UNIX/Linux

Jouer avec le système

Olivier Thauvin

13 novembre 2008

1. Préambule
2. Les fichiers
3. Les processus
4. Executables : format
5. X11

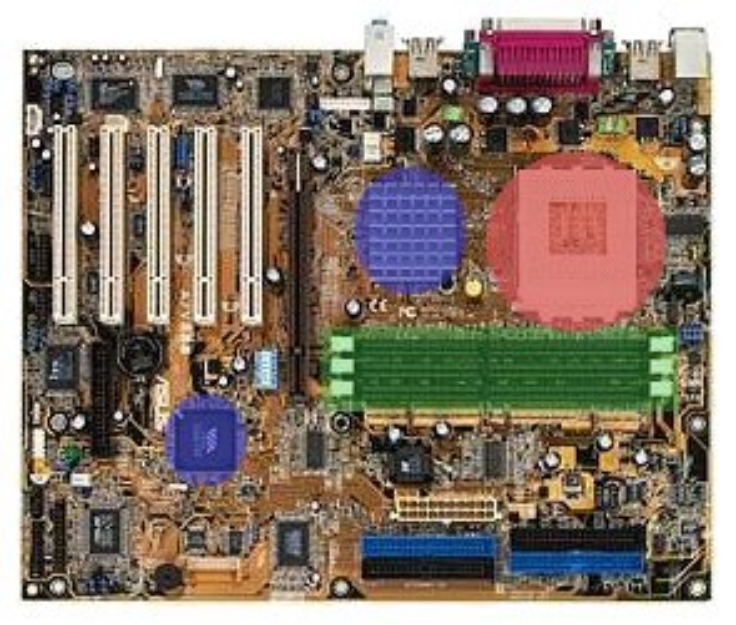
Plan

1. Préambule
   - L'Ordinateur
   - LES UNIX
   - GNU/Linux
2. Les fichiers
3. Les processus
4. Executables : format
5. X11

1 Préambule
- L'Ordinateur
- LES UNIX
- GNU/Linux

Principaux composants

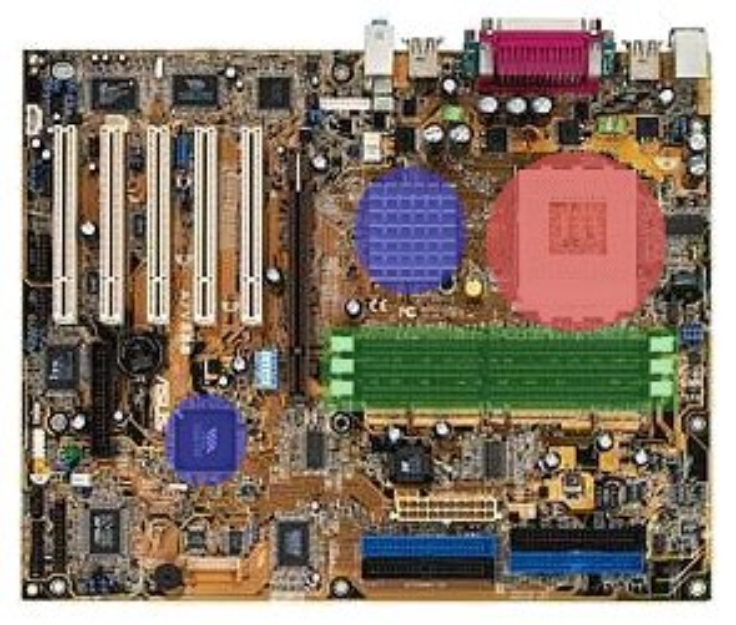
ASUS AV7333



- les éléments vitaux (PC)
  - CPU
  - Mémoire
  - Bus
  - clavier
  - carte graphique

Principaux composants

ASUS AV7333



- les éléments vitaux (PC)
  - CPU
  - Mémoire
  - Bus
  - clavier
  - carte graphique
- le reste
  - mulot et autres rongeurs
  - disques durs
  - carte son
  - ...

Fonctionnement général

Roles de chacun

- CPU (Central processor unit)
  - fait les calculs
  - presque pas de mémoire (registres)

Fonctionnement général

Roles de chacun

- CPU (Central processor unit)
  - fait les calculs
  - presque pas de mémoire (registres)
- Mémoire
  - sert à stocker les données
  - est moins rapide que le CPU (rapport ~ 1/3)

Fonctionnement général

Roles de chacun

- CPU (Central processor unit)
  - fait les calculs
  - presque pas de mémoire (registres)
- Mémoire
  - sert à stocker les données
  - est moins rapide que le CPU (rapport ~ 1/3)
- Bus
  - sur la carte mère
  - transporte les données entre les différents composants

1 Préambule
- L'Ordinateur
- LES UNIX
- GNU/Linux

la grande famille des UNIX

Définition Wikipédia

UNIX est le nom d'un système d'exploitation multitâche et multi-utilisateur créé en 1969, conceptuellement ouvert et fondé sur une approche par laquelle il offre de nombreux petits outils chacun dotés d'une mission spécifique.

la grande famille des UNIX

Définition Wikipédia

UNIX est le nom d'un système d'exploitation multitâche et multi-utilisateur créé en 1969, conceptuellement ouvert et fondé sur une approche par laquelle il offre de nombreux petits outils chacun dotés d'une mission spécifique.

Les grandes lignes :

- mulitâche préemptif
- multiutilisateurs (donc gestion de droit)
- multisessions
- tout est fichier

Vite une Norme !

Exemples d'UNIX :

- GNU/Linux
- *BSD (Net, Free, Open, . . .)
- MacOS X (est un BSD, d'Apple)
- Digital UNIX (de Digital)
- Solaris (de Sun)
- AIX (de IBM)
- . . .

Vite une Norme !

Exemples d'UNIX :

- GNU/Linux
- *BSD (Net, Free, Open, . . .)
- MacOS X (est un BSD, d'Apple)
- Digital UNIX (de Digital)
- Solaris (de Sun)
- AIX (de IBM)
- . . .

Norme POSIX :

standard de fonctionnement et commandes (IEEE).

1 Préambule
- L'Ordinateur
- LES UNIX
- GNU/Linux

GNU/Linux

Contenu du système

- Linux : le noyau lui même (kernel.org)
- des applis éparpillées sur internet

GNU/Linux

Contenu du système

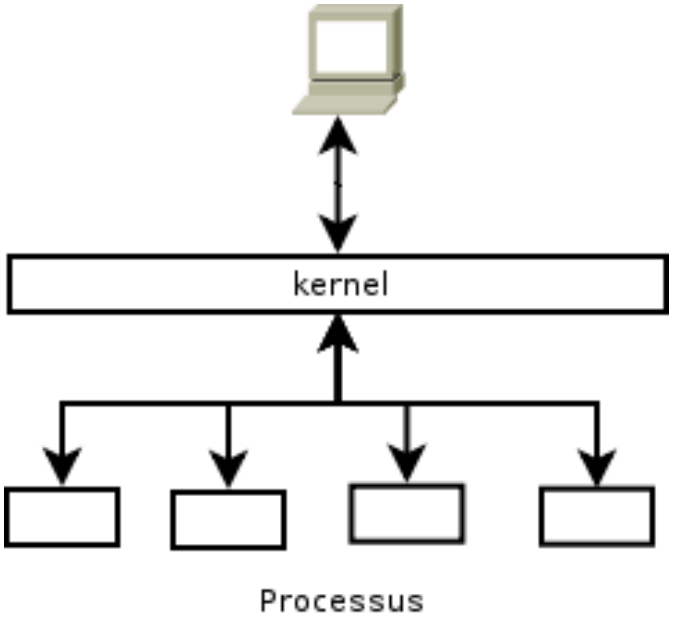
- Linux : le noyau lui même (kernel.org)
- des applis éparpillées sur internet

Les distributions

Ensemble cohérent noyau/applications

- Redhat : Mandriva Suse Centos ...
- Debian : ubuntu
- gentoo
- slackware
- ...

De l'ordinateur à l'utilisateur



Rôle du kernel

- gérer le matériel
- donner le temps CPU aux applications
- vérifier les droits d'accès
- assurer l'intégrité du système

Plan

1. Préambule
2. Les fichiers
   - Systèmes de fichiers
   - Les fichiers
3. Les processus
4. Executables : format
5. X11

2 Les fichiers
- Systèmes de fichiers
- Les fichiers

Vue globale

tree / -L 2 -d

```
/
|-- bin
|   |- ls
|   |- rm
|-- boot
|   `-- grub
|-- dev
|-- etc
|-- home
|   |-- olivier
|-- lib
|-- mnt
|-- net
|-- root
|-- sbin
|-- tmp
|-- usr
|   |-- bin
|   |-- include
|   |-- lib
`-- var
```

Organisation globale

L'emplacement des fichiers

`/home`	les données utilisateurs
`/etc`	fichiers de configuration
`/bin`, `/usr/bin`	les executables
`/sbin`, `/usr/sbin`	idem, réservés à root
`/lib`, `/usr/lib`	les librairies
`/usr/share`	les données statiques des programmes
`/usr/include`	les fichiers .h (C)
`/var`	les données variables (spool, etc...)
`/tmp`, `/var/tmp`	données temporaires
`/opt`	espace pour les apps. propriétaires

Organisation globale (2)

Système de fichiers spéciaux

`/dev`	les fichiers de périphériques
`/proc`	représentation temp réelle du système
`/sys`	configuration du noyau, informations sur le matériel

Organisation globale (2)

Système de fichiers spéciaux

`/dev`	les fichiers de périphériques
`/proc`	représentation temp réelle du système
`/sys`	configuration du noyau, informations sur le matériel

le FHS

Le Filesystem Hierarchy Standard normalise l'empacement des fichiers

principe

le montage

Le système de fichiers est stocké sur un support (disque, fichier, cdrom, ...)
Le noyau le fait apparaitre dans un répertoire.

la commande mount

```
[olivier@andromede unix]$ mount
/dev/sda3 on / type ext3 (rw,noatime)
/dev/sda8 on /home type ext3 (rw,noatime)
[...]
virgo:/home/data on /mnt/disk type nfs
    (ro,addr=192.168.76.1)
```

2 Les fichiers
- Systèmes de fichiers
- Les fichiers

Nom de fichiers

Accès aux fichiers

- le séparateur est /
- plusieurs / n'en font qu'un
- la longueur maximale d'un chemin est de 1024 octets (POSIX)
- tous les caractères sont permis sauf / et *NULL*

On évitera cependant

- l'espace (qu'il faut échaper sur la ligne de commande)
- – (surtout en premier, option en ligne de commande)
- ce qui ne se tape pas au clavier
- les accents (problème d'encodage)

les types de fichiers

ls -l

- rw-r–r– 1 olivier olivier 3049 17 :30 unix2.tex

les différents types

- les simples fichiers (-)
- les répertoires (d)

les types de fichiers

ls -l

- rw-r–r– 1 olivier olivier 3049 17 :30 unix2.tex

les différents types

- les simples fichiers (-)
- les répertoires (d)
- les liens symboliques (l)

les types de fichiers

ls -l

- rw-r–r– 1 olivier olivier 3049 17 :30 unix2.tex

les différents types

- les simples fichiers (-)
- les répertoires (d)
- les liens symboliques (l)
- les fichiers de périphérique (c ou b)

les types de fichiers

ls -l

- rw-r–r– 1 olivier olivier 3049 17 :30 unix2.tex

les différents types

- les simples fichiers (-)
- les répertoires (d)
- les liens symboliques (l)
- les fichiers de périphérique (c ou b)
- les fifo (tubes nommés, p)

les types de fichiers

ls -l

- rw-r–r– 1 olivier olivier 3049 17 :30 unix2.tex

les différents types

- les simples fichiers (-)
- les répertoires (d)
- les liens symboliques (l)
- les fichiers de périphérique (c ou b)
- les fifo (tubes nommés, p)
- les sockets (=)

. et ..

chaque répertoire contient :

- . le répertoire lui même
- . . son répertoire parent

. et ..

chaque répertoire contient :

- . le répertoire lui même
- .. son répertoire parent

trop de ..

/.. pointe sur /

```
$ ls ../../../../../../etc/passwd
../../../../../../etc/passwd
```

On rebondit sur / et on redescend dans /etc/passwd.

L'inode

Le fichier est matérialisé par une inode

```
[olivier@andromede unix]$ ls -li unix.tex
6160386 -rw-r--r-- 1 olivier olivier 3049 17:30 unix.tex
```

L'inode contient

- un numéro (6160386)
- des métadata
  - taille (3049)
  - propriétaires / droits d'accès
  - dates : d'accès (atime), création (ctime), modification (mtime)
- éventuellement un contenu

L'inode

Le fichier est matérialisé par une inode

```
[olivier@andromede unix]$ ls -li unix.tex
6160386 -rw-r--r-- 1 olivier olivier 3049 17:30 unix.tex
```

L'inode contient

- un numéro (6160386)
- des métadata
  - taille (3049)
  - propriétaires / droits d'accès
  - dates : d'accès (atime), création (ctime), modification (mtime)
- éventuellement un contenu

Créer un fichier créer l'inode
Effacer un fichier efface l'inode (si plus référencée)

quand ctime dépasse mtime

Copie en préservant les métadata :

```
[olivier@andromede unix]$ cp -a unix.tex unix2.tex
```

quand ctime dépasse mtime

Copie en préservant les métadata :

```
[olivier@andromede unix]$ cp -a unix.tex unix2.tex
```

Le fichier vient d'être créé :

```
[olivier@andromede unix]$ ls -l --time=c unix2.tex
-rw-r--r-- 1 olivier olivier 3049 18:47 unix2.tex
```

quand ctime dépasse mtime

Copie en préservant les métadata :

```
[olivier@andromede unix]$ cp -a unix.tex unix2.tex
```

Le fichier vient d'être créé :

```
[olivier@andromede unix]$ ls -l --time=c unix2.tex
-rw-r--r-- 1 olivier olivier 3049 18:47 unix2.tex
```

Pourtant la date de modification est antérieur :

```
[olivier@andromede unix]$ ls -l  unix2.tex
-rw-r--r-- 1 olivier olivier 3049 17:30 unix2.tex
```

existence d'un fichier

un fichier existe tant que

- l'inode est référencée par un nom (lien hard)
- l'inode est ouverte

existence d'un fichier

un fichier existe tant que

- l'inode est référencée par un nom (lien hard)
- l'inode est ouverte

liens hard

```
[olivier@andromede unix]$ ln fichier1 fichier2
[olivier@andromede unix]$ ls -li fichier*
6160393 -rw-r--r-- 2 olivier olivier 1101 19:54 fichier1
6160393 -rw-r--r-- 2 olivier olivier 1101 19:54 fichier2
[olivier@andromede unix]$ rm -f fichier1
[olivier@andromede unix]$ ls -li fichier*
6160393 -rw-r--r-- 1 olivier olivier 1101 19:54 fichier2
```

existance d'un fichier

deleteopen.pl

```
1   open(my $handle, '>', '/tmp/opened');
2   unlink('/tmp/opened');
    <STDIN>;
4   close($handle);
```

existance d'un fichier

deleteopen.pl

```
1   open(my $handle, '>', '/tmp/opened');
2   unlink('/tmp/opened');
    <STDIN>;
4   close($handle);
```

Conséquences :

- le fichier n'est accessible que par l'application
- si l'application quitte : l'espace disque est récupéré
- la fermeture du descripteur entraine l'effacement du fichier

existance d'un fichier

deleteopen.pl

```
1   open(my $handle, '>', '/tmp/opened');
2   unlink('/tmp/opened');
    <STDIN>;
4   close($handle);
```

Conséquences :

- le fichier n'est accessible que par l'application
- si l'application quitte : l'espace disque est récupéré
- la fermeture du descripteur entraine l'effacement du fichier

```
perl 23536 ol 3w REG  8,7  0  49065 /tmp/opened (deleted)
```

Utile pour les fichiers temporaires.

Fichiers temporaires

Un fichier qu'on ne va pas garder, mais :

- il ne doit pas exister

Fichiers temporaires

Un fichier qu'on ne va pas garder, mais :

- il ne doit pas exister
- nom aléatoire (écrasement basé sur un lien symbolique)

Fichiers temporaires

Un fichier qu'on ne va pas garder, mais :

- il ne doit pas exister
- nom aléatoire (écrasement basé sur un lien symbolique)
- le placer au bon endroit

Fichiers temporaires

Un fichier qu'on ne va pas garder, mais :

- il ne doit pas exister
- nom aléatoire (écrasement basé sur un lien symbolique)
- le placer au bon endroit

en pratique

- shell : mktemp
- C : mkstemp
- Perl : module File : :Temp

Fichiers temporaires

Un fichier qu'on ne va pas garder, mais :

- il ne doit pas exister
- nom aléatoire (écrasement basé sur un lien symbolique)
- le placer au bon endroit

en pratique

- shell : mktemp
- C : mkstemp
- Perl : module File : :Temp

```
1  FILE=`mktemp /tmp/nanar.XXXXXXX`
2  echo 'toto' > $FILE
3  rm $FILE
```

les fichiers à trous (sparse files)

théorie

- certaines données n'existent pas
- les trous ne prennent pas de place
- le système renvoient des 0 en cas de lecture
- le fichier peut être plus gros que le système de fichiers

les fichiers à trous (sparse files)

théorie

- certaines données n'existent pas
- les trous ne prennent pas de place
- le système renvoient des 0 en cas de lecture
- le fichier peut être plus gros que le système de fichiers

utilisation ?

Chaque fois que des données seront allouées à des emplacements dans le fichier :

- disques virtuels
- bases de données

fichiers à trous : exemple

sparce.c

```
1   #include <stdio.h>
2
3   int main(void) {
4       FILE * foo = fopen("/tmp/sparsed", "w+");
5       fseek(foo, 1024*1024, SEEK_SET);
6       fprintf(foo, "%s", "coucou");
7       fclose(foo);
8   }

[olivier@andromede unix]$ ls -ls sparsed
8 -rw-rw-r-- 1 olivier olivier 1048582 02:57 sparsed
```

Liens symboliques

Fichier qui pointe sur un autre :

- gérés le système
- il ne contient que le nom de l'élément pointé

Liens symboliques

Fichier qui pointe sur un autre :

- gérés le système
- il ne contient que le nom de l'élément pointé
  - l'élément pointer peut être n'importe quoi
  - l'élément peut ne pas existé (lien cassé)

Liens symboliques

Fichier qui pointe sur un autre :

- gérés le système
- il ne contient que le nom de l'élément pointé
  - l'élément pointer peut être n'importe quoi
  - l'élément peut ne pas existé (lien cassé)
- les actions s'appliquent à l'élément pointé sauf :
  - ls/rm
  - les appels systèmes prévus

Liens symboliques

Fichier qui pointe sur un autre :

- gérés le système
- il ne contient que le nom de l'élément pointé
  - l'élément pointer peut être n'importe quoi
  - l'élément peut ne pas existé (lien cassé)
- les actions s'appliquent à l'élément pointé sauf :
  - ls/rm
  - les appels systèmes prévus

exemple

```
$ ln -s  fichier lien
$ ls -l lien
lrwxrwxrwx 1 olivier olivier 7 19:13 lien -> fichier
$ ls -l lien fichier
ls: ne peut accéder fichier: Aucun fichier ou dossier de ce type
lrwxrwxrwx 1 olivier olivier 7 19:13 lien -> fichier
$ echo 'coucou' > lien
$ ls -l lien fichier
-rw-rw-r-- 1 olivier olivier 7 19:14 fichier
lrwxrwxrwx 1 olivier olivier 7 19:13 lien -> fichier
```

Fichier de périphérique

2 types

- block (b)
- caractères (c)

Fichier de périphérique

2 types

- block (b)
- caractères (c)

Tout est fichier

```
brw-rw---- 1 nanardon cdwriter 11, 0 12:26 /dev/sr0
```

C'est le premier lecteur CDROM SCSI (SATA en réalité)

Fichier de périphérique

2 types

- block (b)
- caractères (c)

Tout est fichier

```
brw-rw---- 1 nanardon cdwriter 11, 0 12:26 /dev/sr0
```

C'est le premier lecteur CDROM SCSI (SATA en réalité)
Je peux le lire comme un fichier :

```
$ dd if=/dev/sr0 bs=1024 count=20 | wc
20+0 enregistrements lus
20+0 enregistrements écrits
20480 octets (20 kB) copiés, 0,803619 s, 25,5 kB/s
      0       0   20480
```

les fifos

Première console :

```
$ mkfifo /tmp/fifo
$ cat /etc/passwd > /tmp/fifo
```

les fifos

Première console :

```
$ mkfifo /tmp/fifo
$ cat /etc/passwd > /tmp/fifo
```

Deuxième console :

```
$ grep root < /tmp/fifo
root:x:0:0:root:/root:/bin/bash
```

les fifos

Première console :

```
$ mkfifo /tmp/fifo
$ cat /etc/passwd > /tmp/fifo
```

Deuxième console :

```
$ grep root < /tmp/fifo
root:x:0:0:root:/root:/bin/bash
```

Cet exemple simple revient à

```
cat /etc/passwd | grep root
```

Plan

1. Préambule
2. Les fichiers
3. Les processus
   - le temps CPU
   - La ram
   - plusieurs processus : sur 1 ordinateur
   - Communication inter-processus
4. Executables : format
5. X11

3 Les processus

- le temps CPU
- La ram
- plusieurs processus : sur 1 ordinateur
- Communication inter-processus

scheduler préemptif

préemptif ?

Seul le noyau décide du temps CPU aloué !

scheduler préemptif

préemptif ?

Seul le noyau décide du temps CPU aloué !

Le scheduler

Le scheduler est appelé à interval régulier (plusieurs centaintes de fois par seconde) pour donner du temps CPU à l'application sauf si :

- l'application ne demande pas de temps CPU (sleep)
- est en attente d'une réponse matériel (IO wait)

Le temps CPU est donné en fonction de la priorité

3 Les processus

- le temps CPU
- La ram
- plusieurs processus : sur 1 ordinateur
- Communication inter-processus

La RAM

le contenu

- Mémoire vérouillée :
  - le BIOS (programme interne du PC)
  - mémoire du matériel
  - le code et données du noyau

La RAM

le contenu

- Mémoire vérouillée :
  - le BIOS (programme interne du PC)
  - mémoire du matériel
  - le code et données du noyau
- Mémoire haute (swappable) :
  - code des applications
  - données des applications

La RAM

le contenu

- Mémoire vérouillée :
  - le BIOS (programme interne du PC)
  - mémoire du matériel
  - le code et données du noyau
- Mémoire haute (swappable) :
  - code des applications
  - données des applications
- Mémoire vive libre :
  - buffers
  - cache (disque)

La RAM

le contenu

- Mémoire vérouillée :
  - le BIOS (programme interne du PC)
  - mémoire du matériel
  - le code et données du noyau
- Mémoire haute (swappable) :
  - code des applications
  - données des applications
- Mémoire vive libre :
  - buffers
  - cache (disque)
- swap (sur disque, très lente)

Gestion

la swap ?

Extension sur disque de la mémoire.

Gestion

la swap ?

Extension sur disque de la mémoire.
Elle sert :

- quand la mémoire vive est pleine
- pour mettre de coté des données peu utilisées

Mémoire disponible = mémoire vive + swap

Gestion

la swap ?

Extension sur disque de la mémoire.
Elle sert :

- quand la mémoire vive est pleine
- pour mettre de coté des données peu utilisées

Mémoire disponible = mémoire vive + swap

Un système qui swap **un peu** n'est pas surchargé !

Gestion

la swap ?

Extension sur disque de la mémoire.
Elle sert :

- quand la mémoire vive est pleine
- pour mettre de coté des données peu utilisées

Mémoire disponible = mémoire vive + swap

Un système qui swap **un peu** n'est pas surchargé !

La Ram est virtuelle

De par les aller retour RAM-SWAP, et la fragmentation, le système virutalize les adresses allouées.

Allocation par segment

Un très mauvais code

```
1   #include <string.h>
2   int main(void) {
3       char * cp = malloc(10);
4       char * buf = malloc(1);
5       strncpy(cp, buf, 3);
6   }
```

Et ça ne plante pas !

Allocation par segment

Un très mauvais code

```
1   #include <string.h>
2   int main(void) {
3       char * cp = malloc(10);
4       char * buf = malloc(1);
5       strncpy(cp, buf, 3);
6   }
```

Et ça ne plante pas !

Explication

Le système n'alloue pas des octets mais des segments
Le système n'intervient que si l'application sort du segment !

Mémoire à crédit

malloc.c

```
1   #include <stdlib.h>
2   #include <sdtio.h>
3   int main(void) {
4       char * buf = malloc(1000000000);
5       system("ps aux | grep malloc");
6       memset(buf, 'X', 1000000000);
7       system("ps aux | grep malloc");
8   }
```

Mémoire à crédit

malloc.c

```
1   #include <stdlib.h>
2   #include <sdtio.h>
3   int main(void) {
4       char * buf = malloc(1000000000);
5       system("ps aux | grep malloc");
6       memset(buf, 'X', 1000000000);
7       system("ps aux | grep malloc");
8   }
```

sortie formatée

Mem virtuel	Mem résidente
978116	332
978116	976888

Mémoire à crédit

malloc.c

```
1   #include <stdlib.h>
2   #include <sdtio.h>
3   int main(void) {
4       char * buf = malloc(1000000000);
5       system("ps aux | grep malloc");
6       memset(buf, 'X', 1000000000);
7       system("ps aux | grep malloc");
8   }
```

sortie formatée

Mem virtuel	Mem résidente
978116	332
978116	976888

malloc est une promesse

- malloc promet la ram
- la ram n'est allouée qu'à l'écriture

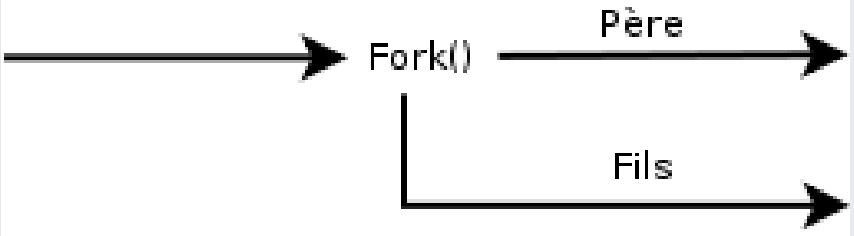
3 Les processus

- le temps CPU
- La ram
- plusieurs processus : sur 1 ordinateur
- Communication inter-processus

fork()

la fourchette

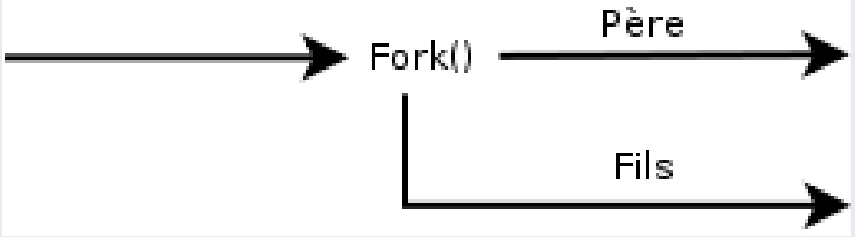
Permet à un processus de se dupliquer, et de donner naissance à un processus "fils" **indépendant**



fork()

la fourchette

Permet à un processus de se dupliquer, et de donner naissance à un processus "fils" **indépendant**



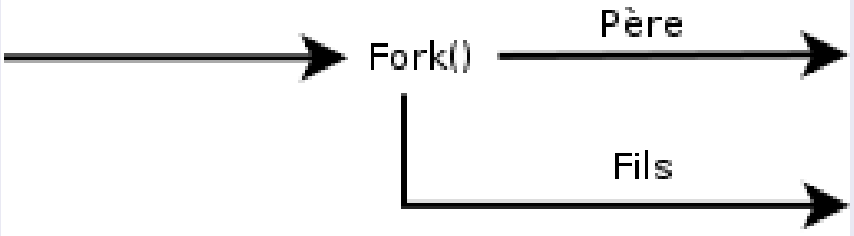
motivation

- excution parallèle
- perte de privilège de fils
- préserver l'environnement du père
- serveur : remettre le père en attente

fork()

la fourchette

Permet à un processus de se dupliquer, et de donner naissance à un processus "fils" **indépendant**



motivation

- excution parallèle
- perte de privilège de fils
- préserver l'environnement du père
- serveur : remettre le père en attente

pros & cons

- existe sur tout les UNIX (glibc)
- simple
- pas de communication

fork et la RAM

fork.c

```
1   #include <unistd.h>
    #define MEMSIZE 1000000000
    int main(void) {
        pid_t pid = 0; int status = 0;
5       char * ptr = malloc(MEMSIZE);
        system("free");
        memset(ptr, 'X', MEMSIZE);
        system("free");
        pid = fork();
10      if (pid < -1) exit(1);
        else if (pid == 0) {
            sleep(10); /* child */
            exit(0);
        } else {
15      sleep(1);
        system("free");
        wait(&status);
        return(0);
        }
20  }
```

fork et la RAM

fork.c

```
1   #include <unistd.h>
    #define MEMSIZE 1000000000
    int main(void) {
        pid_t pid = 0; int status = 0;
5       char * ptr = malloc(MEMSIZE);
        system("free");
        memset(ptr, 'X', MEMSIZE);
        system("free");
        pid = fork();
10      if (pid < -1) exit(1);
        else if (pid == 0) {
            sleep(10); /* child */
            exit(0);
        } else {
15      sleep(1);
        system("free");
        wait(&status);
        return(0);
        }
20  }
```

Ram utilisée

	mem utilisée
Avant allocation	1687848
Après allocation	2664948
Après fork()	2666332

les threads

Sépare un processus en deux

- partage de la mémoire
  - tout n'est pas forcément partagé
  - gestion de lock pour les écriture
- le fonctionnement ressemble à fork mais
  - programmation et gestion différentes
  - complexe mais éfficace

les threads

Sépare un processus en deux

- partage de la mémoire
  - tout n'est pas forcément partagé
  - gestion de lock pour les écriture
- le fonctionnement ressemble à fork mais
  - programmation et gestion différentes
  - complexe mais éfficace

Linux : libpthread (glibc) POSIX

Je ne garantie pas que tout les UNIX fournissent le support des threads

les threads

Sépare un processus en deux

- partage de la mémoire
  - tout n'est pas forcément partagé
  - gestion de lock pour les écriture
- le fonctionnement ressemble à fork mais
  - programmation et gestion différentes
  - complexe mais éfficace

Linux : libpthread (glibc) POSIX

Je ne garantie pas que tout les UNIX fournissent le support des threads

```
[olivier@andromede unix]$ ps aux | grep transcode
PID %CPU %MEM    VSZ   RSS  TIME  COMMAND
3477  124  1.3 119972 41344 36:32 transcode
```

3 Les processus

- le temps CPU
- La ram
- plusieurs processus : sur 1 ordinateur
- Communication inter-processus

Pipe

points à retenir

- flux de donnée continu unique
- sens unique
- bloquants

Pipe

points à retenir

- flux de donnée continu unique
- sens unique
- bloquants

Utilisation

- `fork()`/`pipe()` (`p1 | p2`)
- avec tube nommé (`open`+`write` / `open`+`read`)

IPC/SHM, la boite aux lettres système

fonctions

- file d'attente de message
- zone de mémoire partagée

IPC/SHM, la boite aux lettres système

fonctions

- file d'attente de message
- zone de mémoire partagée

+ et -

- relativement simple
- très performant
- taille allouable limitée (protection du kernel, configurable)
- protection des données par gestion de droit

IPC/SHM, la boite aux lettres système

fonctions

- file d'attente de message
- zone de mémoire partagée

+ et -

- relativement simple
- très performant
- taille allouable limitée (protection du kernel, configurable)
- protection des données par gestion de droit

La doc (pour le C)

man shmat, shmdt, shmop, shm_overview, shmctl, shmget, shm_open, shm_unlink

Socket, Client/Serveur

principe

- un serveur attends une connection
- un client se connecte
- les deux échangent des données

Socket, Client/Serveur

principe

- un serveur attends une connection
- un client se connecte
- les deux échangent des données

modes

- écoute sur le réseau
  - definir un port d'écoute
  - savoir quelle(s) machine(s) contacter
  - communication inter-machines
  - problème de latence

Socket, Client/Serveur

principe

- un serveur attends une connection
- un client se connecte
- les deux échangent des données

modes

- écoute sur le réseau
  - definir un port d'écoute
  - savoir quelle(s) machine(s) contacter
  - communication inter-machines
  - problème de latence
- écoute sur un socket (fichier de type . . .)
  - communication locale
  - restriction d'accès par gestion de droit

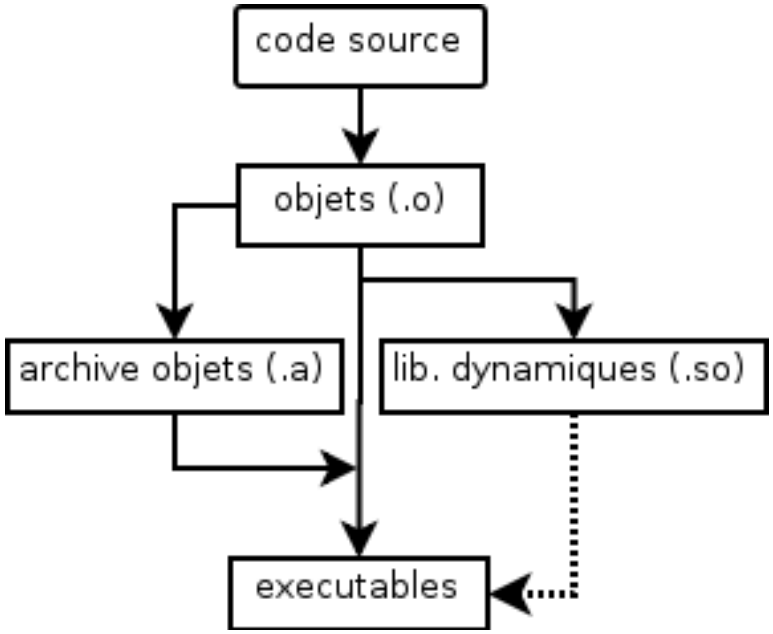
Plan

1. Préambule
2. Les fichiers
3. Les processus
4. Executables : format
   - Code exécutable
   - Bibliothèques
5. X11

4 Executables : format

- Code exécutable
- Bibliothèques

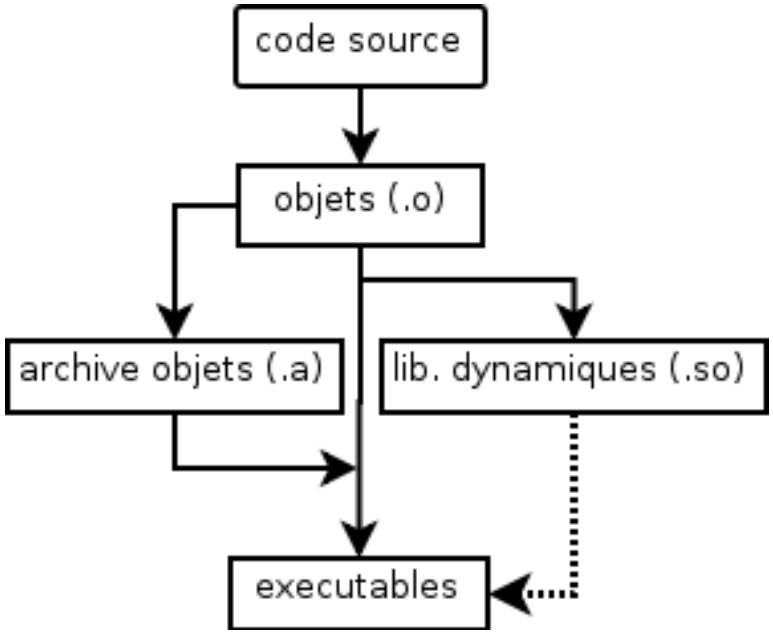
la logique



un quoi ?

- données formatées (format ELF ces jours-ci)

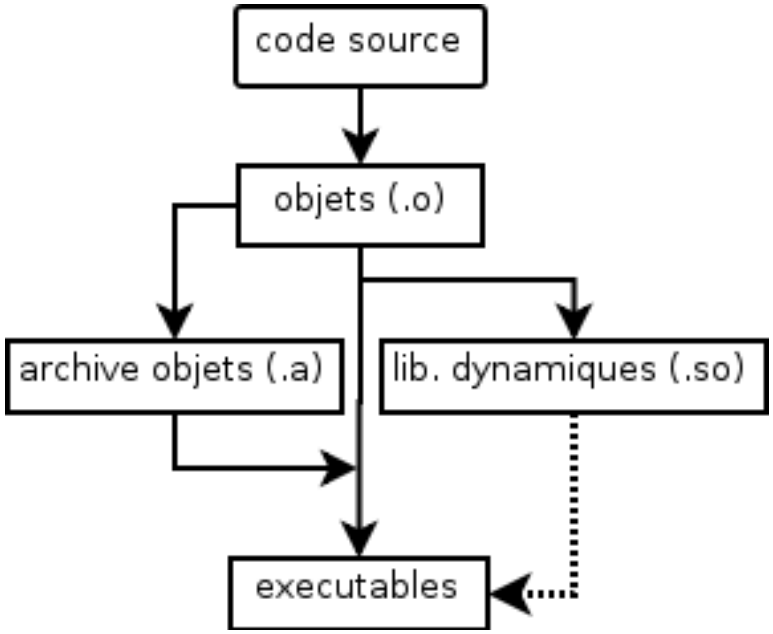
la logique



un quoi ?

- données formatées (format ELF ces jours-ci)
- executable (ou presque) par le processeur

la logique



un quoi ?

- données formatées (format ELF ces jours-ci)
- executable (ou presque) par le processeur
- contient des objets :
  - des variables (globales)
  - des fonctions
  - d'autres objets à trouver

4 Executables : format

- Code exécutable
- Bibliothèques

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation
- libfoo.so.1 : la librairie utilisée, versionnée

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation
- libfoo.so.1 : la librairie utilisée, versionnée
- libfoo.so : (lien) pour l'éditeur de lien

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation
- libfoo.so.1 : la librairie utilisée, versionnée
- libfoo.so : (lien) pour l'éditeur de lien
- libfoo.a : la version statique

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation
- libfoo.so.1 : la librairie utilisée, versionnée
- libfoo.so : (lien) pour l'éditeur de lien
- libfoo.a : la version statique

le point sur les dynamiques

- avantages :
  - les maj sont répercutés sans recompilation des applis
  - factorisation du code dans la RAM et sur le disque

Le standard

Normallement un programme fournit

- foo.h : l'entête pour la compilation
- libfoo.so.1 : la librairie utilisée, versionnée
- libfoo.so : (lien) pour l'éditeur de lien
- libfoo.a : la version statique

le point sur les dynamiques

- avantages :
  - les maj sont répercutés sans recompilation des applis
  - factorisation du code dans la RAM et sur le disque
- inconvénients
  - doit être disponible lors de l'éxecution

Trouver les dynamiques

la recherche

- codé dans l'executable (*rpath*)

Trouver les dynamiques

la recherche

- codé dans l'executable (*rpath*)
- codé en dur : `/lib(64)`, `/usr/lib(64)`

Trouver les dynamiques

la recherche

- codé dans l'executable (*rpath*)
- codé en dur : `/lib(64)`, `/usr/lib(64)`
- indiqués dans `/etc/ld.so.conf`

Trouver les dynamiques

la recherche

- codé dans l'executable (*rpath*)
- codé en dur : `/lib(64)`, `/usr/lib(64)`
- indiqués dans `/etc/ld.so.conf`

ldd

```
[olivier@andromede unix]$ ldd /bin/rm
        linux-gate.so.1 =>  (0xffffe000)
        libc.so.6 => /lib/i686/libc.so.6 (0xb7ee3000)
        /lib/ld-linux.so.2 (0xb8064000)
```

Trouver les dynamiques

la recherche

- codé dans l'executable (*rpath*)
- codé en dur : `/lib(64)`, `/usr/lib(64)`
- indiqués dans `/etc/ld.so.conf`

ldd

```
[olivier@andromede unix]$ ldd /bin/rm
        linux-gate.so.1 =>  (0xffffe000)
        libc.so.6 => /lib/i686/libc.so.6 (0xb7ee3000)
        /lib/ld-linux.so.2 (0xb8064000)
```

linux-gate.so.1

Code injecté directement par le kernel, ignorez le !

l'exemple : la librairie

libsmall.h

```
1   int func(void);
```

libsmall.c

```
1   #include "libsmall.h"
2   int func(void) {
3       return(0);
4   }
```

l'exemple : la librairie

libsmall.h

```
1   int func(void);
```

libsmall.c

```
1   #include "libsmall.h"
2   int func(void) {
3       return(0);
4   }
```

la version statique

```
gcc -c libsmall.o libsmall.c
ar rcs libsmall.a libsmall.o
```

l'exemple : la librairie

libsmall.h

```
1   int func(void);
```

libsmall.c

```
1   #include "libsmall.h"
2   int func(void) {
3       return(0);
4   }
```

la version statique

```
gcc -c libsmall.o libsmall.c
ar rcs libsmall.a libsmall.o
```

la version dynamique

```
gcc -c libsmall.o libsmall.c
gcc -shared -o libsmall.so libsmall.o
```

l'exemple : le programme

small.c

```
1   #include "libsmall.h"
2   int main(void) {
3       return(func());
4   }
```

l'exemple : le programme

small.c

```
1   #include "libsmall.h"
2   int main(void) {
3       return(func());
4   }
```

la version statique

```
gcc -c -o small.o small.c
gcc -o small.static libsmall.a small.o
```

l'exemple : le programme

small.c

```
1   #include "libsmall.h"
2   int main(void) {
3       return(func());
4   }
```

la version statique

```
gcc -c -o small.o small.c
gcc -o small.static libsmall.a small.o
```

la version dynamique

```
gcc -c -o small.o libsmall.c
gcc -o small.dyn -L. -lsmall small.o
    -Wl,-rpath=$(pwd) # $(PWD) pour le chemin courant
```

Et puis il y a *dlopen()*

le mode standard dynamique

Un code lié à une bibliothèque ne fonctionnera jamais sans.

Et puis il y a *dlopen()*

le mode standard dynamique

Un code lié à une bibliothèque ne fonctionnera jamais sans.

les deux clefs :

- *dlopen()* charge une bibliotèque
- *dlsym()* va chercher les fonctions

Et puis il y a *dlopen()*

le mode standard dynamique

Un code lié à une bibliothèque ne fonctionnera jamais sans.

les deux clefs :

- *dlopen()* charge une bibliotèque
- *dlsym()* va chercher les fonctions

exemple d'utilisation

- pour étendre les fonctionnalités d'une appli (perl, python, apache)
- pour eviter de charger du code inutilement

Plan

1. Préambule
2. Les fichiers
3. Les processus
4. Executables : format
5. X11

Histoire

- **1984 création par le MIT**
- X est le successeur de W (sur system V)
- 1985 ajout de la couleur (DEC VAXstation)
- 1986 X10R2, X10R3, X10R4
- 1987 X11, 1988 X11R2, 1991 X11R5
- **1992 création de XFree86 (utilisé par Linux)**
- 1999 X11R6.5
- 2004 Xorg se sépare de XFree86 (pb de licence)
- **depuis tout les UNIX libres utilisent Xorg**

Histoire

- **1984 création par le MIT**
- X est le successeur de W (sur system V)
- 1985 ajout de la couleur (DEC VAXstation)
- 1986 X10R2, X10R3, X10R4
- 1987 X11, 1988 X11R2, 1991 X11R5
- **1992 création de XFree86 (utilisé par Linux)**
- 1999 X11R6.5
- 2004 Xorg se sépare de XFree86 (pb de licence)
- **depuis tout les UNIX libres utilisent Xorg**

X ça pue !

Oui, c'est le prix de la compatibilité...

X : serveur d'affichage

- gestionnaire de matériel
  - carte graphique
  - ecran
  - clavier
  - souris (et autres rongeurs)

X : serveur d'affichage

- gestionnaire de matériel
  - carte graphique
  - ecran
  - clavier
  - souris (et autres rongeurs)
- protocol d'affichage (reseau)

X : serveur d'affichage

- gestionnaire de matériel
  - carte graphique
  - ecran
  - clavier
  - souris (et autres rongeurs)
- protocol d'affichage (reseau)
- **serveur d'affichage**

X : serveur d'affichage

- gestionnaire de matériel
  - carte graphique
  - ecran
  - clavier
  - souris (et autres rongeurs)
- protocol d'affichage (reseau)
- **serveur d'affichage**

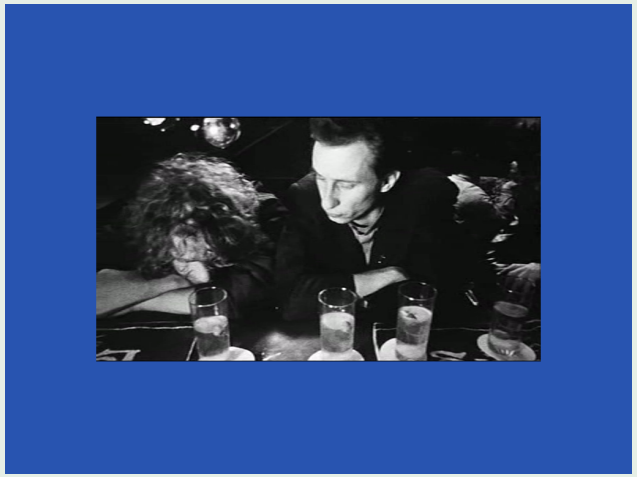
le serveur X, c'est ça :

Applications X11

Une application X11 se connecte au serveur X, et demande un affichage.

Applications X11

Une application X11 se connecte au serveur X, et demande un affichage.



Applications X11

Une application X11 se connecte au serveur X, et demande un affichage.

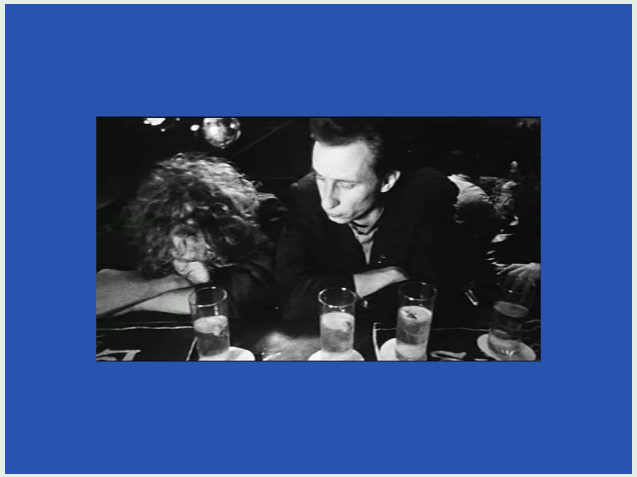
le Window Manager

- application X11
- il créé les fenetres
- fourni un menu (éventuellement)

KDE, GNOME, XFCE : envirronements de bureau

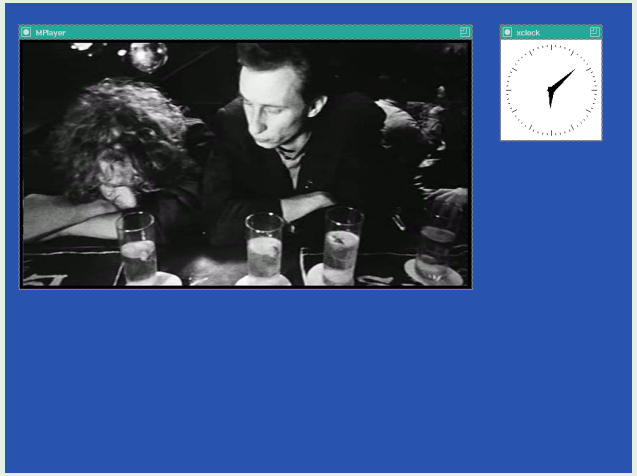
Applications X11

Une application X11 se connecte au serveur X, et demande un affichage.



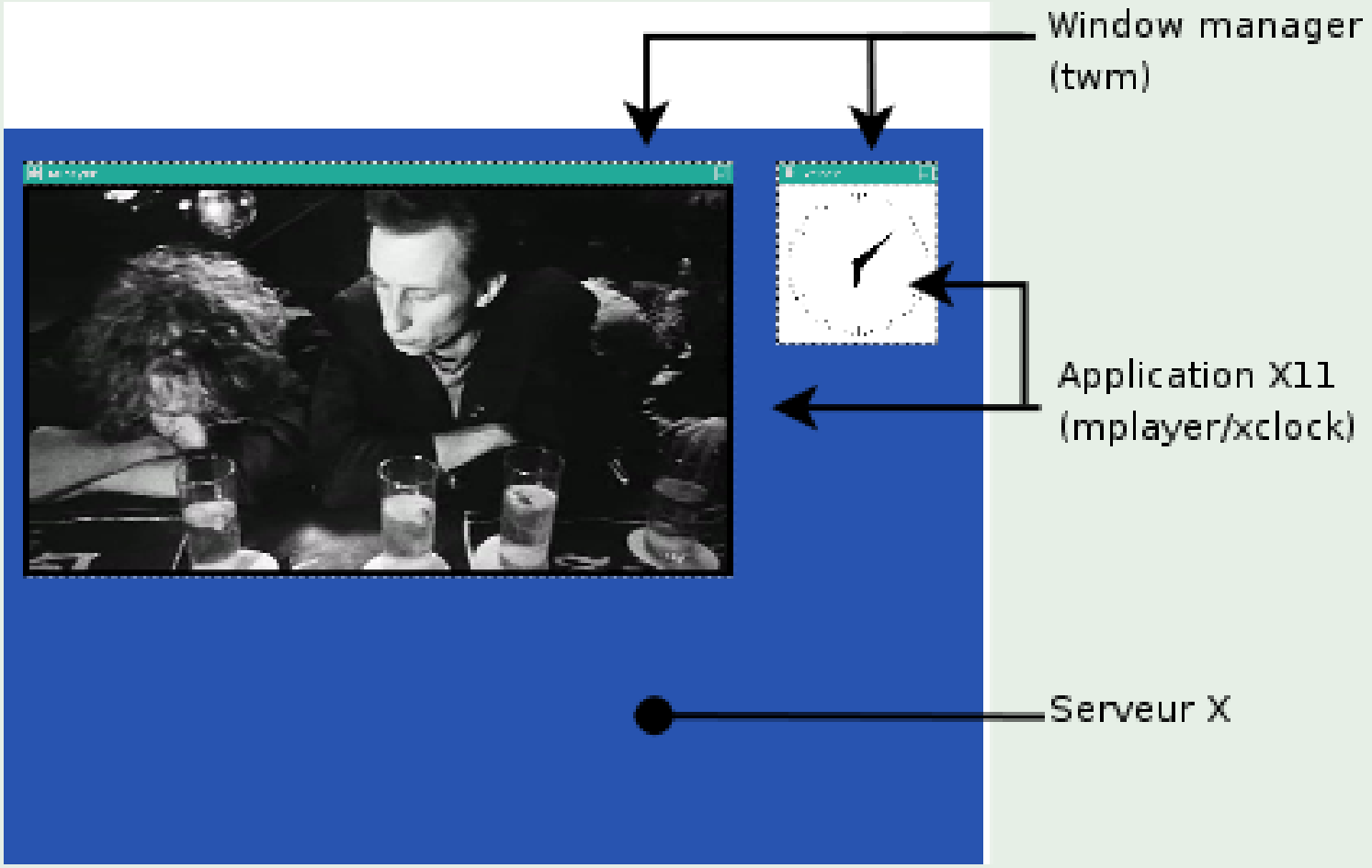
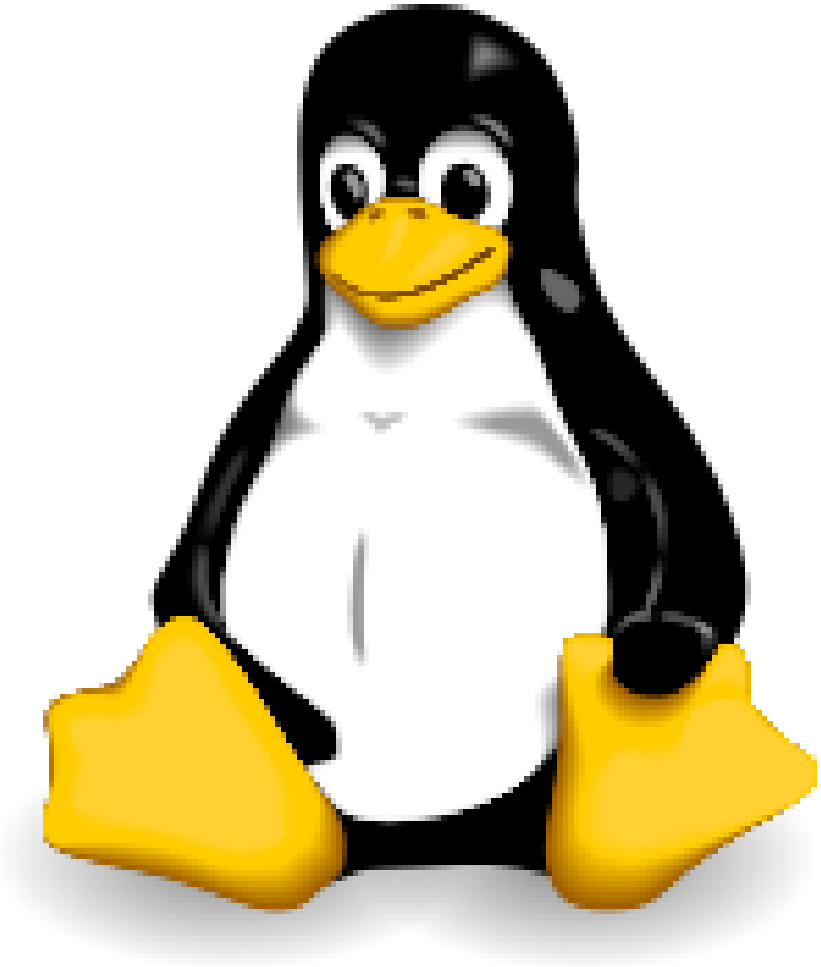
le Window Manager

- application X11
- il créé les fenetres
- fourni un menu (éventuellement)



KDE, GNOME, XFCE : envirronements de bureau

X11 : résumé


Window manager
(twm)
Application X11
(mplayer/xclock)
Serveur X

Des questions