L2 Informatique

Systèmes d'exploitation I

Introduction

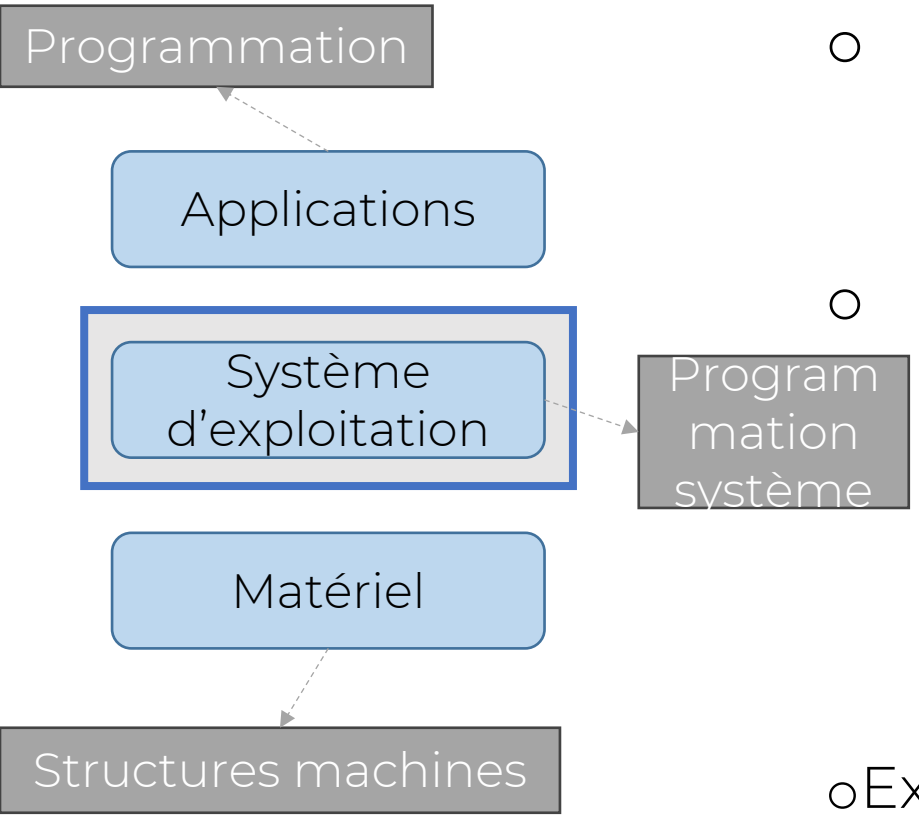
Jessica BECHET

Introduction

- Quelques rappels sur le système d'exploitation
- Pourquoi la programmation système ?
- Normalisation de l'interface des systèmes Unix
- Appel système

Rappels sur le système d’exploitation

Rappels sur le système d'exploitation



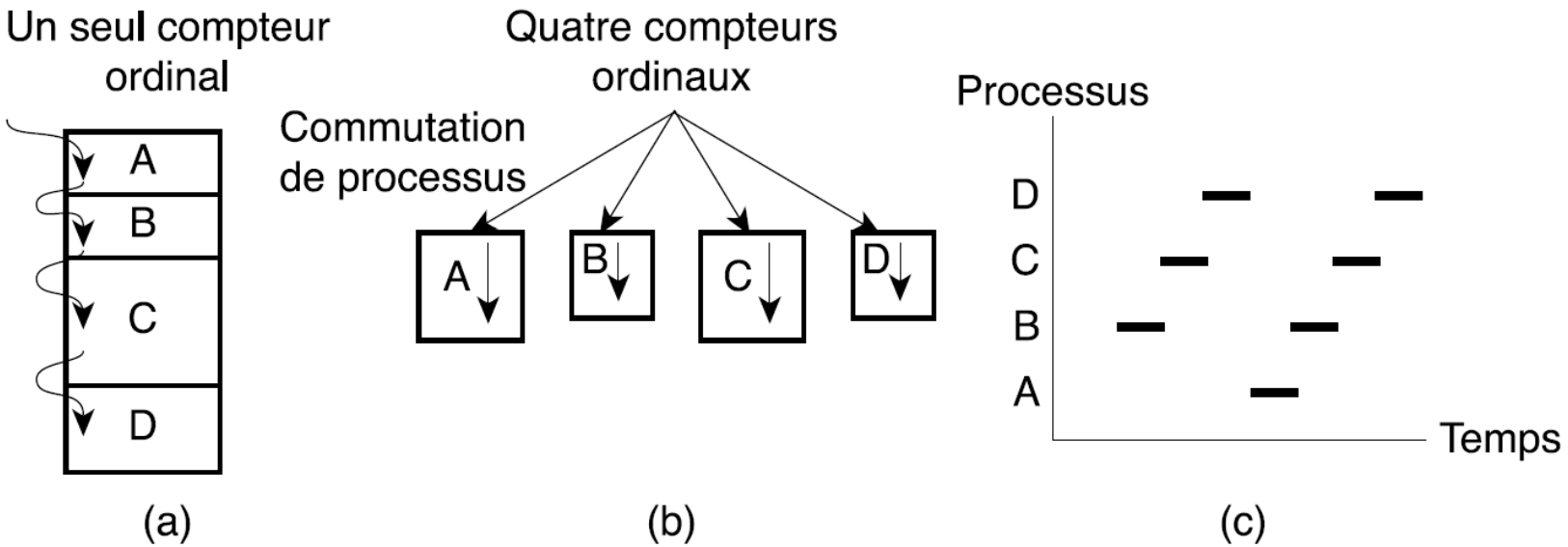
- SE : logiciel entre les applications et le matériel
  - Aujourd'hui en général : multi utilisateur, multi tâches
- SE : 3 fonctions principales
  - Chargement des programmes
  - Abstraction de la machine physique : machine virtuelle
    - Interface de la machine virtuelle composée d'un ensemble d'appels système
  - Gestion des ressources
- Exemples : Linux, Mac OS, Windows // Android, iOS

Rappels sur le système d'exploitation

- Interface
  - Abstraction du matériel
    - Pas besoin de connaître spécifiquement les composants dans la machine
      - Utilisation de syntaxes « compréhensibles » et « uniques »
  - Portabilité des programmes sur les systèmes dont les interfaces suivent la même normalisation

Rappels sur le système d'exploitation

- Chaque processus a son processeur virtuel
- Processeur commute entre plusieurs processus s'exécutant en pseudo-parallélisme (multi-programmation)



(a) Multiprogrammation de quatre programmes. (b) Modèle conceptuel de quatre processus séquentiels indépendants. Un seul programme est actif à un instant donné.

Conception de systèmes d'exploitation – le cas Linux, Patrick Cegielski, 2ème édition.

Pourquoi faire de la programmation système ?

Pourquoi la programmation système ?

- Comprendre le fonctionnement du système d'exploitation
- Créer des commandes spécifiques
- Créer un système d'exploitation

- Ingénieur système

Rappels sur le système d’exploitation

Histoire

Histoire systèmes d'exploitation

- EDSAC, Manchester Mark-1
  - Machine à programme enregistré
- GM/NAA
  - Premier ordinateur de traitement par lots
  - Enchaînement de travaux (lots de programmes utilisateurs)
- IBM OS/360
  - Invention de canaux d'entrée sortie → exécution en parallèle d'entrées-sorties & opérations de calcul
  - OS gère simultanément : lecture des travaux à exécuter, exécution, impression des résultats
  - Initialement un seul programme utilisateur, puis, multi-programmation
- CTSS
  - Système en temps partagé
- Multics
  - Introduction d'un système de fichiers plus efficace
  - Ancêtre de Unix
- UNIX
  - Multi utilisateur
  - Création du langage C pour l'écriture du logiciel de base

Rappels sur le système d'exploitation

Structure interne

Système d'exploitation

- Système à 2 modes : noyau & utilisateur
- Démarrage mode noyau
  - Initialisation des périphériques
  - Mise en place des routines de service pour les appels système
  - Commutation mode utilisateur
- Accès aux périphériques en mode utilisateur : utilisation des appels système
- Changement mode noyau uniquement par compilation du noyau

Système d'exploitation

- Systèmes à couches
  - Extension du système à 2 modes
  - Chaque couche s'appuie sur celle immédiatement inférieure
  - Une couche : demande de services à la couche inférieure & fourniture de services à la couche supérieure

  - Voir CPU

Rappels sur le système d’exploitation

Unix

UNIX

- UNIX
  - Marque déposée de *The Open Group*

  - Flexible
  - Logiciel « ouvert »
  - API Unix couche proche du matériel

  - « Tout est fichier »

UNIX

- Quelques règles de la philosophie UNIX
  - Ecrire des programmes qui coopèrent entre eux
  - Utiliser des algorithmes et des structures de données simples
    - Modularité
    - Préférer la clarté à l'intelligence
    - Placer le savoir dans les données
  - Concevoir un comportement lisible ➔ facilité d'investigation et de débogage
  - Rechercher un bon fonctionnement avant d'optimiser
  - Concevoir en pensant à l'avenir
  - *Se méfier de la bonne solution unique*
  - KISS : Keep It Simple, Stupid

Normalisation de l'interface des systèmes Unix

Normalisation de l'interface – UNIX

- POSIX
  - Portable Operating System Interface X
  - Décrit l'interface entre le SE (noyau) et les applications
  - Standard officiel définissant les interfaces communes à presque tous les systèmes Unix
  - Norme IEEE
  - IEEE non-propriétaire de Unix → ne peut revendiquer proposer un standard

- SUS
  - Single Unix Specification
  - Ecrite par l'Open Group (association d'éditeurs de systèmes compatibles Unix)
  - Sensiblement équivalente à POSIX
  - http://www.unix.org/version4/

Normalisation de l'interface

- Norme
  - Portabilité des fonctions ou d'une option d'une commande

- Interface d'un système Unix
  - Interface POSIX = interface du système
  - Fonction POSIX = appel système

Bibliothèque C et POSIX

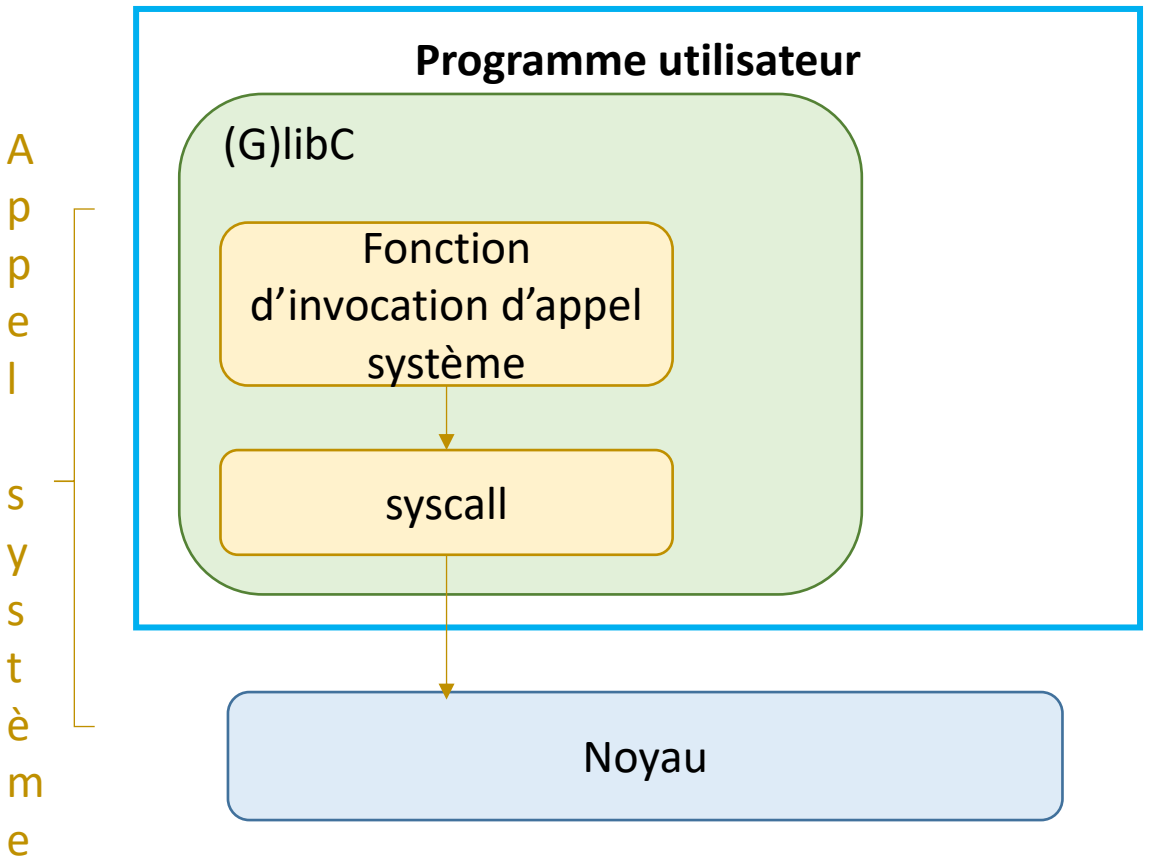
- Bibliothèque C
  - Normalisation ISO
  - Section 3 man
- POSIX
  - Normalisation Single Unix
  - Section 2 man

Bibliothèque et appel système

Système d'exploitation

- Programmes d'application des utilisateurs
  - Programmation « classique »
- Programmes systèmes
  - Assurent le bon fonctionnement de l'ordinateur
- Langage C créé pour le développement du SE UNIX

Appel système



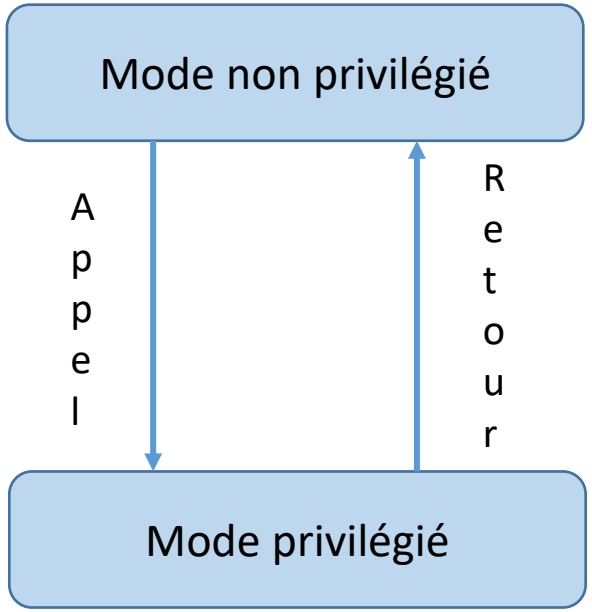
- syscall
  - `long syscall(long number, …)`
    - Numéro de l'appel système
    - Paramètres de l'appel système
    - *`syscall(SYS_chdir, "/home/user/Desktop")`*
  - Appel du code assembleur correspondant
  - Peu dépendre de l'architecture matérielle
- Fonction d'invocation
  - Fait appel à syscall
  - Portable (ne dépend pas de l'architecture matérielle)
  - *`chdir("/home/user/Desktop")`*

Bibliothèque et appel système

- Appels systèmes
  - Fonctions permettant de communiquer avec le noyau
  - Espace utilisateur → demande de services au SE

  - Manipulation du système de fichiers et E/S
  - Gestion des processus
  - Communications entre processus

Bibliothèque et appel système



- Appel système ≠ appel fonction
  - Appel système
    - Appel à un sous-programme intégré au système
    - Points d'entrée dans le noyau de l'OS
    - Accès bas niveau
    - Réception par le noyau → vérification si appel système valide → exécution → renvoi au mode utilisateur
  - Appel fonction
    - Appel à un sous-programme d'un programme
    - Portables (peu importe l'interface)
    - Accès haut niveau
- Appel système + coûteux qu'une fonction classique

Bibliothèque et appel système

- Bibliothèque C
  - Développement plus facile
  - Portabilité vers les systèmes n'ayant pas la même normalisation
  - Encapsulation d'appels système dans des fonctions

Bibliothèque et appel système

- Terminaison d'un appel système
  - Sémantique POSIX d'une primitive
    - Description
    - Même en cas d'erreur
    - Liste des erreurs possibles
    - Voir `man`
  - Valeur de retour d'un appel système
    - En cas de réussite
    - En cas d'erreur

❖Tester systématiquement le retour des fonctions

man

man

- Manuel standard
- Divisé en sections

man

1. Programmes exécutables ou commandes de l'interpréteur de commandes
2. Appels système
3. Appels de bibliothèque (fournies par les bibliothèques des programmes)
4. Fichiers spéciaux (situés généralement dans /dev)
5. Formats des fichiers et conventions
6. Jeux
7. Divers
8. Commandes de gestion du système
9. Sous-programmes du noyau

man

- Exemple
  - `printf(1)`
  - `man 1 printf`
  - `printf(3)`
  - `man 3 printf`

Gestion des erreurs

Gestion des erreurs

Gestion des erreurs

Variables symboliques pour décrire chaque erreur possible

- Exemple : `EACCES, EAGAIN, EINVAL, …`

Difficile de tester toutes les erreurs possibles à chaque appel-système effectué

Que faire alors ?

- Vérifier au moins que l'appel-système a bien fonctionné
- Suivant le cas, on peut tester certaines erreurs
- Si le programme est bien débogué, certaines erreurs ne doivent jamais se produire
  - Exemple : pointeur mal initialisé

Gestion des erreurs

- Variable globale **errno**
  - Valeur d'erreur
  - Mais pas de description de l'erreur produite
  - Effectuer des tests par rapport à la valeur pour trouver la cause

  - Variable globale (ne pas l'utiliser directement)
  - Valeur valable uniquement juste après l'utilisation de la fonction à tester
  - man 3 **errno**

```
#include <errno.h>

extern int errno;
```

Gestion des erreurs

- Fonction **`strerror(3)`**
  - Retourne un pointeur vers une chaîne de caractères décrivant l'erreur dont le code est donné en argument
  - renvoyée par **`errno`**

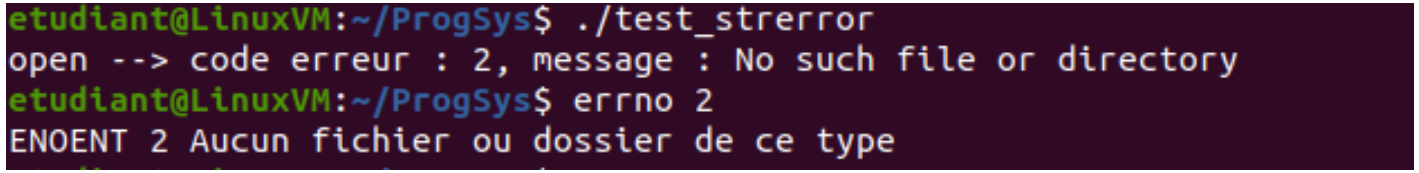
```
#include <string.h>

char * strerror (int errnum);
```

Gestion des erreurs

- Exemple

```
if open("data", O_RDONLY) == -1 )
        fprintf(stderr, "open --> code erreur : %d, message : %s", errno, strerror(errno));
```

- Résultat



- La deuxième ligne de commande permet de voir le nom symbolique associé au numéro d'erreur affiché

Gestion des erreurs

- Fonction **`perror(3)`**
  - Permet d'associer à l'utilisation d'une fonction une description de l'erreur produite
  - Affiche sur **`stderr`** (sortie d'erreur standard) un message de l'erreur décrite par **`errno`**

```
#include <stdio.h>

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

Gestion des erreurs

- Exemple

```
if ( open("data", O_RDONLY) == -1 )
{
    perror("open");
}
```

- Résultat

```
etudiant@LinuxVM:~/ProgSys$ ./test_perror
open: No such file or directory
etudiant@LinuxVM:~/ProgSys$
```

Eléments de compilation

Eléments de compilation

```
gcc -Wall -Werror -std=c18 -pedantic -D_XOPEN_SOURCE=700 -g code.c -o output
```

- `-Wall` : affichage de tous les avertissements
- `-Werror` : affichage de tous les avertissements comme des erreurs
- `-std` : pour vérifier la conformité avec les standards du langage C (c24 est la dernière version mais non encore disponible)
  - https://blog.ansi.org/2017/09/origin-ansi-c-iso-c/
  - https://blog.ansi.org/2018/11/c-language-standard-iso-iec-9899-2018-c18/
- `-pedantic` : avertissements (+ orientés vers la portabilité du code)
- `-D_XOPEN_SOURCE` ≥ 500
  - https://stackoverflow.com/questions/5378778/what-does-d-xopen-source-do-mean
  - La dernière version actuelle correspond à une valeur de 700
- `-g` : inclus dans l'exécutable les informations pour utiliser le débogueur

Debug

Debug

- `(f)printf(3)`
- `assert(3)`

- Commande **`gdb`**

- Commande **`valgrind`**
  - Débogage en particulier pour les fuites mémoire

Type opaque

Un mot sur les types opaques

- Type opaque
  - Structure
  - Accès par pointeur
  - Pas de visibilité du contenu de la structure
    - Définition de la structure et ses champs et fonctions associées dans le .c et définition de la structure dans le .h (sans les champs)
  - Utilisation de la structure sans se préoccuper de son contenu

Merci !