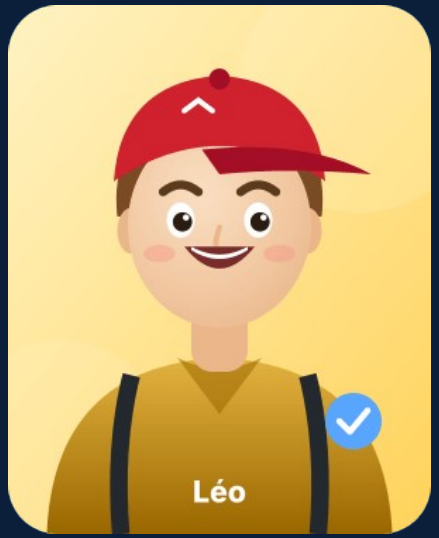
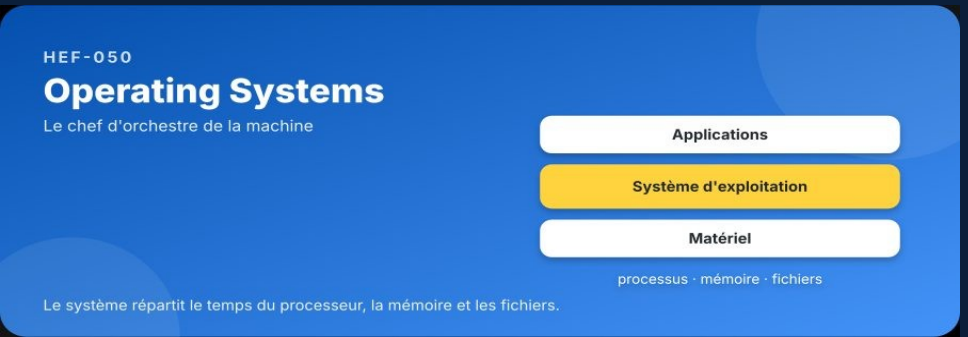
HEF-050 · MISSION 08 / 10

Entrées/sorties : tout est fichier

Clavier, écran, fichier, réseau, tube : en Unix, on lit et on écrit, partout de la même façon.





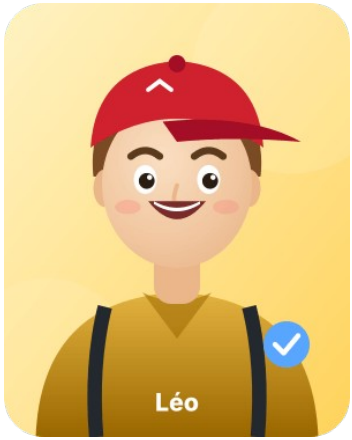
Badge : Plombier des flux

75 min · intermédiaire

L'histoire

L'histoire

Kodi enchaîne deux commandes avec un | et découvre que la sortie de la première devient l'entrée de la seconde. Il redirige un message d'erreur vers un fichier avec 2>.



« Derrière ces symboles, il y a une idée simple, dit Sam : tout est un flux de données accessible par un numéro. Si tu comprends les descripteurs de fichiers, tu comprends comment les programmes se parlent. »

Tu maîtrises le modèle des flux Unix : descripteurs, redirections, tubes et lecture bloquante.

Ta mission

À la fin, je serai capable de…

1. expliquer ce qu'est un descripteur de fichier
2. citer stdin, stdout, stderr et leurs numéros
3. expliquer redirections et tubes (pipes)
4. distinguer lecture bloquante et non bloquante
5. utiliser des flux standard dans un petit programme Python

Défi d'échauffement

Défi d'échauffement

Une usine a une conduite d'entrée, une conduite de sortie et une conduite pour les alarmes. Tu peux brancher chaque conduite sur un réservoir, un évier ou un autre atelier. Pourquoi est-il pratique d'avoir **trois conduites séparées** plutôt qu'une seule ? Que se passe-t-il si l'atelier suivant est plus lent que l'atelier précédent ?

Le concept

Tout est un fichier

Sous Unix, beaucoup de choses s'utilisent comme des fichiers : fichiers réguliers, terminaux, périphériques, tubes, sockets réseau. On y accède par les mêmes opérations : `open`, `read`, `write`, `close`.

Le descripteur de fichier

Quand un processus ouvre un fichier, le noyau renvoie un petit entier : le **descripteur de fichier** (*file descriptor*, fd). Tout processus démarre avec trois descripteurs :

fd	Nom	Usage
0	`stdin` (entrée standard)	clavier ou flux d'entrée
1	`stdout` (sortie standard)	résultats normaux
2	`stderr` (erreur standard)	messages d'erreur et de diagnostic

Le concept

Le descripteur de fichier (suite)

Séparer `stdout` et `stderr` permet de rediriger les résultats sans perdre les erreurs, ou l'inverse.

Redirections et tubes

```
programme > resultat.txt       # stdout vers un fichier
programme 2> erreurs.txt       # stderr vers un fichier
programme < entree.txt         # stdin depuis un fichier
programme1 | programme2        # stdout de 1 devient stdin de 2 (tube)
programme > tout.txt 2>&1      # stdout et stderr vers le même fichier
```

Un **tube** (*pipe*) est un tampon géré par le noyau, avec un côté pour écrire et un côté pour lire. Il permet de composer des petits outils.

Le concept

Lecture bloquante

Par défaut, `read` sur un tube vide **bloque** le processus jusqu'à l'arrivée de données. Le processus passe à l'état *bloqué* (mission 2) et ne consomme pas de CPU.
Alternatives :

- **non bloquant** : `read` rend la main immédiatement ;
- **multiplexage** (`select`, `poll`, `epoll`) : surveiller plusieurs flux à la fois et réagir au premier prêt ;
- **asynchrone** (`asyncio`) : bâti sur ces mécanismes.

Le concept

Un tube en Python

```
import os

lecture, ecriture = os.pipe()
os.write(ecriture, b"bonjour")
os.close(ecriture)
print(os.read(lecture, 100))      # b'bonjour'
os.close(lecture)
```

Brancher deux processus

```
import subprocess, sys

producteur = subprocess.Popen([sys.executable, "-c", "print('un\ndeux\ntrois')"],
                              stdout=subprocess.PIPE)
consommateur = subprocess.run([sys.executable, "-c", "import sys; print(len(sys.st
```

Le concept

Brancher deux processus (suite)

```
    din.read().split()))"],
                               stdin=producteur.stdout, capture_output=True, text=T
    rue)
print(consommateur.stdout.strip())      # 3
```

Écrire proprement

- Résultats utiles → `stdout` ; messages d'erreur → `stderr`.
- Code de sortie ≠ 0 en cas d'erreur.

```
import sys

print("résultat", file=sys.stdout)
print("attention : fichier absent", file=sys.stderr)
```

Le concept

Écrire proprement (suite)

🧠 ***Astuce*** *: les petits programmes qui lisent stdin et écrivent stdout se combinent comme des briques.*

Exemple pas à pas

Problème : écrire un filtre qui lit des lignes sur `stdin`, garde celles qui contiennent un mot, et envoie les erreurs sur `stderr`.

```
import subprocess, sys

filtre = '''
import sys
mot = sys.argv[1]
compte = 0
for ligne in sys.stdin:
    if mot in ligne:
        print(ligne.rstrip())
        compte += 1
print(f"{compte} ligne(s) trouvée(s)", file=sys.stderr)
'''

entree = "erreur disque\nok\nerreur réseau\nok\n"
r = subprocess.run([sys.executable, "-c", filtre, "erreur"], input=entree, capture
```

Exemple pas à pas (suite)

```
   _output=True, text=True)
print("STDOUT :", r.stdout.split("\n")[:-1])
print("STDERR :", r.stderr.strip())
```

Lecture : `stdout` contient les lignes sélectionnées ; `stderr` contient le compte. On peut rediriger l'un sans l'autre : `filtre.py erreur < journal.txt > erreurs.txt`, le compte s'affiche quand même à l'écran.

Surveiller deux flux avec `select` :

```
import os, select, threading, time

r1, w1 = os.pipe()
r2, w2 = os.pipe()

def ecrire(w, delai, texte):
    time.sleep(delai)
    os.write(w, texte)
```

Exemple pas à pas (suite)

```
threading.Thread(target=ecrire, args=(w1, 0.3, b"lent")).start()
threading.Thread(target=ecrire, args=(w2, 0.1, b"rapide")).start()

for _ in range(2):
    prets, _, _ = select.select([r1, r2], [], [], 2)
    for fd in prets:
        print("prêt :", os.read(fd, 100))
```

On lit **dans l'ordre d'arrivée**, sans rester bloqué sur le flux lent.

À toi, avec aide

Écris un programme `compteur.py` (sous forme de chaîne exécutée dans un sous-processus) qui lit des mots sur `stdin` (un par ligne), compte les occurrences, puis écrit sur `stdout` les 3 plus fréquents. Les lignes vides sont ignorées et un message de diagnostic va sur `stderr`.
Lance-le depuis Python avec `subprocess.run(..., input=...)` et vérifie le résultat.

À toi, avec aide · réponse

Indices progressifs

Indice 1

`collections.Counter` aide à compter.

Indice 2

`counter.most_common(3)` renvoie les 3 plus fréquents.

Indice 3

Écris le diagnostic avec `print(..., file=sys.stderr)`.

À toi, avec aide · réponse

Réponse

```
import subprocess, sys

programme = '''
import sys
from collections import Counter
c = Counter()
for ligne in sys.stdin:
    mot = ligne.strip()
    if mot:
        c[mot] += 1
for mot, n in c.most_common(3):
    print(mot, n)
print(f"{sum(c.values())} mots lus", file=sys.stderr)
'''

entree = "a\nb\na\n\nc\na\nb\n"
r = subprocess.run([sys.executable, "-c", programme], input=entree, capture_output
   =True, text=True)
print(r.stdout)
```

À toi, avec aide · réponse

Réponse (suite)

```
print("diagnostic :", r.stderr.strip())
```

Résultat : `a 3`, `b 2`, `c 1`, et le diagnostic « 6 mots lus » séparé du résultat.

À toi, en solo

Chaîne de traitement à la Unix. Construis en Python trois petits programmes (dans des chaînes) reliés par des tubes :

1. `generer` : écrit les nombres de 1 à 20 ;
2. `filtrer_pairs` : lit stdin, garde les nombres pairs ;
3. `somme` : lit stdin, écrit la somme.

Relie-les avec `subprocess.Popen` et des tubes, sans fichier intermédiaire. Résultat attendu : 110. Ajoute la gestion d'erreur : si un des processus échoue, affiche son code de sortie et son `stderr`.

À toi, en solo · réponse

Pistes de correction

```
import subprocess, sys

generer = "for i in range(1, 21): print(i)"
filtrer = "import sys\nfor l in sys.stdin:\n    n = int(l)\n    if n % 2 == 0: pri
   nt(n)"
somme = "import sys\nprint(sum(int(l) for l in sys.stdin))"

p1 = subprocess.Popen([sys.executable, "-c", generer], stdout=subprocess.PIPE, std
   err=subprocess.PIPE)
p2 = subprocess.Popen([sys.executable, "-c", filtrer], stdin=p1.stdout, stdout=sub
   process.PIPE, stderr=subprocess.PIPE)
p1.stdout.close()                    # permet à p1 de recevoir SIGPIPE si p2 se t
   ermine tôt
p3 = subprocess.Popen([sys.executable, "-c", somme], stdin=p2.stdout, stdout=subpr
   ocess.PIPE, stderr=subprocess.PIPE)
p2.stdout.close()

sortie, err3 = p3.communicate()
for nom, p in (("generer", p1), ("filtrer", p2), ("somme", p3)):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    p.wait()
    if p.returncode != 0:
        print(nom, "a échoué avec le code", p.returncode)
print(sortie.decode().strip())       # 110
```

Fermer la copie du côté lecture dans le processus parent est un détail important : sans cela, un processus en amont ne reçoit jamais la fin de flux correctement.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Tout écrire sur stdout	Résultats et erreurs mélangés	Erreurs sur stderr
Lecture bloquante sans délai	Programme figé	Délais d'attente, `select`, asynchrone
Oublier de fermer les extrémités d'un tube	Le lecteur n'obtient jamais la fin de flux	Fermer les descripteurs inutiles
Buffer de sortie	Messages qui arrivent en retard	`flush=True` ou sortie non tamponnée quand nécessaire

Mini-boss

Mini-boss

Superviseur de journaux. Écris un programme qui lance deux processus (qui écrivent des lignes sur leur `stdout` à des rythmes différents) et affiche leurs sorties **fusionnées**, préfixées du nom du processus, au fur et à mesure qu'elles arrivent, en utilisant `select` (ou des threads de lecture + une file).
Quand les deux ont terminé, affiche les codes de sortie. Teste avec un processus rapide (5 lignes en 0,5 s) et un lent (3 lignes en 1,5 s).

Mini-boss · réponse

Pistes de correction

```
import select, subprocess, sys, time

def lanceur(nom, n, delai):
    code = f"import time\nfor i in range({n}):\n    print('{nom}', i, flush=True)\
   n    time.sleep({delai})"
    return subprocess.Popen([sys.executable, "-c", code], stdout=subprocess.PIPE,
   text=True)

processus = {"rapide": lanceur("rapide", 5, 0.1), "lent": lanceur("lent", 3, 0.5)}
ouverts = {p.stdout: nom for nom, p in processus.items()}

debut = time.perf_counter()
while ouverts:
    prets, _, _ = select.select(list(ouverts), [], [], 5)
    for flux in prets:
        ligne = flux.readline()
        if ligne:
            print(f"[{time.perf_counter() - debut:4.1f}s] {ouverts[flux]:7s} {lign
   e.strip()}")
```

Mini-boss · réponse

Pistes de correction (suite)

```
        else:
            del ouverts[flux]

for nom, p in processus.items():
    print(nom, "code", p.wait())
```

Les lignes des deux processus apparaissent entrelacées dans l'ordre d'arrivée : aucun des deux ne bloque l'autre.

Quiz éclair

Quiz éclair

1. Le descripteur de fichier 2 correspond à :

A. stdin
B. stdout
C. stderr

2. Un tube (pipe) relie :

A. la sortie d'un processus à l'entrée d'un autre
B. deux disques
C. deux écrans

3. `select` sert à :

A. trier des listes

Quiz éclair

Quiz éclair (suite)

B. surveiller plusieurs flux et réagir au premier prêt
C. chiffrer des fichiers

4. 🔁 *Révision mission 07* : Les droits `rwxr-x---` correspondent en octal à :

A. 755
B. 640
C. 750

5. 🔁 *Révision mission 07* : Sur un dossier, le droit `x` permet de :

A. le traverser
B. l'exécuter comme un programme
C. le supprimer

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quels « tuyaux » utilises-tu dans ta vie pour faire passer le résultat d'une tâche à la suivante ?
2. Pourquoi séparer les résultats et les messages d'erreur est-il utile pour un utilisateur ?

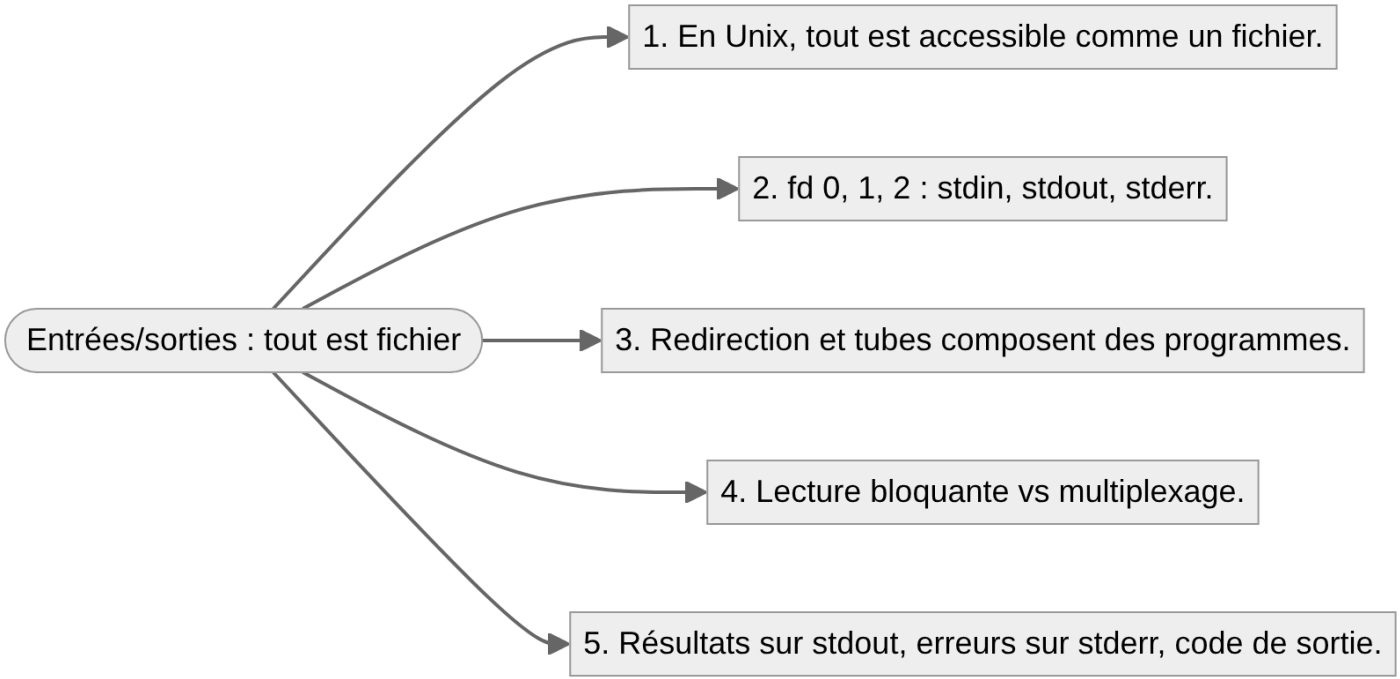
À retenir

À retenir

1. En Unix, tout est accessible comme un fichier.
2. fd 0, 1, 2 : stdin, stdout, stderr.
3. Redirection et tubes composent des programmes.
4. Lecture bloquante vs multiplexage.
5. Résultats sur stdout, erreurs sur stderr, code de sortie.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Plombier des flux

Prochaine mission : Services, signaux et observabilité

