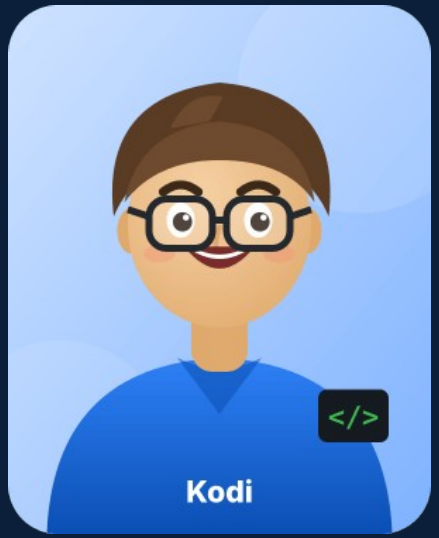
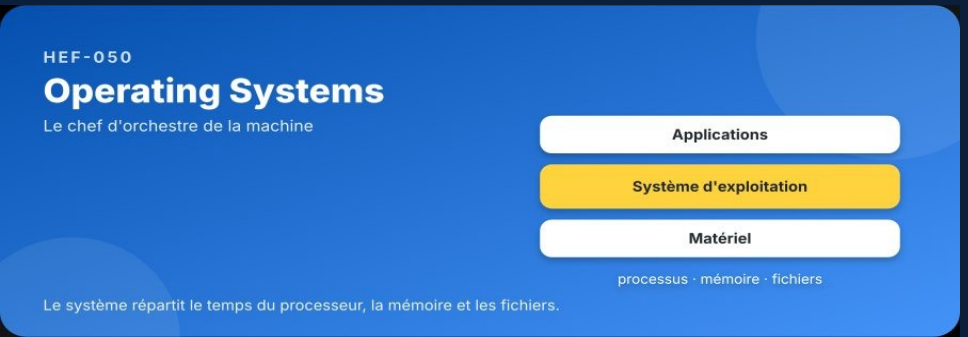
HEF-050 · MISSION 01 / 10

Le rôle d'un système d'exploitation

Entre tes programmes et le matériel, il y a un chef d'orchestre : le système d'exploitation.



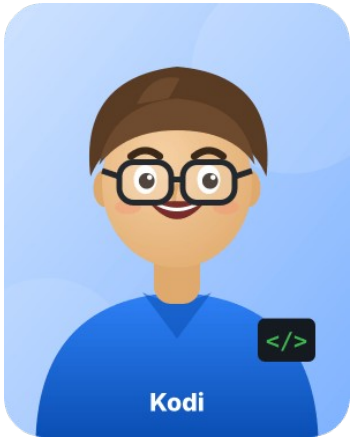


Badge : Directeur d'orchestre

65 min · intermédiaire

L'histoire

Kodi lance en même temps un navigateur, un éditeur, un lecteur de musique et un serveur de test. Chacun croit avoir toute la mémoire et tout le processeur pour lui. Pourtant, la machine ne plante pas.



> *« Quelqu'un organise tout ça, dit Sam. Il distribue le temps du processeur, la mémoire, l'accès aux disques, et il empêche chaque programme de marcher sur les plates-bandes des autres. C'est le système d'exploitation. »*

Dans cette mission, tu découvres ce que fait réellement un système d'exploitation et pourquoi on ne peut pas s'en passer.

Ta mission

À la fin, je serai capable de…

1. énoncer les trois rôles d'un système d'exploitation
2. distinguer noyau, espace utilisateur et applications
3. expliquer les modes utilisateur et noyau
4. expliquer ce qu'est un appel système et en donner des exemples
5. dire comment un programme demande un service au noyau

Défi d'échauffement

Un immeuble partage un seul ascenseur, une seule chaudière et une seule connexion internet entre 20 appartements. Quelles **règles** faut-il pour que ça marche (tour de rôle, droits d'accès, protection) ? Qui fait respecter ces règles ? Que se passerait-il sans règles ?

Le concept

Les trois rôles

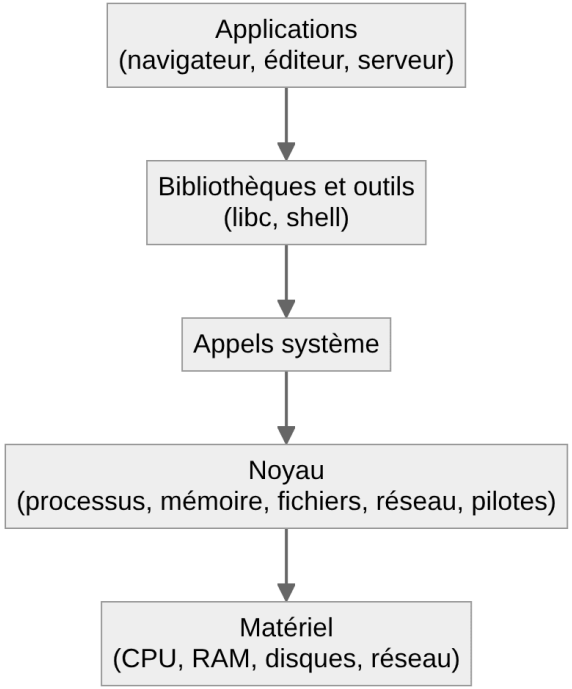
Rôle	Explication	Exemple
Gestionnaire de ressources	partage CPU, mémoire, disques, réseau entre les programmes	l'ordonnanceur donne le CPU à tour de rôle
Abstraction	cache la complexité du matériel derrière des interfaces simples	« fichier » au lieu de « secteurs du disque »
Protection	empêche un programme de nuire aux autres ou au système	un programme ne peut pas lire la mémoire d'un autre

Le noyau (kernel)

Le **noyau** est le cœur du système : il tourne avec tous les privilèges et contrôle le matériel. Autour de lui : les bibliothèques système, les outils (shell, commandes) et les applications.

Le concept

Le noyau (kernel) (suite)



Le concept

Le noyau (kernel) (suite)

Description du schéma : cinq couches empilées. Les applications passent par des bibliothèques et des outils, qui font des appels système vers le noyau. Le noyau dialogue avec le matériel.

Mode utilisateur et mode noyau

Le processeur a (au moins) deux niveaux de privilège :

Mode	Peut faire	Qui l'utilise
Utilisateur	calculer, accéder à sa propre mémoire	les applications
Noyau	tout : accéder au matériel, à toute la mémoire	le noyau uniquement

Une application ne peut pas lire un disque directement. Elle **demande** au noyau.

Le concept

L'appel système

Un **appel système** (*system call*) est la porte d'entrée vers le noyau. Le programme prépare une demande, exécute une instruction spéciale, le processeur passe en mode noyau, le noyau fait le travail, puis rend la main.

Besoin	Appel système typique (Linux)
Ouvrir un fichier	`open`
Lire / écrire	`read`, `write`
Créer un processus	`fork`, `execve`
Allouer de la mémoire	`mmap`, `brk`
Obtenir l'identifiant du processus	`getpid`

Le concept

L'appel système (suite)

Besoin	Appel système typique (Linux)
Se terminer	`exit`

Chaque appel système a un **coût** (changement de mode), ce qui explique l'intérêt du regroupement vu dans la mission « Stockage » du module précédent.

Les systèmes courants

Linux, macOS et Windows ont des noyaux différents mais des rôles identiques. Linux est un noyau **libre**, utilisé dans la plupart des serveurs et dans Android.

Observer un appel système en Python

Le concept

Observer un appel système en Python (suite)

```
import os

print("PID du processus :", os.getpid())           # appel système getpid
print("PID du parent    :", os.getppid())          # appel système getppid
print("Répertoire courant :", os.getcwd())         # appel système getcwd
print("Système :", os.name)
```

Sous Linux, l'outil `strace` permet de **voir** les appels système d'un programme :

```
strace -c python3 -c "print('bonjour')"
```

🧠 ***Astuce*** *: quand un programme est « lent », se demander combien d'appels système il fait et ce qu'ils coûtent.*

Exemple pas à pas

Problème : comprendre pourquoi une simple écriture traverse plusieurs couches.

Le programme Python :

```
import os, tempfile

with tempfile.TemporaryDirectory() as d:
    with open(os.path.join(d, "note.txt"), "w") as f:
        f.write("Bonjour le noyau")
    print(os.path.getsize(os.path.join(d, "note.txt")), "octets écrits")
```

Ce qui se passe, couche par couche :

1. Le **programme Python** appelle `open` et `write` (fonctions de la bibliothèque standard).
2. La **bibliothèque C** traduit ces demandes en **appels système** `openat` et `write`.
3. Le processeur passe en **mode noyau**.

Exemple pas à pas (suite)

4. Le **noyau** vérifie les droits, trouve où écrire dans le **système de fichiers**, et demande au **pilote** du disque d'écrire.
5. Le noyau repasse en mode utilisateur et le programme continue.

À retenir : le programme n'a jamais touché le disque. Il n'en a pas le droit. Il a seulement fait des **demandes** que le noyau a vérifiées et exécutées.

```
import os, time

debut = time.perf_counter()
for _ in range(100_000):
    os.getpid()
duree = time.perf_counter() - debut
print(f"{duree / 100_000 * 1e9:.0f} ns par appel (ordre de grandeur)")
```

À toi, avec aide

Voici cinq actions. Pour chacune, dis si elle nécessite un appel système et lequel (de façon générale) :

Action	Appel système ?	Lequel ?
Additionner deux variables	?	?
Lire un fichier	?	?
Afficher du texte à l'écran	?	?
Demander la liste des fichiers d'un dossier	?	?
Calculer la racine carrée d'un nombre	?	?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Une opération purement calculatoire se fait entièrement dans le CPU.

Indice 2

Tout ce qui touche un fichier, l'écran ou le réseau passe par le noyau.

Indice 3

Lister un dossier demande au noyau de lire la structure du système de fichiers.

À toi, avec aide · réponse

Réponse

Action	Appel système ?	Lequel ?
Additionner deux variables	non	: (calcul pur)
Lire un fichier	oui	`open`, `read`, `close`
Afficher du texte	oui	`write` (vers la sortie standard)
Lister un dossier	oui	`openat` + `getdents` (sous Linux)
Racine carrée	non	: (instruction du CPU ou bibliothèque)

À toi, en solo

Écris un programme qui **mesure le coût d'un appel système** (par exemple `os.getpid`) et le compare au coût d'une opération simple (une addition) :

1. mesure 200 000 appels de chaque ;
2. affiche le rapport entre les deux ;
3. explique le résultat en 3 phrases en utilisant les notions de mode utilisateur et noyau.

À toi, en solo · réponse

Pistes de correction

```
import os, time

N = 200_000

t = time.perf_counter()
for _ in range(N):
    os.getpid()
appel = (time.perf_counter() - t) / N

t = time.perf_counter()
x = 0
for _ in range(N):
    x = x + 1
addition = (time.perf_counter() - t) / N

print(f"appel système : {appel * 1e9:.0f} ns ; addition : {addition * 1e9:.0f} ns
   ; rapport ≈ {appel / addition:.1f}")
```

Pistes de correction (suite)

L'appel système doit changer de mode (utilisateur → noyau → utilisateur), ce qui coûte plus qu'une addition. Le rapport mesuré en Python est modeste car l'interpréteur Python ajoute lui-même beaucoup de travail à chaque opération ; en langage compilé, l'écart est bien plus grand. L'enseignement reste : regrouper les appels système quand c'est possible.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Confondre OS et noyau	L'OS inclut aussi outils et bibliothèques	Noyau = cœur, OS = ensemble
Croire qu'un programme accède au matériel	Il passe toujours par le noyau	Penser en « demandes »
Ignorer le coût des appels système	Beaucoup de petits appels = lent	Regrouper les opérations
Mélanger OS et distribution	Linux est un noyau, Ubuntu une distribution	Noyau, système, distribution

Mini-boss

Mini-boss

Carte d'identité d'une machine. Écris un script qui affiche : le système, la version du noyau, l'architecture, le nombre de cœurs logiques, le PID du script et celui de son parent, et le nom du processus parent (si disponible sous Linux via `/proc`).

Puis rédige un schéma Mermaid de ce qui se passe quand ton script lit son propre code source (couches et appels système), avec sa description en texte.

Mini-boss · réponse

Pistes de correction

```
import os, platform

print("Système       :", platform.system(), platform.release())
print("Architecture  :", platform.machine())
print("Cœurs logiques:", os.cpu_count())
print("PID / parent  :", os.getpid(), os.getppid())
try:
    with open(f"/proc/{os.getppid()}/comm") as f:
        print("Processus parent :", f.read().strip())
except OSError:
    print("Processus parent : information non disponible sur ce système")
```

Le schéma doit montrer : script Python → bibliothèque → appels système `openat`/`read`/`close` → noyau (système de fichiers, cache de pages) → disque. Il doit être accompagné d'une phrase décrivant chaque flèche.

Quiz éclair

1. Le noyau est :

A. le cœur du système qui contrôle le matériel
B. une application
C. un type de fichier

2. Un appel système permet :

A. d'éteindre l'ordinateur
B. à un programme de demander un service au noyau
C. d'écrire du code plus vite

3. En mode utilisateur, un programme :

A. peut tout faire

Quiz éclair

Quiz éclair (suite)

B. est plus rapide
C. ne peut pas accéder directement au matériel

4. 🔁 *Révision HEF-040, mission 09* : Un pipeline permet de :

A. traiter plusieurs instructions à des étapes différentes
B. augmenter la taille du cache
C. réduire la RAM

5. 🔁 *Révision HEF-040, mission 07* : Une interruption est :

A. une panne
B. un signal d'un périphérique vers le CPU
C. un type de mémoire

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

1. Quelles « règles de vie en commun » du système d'exploitation retrouves-tu dans ta vie (tour de parole, droits d'accès) ?
2. Pourquoi est-il bon qu'un programme défectueux ne puisse pas tout casser ?

À retenir

À retenir

1. L'OS gère les ressources, abstrait le matériel et protège.
2. Le noyau tourne en mode privilégié.
3. Les applications demandent via des appels système.
4. Chaque appel système a un coût.
5. Linux, macOS, Windows : mêmes rôles, noyaux différents.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Directeur d'orchestre

Prochaine mission : Les processus

