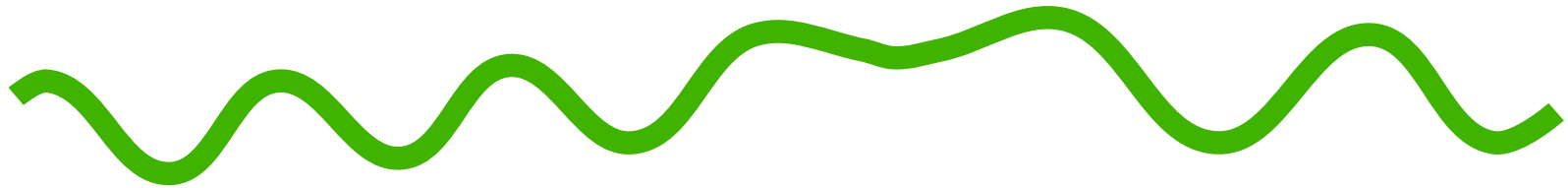


Programmation des systèmes

Philippe MARQUET

Philippe.Marquet@lifl.fr

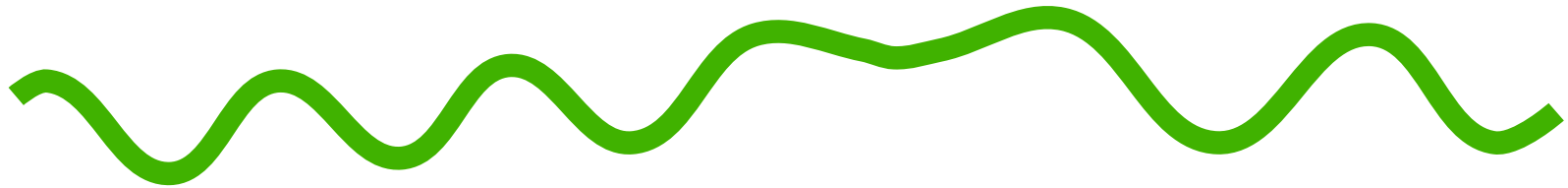
Laboratoire d'informatique fondamentale de Lille

Université des sciences et technologies de Lille

Licence d'informatique de Lille

décembre 2004

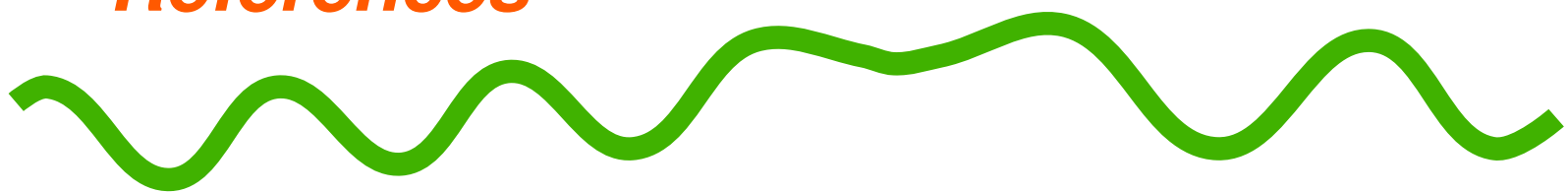
révision de janvier 2010



- Ce cours est diffusé sous la GNU Free Documentation License, `www.gnu.org/copyleft/fdl.html`
- La dernière version de ce cours est accessible à `www.lifl.fr/~marquet/cnl/pds/`
- `$Id: intro.tex,v 1.17 2009/12/11 07:29:22 marquet Exp $`

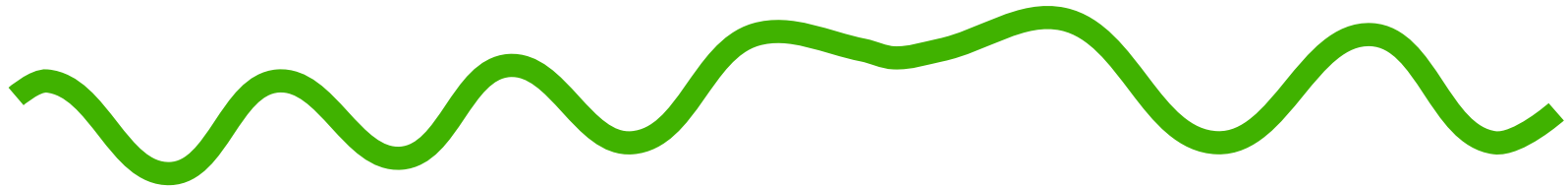
- Page web du cours
  - Portail du FIL `www.fil.univ-lille1.fr/portail/`
  - Portail de PDS `www.fil.univ-lille1.fr/portail/ls6/pds`
  - supports de cours TD TP
  - documents divers, références et pointeurs web
  - informations pratiques, calendrier

Références



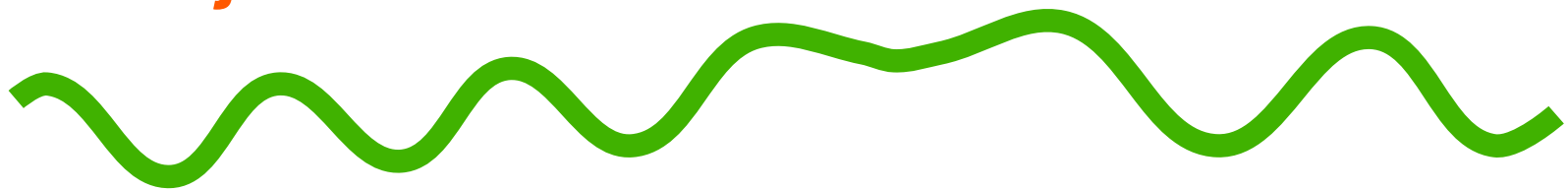
- Système d'exploitation
  - *Systèmes d'exploitation*, 3e ed.
    Andrew Tanenbaum
    Prentice Hall, 2007, trad. française Pearson Education France, 2008
- Programmation des systèmes d'exploitation
  - *Unix, programmation et communication*
    Jean-Marie Rifflet et Jean-Baptiste Yunès
    Dunod, 2003
  - *The Single Unix Specification*
    The Open Group
    `www.unix.org/single_unix_specification/`

Table des matières



Objectifs et organisation

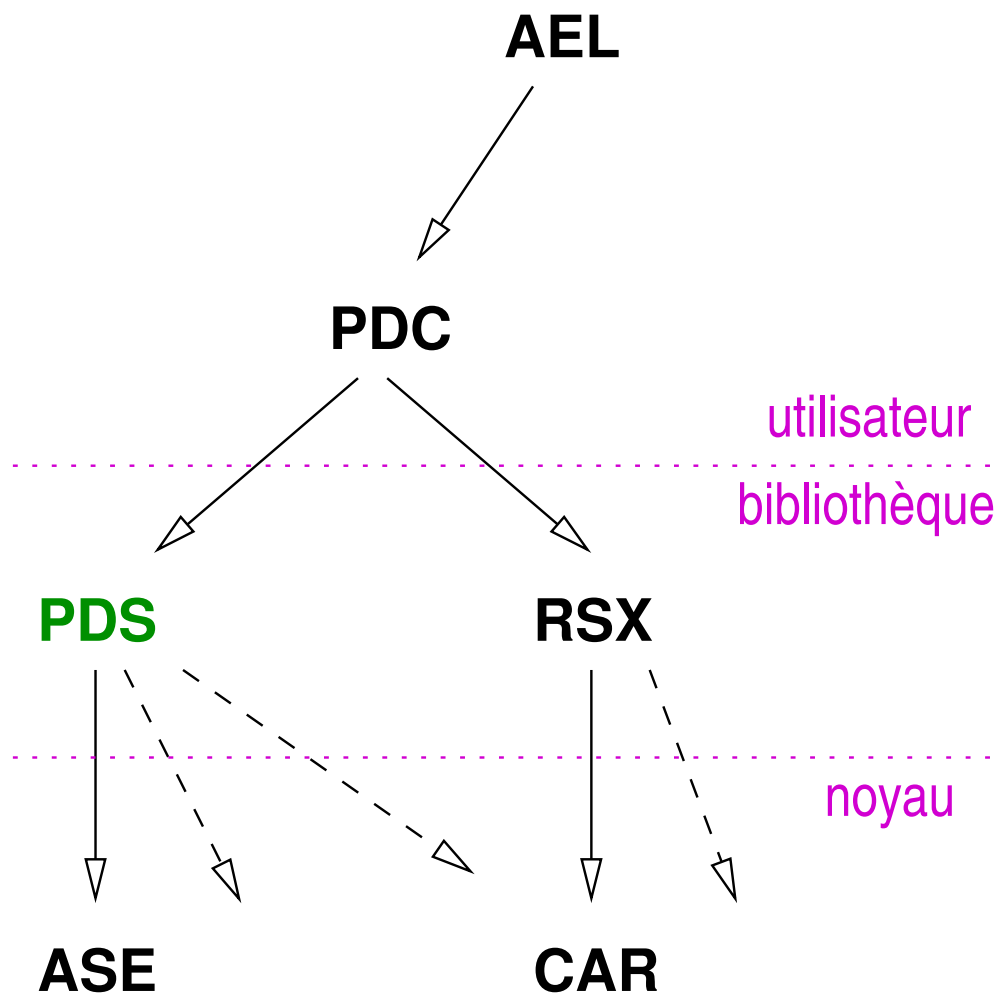
Objectif du cours



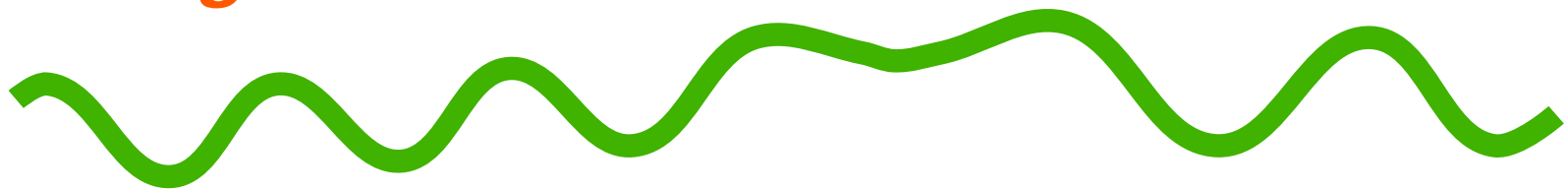
- Maîtrise des paradigmes de la programmation des systèmes d'exploitation
  - systèmes Unix, Linux, Windows...
- Étude des concepts fournis par l'interface des systèmes d'exploitation
  - fichier, système de fichiers, processus, communication inter-processus...
- Principe d'utilisation de l'interface système
  - interface normalisée POSIX
  - manipulations pratiques
- Pas de vue du fonctionnement interne du système
  - normalement !
  - cours de master ASE

Cursus architecture et système d'exploitation

- AEL — Architecture élémentaire, S3/S4 licence
- PDC — Pratique du C, S5 licence
- **PDS — Programmation des systèmes, S6 licence**
- RSX — Réseaux, S6 licence
- AEV — Architecture évoluée, M1 master
- ASE — Architecture et conceptions des systèmes d'exploitation, M1 master
- CAR — Construction d'applications réparties, M1 master
- options...

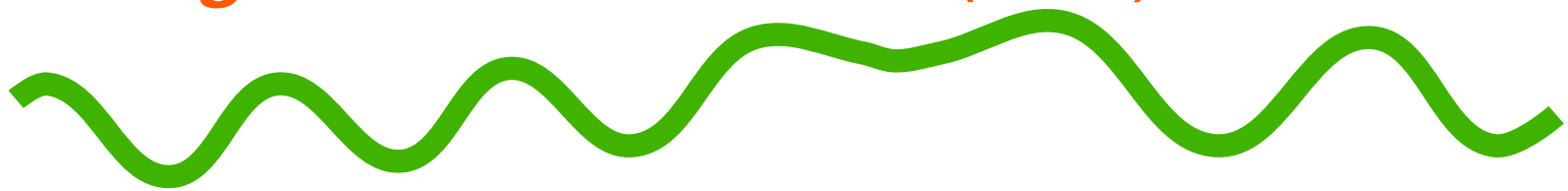


Organisation du cours



- Cours, TD et TP hebdomadaires
  - présence obligatoire
  - ce qui est vu en cours est supposé connu
  - mardis 13h30 + certains lundis 13h30
- Interaction enseignants / étudiants
  - questions bienvenues
  - pendant / après le cours, mail...
  - commentaires sur le cours
- Implication personnelle étudiant
  - manipulation : programmez !
  - documentation : surfez ! lisez !

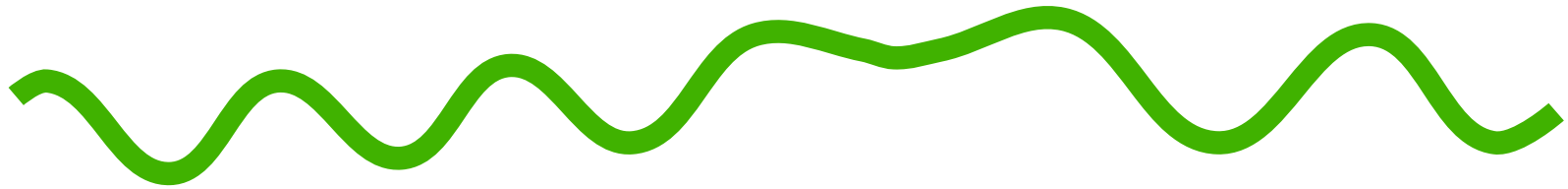
Organisation du cours (cont'd)



- Évaluation
  - examen final sur table, 3 heures, documents autorisés
  - ramassage de tous les TP via PROF sur
    `www.fil.univ-lille1.fr`
  - démonstration de TP en fin de semestre
  - correction de certains TP
- Note finale
  - pas de règle du sup

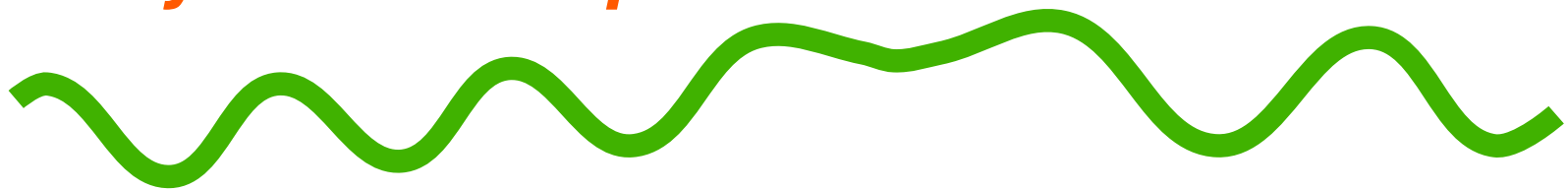
$$60\% \text{ Examen} + 40\% \text{ TP}$$

  - deux sessions d'examen, une seule "session" de TP



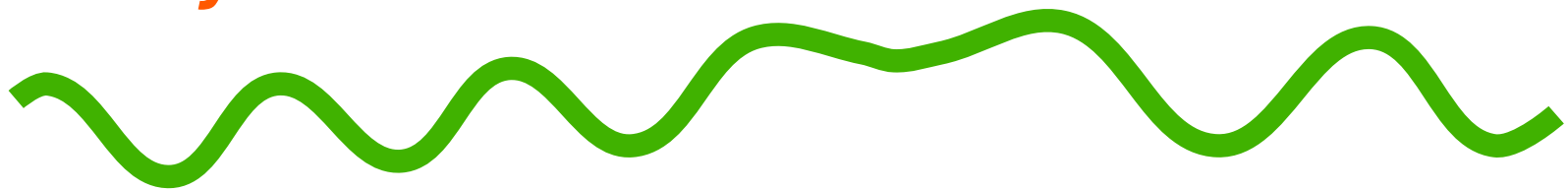
Interface avec le système d'exploitation

Système d'exploitation



- Système d'exploitation pour gérer le matériel
  - `read()` *vs* registre de commande du contrôleur d'entrées/sorties
- Système d'exploitation pour virtualiser le matériel
  - une machine pour chaque utilisateur
  - une machine pour chaque programme
- Système d'exploitation pour abstraire le matériel
  - une même abstraction de matériels différents
    ex. disque dur et mémoire Flash
  - une même abstraction de machines différentes
- Système d'exploitation pour protéger / sécuriser
  - protéger le matériel
    utilisation cohérente, en bon ordre, des commandes matérielles
  - protéger les autres programmes des erreurs de votre programme

Multiprogrammation et asynchronisme

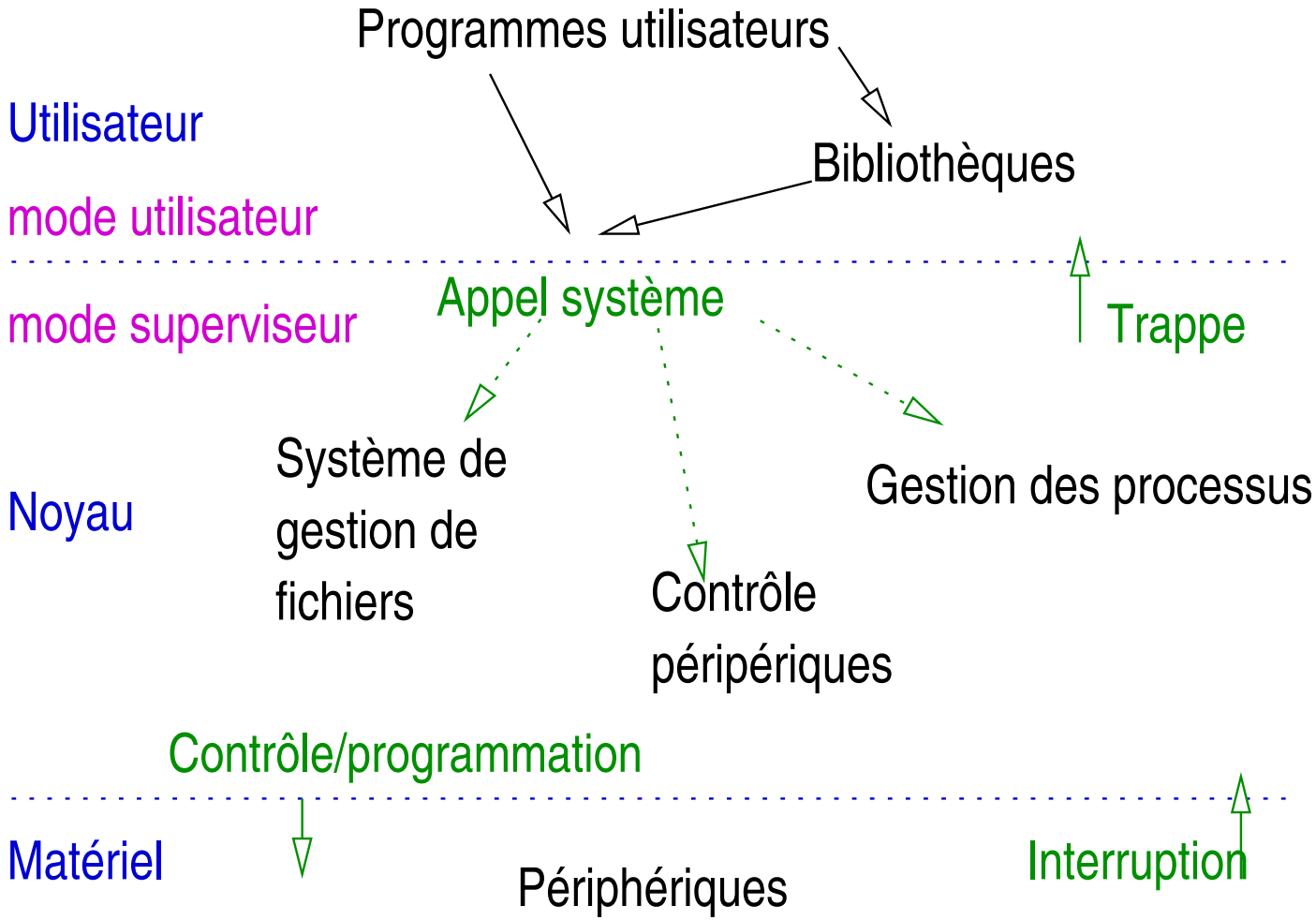


- Activités « simultanées »
  - progression pseudo parallèle
- Pas d'attente active
  - état bloqué
- Cycle
  - requête, attente = état bloqué, interruption matérielle, réveil
- Concurrence, synchronisation, communication

Modes d'exécution d'un processeur

- Utilisateur
- Superviseur

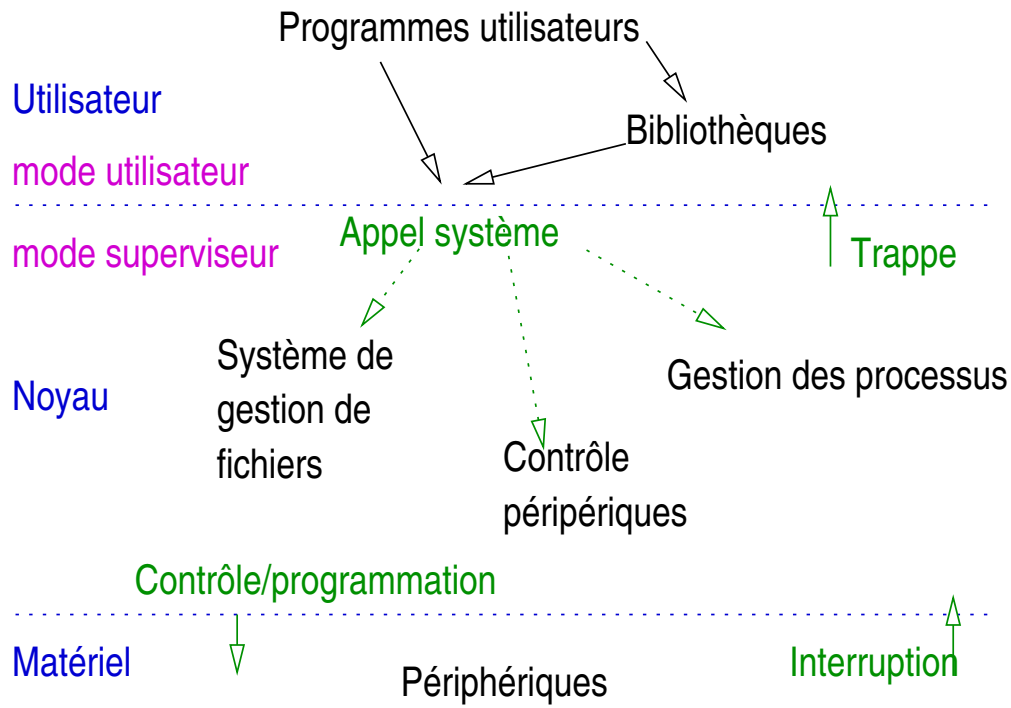
- Passage mode superviseur : appels système, trappes, IT



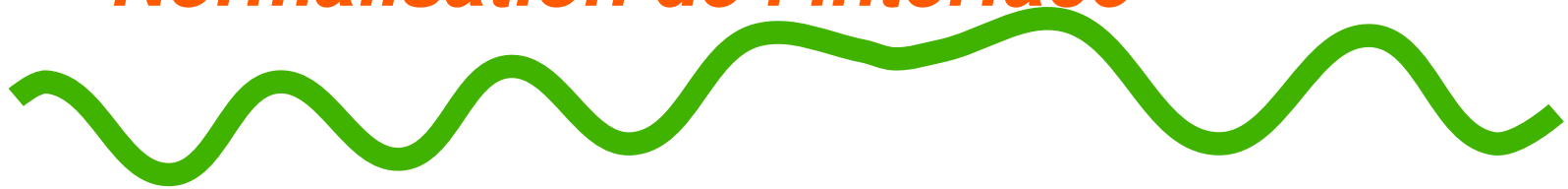
Structuration en couches d'un système d'exploitation

- Niveau utilisateur
  - applications utilisateurs
  - logiciel de base
  - bibliothèques système
  - appels de fonctions
- Niveau noyau
  - gestion des processus
  - système de fichiers
  - gestion de la mémoire
- Niveau matériel

- Système d'exploitation
  - bibliothèques système : PDS
  - noyau : ASE (master)



Normalisation de l'interface

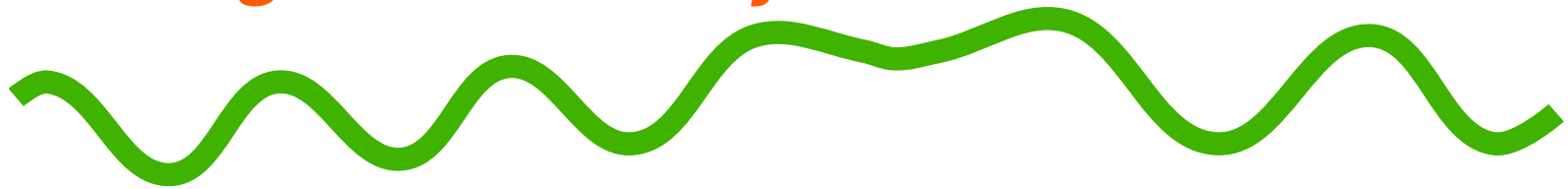


- Unix
  - système d'exploitation
  - Ken THOMPSON et Dennis RITCHIE, Bell Labs, 1969
  - distribution du code source
  - multiples versions (branches BSD, System III...)
- POSIX
  - *Portable Open System Interface eXchange*
  - *Portable Open System Interface X for Unix*
  - standard IEEE, 1985
  - interface standardisée des services fournis par le système
- Single Unix Specification, SUS
  - X/Open reprend les activités de normalisation POSIX, 1999
  - The Open Group (ex X/Open) propose *Single Unix Specification version 3*, 2001
  - `www.unix.org/version3/`

Fourniture de l'interface POSIX

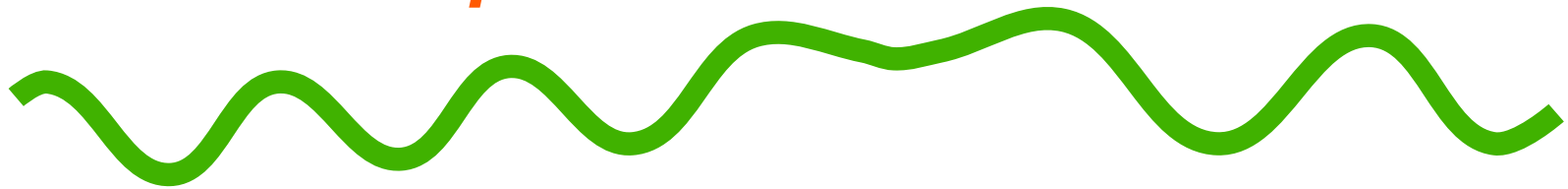
- Norme POSIX = interface d'utilisation du système
  - description des fonctions d'appel des services système fournis par le noyau
  - portabilité des applications
  - ne définit pas la construction du système d'exploitation, noyau
- POSIX et l'interface d'un système Unix
  - interface POSIX = l'interface du système !
  - interface native
  - une fonction POSIX = un appel système Unix
- POSIX et l'interface de systèmes propriétaires
  - Windows, VMS, Mach...
  - interface POSIX $\neq$ interface du système
  - bibliothèque niveau utilisateur au dessus des appels système
  - une fonction POSIX = un appel fonction bibliothèque utilisateur = un / multiples appels système

Programmation système en C



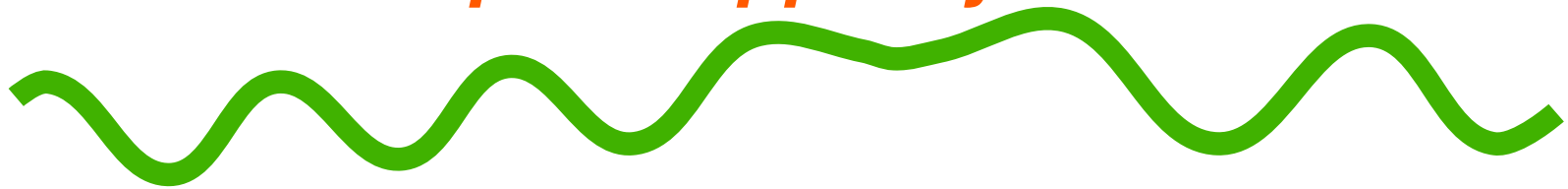
- Langage C pour programmer le système
  - sémantique claire
  - efficacité
  - accès à toutes les structures de la machine (registres, bits...)
  - allocation mémoire explicite
  - autres approches possibles : langage dédié
- Langage C interface naturelle avec le système
  - bibliothèques écrites en C
  - utilisation de la bibliothèque depuis le C
  - autres approches possibles : Java, OCaml

Bibliothèque C standard et POSIX



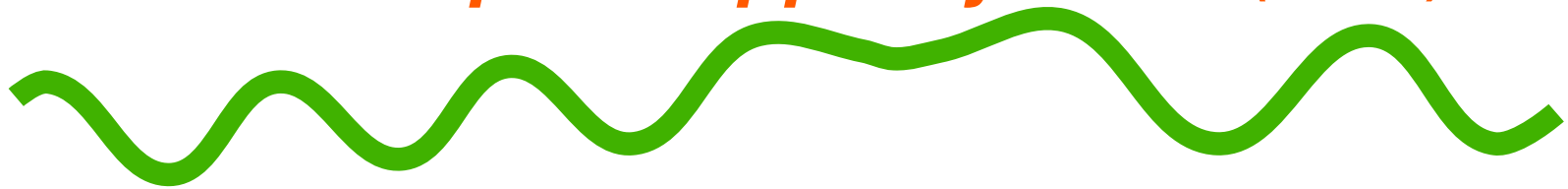
- Bibliothèque C
  - normalisation ISO du langage C
  - section 3 de `man`
    ```
    % man 3 malloc
    ```
- POSIX
  - normalisation Single Unix
  - section 2 de `man`
    ```
    % man 2 sbrk
    ```
- Confusion
  - historique commun
  - imbrication
  - définitions communes

Bibliothèque et appel système



- Appel système semblable à un appel de fonction de bibliothèque
  - comme des appels de fonctions C
- Appel système différent d'un appel de fonction de bibliothèque
  - appel système
    - pas d'édition de liens
    - exécution de code système
  - bibliothèque standard
    - abstraction de plus haut niveau
    - édition de liens avec la bibliothèque

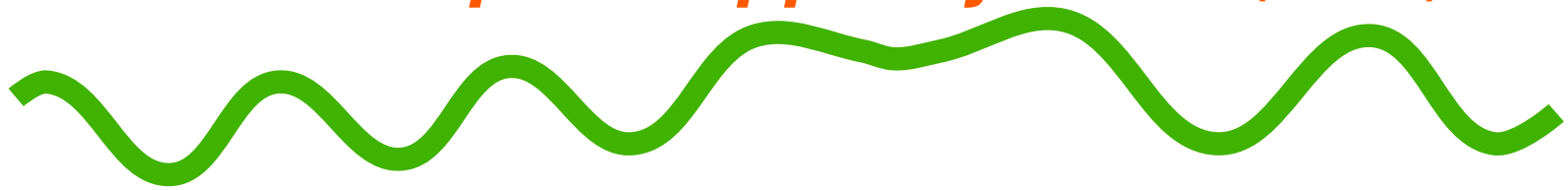
Bibliothèque et appel système (cont'd)



Appels système

- Manipulation du système de fichiers et entrées/sorties
- Gestion des processus
  - processus $\equiv$ exécution d'un programme
  - allocation de ressources pour les processus (mémoire...)
  - lancement, arrêt, ordonnancement des processus
- Communications entre processus
- Informations

Bibliothèque et appel système *(cont'd)*



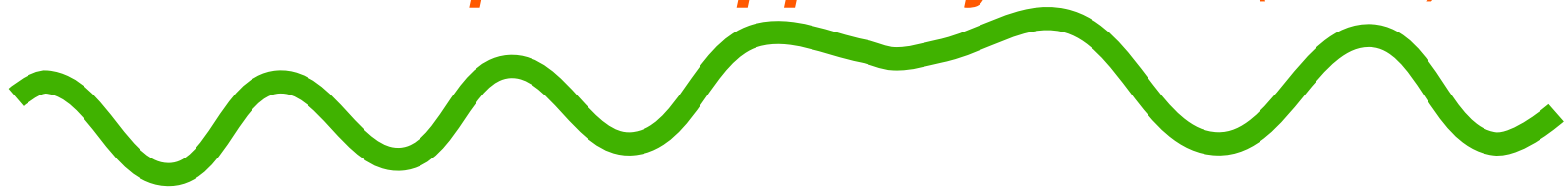
Terminaison d'un appel système

- Sémantique POSIX d'une primitive
  - comportement
  - y compris en cas d'erreur
  - liste des erreurs pouvant être retournées
  - voir le manuel `man`
- Valeur de retour d'un appel système
  - retourne -1 en cas d'erreur
  - positionne la variable globale `errno`
  - `perror()` produit une message décrivant la dernière erreur (appel système ou fonction bibliothèque)
- Exemple typique

```
if (creat(pathname, O_RDWR) == -1) {
    perror("Creation de mon fichier");
}
```

- Test systématique des retours des fonctions (laborieux...)

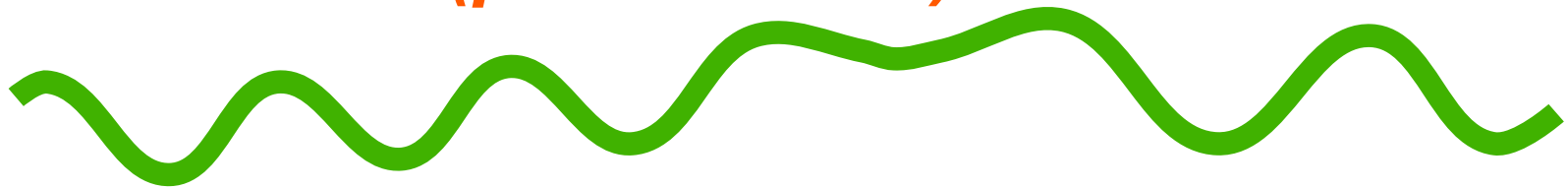
Bibliothèque et appel système (cont'd)



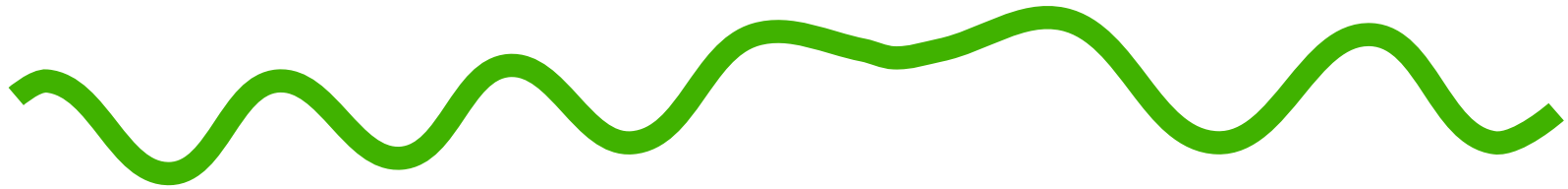
Bibliothèques standard

- Nombreuses bibliothèques
  - entrées/sorties formatées & bufferisées
  - localisation (français...)
  - fonctions mathématiques
  - allocation mémoire dynamique
  - etc.
- Abstraction de plus haut niveau
- Performance
  - nombre appels système réduits
  - exemple : allocation mémoire `malloc()`/`sbrk()`

Contenu (prévisionnel) du cours

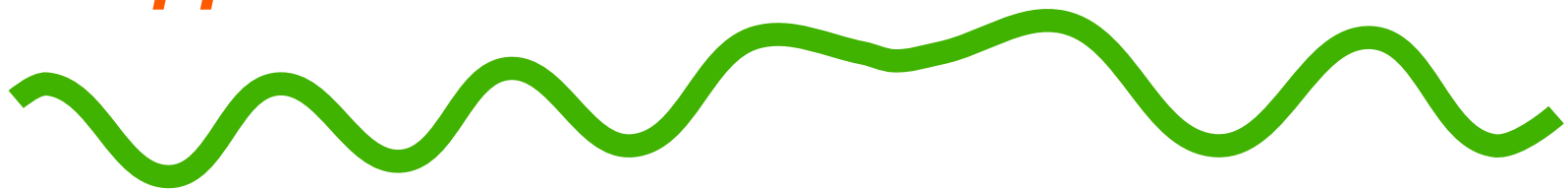


- Utilisation de la bibliothèque C standard et de l'API POSIX
  - interagir avec le système d'exploitation
- Entrées/sorties
  - (rappels) utilisation bibliothèque C
  - utilisation appels système
  - implantation de la bibliothèque au dessus des appels système
  - quelques éléments d'organisation d'un système de fichiers
- Processus
  - création, terminaison, interruptions, ordonnancement
- Communications inter-processus
  - signaux, pipes, sockets
- Mémoire
  - allocation, projection mémoire de périphériques, partage de mémoire
- Processus légers ou threads
  - création, synchronisation...



Interface avec l'environnement

Interface avec le programme appelant



- Programme = nouvelle commande
  - appel depuis une autre commande
  - depuis un shell

    ```
    % mprogram [arguments]...
    ```
- Accès aux arguments de la commande
  - `int main (int argc, char *argv[]);`

`argc`	nombre d'arguments +1
`argv[0]`	nom de la commande
`argv[1]` à `argv[argc]`	arguments

Interface avec le programme appelant *(cont'd)*

- Exemple, `mecho` : ma commande `echo`

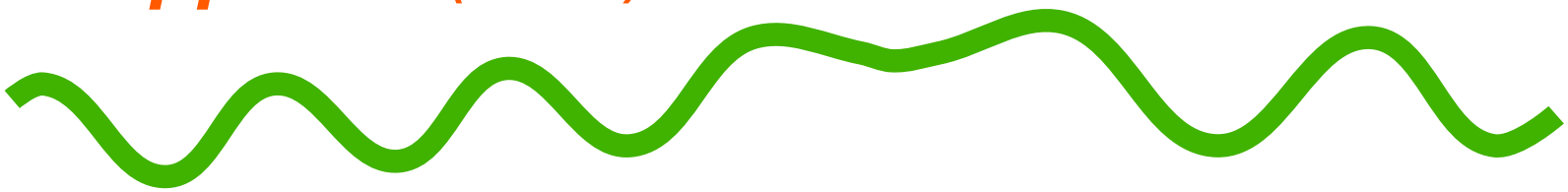
```
#include <stdio.h>
#include <stdlib.h>

int
main (int argc, char *argv[])
{
    int i;

    for(i = 1; i < argc ; i++) {
        printf("%s ", argv[i]);
    }
    putchar('\n');

    exit(EXIT_SUCCESS);
}
```

⇓

```
% ./mecho Hello $USER
Hello phm
```

- erreur : espace après le dernier argument
- pas d'option (`-n` : pas de retour à la ligne)

Interface avec le programme appelant *(cont'd)*



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

int
main (int argc, char *argv[])
{
    int i;
    int arg1 = 1;
    int println = TRUE;

    if (argc>1 && !strcmp(argv[1], "-n")) {
        arg1 = 2;
        println = FALSE;
    } else if (argc > 1 && *argv[1] == '-') {
        fprintf(stderr, "%s: invalid option %s\n",
                argv[0], argv[1]);
        exit(EXIT_FAILURE);
    }
    if (arg1 < argc)
        printf("%s", argv[arg1]);
    for(i = arg1+1; i < argc ; i++) {
        printf(" %s", argv[i]);
    }
    if (println)
        putchar('\n');

    exit(EXIT_SUCCESS);
}
```

```
% ./mecho Hello; ./mecho $USER
Hello
phm
% ./mecho -n Hello; ./mecho $USER
Hellophm
% ./mecho -n "Hello "; ./mecho $USE
Hello phm
% ./mecho -d Hello; ./mecho $USER
./mecho: invalid option -d
```

Accès à l'environnement

- Accès aux variables d'environnement (`$PATH`, `$USER`...)
  - forme générale de `main()`

    `int main (int argc, char *argv[], char **arge);`

    tableau, terminé par `NULL`, de chaînes de caractères de la forme

    `varname=value`
  - variable globale `environ`
  - exemple : mon `printenv`

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

extern char **environ;

int
main (int argc, char *argv[])
{
    char **envp = environ;

    while (*envp)
        printf("%s\n", *envp++);

    exit(EXIT_SUCCESS);
}
```

⇓

```
% ./mprintenv
USER=phm
PATH=/bin:/usr/local/bin:/usr/X1
EDITOR=emacs
...
```

Accès à l'environnement *(cont'd)*

Fonction POSIX `getenv()`

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int
main (int argc, char *argv[])
{
    char *username;

    username = getenv("USER");
    assert(username != NULL);

    printf("Hello %s\n", username);

    exit(EXIT_SUCCESS);
}
```

⇓

```
% ./echouser
Hello phm
```

Terminaison

- Un programme termine à la fin de `main()`
- Retourne une valeur à l'environnement
  - succès : `EXIT_SUCCESS`
  - échec : `EXIT_FAILURE`
  - fonction `exit()` de la bibliothèque C
  - qui fait appel à fonction POSIX `_exit()`
- Enregistrement de fonctions de terminaison par `atexit()`

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

void
bye(void)
{
    printf("A la semaine prochaine!\n");
}

int
main (int argc, char *argv[])
{
    atexit(bye);
    printf("Hello...\n");

    exit(EXIT_SUCCESS);
}
```

⇓

```
% ./atexit
Hello...
A la semaine prochaine!
```