

Correction : Examen Programmation Système (I/O, Concurrency, Réseau, Process, VFS)

Rédigé par : Dr. Jehan-Antoine Vayssade

20 mars 2025

Instructions

Durée : 2 heures – Note totale : 20 points. Les réponses sont argumentées et les codes sont corrects (Java 11+).

Q1 Programme et processus (2 pt)

Un **programme** est un fichier exécutable stocké sur le disque (code binaire + données). Il est passif.

Un **processus** est une instance en cours d'exécution de ce programme. Il est actif et géré par le noyau.

Différences clés :

- **Format / stockage** : Programme = fichier sur disque ; Processus = structure en mémoire RAM et différents caches.
- **Espace mémoire** : Chaque processus possède son propre espace d'adressage virtuel (isolation complète).
- **Exécution** : Un programme peut engendrer plusieurs processus ; un processus exécute le code du programme.
- **Identifiants** : Programme = nom de fichier et stocke sur un ou plusieurs inode ; Processus = PID (Process ID) + PPID (celui qui la crée).

Q2 Commutation de contexte (2 pts)

Un **context switch** (commutation de contexte) est l'opération par laquelle le noyau suspend l'exécution d'un processus (ou thread) pour en exécuter un autre sur le CPU y compris lui-même (mode kernel).

Éléments sauvegardés et restaurés :

- Registres CPU (généraux, pointeur de pile, compteur de programme, registre d'état).
- Informations de gestion de la mémoire (table des pages, pointeur vers le répertoire de pages).
- État du processus (running, ready, blocked).

Coûts :

- **Coût direct** : Temps CPU nécessaire pour sauvegarder/restaurer les registres.
- **Coût indirect** : Vidage partiel des caches (TLB, cache L1/L2) et pollution du cache par le nouveau processus.

Q3 Copier un fichier (2 pts)

```
1 import java.io.*;
2 import java.nio.file.*;
3
4 public static void copyFile(Path src, Path dst) throws IOException {
5     FileInputStream fis = new FileInputStream(src.toFile());
6     FileOutputStream fos = new FileOutputStream(dst.toFile())
7
8     byte[] buffer = new byte[8192]; // Tampon de 8 Ko
9     int bytesRead;
10    while ((bytesRead = fis.read(buffer)) != -1) {
11        fos.write(buffer, 0, bytesRead);
12    }
13
14    fis.close();
15    fos.close();
16 }
```

Q4 Lecture de fichier PPM P3 (4 pts)

Stratégie robuste : Utiliser `BufferedReader` combiné à `StreamTokenizer` qui ignore automatiquement les commentaires commençant par `#` et traite les espaces multiples comme séparateurs.

Structure de données recommandée : `int[][][] image = new int[height][width][3];`
Justification : accès en temps constant $O(1)$, très adapté aux traitements d'image (rotation, filtres, etc.), mémoire raisonnable pour les fichiers PPM.

Implémentation :

```
1 import java.io.*;
2 import java.util.*;
3
4 public static int[][][] readP3(String filename) throws IOException {
5     try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
6         StreamTokenizer st = new StreamTokenizer(br);
7         st.commentChar('#');
8         st.parseNumbers();
9
10        // Magic number
11        st.nextToken();
12        if (!"P3".equals(st.sval)) {
13            throw new IOException("Format PPM P3 attendu");
14        }
15
16        st.nextToken(); int width = (int) st.nval;
17        st.nextToken(); int height = (int) st.nval;
```

```

18     st.nextToken(); int maxVal = (int) st.nval;
19
20     if (width <= 0 || height <= 0 || maxVal != 255) {
21         throw new IOException("Dimensions ou valeur maximale
22             invalides");
23     }
24
25     int[][][] image = new int[height][width][3];
26
27     for (int y = 0; y < height; y++) {
28         for (int x = 0; x < width; x++) {
29             for (int c = 0; c < 3; c++) {
30                 st.nextToken();
31                 if (st.ttype != StreamTokenizer.TT_NUMBER) {
32                     throw new IOException("Valeur RGB manquante ou
33                         invalide");
34                 }
35                 int val = (int) st.nval;
36                 if (val < 0 || val > 255) {
37                     throw new IOException("Valeur hors plage
38                         [0-255]");
39                 }
40                 image[y][x][c] = val;
41             }
42         }
43     }
44     return image;
45 }

```

Version Scanner :

```

1  import java.io.*;
2  import java.util.*;
3
4  public static int[][][] readP3(String filename) throws IOException {
5      Scanner sc = new Scanner(new File(filename));
6
7      // Gestion des commentaires (# ...)
8      List<String> tokens = new ArrayList<>();
9      while (sc.hasNext()) {
10         String token = sc.next();
11         if (token.startsWith("#")) {
12             if (sc.hasNextLine()) sc.nextLine(); // ignorer le reste de
13                 la ligne
14             continue;
15         }
16         tokens.add(token);
17     }
18
19     Iterator<String> it = tokens.iterator();
20
21     // Magic number
22     if (!it.hasNext() || !"P3".equals(it.next())) {

```

```

22         throw new IOException("Format PPM P3 attendu");
23     }
24
25     // Dimensions et maxVal
26     if (!it.hasNext()) throw new IOException("Largeur manquante");
27     int width = Integer.parseInt(it.next());
28
29     if (!it.hasNext()) throw new IOException("Hauteur manquante");
30     int height = Integer.parseInt(it.next());
31
32     if (!it.hasNext()) throw new IOException("Valeur max manquante");
33     int maxVal = Integer.parseInt(it.next());
34
35     if (width <= 0 || height <= 0 || maxVal != 255) {
36         throw new IOException("Dimensions ou valeur maximale invalides");
37     }
38
39     int[][][] image = new int[height][width][3];
40
41     // Lecture des pixels
42     for (int y = 0; y < height; y++) {
43         for (int x = 0; x < width; x++) {
44             for (int c = 0; c < 3; c++) {
45                 if (!it.hasNext()) {
46                     throw new IOException("Valeur RGB manquante");
47                 }
48                 int val = Integer.parseInt(it.next());
49                 if (val < 0 || val > 255) {
50                     throw new IOException("Valeur hors plage [0-255]");
51                 }
52                 image[y][x][c] = val;
53             }
54         }
55     }
56
57     return image;
58 }
59 }

```

Q5 Concurrency (3 pts)

Problème : Race condition. L'opération `c.value++` n'est pas atomique (lecture + incrémentation + écriture).

Scénario d'erreur : Les deux threads lisent la même valeur avant que l'un n'ait écrit la nouvelle, entraînant la perte d'incrémentations.

Correction (avec synchronisation, boucles en parallèle) :

```

1 class Counter {
2     private int value = 0;
3
4     public synchronized void increment() {

```

```

5         value++;
6     }
7
8     public int getValue() {
9         return value;
10    }
11}
12
13// Utilisation :
14Counter c = new Counter();
15Thread t1 = new Thread(() -> {
16    for (int i = 0; i < 1_000_000; i++) c.increment();
17});
18Thread t2 = new Thread(() -> {
19    for (int i = 0; i < 1_000_000; i++) c.increment();
20});
21t1.start(); t2.start();
22t1.join(); t2.join();
23System.out.println(c.getValue()); // 2_000_000

```

Q6 Sémaphore (2 pts)

- **Sémaphore binaire** : compteur initialisé à 1 (similaire à un mutex).
- **Mutex** : sémaphore binaire avec propriété (seul le thread propriétaire peut le libérer).
- **Sémaphore comptant** : compteur initialisé à $N > 1$, autorise N accès simultanés.

Exemple d'utilisation (limiter à 3 threads) :

```

1 Semaphore dbSemaphore = new Semaphore(3); // Maximum 3 connexions
   simultan es
2
3 // Dans un thread :
4 try {
5     dbSemaphore.acquire();
6     // Utilisation de la ressource (connexion DB)
7 } finally {
8     dbSemaphore.release();
9 }

```

Q7 Adressage des données d'un inode (3 pts)

Un inode contient typiquement 12 pointeurs directs et des pointeurs d'indirection.

Données : taille de bloc = 4 Ko (4096 octets), taille d'un pointeur = 4 octets.

- **12 pointeurs directs** : $12 \times 4096 = 49\,152$ octets (≈ 48 KiB).
- **Indirection simple** : Un bloc d'indirection contient $4096/4 = 1024$ pointeurs. Capacité : $1024 \times 4096 = 4\,194\,304$ octets (≈ 4 MiB).

Q8 Liens (2 pt)

- **Lien dur (hard link)** : Nouvelle entrée de répertoire pointant vers le même **inode**. Le fichier est supprimé uniquement quand le compteur de liens durs atteint 0. **Risque** : Ne fonctionne pas entre systèmes de fichiers différents ; peut créer des confusions.
- **Lien symbolique (symlink)** : Fichier contenant le chemin vers la cible. **Risque** : Devient invalide (« dangling ») si la cible est supprimée ou déplacée.