:500:a8::e	ns.nasa.gov	NASA	Mountain View, Californie, États-Unis	BIND
F[18]	192.5.5.241[19]	2001:500:2f::f	ns.isc.org	Internet Systems Consortium	trafic distribué par anycast	BIND
G[20]	192.112.36.4Notes 3	2001:500:12::d0d	ns.nic.ddn.mil	Defense Information Systems Agency	trafic distribué par anycast	BIND
H[21]	198.97.190.53[22],Notes 4	2001:500:1::53	aos.arl.army.mil	United States Army Research Laboratory (en)	Aberdeen, Maryland, États-Unis	NSD
I[23]	192.36.148.17[24]	2001:7fe::53	nic.nordu.net	Autonomica (Netnod (en))	trafic distribué par anycast	BIND
J[25]	192.58.128.30[25],Notes 5	2001:503:c27::2:30		VeriSign	trafic distribué par anycast	BIND
K[27]	193.0.14.129[27]	2001:7fd::1		RIPE NCC	trafic distribué par anycast	BIND, NSD[27]
L[29]	199.7.83.42Notes 6	2001:500:3::42		ICANN	trafic distribué par anycast	NSD[29]
M[31]	202.12.27.33[32]	2001:dc3::35		WIDE Project (en)	trafic distribué par anycast	BIND

Serveur DNS

13 Serveur racine (source wikipedia)

gethostbyname

struct hostent *gethostbyname(const char *name);

The hostent structure is defined in <netdb.h> as follows:

```
struct hostent {
    char  *h_name;          /* official name of host */
    char **h_aliases;       /* alias list */
    int    h_addrtype;      /* host address type */
    int    h_length;        /* length of address */
    char **h_addr_list;     /* list of addresses */
}

#define h_addr h_addr_list[0] /* for backward compatibility */
```

The members of the hostent structure are:

h_name The official name of the host.

h_aliases
An array of alternative names for the host, terminated by a null pointer.

h_addrtype
The type of address; always AF_INET or AF_INET6 at present.

h_length
The length of the address in bytes.

h_addr_list
An array of pointers to network addresses for the host (in network byte order), terminated by a null pointer.

h_addr The first address in h_addr_list for backward compatibility.

gethostbyname

```c
#include <netdb.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>


int main(int argc, char **argv)
{
        if(argc < 2)
                return 1;

        struct hostent *ret = gethostbyname(argv[1]);

        if(!ret)
        {
                perror("gethostbyname");
                return 1;
        }


        printf("Host resolves to %s\n", ret->h_name);

        unsigned int i=0;
        while ( ret->h_addr_list[i] != NULL) {
                printf( "%s\n", inet_ntoa( *( struct in_addr*)( ret->h_addr_list[i])));
                i++;
        }


        return 0;
}
```

gethostbyname

```
$ ./a.out cnn.com
Host resolves to cnn.com
151.101.65.67
151.101.1.67
151.101.193.67
151.101.129.67
```

```
$ ./a.out elysee.fr
Host resolves to elysee.fr
45.60.151.214
45.60.155.214
```

```
$ ./a.out facebook.com
Host resolves to facebook.com
157.240.21.35
```

```
$ ./a.out www.uvsq.fr
Host resolves to preprod.uvsq.fr
193.51.33.8
```

UNIQUEMENT IPV4

getaddrinfo

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

int getaddrinfo(const char *node,     // Domaine par exemple google.com
                const char *service,  // Port ou service (80 ou www par exemple)
                  const struct addrinfo *hints, // Hints souvent un memset 0 de la struct
                  struct addrinfo **res); // Valeur de retour
```

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```

Convertir adresse DNS google.fr en:

IPV4 : 172.217.22.131

IPV6 : 2a00:1450:4007:815::2003

getaddrinfo

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```

Convertir adresse DNS google.fr en:

IPV4 : 216.58.215.35

IPV6 : 2a00:1450:4007:808::2003

Résolution de Nom DNS

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>

int main( int argc, char ** argv ) {
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    int ret = getaddrinfo(argv[1], argv[2],
            &hints,
            &res);
    if( ret < 0) {
        herror("getaddrinfo");
        return 1;
    }
    struct addrinfo *tmp;
    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        char ip[INET6_ADDRSTRLEN];
        ip[0] = '\0';
        if(tmp->ai_family == AF_INET) {
            struct sockaddr_in* saddr = (struct sockaddr_in*)tmp->ai_addr;
            inet_ntop(AF_INET, &saddr->sin_addr, ip, sizeof(ip));
            printf("IPV4 : %s\n", ip);
        }
        else if(tmp->ai_family == AF_INET6) {
            struct sockaddr_in6* saddr6 = (struct sockaddr_in6*)tmp->ai_addr;
            inet_ntop(AF_INET6, &saddr6->sin6_addr, ip, sizeof(ip));
            printf("IPV6 : %s\n", ip);
        }
    }
    return 0;
}
```

```c
hints.ai_family = AF_UNSPEC;
```

```
IPV4 : 172.217.22.131
IPV6 : 2a00:1450:4007:815::2003
```

```c
hints.ai_family = AF_INET;
```

```
IPV4 : 172.217.22.131
```

```c
hints.ai_family = AF_INET6;
```

```
IPV6 : 2a00:1450:4007:815::2003
```

getaddrinfo

```
$ ./getaddr cnn.com
IPV4 : 151.101.193.67
IPV4 : 151.101.129.67
IPV4 : 151.101.1.67
IPV4 : 151.101.65.67
IPV6 : 2a04:4e42:600::323
IPV6 : 2a04:4e42:400::323
IPV6 : 2a04:4e42::323
IPV6 : 2a04:4e42:200::323
```

```
$ ./getaddr google.com
IPV4 : 216.58.209.238
IPV6 : 2a00:1450:4007:812::200e
```

```
$ ./getaddr free.fr
IPV4 : 212.27.48.10
IPV6 : 2a01:e0c:1::1
```

```
$ ./getaddr facebook.com
IPV4 : 157.240.21.35
IPV6 : 2a03:2880:f130:83:face:b00c:0:25de
```

Créer un Socket

Créer un Socket TCP

Un socket est un descripteur de fichier correspondant à un flux réseau

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

int socket(int domain, int type, int protocol);
```

- Domain -> type de socket
  - ➡ AF_UNIX -> socket UNIX reposant sur un fichier (socket local)
  - ➡ AF_INET -> socket IPV4
  - ➡ AF_INET6 -> socket IPV6

- Type -> Mode de communications
  - ➡ SOCK_STREAM -> flux de donnée TCP
  - ➡ SOCK_DGRAM -> datagramme (UDP)
  - ➡ SOCK_RAW -> accès bas niveau à l'interface (uniquement root)

- Protocol -> Protocole à utiliser
  - ➡ En général on laisse 0

Etats d’un Socket

Connecter un Socket

Connect

```
NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Astuce : utiliser les valeurs de retour de gethostbyname !

Socket Client

Connect via Gethostbyname

```
/* Argument check */
if ( argc != 3 )
{
        return 1;
}

/* Name resolution */
struct hostent *server_info = gethostbyname( argv[1] );

if ( !server_info )
{
        herror( "gethostbyname" );
        return 1;
}

/* Create Socket */
int sock = socket( AF_INET, SOCK_STREAM, 0 );

if ( sock < 0 )
{
        perror( "socket" );
        return 1;
}

/* Configure client socket */
struct sockaddr_in server_conf;
/* Insert address family */
server_conf.sin_family = server_info->h_addrtype;
/* Insert PORT */
server_conf.sin_port = htons( atoi( argv[2] ) );
/* Copy destination addr from server_info */
memcpy( &server_conf.sin_addr, server_info->h_addr_list[0], server_info->h_length );

/* Connect the socket to the server */
if ( connect( sock, ( struct sockaddr * )&server_conf, sizeof( struct sockaddr_in ) ) < 0 )
{
        perror( "Connect" );
        return 1;
}
```

Connect via getaddrinfo

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main( int argc, char ** argv )
{
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    int ret = getaddrinfo(argv[1], argv[2],
            &hints,
            &res);

    if( ret < 0)
    {
        herror("getaddrinfo");
        return 1;
    }
```

Suite du code ici …

```c
    struct addrinfo *tmp;
    int sock = -1;
    int connected = 0;

    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        sock = socket(tmp->ai_family,
                      tmp->ai_socktype,
                      tmp->ai_protocol);
        if( sock < 0) {
            perror("sock");
            continue;
        }
        int ret = connect( sock, tmp->ai_addr,
                           tmp->ai_addrlen);
        if( ret < 0 ) {
            close(sock);
            perror("connect");
            continue;
        }
        connected = 1;
        break;
    }

    if(!connected) {
        fprintf(stderr, "Failed to connect to %s:%s\n »,
                        argv[1], argv[2]);
        return 1;
    }
    /* Use the socket */
    close(sock);
    return 0;
}
```