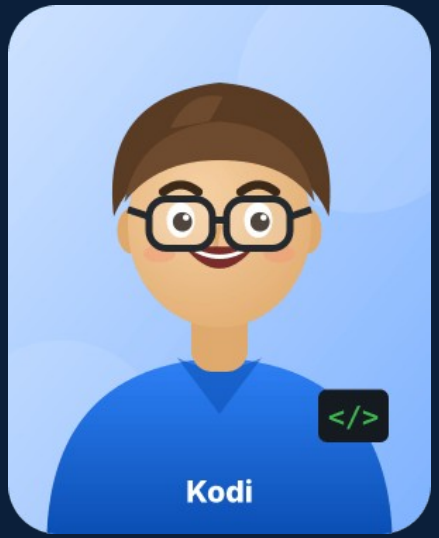
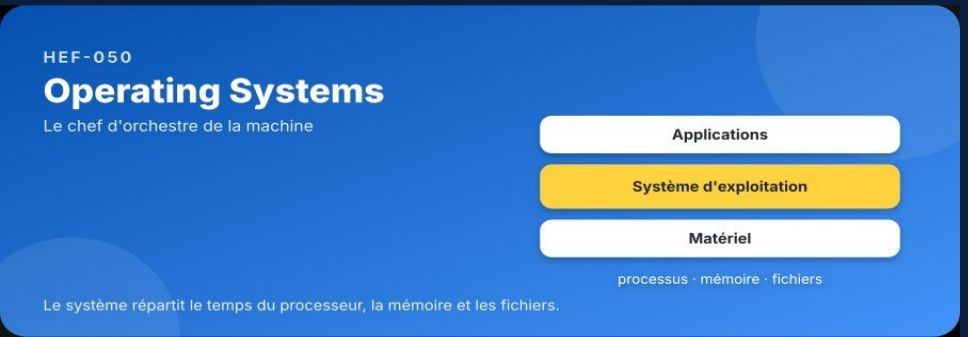
HEF-050 · MISSION 09 / 10

Services, signaux et observabilité

Un service tourne en fond, répond aux signaux, écrit des journaux. Comment le piloter et le comprendre ?



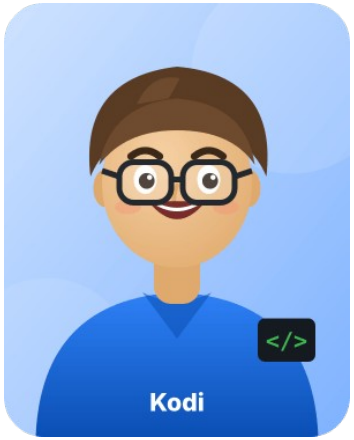


Badge : Surveillant de service

75 min · intermédiaire

L'histoire

Le serveur de HashBox s'arrête brutalement quand Kodi ferme son terminal. Les logs sont perdus, une opération était à moitié terminée. Plus tard, il tue un processus bloqué avec `kill -9` et la base de données reste dans un état incohérent.



> *« Il y a des façons polies et des façons brutales de demander à un programme de s'arrêter, dit Sam. Et un bon service raconte ce qu'il fait. »*

Tu apprends à faire tourner un programme comme un service, à lui parler avec des signaux, et à observer ce qu'il fait.

Ta mission

À la fin, je serai capable de…

1. expliquer ce qu'est un service (démon) et son cycle de vie
2. utiliser SIGTERM, SIGINT, SIGKILL et expliquer leurs différences
3. écrire un gestionnaire de signal pour un arrêt propre
4. utiliser le module `logging` avec des niveaux de gravité
5. lire des informations de supervision dans `/proc`

Défi d'échauffement

Défi d'échauffement

Un restaurant ferme à 23 h. Option A : on éteint les lumières brutalement, les clients doivent sortir tout de suite, les plats en préparation sont jetés. Option B : on annonce « dernière commande », on termine les plats en cours, on range et on éteint. Quelle option préserves-tu pour un serveur qui traite des paiements ? Que faut-il prévoir pour que B soit possible ?

Le concept

Un service (ou démon)

Un **service** (*daemon*) est un processus qui tourne **en arrière-plan**, démarré par le système (souvent par `systemd`), sans terminal attaché. Exemples : serveur web, base de données, planificateur.

Un gestionnaire de services permet de :

- **démarrer / arrêter / redémarrer** ;
- **relancer** automatiquement en cas de plantage ;
- gérer les **dépendances** (la base avant l'application) ;
- collecter les **journaux**.

Le concept

Un service (ou démon) (suite)

```
systemctl status hashbox
systemctl restart hashbox
journalctl -u hashbox -n 50
```

Les signaux

Un **signal** est un petit message asynchrone envoyé à un processus.

Signal	Numéro	Sens	Interceptable ?
SIGINT	2	interruption (Ctrl+C)	oui
SIGTERM	15	« merci de terminer proprement » (défaut de `kill`)	oui
SIGHUP	1	terminal fermé / recharger la config (par convention)	oui
SIGKILL	9	arrêt immédiat, imposé par le noyau	**non**

Le concept

Les signaux (suite)

Signal	Numéro	Sens	Interceptable ?
SIGSTOP	19	suspendre	non

Règle : essayer d'abord `SIGTERM`, laisser un délai, et n'utiliser `SIGKILL` qu'en dernier recours (pas de nettoyage possible).

Un arrêt propre en Python

```
import signal, time

arret_demande = False

def gerer_arret(signum, frame):
    global arret_demande
    arret_demande = True
```

Le concept

Un arrêt propre en Python (suite)

```
signal.signal(signal.SIGTERM, gerer_arret)
signal.signal(signal.SIGINT, gerer_arret)

while not arret_demande:
    # ... traite une tâche ...
    time.sleep(0.1)
# nettoyage : terminer les tâches, fermer les fichiers, libérer les verrous
print("arrêt propre")
```

Les journaux (logs)

On n'utilise pas `print` pour un service : on utilise un système de **journalisation** avec niveaux.

Niveau	Usage
DEBUG	détails pour développer
INFO	événements normaux

Le concept

Les journaux (logs) (suite)

Niveau	Usage
WARNING	anomalie sans gravité
ERROR	échec d'une opération
CRITICAL	le service est en danger

```
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(messag
   e)s")
log = logging.getLogger("hashbox")
log.info("service démarré")
log.warning("disque à 85 pour cent")
log.error("paiement refusé pour le membre %s", 42)
```

Le concept

L'observabilité de base

Trois sources :

Source	Question
Journaux	que s'est-il passé ?
Métriques	combien, à quelle vitesse ? (CPU, mémoire, requêtes)
Traces	par où la requête est-elle passée ?

`/proc` sous Linux

Le noyau expose des informations en lecture sous forme de fichiers virtuels :

Le concept

`/proc` sous Linux (suite)

```
import os

pid = os.getpid()
try:
    with open(f"/proc/{pid}/status") as f:
        for ligne in f:
            if ligne.startswith(("Name", "State", "VmRSS", "Threads")):
                print(ligne.strip())
except OSError:
    print("/proc non disponible sur ce système")
```

`/proc/<pid>/status` donne le nom, l'état, la mémoire résidente (`VmRSS`), le nombre de threads.

🧠 ***Astuce*** *: un service sans journal est une boîte noire. Un journal sans niveaux est du bruit.*

Exemple pas à pas

Problème : un travailleur qui traite des tâches doit s'arrêter proprement sur `SIGTERM`, en finissant la tâche en cours et en journalisant ce qu'il fait.

```
import logging, os, signal, time

logging.basicConfig(level=logging.INFO, format="%(levelname)s %(message)s")
log = logging.getLogger("travailleur")

arret = False

def sur_sigterm(signum, frame):
    global arret
    log.info("signal %s reçu : arrêt après la tâche en cours", signal.Signals(sign
   um).name)
    arret = True

signal.signal(signal.SIGTERM, sur_sigterm)
```

Exemple pas à pas (suite)

```
taches_faites = []
numero = 0
while not arret:
    numero += 1
    log.info("tâche %d : début", numero)
    time.sleep(0.05)                        # travail
    taches_faites.append(numero)
    log.info("tâche %d : terminée", numero)
    if numero == 3:
        os.kill(os.getpid(), signal.SIGTERM)    # on simule un arrêt demandé de l'
   extérieur

log.info("arrêt propre, %d tâches terminées", len(taches_faites))
assert taches_faites == [1, 2, 3]
```

Lecture : la tâche 3 n'est pas interrompue au milieu : elle se termine, puis la boucle s'arrête. Avec `SIGKILL`, le processus serait tué sur-le-champ, au milieu d'une tâche, sans journal de fin.

Exemple pas à pas (suite)

Journaux dans un fichier avec rotation :

```
import logging
from logging.handlers import RotatingFileHandler

handler = RotatingFileHandler("service.log", maxBytes=1_000_000, backupCount=3)
```

La **rotation** évite qu'un journal sature le disque (voir la mission « Système de fichiers »).

À toi, avec aide

Écris un petit service de simulation qui :

1. journalise « démarré » ;
2. toutes les 0,1 s, journalise un battement de cœur (`heartbeat`) ;
3. s'arrête proprement sur `SIGINT` ou `SIGTERM` en journalisant « arrêt » ;
4. s'envoie lui-même `SIGTERM` après 5 battements pour que le test se termine.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Une variable globale `arret` change dans le gestionnaire.

Indice 2

Utilise `os.kill(os.getpid(), signal.SIGTERM)`.

Indice 3

N'oublie pas d'enregistrer le même gestionnaire pour les deux signaux.

À toi, avec aide · réponse

Réponse

```
import logging, os, signal, time

logging.basicConfig(level=logging.INFO, format="%(levelname)s %(message)s")
log = logging.getLogger("service")
arret = False

def gerer(signum, frame):
    global arret
    arret = True

signal.signal(signal.SIGINT, gerer)
signal.signal(signal.SIGTERM, gerer)

log.info("démarré")
battements = 0
while not arret:
    battements += 1
    log.info("heartbeat %d", battements)
    time.sleep(0.1)
```

Réponse (suite)

```
    if battements == 5:
        os.kill(os.getpid(), signal.SIGTERM)
log.info("arrêt après %d battements", battements)
```

À toi, en solo

Moniteur de processus. Écris `etat_processus(pid)` qui lit `/proc/<pid>/status` (Linux) et renvoie un dictionnaire `{nom, etat, memoire_ko, threads}`. Puis écris un programme qui lance un fils qui alloue de la mémoire progressivement, observe sa consommation toutes les 0,1 s, et **le tue avec SIGTERM** (puis SIGKILL après 1 s s'il ne s'arrête pas) quand il dépasse 100 Mo. Adapte pour que ce soit testable : le fils alloue par paliers de 20 Mo jusqu'à 200 Mo.

À toi, en solo · réponse

Pistes de correction

```
import os, signal, subprocess, sys, time

def etat_processus(pid):
    infos = {}
    with open(f"/proc/{pid}/status") as f:
        for ligne in f:
            cle, _, valeur = ligne.partition(":")
            infos[cle] = valeur.strip()
    return {
        "nom": infos.get("Name"),
        "etat": infos.get("State"),
        "memoire_ko": int(infos.get("VmRSS", "0 kB").split()[0]),
        "threads": int(infos.get("Threads", "1")),
    }

fils_code = '''
import time
blocs = []
for _ in range(10):
```

Pistes de correction (suite)

```
    blocs.append(bytearray(20 * 1024 * 1024))
    for i in range(0, len(blocs[-1]), 4096):
        blocs[-1][i] = 1
    time.sleep(0.1)
time.sleep(5)
'''
p = subprocess.Popen([sys.executable, "-c", fils_code])
limite_ko = 100 * 1024
try:
    while p.poll() is None:
        try:
            info = etat_processus(p.pid)
        except OSError:
            break
        print(f"{info['nom']} {info['etat']} {info['memoire_ko'] // 1024} Mo")
        if info["memoire_ko"] > limite_ko:
            print("limite dépassée : SIGTERM")
            p.send_signal(signal.SIGTERM)
            try:
```

À toi, en solo · réponse

Pistes de correction (suite)

```
                p.wait(timeout=1)
            except subprocess.TimeoutExpired:
                print("pas d'arrêt : SIGKILL")
                p.kill()
            break
        time.sleep(0.1)
finally:
    p.wait()
print("code de sortie :", p.returncode)       # -15 (tué par SIGTERM)
```

Un code de sortie négatif en Python signifie « tué par le signal de ce numéro » (-15 = SIGTERM, -9 = SIGKILL).

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
`kill -9` en premier	Pas de nettoyage, données incohérentes	SIGTERM d'abord, puis délai, puis SIGKILL
`print` dans un service	Pas de niveaux, difficile à filtrer	Module `logging`
Journaux sans rotation	Le disque se remplit	Rotation et rétention
Gestionnaire de signal trop lourd	Comportement fragile	Se contenter de lever un drapeau, traiter dans la boucle

Mini-boss

Mini-boss

Service de traitement robuste. Écris un service qui :

1. lit des « commandes » dans une `queue.Queue` alimentée par un thread simulant des requêtes ;
2. traite chaque commande (petite pause) et journalise son début et sa fin avec un identifiant ;
3. s'arrête **proprement** sur SIGTERM : il cesse d'accepter de nouvelles commandes, termine celle en cours, journalise le nombre de commandes restantes non traitées ;
4. expose une fonction `sante()` qui renvoie un dictionnaire (traitées, en attente, état) — un « point de santé » ;
5. est testé en s'envoyant SIGTERM après 1 seconde.

Mini-boss · réponse

Pistes de correction

```
import logging, os, queue, signal, threading, time

logging.basicConfig(level=logging.INFO, format="%(asctime)s.%(msecs)03d %(levelnam
   e)s %(message)s", datefmt="%H:%M:%S")
log = logging.getLogger("service")

q = queue.Queue()
etat = {"traitees": 0, "etat": "demarrage"}
arret = threading.Event()

def sante():
    return {"traitees": etat["traitees"], "en_attente": q.qsize(), "etat": etat["e
   tat"]}

def gerer(signum, frame):
    log.info("SIGTERM reçu")
    arret.set()

signal.signal(signal.SIGTERM, gerer)
```

Mini-boss · réponse

Pistes de correction (suite)

```
def requetes():
    i = 0
    while not arret.is_set():
        i += 1
        q.put(i)
        time.sleep(0.05)

threading.Thread(target=requetes, daemon=True).start()
threading.Timer(1.0, lambda: os.kill(os.getpid(), signal.SIGTERM)).start()

etat["etat"] = "actif"
while not arret.is_set():
    try:
        cmd = q.get(timeout=0.1)
    except queue.Empty:
        continue
    log.info("commande %d : début", cmd)
    time.sleep(0.08)
```

Pistes de correction (suite)

```
    etat["traitees"] += 1
    log.info("commande %d : fin", cmd)

etat["etat"] = "arret"
log.info("arrêt propre : %d traitées, %d en attente non traitées", etat["traitees"
   ], q.qsize())
print(sante())
```

Le service finit la commande en cours et rapporte précisément ce qu'il laisse derrière lui : un arrêt propre est aussi un arrêt **explicable**.

Quiz éclair

Quiz éclair

1. Pour demander poliment à un processus de se terminer, on envoie :

A. SIGKILL
B. SIGTERM
C. SIGSTOP

2. Quel signal ne peut pas être intercepté ?

A. SIGTERM
B. SIGINT
C. SIGKILL

3. Un journal avec niveaux (DEBUG, INFO, ERROR) permet de :

A. filtrer par gravité

Quiz éclair

Quiz éclair (suite)

B. compresser les fichiers
C. accélérer le programme

4. 🔁 *Révision mission 08* : Le descripteur de fichier 2 correspond à :

A. stdin
B. stderr
C. stdout

5. 🔁 *Révision mission 08* : Un tube (pipe) relie :

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

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quand tu quittes une tâche, te donnes-tu le temps de la « fermer proprement » ?
2. Quelles « traces » laisses-tu pour qu'un collègue puisse comprendre ce que tu as fait ?

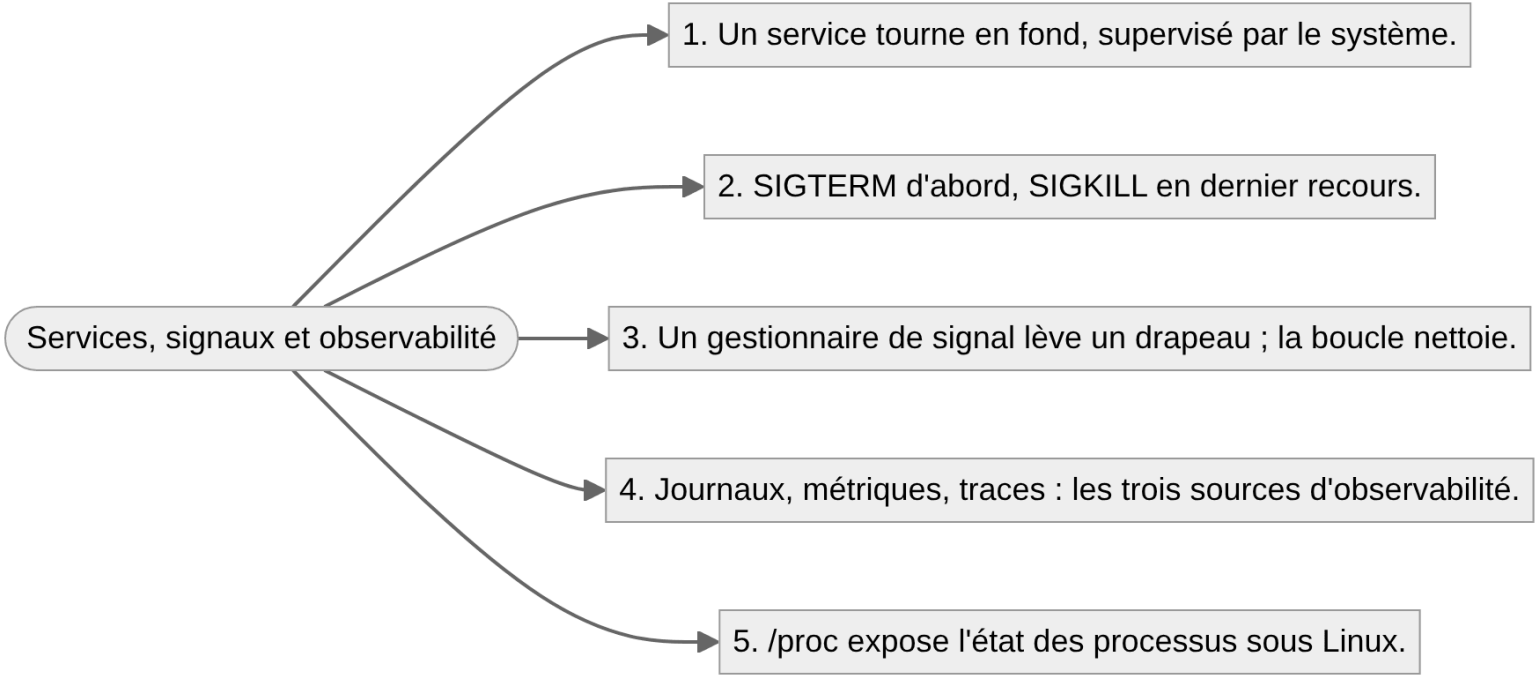
À retenir

À retenir

1. Un service tourne en fond, supervisé par le système.
2. SIGTERM d'abord, SIGKILL en dernier recours.
3. Un gestionnaire de signal lève un drapeau ; la boucle nettoie.
4. Journaux, métriques, traces : les trois sources d'observabilité.
5. `/proc` expose l'état des processus sous Linux.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Surveillant de service

Prochaine mission : Investigation Linux

