Module SY5 – Systèmes d'Exploitation

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université Paris Cité
L3 Informatique & DL Bio-Info, Jap-Info, Math-Info
Année universitaire 2022-2023

Équipe enseignante

Responsable du cours : Dominique Poulalhon

dominique.poulalhon@irif.fr

Chargés de TD/TP

- Groupe INFO 1 (lundi 8h30) : Dominique Poulalhon
- Groupe INFO 2 (lundi 10h45) : Mirna Dzamonja mdzamonaj@irif.fr
- Groupe INFO 3 (vendredi 10h45) : Isabelle Fagnot fagnot@irif.fr
- Groupe INFO 4 (mardi 8h30) : Guillaume Geoffroy
- Groupe INFO 5 (mardi 14h) : guillaume.geoffroy@irif.fr
- Groupe MI (mercredi 16h15) : Anne Micheli anne.micheli@irif.fr

toujours mentionner [SY5] dans le sujet du mail, et bien sûr, signer...

Organisation

Enseignements

- CM : 2 heures par semaine, présence naturellement obligatoire
- TP ou (occasionnellement) TD : 2 heures par semaine, avec vérification de la présence

Communication

- principalement via un dépôt git (supports de cours, énoncés de TD/TP, de projet...) :

`gaufre.informatique.univ-paris-diderot.fr/poulalho/sy5-2022-2023`

- éventuellement moodle en solution de repli
- probablement un serveur discord, mais il n'est pas encore mis en place

Modalités de contrôle des connaissances

Session 1 : examen écrit + projet (en bi- ou trinômes) + partiel + assiduité
Session 2 : examen écrit

Prérequis

maîtrise du langage C, en particulier gestion de la mémoire (pointeurs, allocation dynamique), manipulations bit à bit (`|`, `&`, `>>`...)

utilisation courante d'UNIX via un shell (`bash` par exemple)

structures de données usuelles : tableaux, listes chaînées, tableaux dynamiques, tampons circulaires, arbres...

Quel est le rôle d'un système d'exploitation ?

faire l'*interface* entre le matériel et l'utilisateur (humain ou applications) $\Longrightarrow$ fournir le bon niveau d'*abstraction*

Quel est le rôle d'un système d'exploitation ?

faire l'*interface* entre le matériel et l'utilisateur (humain ou applications) $\Longrightarrow$ fournir le bon niveau d'*abstraction*

gérer les ressources, *équitablement et de façon sûre* : temps de calcul du processeur, mémoire, périphériques...

Quel est le rôle d'un système d'exploitation ?

faire l'*interface* entre le matériel et l'utilisateur (humain ou applications) $\Longrightarrow$ fournir le bon niveau d'*abstraction*

gérer les ressources, *équitablement et de façon sûre* : temps de calcul du processeur, mémoire, périphériques...

en particulier, il effectue des opérations *critiques* ; la plupart des ordinateurs ont deux modes opératoires : le mode *noyau*, où toutes les opérations sont permises, et le mode *utilisateur*, bridé.

Quel est le rôle d'un système d'exploitation ?

faire l'*interface* entre le matériel et l'utilisateur (humain ou applications) $\Longrightarrow$ fournir le bon niveau d'*abstraction*

gérer les ressources, *équitablement et de façon sûre* : temps de calcul du processeur, mémoire, périphériques...

en particulier, il effectue des opérations *critiques* ; la plupart des ordinateurs ont deux modes opératoires : le mode *noyau*, où toutes les opérations sont permises, et le mode *utilisateur*, bridé.

But du cours : comprendre (les grandes lignes de) la partie entre les deux traits, trait du haut inclus, trait du bas exclu, dans le cas des systèmes POSIX

Quelques mots sur le matériel

le processeur :

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter
- ensemble d'instructions spécifique agissant sur la mémoire et les registres

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter
- ensemble d'instructions spécifique agissant sur la mémoire et les registres
- parmi les registres : *compteur ordinal* (*program counter*), *pointeur de pile* (*stack pointer*)

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter
- ensemble d'instructions spécifique agissant sur la mémoire et les registres
- parmi les registres : *compteur ordinal* (*program counter*), *pointeur de pile* (*stack pointer*)
- changement de processus $\Longrightarrow$ sauvegarde des registres

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter
- ensemble d'instructions spécifique agissant sur la mémoire et les registres
- parmi les registres : *compteur ordinal* (*program counter*), *pointeur de pile* (*stack pointer*)
- changement de processus $\Longrightarrow$ sauvegarde des registres

la mémoire : plusieurs niveaux, caractérisés par des vitesses et des capacités (très) différentes...

Quelques mots sur le matériel

le processeur :

- cycle de base : extraire une instruction – la décoder – l'exécuter
- ensemble d'instructions spécifique agissant sur la mémoire et les registres
- parmi les registres : *compteur ordinal* (*program counter*), *pointeur de pile* (*stack pointer*)
- changement de processus $\Longrightarrow$ sauvegarde des registres

la mémoire : plusieurs niveaux, caractérisés par des vitesses et des capacités (très) différentes...

les périphériques d'entrée/sortie : en général, contrôleur + périphérique lui-même
nécessitent des *pilotes* (*device drivers*)

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

- *espace d'adressage* contenant le programme exécutable, ses données statiques, son tas, sa pile

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

- *espace d'adressage* contenant le programme exécutable, ses données statiques, son tas, sa pile
- registres... à sauvegarder en cas d'interruption

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

- *espace d'adressage* contenant le programme exécutable, ses données statiques, son tas, sa pile
- registres... à sauvegarder en cas d'interruption
- caractéristiques telles que identité, propriétaire, état, répertoire courant...

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

- *espace d'adressage* contenant le programme exécutable, ses données statiques, son tas, sa pile
- registres... à sauvegarder en cas d'interruption
- caractéristiques telles que identité, propriétaire, état, répertoire courant...
- $\Longrightarrow$ table des processsus

Quelques concepts importants

les processus : objet dynamique représentant un programme en cours d'exécution

- *espace d'adressage* contenant le programme exécutable, ses données statiques, son tas, sa pile
- registres... à sauvegarder en cas d'interruption
- caractéristiques telles que identité, propriétaire, état, répertoire courant...
- $\Longrightarrow$ table des processsus

mode utilisateur *vs* mode noyau : actions inoffensives *vs* actions sensibles $\Longrightarrow$ appel système pour passer de l'un à l'autre

Quelques concepts importants

mode utilisateur *vs* mode noyau : actions inoffensives *vs* actions sensibles $\Longrightarrow$ appel système pour passer de l'un à l'autre

Quelques concepts importants

mode utilisateur *vs* mode noyau : actions inoffensives *vs* actions sensibles $\Longrightarrow$ appel système pour passer de l'un à l'autre

appel système : syntaxiquement semblable à un appel de fonction, mais beaucoup plus lent à cause des changements de contexte

Quelques concepts importants

mode utilisateur *vs* mode noyau : actions inoffensives *vs* actions sensibles $\Longrightarrow$ appel système pour passer de l'un à l'autre

appel système : syntaxiquement semblable à un appel de fonction, mais beaucoup plus lent à cause des changements de contexte

documentés dans la section 2 du manuel (section 3 pour les fonctions C usuelles)

Quelques concepts importants

mode utilisateur *vs* mode noyau : actions inoffensives *vs* actions sensibles $\Longrightarrow$ appel système pour passer de l'un à l'autre

appel système : syntaxiquement semblable à un appel de fonction, mais beaucoup plus lent à cause des changements de contexte

documentés dans la section 2 du `manuel` (section 3 pour les fonctions C usuelles)

Sous linux, on peut voir la liste des appels système effectués par un processus grâce à la commande `strace`.

Quelques concepts importants

les fichiers : abstraction qui masque les particularités des disques de stockage et autres périphériques d'entrée/sortie

Quelques concepts importants

les fichiers : abstraction qui masque les particularités des disques de stockage et autres périphériques d'entrée/sortie

appels système nécessaires : pour la création, la destruction, les accès en lecture et écriture

Quelques concepts importants

les fichiers : abstraction qui masque les particularités des disques de stockage et autres périphériques d'entrée/sortie

appels système nécessaires : pour la création, la destruction, les accès en lecture et écriture

dans la plupart des systèmes, structuration de l'ensemble des fichiers grâce à la notion de répertoires, qui permet d'établir une *hiérarchie*

Appels système pour les entrées/sorties

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

renvoie un entier appelé descripteur de fichier, ou -1 en cas d'échec

Appels système pour les entrées/sorties

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

renvoie un entier appelé descripteur de fichier, ou -1 en cas d'échec

- table des descripteurs *du processus*
- table des fichiers ouverts *du système*

Appels système pour les entrées/sorties

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

renvoie un entier appelé descripteur de fichier, ou -1 en cas d'échec

- table des descripteurs *du processus*
- table des fichiers ouverts *du système*

vous connaissez déjà des descripteurs :

- 0 est le descripteur associé à l'entrée standard ;
- 1 est le descripteur associé à la sortie standard ;
- 2 est le descripteur associé à la sortie erreur standard.

note : ces trois descripteurs sont aussi définis par des macros dans `unistd.h` : `STDIN_FILENO`, `STDOUT_FILENO` et `STDERR_FILENO`.

Appels système pour les entrées/sorties

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

(les types `size_t` et `ssize_t` sont des entiers respectivement non signés et signés pour POSIX.1. Ils servent essentiellement à conserver la compatibilité entre les différentes versions de POSIX)

- `fd` est un descripteur
- `count` est la taille des données à lire ou écrire
- `buf` est l'adresse d'un emplacement mémoire pour stocker les données lues ou lire les données à écrire