Introduction à la Programmation Système

Système d'exploitation
Principes généraux
Précautions

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

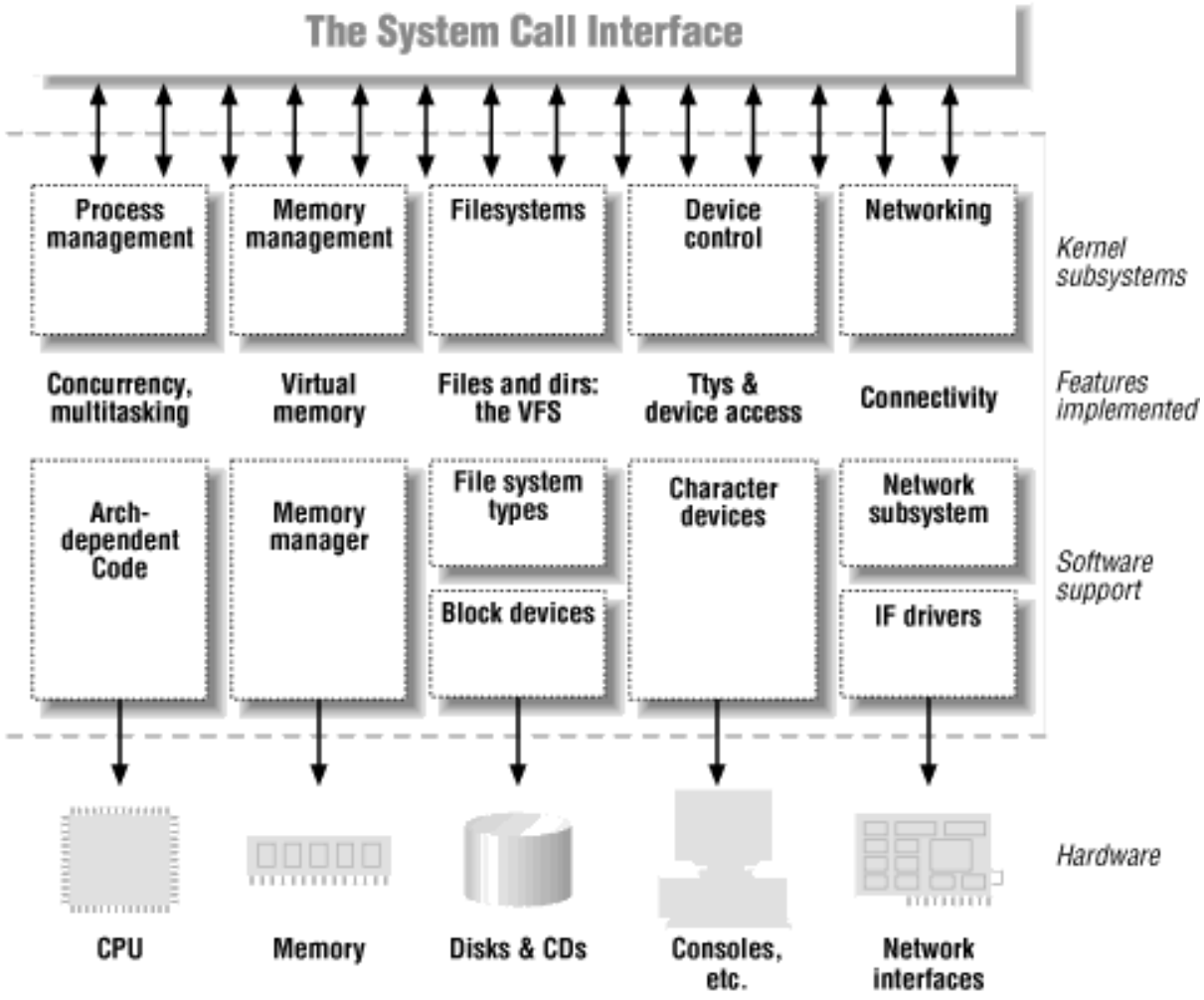
Un système d'exploitation

- **Son rôle**
  - Interfacer le matériel et les services applicatifs
  - Piloter le matériel, fournir des abstractions
  - Gérer la mémoire (réelle/virtuelle)
  - Gérer les fichiers
    - Disques/systèmes de fichiers
  - Gérer la multi-programmation
    - Partage du temps, des ressources, communication
    - **Processus** $\simeq$ programme en cours d'exécution

Un système d'exploitation

- **Ce que c'est principalement**
  - Un noyau (ou plusieurs) contrôlant la machine
  - Un ensemble de points d'entrée pour le solliciter
    - Les *appels-système*
- **Ce n'est pas** *"exclusivement"*
  - Un environnement de développement
  - Un interpréteur de commandes
  - Une interface graphique

Un système d'exploitation



Un système d'exploitation

- **Architecture monolithique**
  - Un noyau unique se charge de tout
  - Structure interne complexe
  - Solution efficace et la plus employée
  - Des modules peuvent le compléter
- **Architecture micro-noyau**
  - Le noyau est minimal
  - Traitements confiés à des serveurs spécialisés
  - Structure individuelle simple mais
    - Communication, synchronisation
    - Structure globale bien plus complexe
  - Solution peu efficace, domaine universitaire (*Mach*, *GNU-HURD*)

Quelques points d'histoire

▷ **À propos d'*Unix***

- ◇ 1969 : K. Thompson & D. Ritchie (*AT&T*, *Bell Labs*) ancêtre d'*UNIX*
- ◇ 1973 : Première version d'*UNIX* en *C* (portable, sources disponibles)
- ◇ 1978 : *UNIX* version 7 devient commercial !
- ◇ 1980 : Variante *BSD* (Université de Californie, Berkeley)
- ◇ 1983 : *AT&T UNIX System V*
- ◇ 1984 : Naissance du projet *GNU*, *Free Software Fundation*
- ◇ 1987 : *AT&T* et *Sun* unifient *BSD* et *System V* (*SunOS*)
  (systèmes propriétaires *HP-UX*, *AIX* ...)
- ◇ 1990 : *System VR4* apporte de nouveaux standards d'unification
- ◇ 1991 : *OSF/1*, *Open Software Fundation*
- ◇ 1991 : Clones gratuits
- ◇ 1992 : *Solaris* de *Sun* basé sur *System VR4*

Quelques points d'histoire

▷ **À propos de *DOS***

- ◇ 1974 : G. Kildall (*Digital Research*), *CP/M* pour *8080* et *Z80*
- ◇ 1981 : *IBM PC*, portage de *CP/M* (*PCDOS*)
- ◇ 1981 : *Microsoft* achète *QDOS*
  (inspiré de *CP/M*, T. Paterson, *Seattle Computer Products*)
  commercialisé sous le nom *MSDOS 1.0*
- ◇ 1985 : *Windows 1.0*, *Microsoft*
- ◇ 1988 : *DR-DOS*, *Digital Research*
- ◇ 1990 : *Windows 3.0*, *Microsoft* (énorme succès commercial)
- ◇ 1991 : *MSDOS 5.0*, *Microsoft* (gestion *"subtile"* de la mémoire)
- ◇ 1995 : *Windows95*, *Microsoft* (*MSDOS "habilement"* dissimulé)

La programmation système

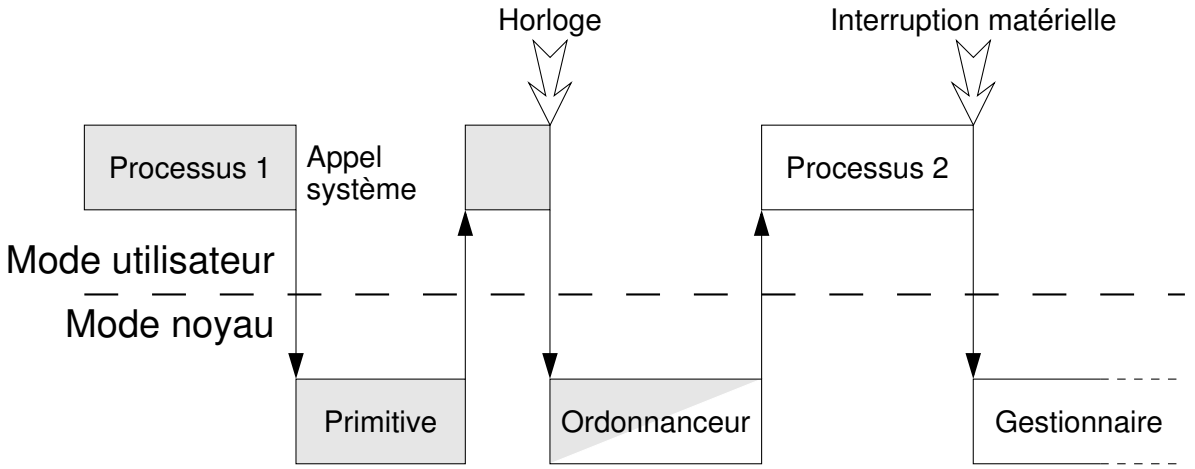
- **Objectifs**
  - Comprendre les principes d'un système d'exploitation
  - En maîtriser les principaux services
    - Appels système
    - Bibliothèques usuelles
  - Tirer profit de ce qui existe en standard
  - Prendre en considération la portabilité
    - Normes existantes
    - Conventions d'écriture
  - Il ne s'agit pas ici de réaliser un système !

La programmation système

- **Les standards**
  - Forte influence de l'histoire des systèmes *UNIX*
  - *ANSI-C* (*Iso-C*) : Services standards accompagnant le langage
  - *SystemV* / *BSD* : Services issus d'une famille particulière
  - *POSIX* : Services attendus des systèmes *UNIX* (et autres)
    - *POSIX.1* : Appels système et fonctions d'usage général
    - *POSIX.1b* : Extensions temps-réel
    - *POSIX.1c* : Les *threads*
    - *POSIX.1e* : Sécurité, autorisations d'accès
    - ...
  - *UNIX98* : Autre effort de normalisation des systèmes *UNIX*

Mode noyau / mode utilisateur

- **Mode utilisateur**
  - Mode de fonctionnement *"normal"* d'un processus
  - Traitements dans son propre espace d'adressage
  - Pas de risque majeur (sauf pour lui)
- **Mode noyau**
  - Accès au matériel, à la mémoire physique
  - Gestion des processus



Appel système $\neq$ fonction

- **Appel système**
  - Permet d'accéder aux services du système
    - Passage en mode noyau
    - Vérification si processus autorisé
  - Mécanisme d'interruption très couteux en temps
    - Sauvegarde des données du processus ...
- **Fonction**
  - Calcul dans l'espace utilisateur
  - Encapsulation des appels système
    - Services de plus haut niveau
    - Regrouper plusieurs invocations en une seule
    - Rôle de la bibliothèque standard du *C*

Appel système $\neq$ fonction

- **Point de vue du programmeur**
  - Langage de prédilection : *C*
  - L'aspect extérieur est le même (un prototype de fonction)
  - L'ensemble $\simeq$ une grande bibliothèques de services
  - La lecture de la documentation est nécessaire
    - Appel système : `$ man 2 service`
    - Fonction : `$ man 3 service`

Quelques précautions

- **De la rigueur !**
  - *"C'est facile lorsque ça fonctionne !"*
    - Difficile d'expliquer les dysfonctionnements *a posteriori*
  - Compiler le plus sévèrement possible
    - `$ g++ -W -Wall -pedantic -Werror ...`
    - Il est **toujours** possible d'éliminer un avertissement
  - Tester le succès ou l'échec des invocations
    - Lire attentivement les pages de manuel
    - En particulier les sections `RETURN VALUE` et `ERRORS`
  - Faciliter la portabilité
    - Vérifier l'origine (*ANSI-C*, *POSIX*, *SystemV*, *BSD* ...)
    - Éviter les extensions spécifiques à l'environnement
    - Ne pas compter sur un comportement spécifique

Quelques précautions

- **Utilisation de** `errno` (`man 3 errno`)
  - `#include <errno.h>`
    `extern int errno;`
  - Indicateur positionné en cas d'erreur
    - Depuis les appels système
    - Depuis quelques fonctions de bibliothèque
  - Un ensemble de constantes indiquant la nature de l'erreur
  - Utilisation classique
    - `errno=0;` (facultatif)
    - Effectuer l'appel
    - Si le résultat indique une erreur (`-1` généralement)
    - $\rightarrow$ Lire la valeur de `errno`

Quelques précautions

- **Utilisation de `errno`**
  - Description d'une erreur (`man 3 strerror`)

```
#include <string.h>
char * strerror(int errnum);
```

  - Retourne une chaîne allouée statiquement décrivant l'erreur `errnum`
  - Signaler une erreur (`man 3 perror`)

```
#include <stdio.h>
void perror(const char * msg);
```

  - Écrit `msg` et une description de `errno` dans la sortie d'erreur
  - `msg` peut être un pointeur nul

Quelques précautions

- **Les appels système** *"lents"*
  - Certains appels système sont atomiques
    - Information immédiatement disponible dans le noyau
  - D'autres peuvent prendre du temps (*"lents"*)
    - Il faut attendre les informations
  - Les processus peuvent recevoir des signaux
  - Si un processus reçoit un signal en cours d'appel système
    - Le processus est relancé
    - Les informations ne sont pas forcément disponibles !
    - L'appel système échoue
    - `errno` vaut `EINTR`
  - Sans gravité → recommencer l'appel
  - Précisé dans la section `ERRORS` du manuel de l'appel

Utilisation de `errno`

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

int main(void)
{
int status=EXIT_SUCCESS;
int fd=open("file.txt",O_RDONLY);
if(fd==-1)
  {
  perror("open()");
  status=EXIT_FAILURE;
  }
else
  {
  char buffer[256];
  ssize_t nb;
```

```
$ ./prog
open(): No such file or directory
$ echo Hello > file.txt
$ ./prog
Success ! --> Hello

$ chmod -r file.txt
$ ./prog
open(): Permission denied
$ rm file.txt
$ mkdir file.txt
$ ./prog
read(): Is a directory
$
```

Utilisation de `errno`

```
  do
    {
    nb=read(fd,buffer,255);
    } while((nb==-1)&&
            ((errno==EINTR)||(errno==EAGAIN)));
  if(nb==-1)
    {
    perror("read()");
    status=EXIT_FAILURE;
    }
  else
    {
    buffer[nb]='\0';
    fprintf(stderr,"Success ! --> %s\n",buffer);
    }
  close(fd);
  }
return(status);
}
```

Quelques précautions

- **Au delà du code strict**
  - S'imaginer dans les pires conditions !
  - Présentation, indentation
    - 80 colonnes, espaces/tabulations ...
    - *"Aurai-je toujours ma super-imprimante sous la main ?"*
  - Capitaliser son expérience
    - Une difficulté rencontrée → une nouvelle règle pour l'éviter
    - Un problème de portabilité → retenir la solution commune
  - De nombreuses opérations s'utilisent dans des contextes variés
    - S'efforcer de les effectuer de la même façon
    - Utilisation systématique de *patrons* de programmation
  - Pas de règles universelles mais de l'homogénéité