01 - Introduction

Gaétan Richard
gaetan.richard@unicaen.fr

L2 – S4 2010/2011

0. Administratif

Équipe pédagogique

Membres :

- Gaétan Richard *(responsable – S3-354)* ;
- Mariya Georgieva ;
- Abir Karami ;
- Thu Hien Nguyen Thi.

Nous contacter :

`prenom.nom@unicaen.fr`

Séances :

- 9 séances de Cours ;
- 12 séances de TD ;
- 10 séances de TP.

Emploi du temps

Cours :

Le mardi 15h00 – 17h00 (GR – S3 057) ;

Groupe 1 et 2 :

(étudiants info suivant l'option logique séquentielle)

TD jeudi 13h45 à 15h15 (Thu Hien Nguyen Thi – S1 113) ;

TP 1 jeudi 16h30 à 18h00 (Thu Hien Nguyen Thi – S1 128) ;

TP 2 vendredi 9h00 à 10h30 (Mariya Georgieva – S1 127).

Groupe 3 :

(autres étudiants info et étudiants maths)

TD vendredi 14h00 – 15h30 (Abir Karami – S3 137) ;

TP mardi 16h00 – 17h30 (Abir Karami – S1 126) ;

Modalités de contrôle de connaissances

Épreuves :

- Un Projet (CC) ;
- Un Examen (E).

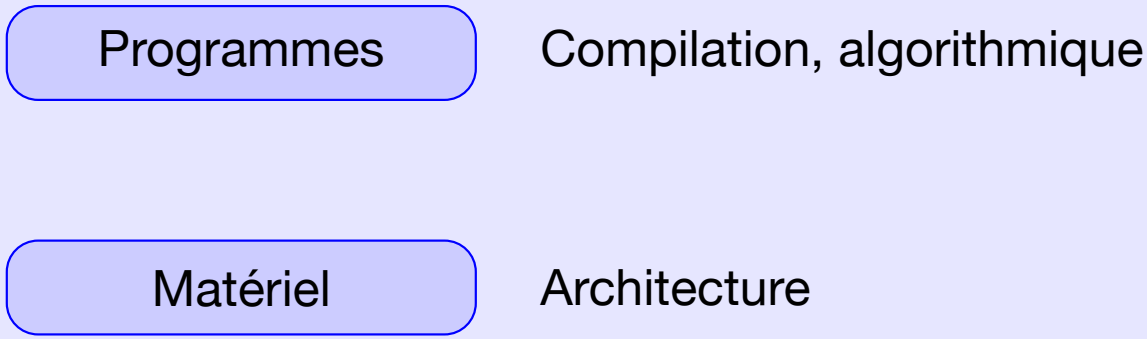
Note :

- Note finale du module : $F = \frac{1}{3}CC + \frac{2}{3}E$.

1. Système d'exploitation

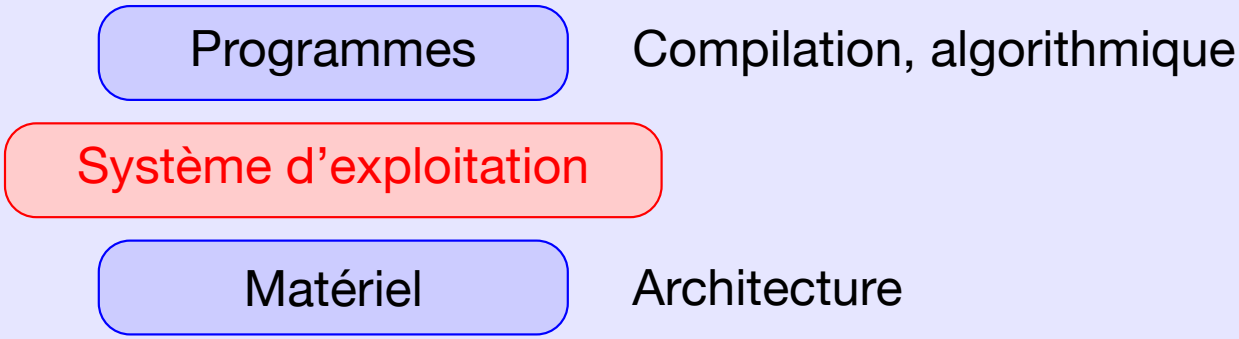
Le programme unique

Principe : couche entre le matériel et les programmes utilisateurs.



Le programme unique

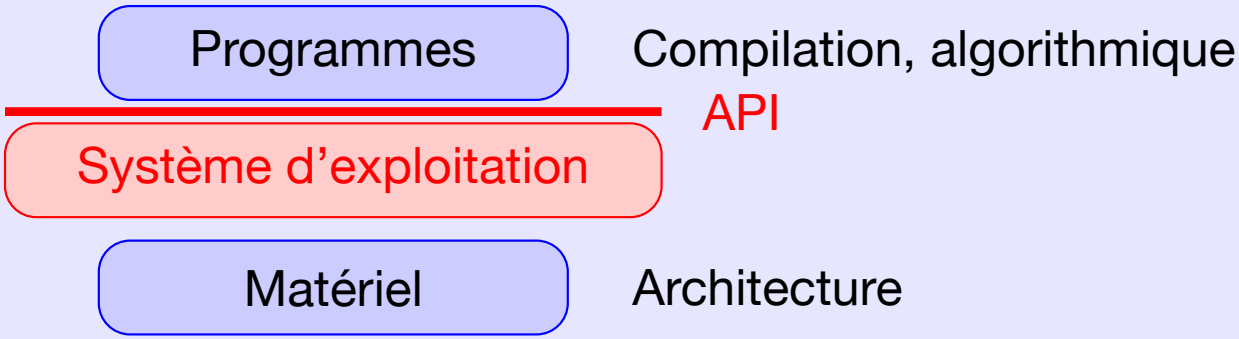
Principe : couche entre le matériel et les programmes utilisateurs.



Le programme unique

Principe : couche entre le matériel et les programmes utilisateurs.

Dans ce cours : aspects utilisation



Problématiques

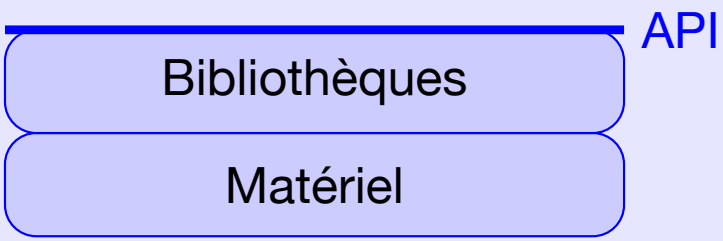
Rôle d'un système d'exploitation :

- Gérer le matériel ;

Problématiques

Rôle d'un système d'exploitation :

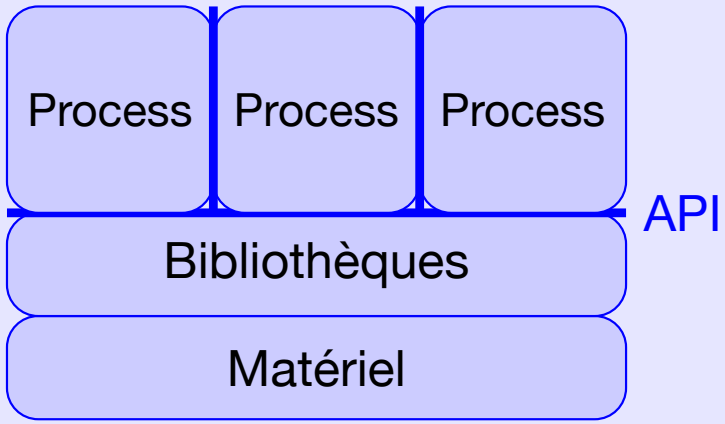
- Gérer le matériel ;
- Fournir une abstraction du matériel (bibliothèques) ;



Problématiques

Rôle d'un système d'exploitation :

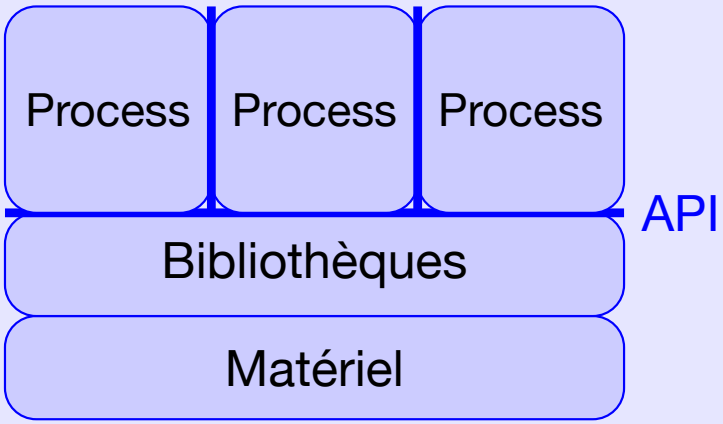
- Gérer le matériel ;
- Fournir une abstraction du matériel (bibliothèques) ;
- Gérer et protéger des processus en parallèles ;



Problématiques

Rôle d'un système d'exploitation :

- Gérer le matériel ;
- Fournir une abstraction du matériel (bibliothèques) ;
- Gérer et protéger des processus en parallèles ;
- Permettre la gestion et administration de la machine.



Les services

Présentation :

- Persistance : système de fichier ;
- Temps : processus ;
- Espace : mémoire ;
- Interaction : signaux, synchronisation.

Tentative de définition

Système d'exploitation :

Ensemble d'éléments permettant la prise en charge de l'aspect matériel de l'ordinateur par la présence d'interfaces abstraites et assurant le partage et la protection de ce matériel.

Exemples :

UNIX, Mac OS X, GNU/Linux, Windows, *BSD, Android, iOS, …

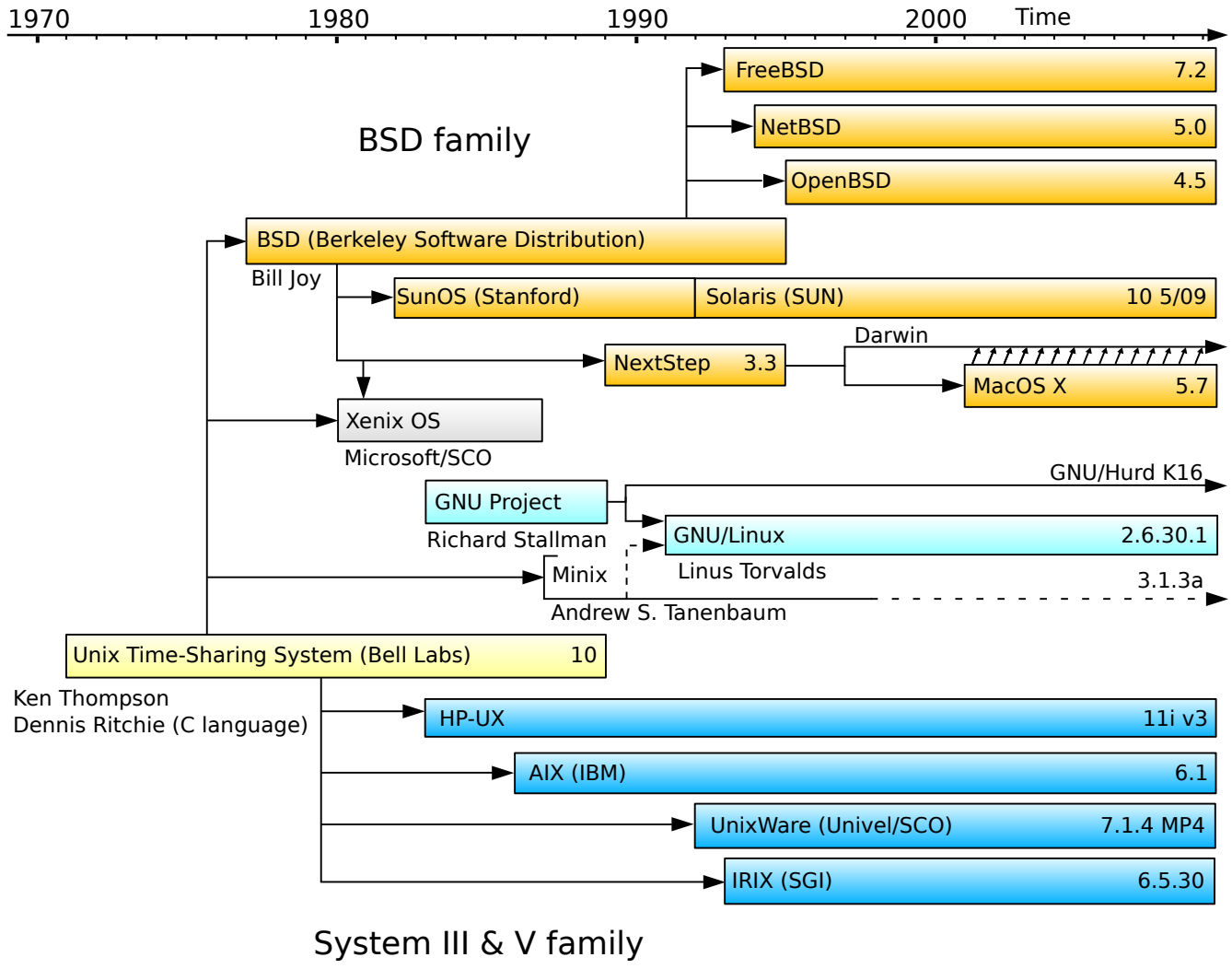
2. UNIX and co

Qu'est ce que UNIX ?

Un peu d'historique :

- En 1965, un projet ambitieux de réaliser un système d'exploitation multi-utilisateurs : Multics (Multiplexed Information and Computing Service).
- En 1969, Bell Labs se retire du projet. K. Thompson, D.Ritchie initient un projet de système d'exploitation pour PDP-7.
- En 1973, UNIX est réécrit en C, devient portable et est commercialisé.
- En 1977, un fork apparaît à Berkeley : **BSD**.

L'explosion


1970
1980
1990
2000
Time
FreeBSD 7.2
NetBSD 5.0
OpenBSD 4.5
BSD family
BSD (Berkeley Software Distribution)
Bill Joy
SunOS (Stanford)
Solaris (SUN) 10 5/09
Darwin
NextStep 3.3
MacOS X 5.7
Xenix OS
Microsoft/SCO
GNU/Hurd K16
GNU Project
Richard Stallman
GNU/Linux 2.6.30.1
Linus Torvalds
Minix 3.1.3a
Andrew S. Tanenbaum
Unix Time-Sharing System (Bell Labs) 10
Ken Thompson
Dennis Ritchie (C language)
HP-UX 11i v3
AIX (IBM) 6.1
UnixWare (Univel/SCO) 7.1.4 MP4
IRIX (SGI) 6.5.30
System III & V family

Pourquoi UNIX ?

UNIX

- est un système standardisé,
- est un système répandu,
- a un grand impact historique et pratique,
- dispose de concept fondamentaux simples,
- disposait d'une API concise.
- possède une tradition d'ouverture du code.

Normalisation

Il n'existe pas une normalisation d'UNIX mais un ensemble de normalisations qui disposent chacune de plusieurs versions. En particulier, on citera :

- **POSIX** (Portable Operating System Interface) qui normalise les appels système, les commandes et utilitaires.
- **SUS** (Single Unix Specification) qui est une famille de normes recoupant partiellement les normes POSIX et servant de références pour UNIX.

Respect des normes

Il existe un programme de certification des systèmes d'exploitations. Seuls ceux conformes à la norme Single Unix Specification ont le droit de porter officiellement le nom UNIX (actuellement : AIX, HP/UX, Mac OS X, Solaris et z/OS).

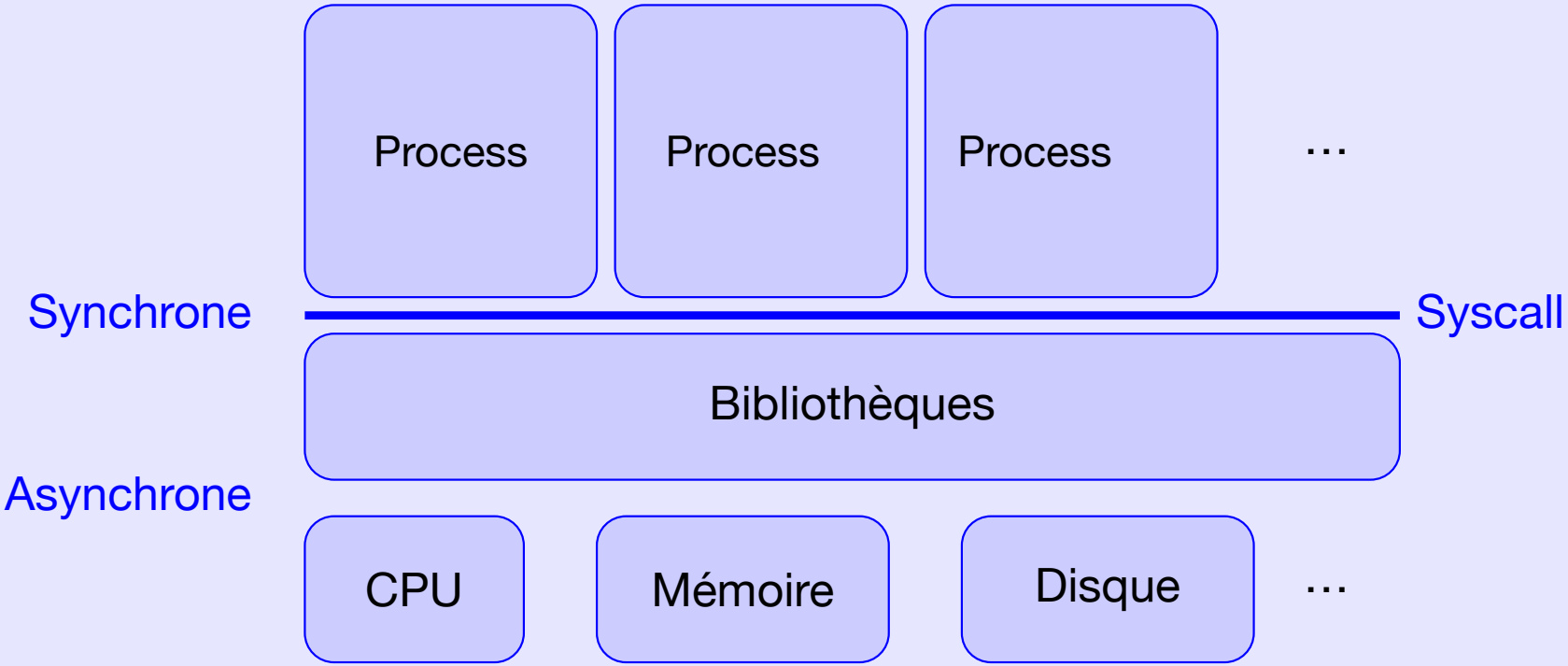
Cependant, un très grand nombre d'autres systèmes offrent un niveau suffisant de compatibilité. Parmi ceux-ci, il existe en certain nombre de programmes libres : GNU/Linux, OpenSolaris, NetBSD, FreeBSD, OpenBSD, …

Modèle

Le modèle UNIX incarne un système d'exploitation :

- **monolithique :** le système se compose d'un seul bloc s'exécutant en mode privilégié, les utilisateurs interagissent avec lui à l'aide d'appels système (syscall).
- **multi-programmé :** les temps d'attentes des appels systèmes ne bloquent pas tout le système.
- **multitâche préemptif :** le temps de calcul est partagé entre processeurs et utilisateurs avec ou sans leur coopération.

Principe


Process
Process
Process
...
Synchrone
Syscall
Bibliothèques
Asynchrone
CPU
Mémoire
Disque
...

3. Du programme au processus

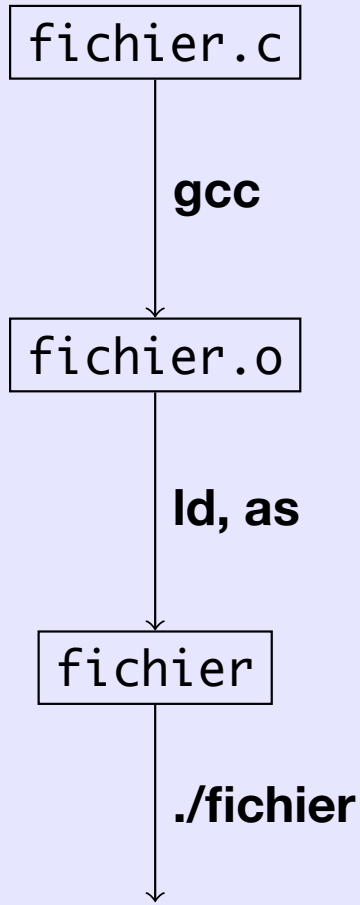
Définitions

- **Programme :** ensemble des fichiers textes décrivant un ensemble d'opérations à effectuer.
- **Exécutable :** fichier binaire contenant les instructions machine correspondant aux opérations du programme ainsi que des instructions pour le charger en mémoire.
- **Processus :** Exécutable en cours d'exécution sur la machine.

Du code source à l'exécution

Cas des langages compilés :

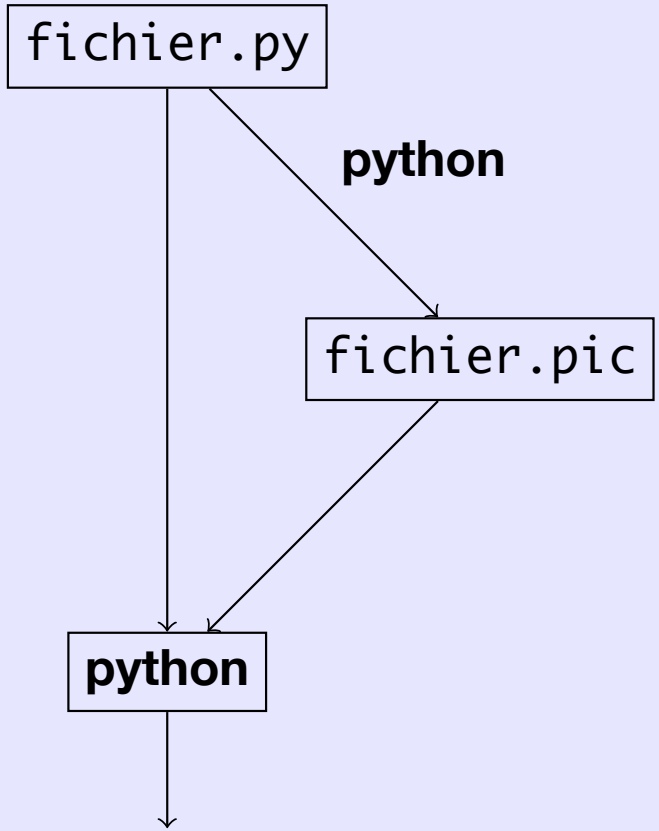
- le fichier source contient le code du programme ;
- on compile le fichier pour produire du code exécutable ;
- on fait le liens avec les fonctions de bibliothèques (linkage) ;
- on lance le programme.



Du code source à l'exécution

Cas des langages interprétés :

- le fichier source contient le code du programme ;
- on convertit le langage vers un langage intermédiaire ;
- un interpréteur se charge alors d'exécuter le source et de gérer les fonctions de bibliothèques.



4. Utilisation d'appel système

Python **os.times**

Documentation

```
>>> import os
>>> help(os.times)

Help on built-in function times in module posix :

times(...)
    times() -> (utime, stime, cutime, cstime, elapsed_time)

   Return a tuple of floating point numbers indicating process times.

>>> os.times()
(0.04000000000000001, 0.02, 0.0, 0.0, 18191555.66)
```

en C

man 3 times

```
TIMES(3)            BSD Library Functions Manual            TIMES(3)

NAME
     times -- process times

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <sys/times.h>

     clock_t
     times(struct tms *buffer);

DESCRIPTION
     ...
```

En shell

Utilitaire times :

```
sh-3.2$ man 1 times
....
sh-3.2$ help times
times : times
   Print the accumulated user and system times for processes run from
    the shell.
sh-3.2$ times
0m0.005s 0m0.010s
0m0.243s 0m0.063s
```

Des erreurs

Dans certains cas, les appels système peuvent échouer (arguments invalides, périphérique indisponible, …). L'appel retourne alors une valeur particulière (langage C, shell, …) ou lève une exception (python, ...). Ces comportements sont décrits dans les pages de manuel des appels systèmes.

Exemple : Extrait de **man 2 chdir**

```
ERRORS
   The Chdir() system call will fail and the current working directory
 will be unchanged if one or more of the following are true :

    [EACCES]          Search permission is denied for any component
                         of the path name.

    ...
```

Et des hommes

Méthodologie :

Lors de l'utilisation d'un appel système, il faut vérifier le bon déroulement de l'appel et gérer les erreurs.

Erreurs en C

La norme POSIX utilise errno

Principe :

Lorsqu'un appel système rencontre une erreur, il positionne la valeur de la variable `errno` et retourne avec une valeur définie (en général -1). Il est alors possible d'afficher un message d'erreur avec le fonction `perror`.

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

void perror(const char *s);
```

Erreurs en C

Exemple :

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

...
if (chdir(”toto”) == -1) {
    /* En cas d’erreur */
    perror(”chdir a échoué”);
    exit(-1);}

/* suite du programme */
...
```

Erreurs en shell

Principe : En cas d'erreur, le programme affiche un message sur la sortie d'erreur et renvoie une valeur particulière (en général une valeur différente de 0).

Exemple :

```
sh-3.2$ ls toto
ls : toto : No such file or directory
sh-3.2$ echo $?
1
```

Erreurs en python

Principe :

En cas d'erreur, un appel système retourne une exception. Ce système est basé sur le principe de la variable `errno`.

Exemple :

```
from os import *
import errno

...
    try :
        os.chdir("toto")
    except OSError as err :
        print("Erreur chdir :",err)
        # gestion de l'erreur
...
```

5. multi-utilisateurs

Une machine / plusieurs utilisateurs

Sur une machine, il peut y avoir plusieurs utilisateurs. Chacun dispose d'un identifiant uid.
Il existe également des groupes. Chaque groupe possède également un identifiant unique : le gid.
Chaque utilisateur peut appartenir à plusieurs groupes. En shell, il est possible de connaître ces informations à l'aide de la commande **id**.

Exemple :

```
sh-3.2$ id
uid=4064(grichard) gid=140(ens) groupes=100(users),130(fuse),
140(ens),669(audit),999(grsec)
```

Droits

- Les utilisateurs peuvent correspondre à des personnes physiques (ex : vous) ou à des service (ex : www pour le serveur web).
- À chaque utilisateur et chaque groupe sont associées des permissions.
- Il existe un super-utilisateur qui possède tous les droits. Usuellement, son nom est **root** et son uid est **0**.
- Dans un système UNIX, ces permissions sont mises en place par l'intermédiaire du système de fichier. Nous verrons cela plus tard.

Accès à distance

Principe :

- Il peut y avoir plusieurs claviers / écrans reliés à un même ordinateur permettant ainsi à plusieurs personnes de l'utiliser simultanément. On parle alors de client léger.
- Il est également possible de se connecter à un ordinateur à distance par exemple à l'aide de la commande **ssh**.

Remarque :

Il est important de **ne pas éteindre** les machines mais uniquement de se déconnecter (logout).

6. shell

Interaction avec l'ordinateur

Il est possible d'interagir avec l'ordinateur à l'aide de plusieurs périphériques :

- **clavier** ;
- **souris** ;
- **tablette** ;
- **micro** ;
- …

Présentation des programmes

Il existe deux types principaux d'interfaces :

- **interface en ligne de commande :** l'utilisateur entre des commande aux clavier, l'ordinateur affiche sous forme de texte le résultat et les éventuelles questions
- **GUI :** (Graphical User Interface) l'interface graphique que vous connaissez sans doute déjà.

Le shell

Principe :

Le shell est un programme d'une interface en ligne de commande permettant de lancer n'importe quel programme et de **tout** faire.

Et même le café :

La plupart des shell fournissent de nombreuses fonctionnalités additionnelles :

- complétion ;
- facilité de rappel ;
- facilités de programmation ;
- raccourcis d'éditions ;
- …

Un, deux, trois, ...

Il existe de nombreux shell :

- **sh :** le shell historique et normalisé par les normes UNIX ;
- **csh :** un shell munie entre autre de facilités de programmation basées sur le langage C ;
- **ksh :** est une “évolution” de csh ;
- **bash :** (Bourne again shell) un shell courant et assez évolué ;
- **dash :** (Debian Almquist shell) un shell plus minimaliste et qui tente d’être proche de la norme POSIX.

Commande

Dans un shell, on rentre des **commandes**.

Exemple :

```
sh-3.2$ls -l -h /bin /var
```

Les éléments de la commande sont appelés arguments et numérotés à partir de 0. Usuellement, les arguments commençant par des signes - sont des options de la commande et modifient son comportement.

7. Contenu du cours

Cours

Dans ce cours, on parlera principalement des objets suivants :

- Shell : dash
- Système d'exploitation : GNU/Linux

Une séance sera consacré au cours d'année à l'administration d'un système sous différentes distributions (Ubuntu / Fédora / ...).

Cours (suite)

Au niveau du système d'exploitation, les cours seront organisés autour des concepts suivants :

- Système de fichier ;
- Processus :
- Mémoire ;
- Signaux ;
- Synchronisation.

Cours (méthode)

La présentation des appels système se fera principalement en insistant sur les concepts et à l'aide des langages suivants :

- En **C**
- En **python**
- Par l'intermédiaire des utilitaires.

TD / TP

TD :

- Utilisation des appels systèmes ;
- Manipulation des concepts.

TP :

- Programmation de scripts shell ;
- Un peu de python ;
- Un projet à réaliser.