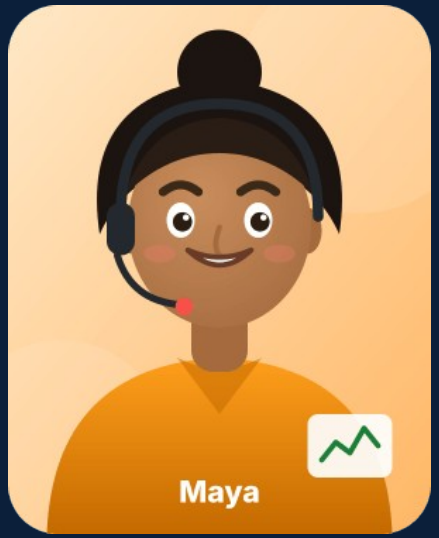
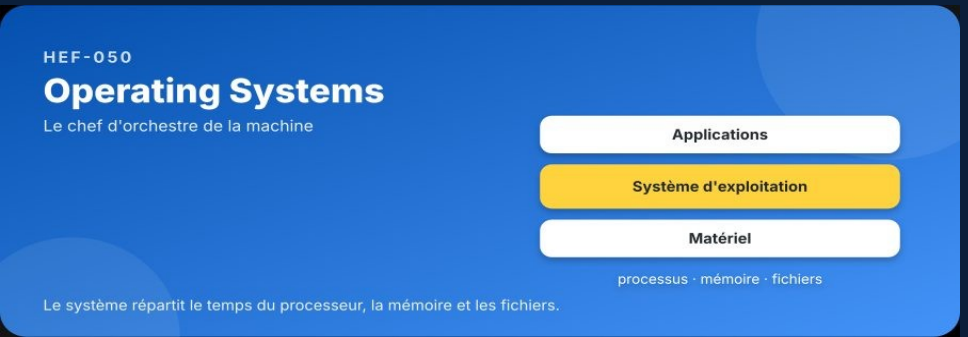
HEF-050 · MISSION 06 / 10

Les systèmes de fichiers

Un fichier n'est pas ce que tu crois : c'est un nom, un numéro et des blocs de données.





Badge : Archiviste

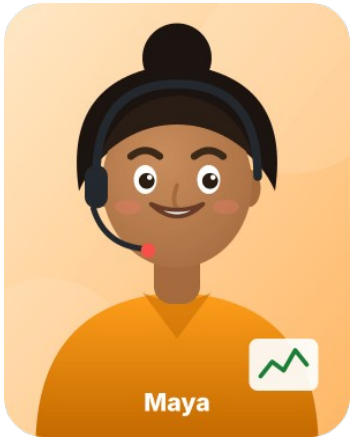
75 min · intermédiaire

L'histoire

Kodi supprime un fichier de 5 Go : le disque ne se vide pas. Il supprime un autre fichier : l'espace est libéré. Il crée un « raccourci » et s'étonne qu'il ne fonctionne plus quand le fichier d'origine est déplacé, alors qu'une autre copie, elle, résiste.

> *« Les fichiers, les noms et les raccourcis sont trois choses différentes, dit Sam. Regardons ce qu'il y a derrière. »*

Tu comprends comment le noyau organise et retrouve tes données sur le disque.



Ta mission

À la fin, je serai capable de…

1. décrire inode, bloc de données, répertoire et nom de fichier
2. distinguer lien physique (hard link) et lien symbolique
3. expliquer pourquoi supprimer un fichier ne libère pas toujours l'espace
4. expliquer la journalisation (journaling)
5. lire les métadonnées d'un fichier avec `os.stat`

Défi d'échauffement

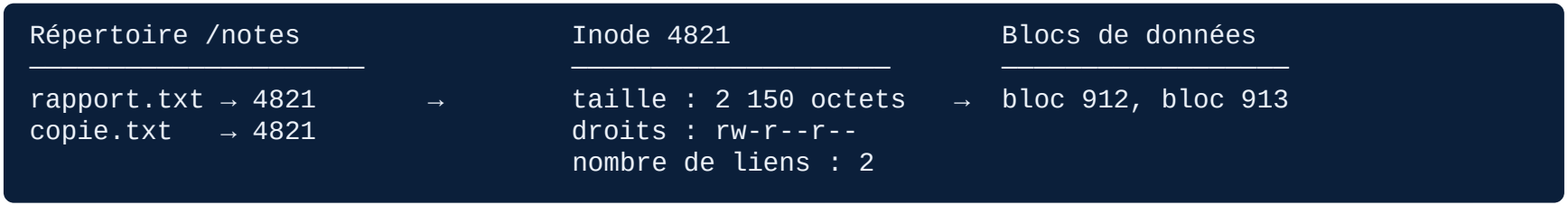
Défi d'échauffement

Une bibliothèque a un **catalogue** (titre → numéro de rayonnage) et des **rayonnages** (livres). Deux fiches du catalogue peuvent pointer vers le même livre. Si tu retires **une** fiche, le livre disparaît-il ? Quand le livre peut-il être jeté ? Et si une fiche indique « voir le titre X » au lieu d'un rayonnage, que se passe-t-il quand le titre X est renommé ?

Le concept

Les couches d'un fichier

Élément	Rôle
Bloc de données	contient le contenu (par morceaux de 4 Ko, par exemple)
Inode	la « fiche d'identité » : taille, propriétaire, droits, dates, emplacement des blocs. **Pas le nom !**
Répertoire	un fichier spécial qui associe des **noms** à des **numéros d'inode**



Le **nom** appartient au répertoire, pas au fichier.

Le concept

Lien physique et lien symbolique

	Lien physique (*hard link*)	Lien symbolique (*symlink*)
Nature	un deuxième nom pour le **même inode**	un fichier spécial qui contient un **chemin**
Si l'original est supprimé	le fichier existe toujours	le lien est **cassé**
Entre systèmes de fichiers	impossible	possible
Sur un dossier	généralement interdit	possible

Pourquoi l'espace n'est pas libéré ?

Le contenu est supprimé quand **le nombre de liens tombe à 0 ET qu'aucun processus ne garde le fichier ouvert**. Un fichier de log supprimé alors qu'un serveur l'écrit encore continue d'occuper le disque.

Le concept

Les métadonnées avec Python

```
import os, stat, tempfile

with tempfile.TemporaryDirectory() as d:
    chemin = os.path.join(d, "note.txt")
    with open(chemin, "w") as f:
        f.write("bonjour")
    s = os.stat(chemin)
    print("inode :", s.st_ino)
    print("taille :", s.st_size, "octets")
    print("droits :", stat.filemode(s.st_mode))
    print("liens :", s.st_nlink)
```

Journalisation

Le concept

Journalisation (suite)

Une écriture touche plusieurs structures (données, inode, répertoire). Une coupure de courant au milieu laisse un système incohérent. Un **système de fichiers journalisé** (ext4, NTFS, APFS…) note d'abord l'opération dans un **journal** : au redémarrage, il la rejoue ou l'annule. Cela évite de longues vérifications.

L'arborescence et les points de montage

L'ensemble des fichiers forme un arbre unique partant de `/`. Chaque disque ou partition est **monté** à un endroit de l'arbre (par exemple `/mnt/donnees`).

> 🧠 ***Astuce** : un « fichier » est un **nom** (dans un répertoire) vers un **inode** vers des **blocs**.*

Exemple pas à pas

Problème : observer la différence entre lien physique et lien symbolique, avec un fichier supprimé.

```
import os, tempfile

with tempfile.TemporaryDirectory() as d:
    original = os.path.join(d, "original.txt")
    dur = os.path.join(d, "lien_dur.txt")
    symbolique = os.path.join(d, "lien_sym.txt")

    with open(original, "w") as f:
        f.write("contenu important")

    os.link(original, dur)                # lien physique
    os.symlink(original, symbolique)      # lien symbolique

    print("même inode (original / lien dur) :", os.stat(original).st_ino == os.sta
   t(dur).st_ino)
    print("nombre de liens :", os.stat(original).st_nlink)
```

Exemple pas à pas (suite)

```
os.remove(original)                    # on supprime le nom d'origine

print("lien dur lisible :", open(dur).read())
print("lien symbolique cassé :", not os.path.exists(symbolique))
```

Lecture :

- `original.txt` et `lien_dur.txt` sont **deux noms pour le même inode** : le nombre de liens vaut 2.
- Après `os.remove(original)`, le contenu **existe toujours**, car il reste un nom.
- Le lien symbolique stocke seulement le chemin `original.txt` : une fois supprimé, il pointe dans le vide.

Fichier supprimé mais ouvert :

Exemple pas à pas (suite)

```
import os, tempfile

with tempfile.TemporaryDirectory() as d:
    chemin = os.path.join(d, "log.txt")
    f = open(chemin, "w")
    f.write("x" * 1000)
    f.flush()
    os.remove(chemin)                       # le nom disparaît
    print(os.path.exists(chemin))           # False
    f.write("encore")                       # mais le processus peut continuer à éc
   rire
    f.close()                               # le contenu est libéré ici
```

À toi, avec aide

Crée dans un dossier temporaire un fichier `a.txt`, un lien physique `b.txt` et un lien symbolique `c.txt`. Pour chacun, affiche l'inode et le nombre de liens. Puis modifie le contenu via `b.txt` et vérifie que `a.txt` et `c.txt` montrent le nouveau contenu.

À toi, avec aide · réponse

Indices progressifs

Indice 1

`os.link(src, dst)` et `os.symlink(src, dst)`.

Indice 2

`os.stat(chemin).st_ino` pour l'inode ; `os.lstat` ne suit pas les liens symboliques.

Indice 3

Écrire dans `b.txt` modifie les blocs de données partagés par `a.txt`.

À toi, avec aide · réponse

Réponse

```
import os, tempfile

with tempfile.TemporaryDirectory() as d:
    a, b, c = (os.path.join(d, n) for n in ("a.txt", "b.txt", "c.txt"))
    open(a, "w").write("v1")
    os.link(a, b)
    os.symlink(a, c)
    for nom, p in (("a", a), ("b", b), ("c (suit le lien)", c)):
        s = os.stat(p)
        print(nom, "inode", s.st_ino, "liens", s.st_nlink)
    print("c lui-même (lstat) :", os.lstat(c).st_ino)
    with open(b, "w") as f:
        f.write("v2")
    print(open(a).read(), open(c).read())    # v2 v2
```

`a` et `b` ont le même inode et 2 liens. `c` est un autre inode (le lien lui-même) mais `os.stat(c)` suit le lien et montre l'inode de `a`.

À toi, en solo

Détective de doublons. Écris `trouver_doublons(dossier)` qui parcourt un dossier (récursivement) et renvoie des groupes de chemins qui correspondent au **même contenu**. Pour être efficace : regroupe d'abord par taille, puis calcule une empreinte (`hashlib.sha256`) seulement pour les fichiers de même taille. Distingue deux cas : (a) copies indépendantes (inodes différents) ; (b) liens physiques (même inode).

À toi, en solo · réponse

Pistes de correction

```
import hashlib, os, tempfile
from collections import defaultdict

def empreinte(chemin):
    h = hashlib.sha256()
    with open(chemin, "rb") as f:
        for bloc in iter(lambda: f.read(65536), b""):
            h.update(bloc)
    return h.hexdigest()

def trouver_doublons(dossier):
    par_taille = defaultdict(list)
    for racine, _, fichiers in os.walk(dossier):
        for nom in fichiers:
            p = os.path.join(racine, nom)
            if not os.path.islink(p):
                par_taille[os.path.getsize(p)].append(p)
    groupes = []
    for taille, chemins in par_taille.items():
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        if len(chemins) < 2:
            continue
        par_hash = defaultdict(list)
        for p in chemins:
            par_hash[empreinte(p)].append(p)
        for liste in par_hash.values():
            if len(liste) > 1:
                inodes = {os.stat(p).st_ino for p in liste}
                groupes.append((liste, "liens physiques" if len(inodes) == 1 else
   "copies"))
    return groupes

with tempfile.TemporaryDirectory() as d:
    for nom, contenu in (("a.txt", "x"), ("b.txt", "x"), ("c.txt", "yy")):
        open(os.path.join(d, nom), "w").write(contenu)
    os.link(os.path.join(d, "a.txt"), os.path.join(d, "lien.txt"))
    for g in trouver_doublons(d):
        print([os.path.basename(p) for p in g[0]], g[1])
```

Pistes de correction (suite)

Comparer d'abord la taille évite de calculer des empreintes inutiles : c'est le même principe que filtrer d'abord par une clé peu coûteuse.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Croire que le nom est dans le fichier	Confusion entre lien dur et copie	Nom = répertoire ; données = inode
Supprimer un log encore ouvert	L'espace n'est pas libéré	Redémarrer ou tronquer le fichier
Lien symbolique relatif cassé	Le lien ne résout plus après déplacement	Utiliser des chemins absolus ou vérifier
Débrancher pendant une écriture	Système incohérent sans journal	Systèmes journalisés, arrêt propre

Mini-boss

Mini-boss

Mini système de fichiers en mémoire. Implémente `MiniFS` avec :

- des **inodes** (dictionnaire numéro → {contenu, nlink}) ;
- des **répertoires** (dictionnaire nom → numéro d'inode) ;
- `creer(chemin, contenu)`, `lire(chemin)`, `lier(source, destination)` (lien physique), `supprimer(chemin)` ;
- le contenu n'est effacé que lorsque `nlink` tombe à 0.

Teste : créer un fichier, lui faire un lien, supprimer l'original, relire via le lien, supprimer le lien et vérifier que l'inode a disparu.

Mini-boss · réponse

Pistes de correction

```
class MiniFS:
    def __init__(self):
        self.inodes = {}          # numéro -> {"contenu": str, "nlink": int}
        self.repertoire = {}      # chemin -> numéro d'inode
        self._suivant = 1

    def creer(self, chemin, contenu):
        if chemin in self.repertoire:
            raise FileExistsError(chemin)
        n = self._suivant; self._suivant += 1
        self.inodes[n] = {"contenu": contenu, "nlink": 1}
        self.repertoire[chemin] = n

    def lire(self, chemin):
        return self.inodes[self.repertoire[chemin]]["contenu"]

    def lier(self, source, destination):
        n = self.repertoire[source]
        self.repertoire[destination] = n
```

Mini-boss · réponse

Pistes de correction (suite)

```
        self.inodes[n]["nlink"] += 1

    def supprimer(self, chemin):
        n = self.repertoire.pop(chemin)
        self.inodes[n]["nlink"] -= 1
        if self.inodes[n]["nlink"] == 0:
            del self.inodes[n]

fs = MiniFS()
fs.creer("/a", "secret")
fs.lier("/a", "/b")
fs.supprimer("/a")
assert fs.lire("/b") == "secret" and len(fs.inodes) == 1
fs.supprimer("/b")
assert len(fs.inodes) == 0
print("ok")
```

Mini-boss · réponse

Pistes de correction (suite)

Extension possible : ajouter un compteur de « processus qui ont le fichier ouvert » pour reproduire le comportement des fichiers supprimés encore ouverts.

Quiz éclair

Quiz éclair

1. Un inode contient :

A. le nom du fichier
B. les métadonnées du fichier (taille, droits, emplacement)
C. le contenu en clair

2. Un lien physique :

A. contient un chemin
B. ne peut exister qu'une fois
C. est un deuxième nom pour le même inode

3. Un lien symbolique devient cassé quand :

A. le fichier cible est supprimé ou déplacé

Quiz éclair

Quiz éclair (suite)

B. on le lit
C. on le copie

4. 🔁 *Révision mission 05* : La table des pages sert à :

A. stocker les fichiers
B. traduire les adresses virtuelles en adresses physiques
C. chiffrer la mémoire

5. 🔁 *Révision mission 05* : Le thrashing est :

A. un virus
B. un type de cache
C. un système qui passe son temps à échanger des pages

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quel équivalent des « liens physiques » et « symboliques » trouves-tu dans ta vie (deux adresses pour un même lieu, un panneau « voir ailleurs ») ?
2. Pourquoi la journalisation est-elle précieuse en cas de coupure de courant ?

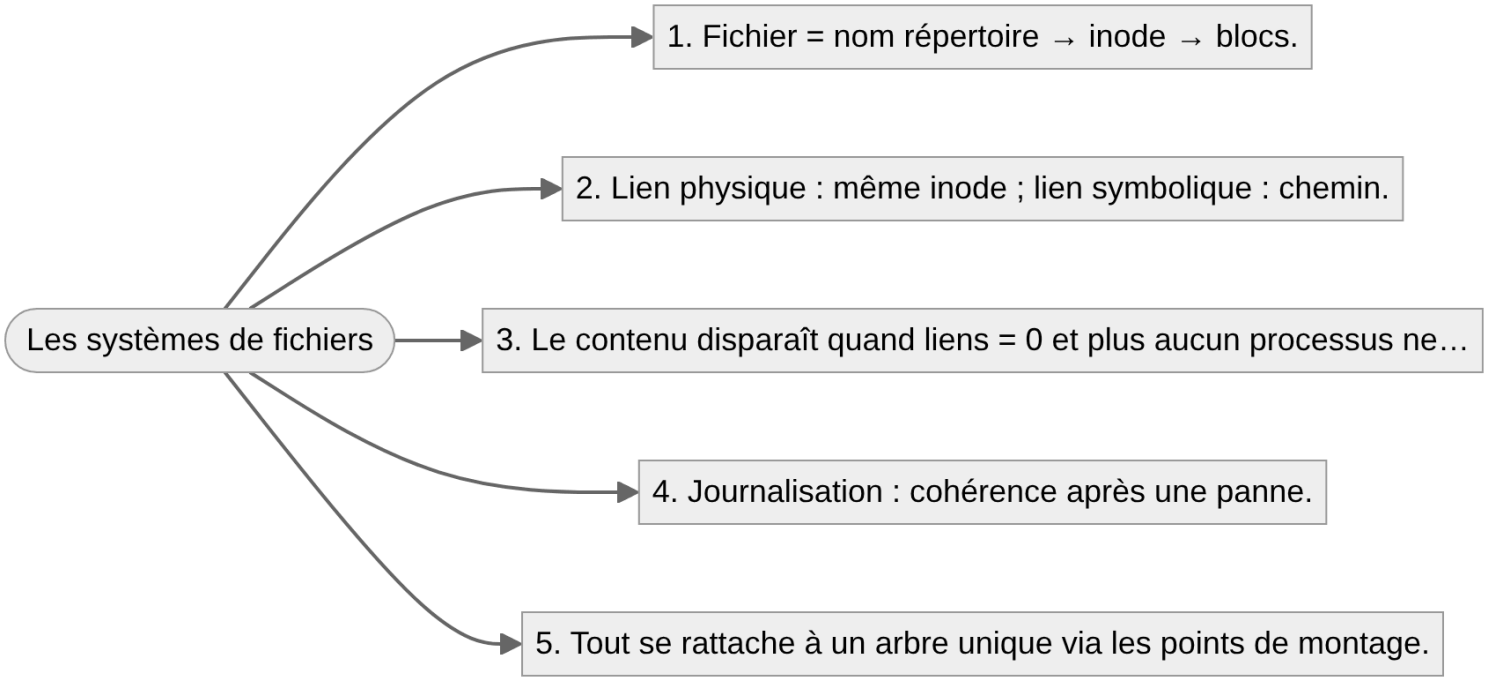
À retenir

À retenir

1. Fichier = nom (répertoire) → inode → blocs.
2. Lien physique : même inode ; lien symbolique : chemin.
3. Le contenu disparaît quand liens = 0 et plus aucun processus ne l'ouvre.
4. Journalisation : cohérence après une panne.
5. Tout se rattache à un arbre unique via les points de montage.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Archiviste

Prochaine mission : Permissions et identité

