

# Correction — DS IF110 : Systèmes d'Exploitation

**Durée : 2 heures — Barème total : 100 points**

## Partie A — Questions de cours (20 points)

1. **Définitions :**

- **Processus :** programme en cours d'exécution, identifié par un PID.
- **Signal :** message asynchrone envoyé à un processus pour provoquer une action.
- **Tube :** canal de communication unidirectionnel entre processus.
- **Mutex :** mécanisme de synchronisation empêchant l'accès concurrent à une ressource.

2. **Tube nommé vs anonyme :**

- **Anonyme :** créé avec `|`, disparaît à la fin du processus. (ex : `cat /etc/passwd | grep root`)
- **Nommé :** créé avec `mkfifo`, persiste dans le système de fichiers. (ex : `mkfifo canal`)

3. **Lien dur vs symbolique :**

- Lien **dur** : même inode, nécessite même partition.
- Lien **symbolique** : raccourci vers un autre fichier (ex : raccourci de bureau).

4. **Flux standards :**

`stdin (0)` : entrée standard ; `stdout (1)` : sortie ; `stderr (2)` : erreur. Permettent la redirection et la communication entre programmes.

5. **Modification de priorité :**

Commande : `nice -n 10 commande` ou `renice -n 5 PID`. Augmente ou réduit la priorité CPU d'un processus.

6. **Section critique :**

Zone du code accédant à une ressource partagée. Pour éviter les conflits, on utilise un **mutex** (exclusion mutuelle) via `P.sh` et `V.sh`.

## Partie B — Commandes et scripts Bash (40 points)

1. **Analyse de commande :**

`cat /etc/passwd | grep root | tee /tmp/result 2>&1` affiche les lignes contenant `*root*` tout en enregistrant la sortie et les erreurs dans `/tmp/result`.

2. **Analyse de script :**

`$$` = PID du shell courant ; `$_` = PID du dernier processus lancé en arrière-plan ; `$_?` = code de retour de la dernière commande.

→ Affiche le PID du shell, celui du processus lancé avec `ls /etc &`, et le code de retour (0 si succès).

3. **Commande `kill -9 \$\$` :**

Envoie le signal `SIGKILL` au shell lui-même → le processus courant se termine immédiatement.

4. **Script de comptage d'utilisateurs :**

```
#!/bin/bash
n=$(who | wc -l)
if [ $n -gt 5 ]; then
echo "Alerte : $n utilisateurs connectés !"
else
echo "Nombre d'utilisateurs : $n"
fi
```

5. **fg / bg :**

Permettent de ramener un processus suspendu (`Ctrl+Z`) au premier plan (`fg`) ou à l'arrière-plan (`bg`).

6. **Commande `exec 3<>tube` :**

Ouvre le tube nommé en lecture/écriture simultanément, évitant les blocages à l'ouverture. Utile pour les communications persistantes.

## Partie C — Étude de cas pratique : Concurrency et synchronisation (40 points)

1. **Problème sans verrouillage :**

Les deux scripts lisent et écrivent simultanément sur le même fichier `compte-igor`, provoquant une perte d'opérations (incohérence des données).

2. **Résolution avec P.sh et V.sh :**

Chaque script entoure la section critique par un verrou :

```
P.sh compte-igor.lock
read a < compte-igor
a=$((a + 2))
echo $a > compte-igor
V.sh compte-igor.lock
```

Ainsi, un seul processus accède à la ressource à la fois.

3. **Pseudo-code :**

boucle infinie : prendre\_verrou(compte-igor.lock) lire solde modifier solde écrire nouveau solde libérer\_verrou(compte-igor.lock)

4. **Interblocage :**

Se produit si plusieurs verrous sont pris dans un ordre différent. Solution : acquérir les verrous **\*\*dans le même ordre global\*\***.

**Total : 100 points**