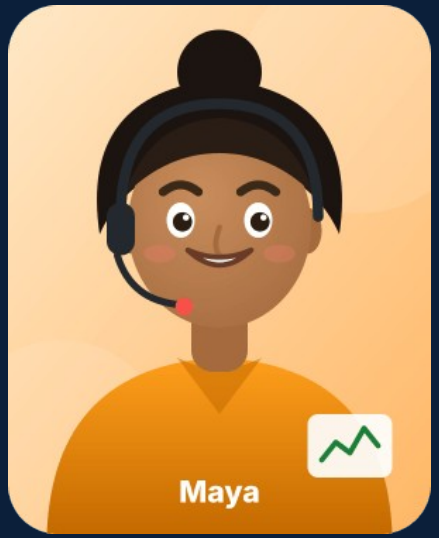
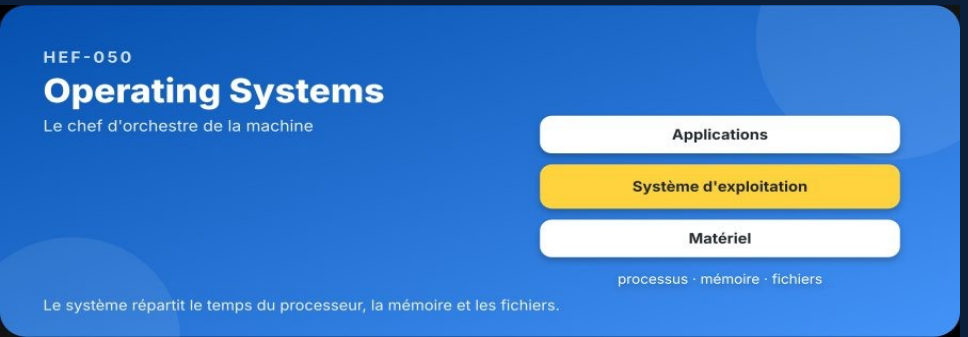
HEF-050 · MISSION 02 / 10

Les processus

Un programme est une recette. Un processus est un cuisinier en train de la suivre.



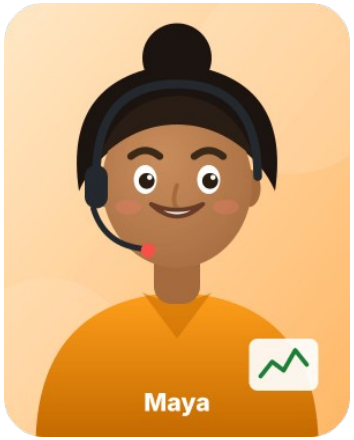


Badge : Chef de processus

75 min · intermédiaire

L'histoire

Kodi lance son serveur de test deux fois. Il y a maintenant **deux serveurs** en même temps, avec deux identifiants différents et chacun ses propres données. Il s'étonne : « C'est le même programme ! »



> *« Le programme est un fichier, dit Sam. Un processus est un programme **en cours d'exécution**, avec sa propre mémoire et son propre état. »*

Tu apprends à créer, observer et contrôler des processus.

Ta mission

À la fin, je serai capable de…

1. distinguer programme et processus
2. décrire les états d'un processus
3. expliquer PID, parent, enfant
4. créer un processus avec `subprocess` et lire son code de sortie
5. expliquer ce qu'est un processus zombie ou orphelin

Défi d'échauffement

Défi d'échauffement

Une recette de cuisine (le **programme**) peut être suivie par 3 cuisiniers dans 3 cuisines différentes en même temps. Chacun en est à une étape différente, avec ses propres ingrédients. Qu'est-ce qui appartient à la **recette**, et qu'est-ce qui appartient à chaque **cuisinier** ? Que se passe-t-il si l'un se brûle ?

Le concept

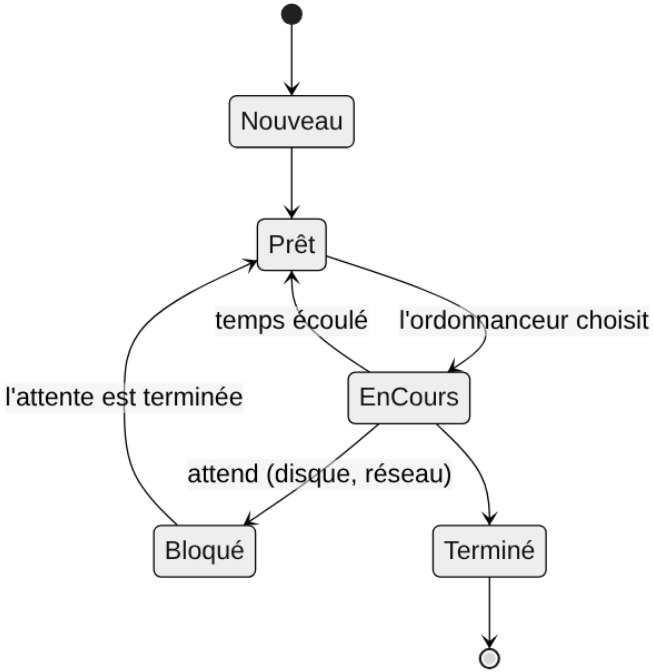
Programme et processus

- **Programme** : un fichier exécutable, inerte.
- **Processus** : une instance en cours d'exécution, avec :
  - un **PID** (identifiant) ;
  - son **espace mémoire** (code, données, tas, pile) ;
  - ses **fichiers ouverts** ;
  - son **état** et son **compteur de programme** ;
  - un **utilisateur propriétaire**.

Les états d'un processus

Le concept

Les états d'un processus (suite)



Le concept

Les états d'un processus (suite)

Description du schéma : un processus naît (nouveau), devient prêt, puis « en cours » quand il obtient le CPU. Il peut repasser à prêt si son temps est écoulé, ou passer à bloqué s'il attend une ressource, puis revenir à prêt. Il finit à l'état terminé.

Parent et enfants

Un processus est créé par un autre (son **parent**). Sous Unix, `fork` crée une copie du processus, puis `exec` remplace le programme de la copie par un autre. Le premier processus (`init` ou `systemd`) a le PID 1.

```
systemd (PID 1)
└── bash (PID 812)
    └── python3 script.py (PID 4120)
```

Le concept

Créer un processus en Python

```
import subprocess, sys

resultat = subprocess.run([sys.executable, "-c", "print('bonjour du fils')"],
                          capture_output=True, text=True)
print(resultat.stdout.strip())     # bonjour du fils
print(resultat.returncode)         # 0
```

Le code de sortie

Un processus termine avec un **code de sortie** : **0 = succès**, autre valeur = erreur. Les scripts, les outils d'intégration continue et les shells s'appuient dessus.

Le concept

Le code de sortie (suite)

```
import subprocess, sys

ok = subprocess.run([sys.executable, "-c", "raise SystemExit(0)"]).returncode
ko = subprocess.run([sys.executable, "-c", "raise SystemExit(3)"]).returncode
print(ok, ko)     # 0 3
```

Cas particuliers

- **Zombie** : un enfant terminé dont le parent n'a pas encore lu le code de sortie. Il ne consomme presque rien, mais garde une entrée dans la table des processus.
- **Orphelin** : un enfant dont le parent est mort. Il est adopté par `init`.

Isolation

Le concept

Isolation (suite)

Deux processus ne partagent **pas** leur mémoire. Pour échanger des données : fichiers, tubes (pipes), sockets, mémoire partagée explicitement.

🧠 ***Astuce*** *: un processus qui plante n'emporte pas les autres. C'est la grande force de l'isolation.*

Exemple pas à pas

Problème : lancer plusieurs processus fils en parallèle, attendre leurs résultats et récupérer leurs codes de sortie.

```
import subprocess, sys

travaux = [
    "print('A'); raise SystemExit(0)",
    "print('B'); raise SystemExit(2)",
    "import time; time.sleep(0.2); print('C')",
]

processus = [
    subprocess.Popen([sys.executable, "-c", code], stdout=subprocess.PIPE, text=Tr
   ue)
    for code in travaux
]

for p in processus:
```

Exemple pas à pas (suite)

```
    sortie, _ = p.communicate()         # attend la fin et lit la sortie
    print(f"PID {p.pid}: sortie={sortie.strip()!r} code={p.returncode}")
```

Lecture :

- les trois processus tournent **en même temps** (le dernier attend 0,2 s sans retarder les autres) ;
- chaque processus a son propre **PID** et son propre **code de sortie** ;
- `communicate()` attend la fin : c'est aussi ce qui évite de laisser des processus zombies.

Isolation : une variable modifiée dans un fils n'est jamais vue par le parent.

```
import subprocess, sys

x = 10
subprocess.run([sys.executable, "-c", "x = 99; print('fils x =', x)"])
print("parent x =", x)     # parent x = 10
```

À toi, avec aide

Écris `executer(commande_python)` qui lance un petit programme Python dans un **processus séparé**, avec un délai maximum de 2 secondes, et renvoie un dictionnaire `{"sortie": …, "code": …, "expire": True/False}`.
Teste avec un programme qui affiche un texte, un programme qui échoue (code 1), et un programme qui dort 5 secondes.

À toi, avec aide · réponse

Indices progressifs

Indice 1

`subprocess.run(..., timeout=2)` lève `subprocess.TimeoutExpired`.

Indice 2

Attrape l'exception et renvoie `expire: True`.

Indice 3

Capture la sortie avec `capture_output=True, text=True`.

À toi, avec aide · réponse

Réponse

```
import subprocess, sys

def executer(code, delai=2):
    try:
        r = subprocess.run([sys.executable, "-c", code], capture_output=True, text
   =True, timeout=delai)
        return {"sortie": r.stdout.strip(), "code": r.returncode, "expire": False}
    except subprocess.TimeoutExpired:
        return {"sortie": "", "code": None, "expire": True}

print(executer("print('ok')"))
print(executer("raise SystemExit(1)"))
print(executer("import time; time.sleep(5)", delai=1))
```

À toi, en solo

Superviseur minimal. Écris `superviser(code, essais)` qui lance un programme dans un processus fils et, s'il échoue (code ≠ 0), le relance jusqu'à `essais` fois, en affichant chaque tentative. Teste avec un programme qui échoue deux fois avant de réussir (utilise un fichier compteur temporaire pour mémoriser le nombre de tentatives).

À toi, en solo · réponse

Pistes de correction

```
import os, subprocess, sys, tempfile

def superviser(code, essais):
    for tentative in range(1, essais + 1):
        r = subprocess.run([sys.executable, "-c", code], capture_output=True, text
   =True)
        print(f"tentative {tentative}: code={r.returncode}")
        if r.returncode == 0:
            return True
    return False

with tempfile.TemporaryDirectory() as d:
    compteur = os.path.join(d, "n.txt")
    programme = f'''
import os
chemin = {compteur!r}
n = int(open(chemin).read()) if os.path.exists(chemin) else 0
open(chemin, "w").write(str(n + 1))
raise SystemExit(0 if n >= 2 else 1)
```

À toi, en solo · réponse

Pistes de correction (suite)

```
'''
    print(superviser(programme, 5))      # réussit à la 3e tentative
```

C'est le principe des superviseurs de services (systemd, supervisord) : relancer un processus en échec, avec une limite pour éviter les boucles infinies.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Confondre programme et processus	Un fichier n'est pas un processus	Programme = recette, processus = exécution
Oublier d'attendre les fils	Processus zombies, résultats perdus	`wait` / `communicate` / `run`
Ignorer le code de sortie	Une erreur passe inaperçue	Toujours tester `returncode`
Supposer que la mémoire est partagée	Le fils ne modifie pas le parent	Utiliser tubes, fichiers ou sockets

Mini-boss

Pool de travailleurs. Écris `executer_en_parallele(taches, max_simultanes)` : `taches` est une liste de petits programmes Python (chaînes). La fonction les lance dans des processus séparés, **au plus `max_simultanes` à la fois**, et renvoie la liste des résultats `(code, sortie)` dans l'ordre des tâches.

Teste avec 6 tâches de 0,2 s et `max_simultanes` = 1, 2, 6. Mesure le temps total et explique le résultat.

Mini-boss · réponse

Pistes de correction

```
import subprocess, sys, time

def executer_en_parallele(taches, max_simultanes):
    resultats = [None] * len(taches)
    en_cours = {}                    # index -> Popen
    suivant = 0
    while suivant < len(taches) or en_cours:
        while suivant < len(taches) and len(en_cours) < max_simultanes:
            en_cours[suivant] = subprocess.Popen([sys.executable, "-c", taches[sui
   vant]],
                                                 stdout=subprocess.PIPE, text=Tru
   e)
            suivant += 1
        for i, p in list(en_cours.items()):
            if p.poll() is not None:
                sortie = p.stdout.read().strip()
                resultats[i] = (p.returncode, sortie)
                del en_cours[i]
        time.sleep(0.01)
```

Mini-boss · réponse

Pistes de correction (suite)

```
    return resultats

taches = [f"import time; time.sleep(0.2); print({i})" for i in range(6)]
for m in (1, 2, 6):
    t = time.perf_counter()
    r = executer_en_parallele(taches, m)
    print(m, round(time.perf_counter() - t, 2), "s", [x[1] for x in r])
```

Temps attendu : environ 1,2 s avec 1 travailleur (6 × 0,2 s), 0,6 s avec 2, 0,2 s avec 6 : le parallélisme réduit le temps tant que le nombre de cœurs et de tâches le permet.

Quiz éclair

Quiz éclair

1. Un processus est :

A. un fichier exécutable
B. un type de mémoire
C. un programme en cours d'exécution

2. Un code de sortie 0 signifie généralement :

A. succès
B. erreur
C. attente

3. Deux processus partagent par défaut :

A. toute leur mémoire

Quiz éclair

Quiz éclair (suite)

B. rien de leur mémoire
C. leur pile

4. 🔁 *Révision mission 01* : Un appel système permet :

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

5. 🔁 *Révision HEF-040, mission 09* : La loi d'Amdahl dit que l'accélération est limitée par :

A. la partie non parallélisable
B. la fréquence
C. la taille de la RAM

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quel processus de ta vie quotidienne a un « parent » (une tâche qui en déclenche d'autres) ?
2. Pourquoi l'isolation entre processus est-elle une protection précieuse ?

À retenir

1. Programme = fichier ; processus = exécution avec PID et mémoire propres.
2. États : prêt, en cours, bloqué, terminé.
3. Parent/enfant ; PID 1 est le premier processus.
4. Code de sortie : 0 = succès.
5. Zombie : enfant terminé non « récolté » ; orphelin : adopté par init.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Chef de processus

Prochaine mission : Threads et concurrence

