

02A : Mécanismes d'entrée dans le noyau

420-3A6-ST : Programmation système

Dany Gagnon

Cégep de Saint-Jean-sur-Richelieu

Lundi 31 août 2026

Table des matières

- ➊ Observer les appels système
- ➋ Mécanismes matériels
- ➌ Terminer la machine virtuelle
- ➍ Atelier formatif

La question de la séance

Un programme, un périphérique et une instruction fautive
peuvent tous attirer l'attention du noyau.
Empruntent-ils le même chemin ?

But précis

À la fin de la séance, vous pourrez distinguer :

appel système

interruption

faute

Nous observerons une transition avec **strace**, puis nous terminerons l'environnement nécessaire au laboratoire.

Observer les appels système

Observer la demande avec strace

```
$ strace -e trace=write -o trace.txt ./bonjour  
Bonjour, système!
```

```
$ cat trace.txt  
write(1, "Bonjour, système!\n", 19) = 19  
+++ exited with 0 +++
```

Cette trace décrit ce qui s'est passé ici. Elle ne garantit pas que chaque appel à `puts` produit exactement un appel à `write`.

Lire la ligne de gauche à droite

```
write(1, "Bonjour, système!\n", 19) = 19
```

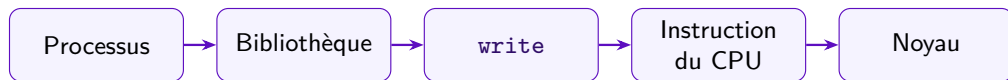
<code>write</code>	service demandé au noyau
<code>1</code>	descripteur de la sortie standard
chaîne	octets que le processus veut écrire
premier <code>19</code>	nombre d'octets demandés
second <code>19</code>	nombre d'octets réellement écrits

Pourquoi 19 octets ?

- les lettres et la ponctuation occupent des octets ;
- le caractère è est encodé sur plus d'un octet en UTF-8 ;
- `puts` ajoute le saut de ligne `\n`.

Le noyau reçoit des octets, pas une phrase avec une intention. La bibliothèque C peut accumuler ces octets avant de faire un appel système.

La chaîne de l'appel système



La fonction C cache les détails propres au processeur. `strace` nous laisse voir la demande et son résultat.

Mécanismes matériels

Interruption : le matériel demande l'attention

Une interruption matérielle peut signaler :

- une touche reçue ;
- un paquet réseau arrivé ;
- l'expiration d'une minuterie.

Le CPU sauvegarde l'état minimal nécessaire, passe en mode noyau et branche vers une routine préparée par le noyau. Le code d'entrée complète ensuite la sauvegarde du contexte.

Faute : l'instruction courante pose problème

Le CPU produit une faute lorsqu'il détecte, par exemple :

- une division par zéro ;
- une violation de privilège ;
- un accès mémoire interdit.

Le noyau reprend le contrôle et décide quoi faire du processus responsable.

Exemple : un accès mémoire interdit

Le fichier `faute.c` contient :

```
int main(void)
{
    int *adresse = (int *)0;
    return *adresse;
}
```

Déréférencer un pointeur nul produit un **comportement indéfini en C**.

Avec `-00`, dans l'environnement Linux du cours, nous nous attendons à une faute suivie de l'arrêt du processus.

Trois chemins vers le noyau

Événement	Origine	Exemple
Appel système	volontaire, processus	demander <code>write</code>
Interruption	externe, matériel	minuterie expirée
Faute	instruction courante	accès mémoire interdit

Les causes diffèrent, mais le matériel protège toujours l'entrée dans le noyau.

Pourquoi l'horloge compte

Sans minuterie programmable, un processus pris dans une boucle infinie pourrait garder le CPU.

- ① Le noyau programme une minuterie.
- ② Il laisse un processus s'exécuter.
- ③ La minuterie expire et produit une interruption.
- ④ Le noyau récupère le CPU.

Ce mécanisme permet le multitâche préemptif. Nous le reprendrons avec l'ordonnancement.

Une même idée pour la mémoire

Le matériel vérifie aussi chaque accès mémoire :

- le noyau configure les zones permises ;
- le processus accède à sa propre mémoire ;
- un accès interdit produit une faute ;
- le noyau reprend le contrôle.

Les modes du CPU, les fautes et la protection mémoire travaillent ensemble. Les détails viendront à la semaine 10.

Terminer la machine virtuelle

18. Démarrer le système installé

Après le redémarrage :

- ① retirez l'ISO du lecteur virtuel si elle est encore montée ;
- ② démarrez sur le disque virtuel ;
- ③ ouvrez la session créée ;
- ④ lancez l'application Terminal.

Si le menu d'installation revient, éteignez la VM et retirez l'ISO dans *Configuration* → *Stockage*.

19. Mettre Arch Linux à jour

```
$ sudo pacman -Syu
```

`pacman` est le gestionnaire de paquets d'Arch Linux.

- `-S` synchronise et installe ;
- `-y` actualise la liste des paquets ;
- `-u` met à jour les paquets installés.

20. Confirmer les outils du cours

Cette commande installe ce qui manquerait encore :

```
$ sudo pacman -S --needed base-devel gcc make gdb \  
    strace man-db man-pages git
```

L'option `-needed` évite de réinstaller les paquets déjà présents.

21. Vérifier l'environnement

```
$ gcc --version  
$ make --version  
$ gdb --version  
$ strace --version  
$ man 2 write
```

Quittez le manuel avec `q`.

Si chaque commande s'exécute, les outils nécessaires aux premières semaines sont présents.

22. Conserver un point de retour propre

- ❶ Fermez les applications dans Arch Linux.
- ❷ Éteignez la VM depuis le menu système.
- ❸ Dans VirtualBox, créez un instantané nommé `installation-propre`.

L'instantané permet de revenir à cet environnement si une manipulation future rend la VM inutilisable.

La VM est prête pour mercredi

- Arch Linux ouvre une session GNOME ;
- le réseau fonctionne ;
- `sudo`, GCC, Make, GDB, `strace` et `man` fonctionnent ;
- l'instantané `installation-propre` existe.

Mercredi, cette machine servira à naviguer dans l'arborescence et à lire la documentation du système.

Atelier formatif

Mission

À partir de deux petits programmes, vous allez observer :

- un appel système `write` qui réussit ;
- une faute causée par un accès mémoire interdit.

Aucune réponse écrite n'est demandée. Le but est de voir ces deux chemins pendant qu'ils se produisent.

Préparer les outils

Vérifiez d'abord l'environnement :

```
$ uname -a  
$ gcc --version  
$ strace --version
```

Si un outil manque :

```
$ sudo pacman -Syu  
$ sudo pacman -S --needed base-devel strace man-db man-pages
```

1. Compiler et exécuter

```
$ mkdir -p ~/cours/3a6/semaine01
$ cd ~/cours/3a6/semaine01
$ gcc -std=c17 -Wall -Wextra -Wpedantic \
    bonjour.c -o bonjour
$ ./bonjour
$ echo $?
```

Observez le résultat du programme et son code de sortie.

2. Voir seulement write

```
$ strace -e trace=write -o trace-write.txt ./bonjour  
$ cat trace-write.txt
```

Dans la ligne `write`, repérez :

- le numéro du descripteur ;
- les données que le processus veut écrire ;
- le nombre d'octets demandés ;
- la valeur retournée.

3. Utiliser directement l'enveloppe POSIX

Remplacez le programme par :

```
#include <unistd.h>

int main(void)
{
    const char message[] = "Bonjour, noyau!\n";
    ssize_t resultat = write(STDOUT_FILENO, message,
                             sizeof(message) - 1);
    return resultat == -1;
}
```

Compilez, exécutez et tracez ce programme de la même façon.

4. Constater le passage par le noyau

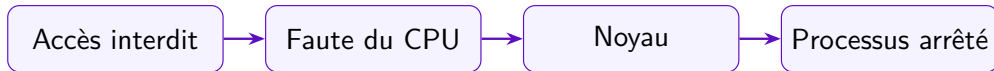
Relancez le programme avec `strace`, puis suivez une seule ligne `write` :

- ① les arguments partent du processus ;
- ② le noyau retourne un résultat ;
- ③ le programme reprend et se termine.

`strace` rend la frontière visible. Il n'affiche pas toutes les instructions exécutées dans le noyau.

5. Observer la faute sous Linux

```
$ gcc -O0 -std=c17 -Wall -Wextra -Wpedantic \  
    faute.c -o faute  
$ ./faute  
$ echo $?
```



Le message exact peut varier. La VM standardise l'expérience ; le programme ne menace pas le poste hôte.