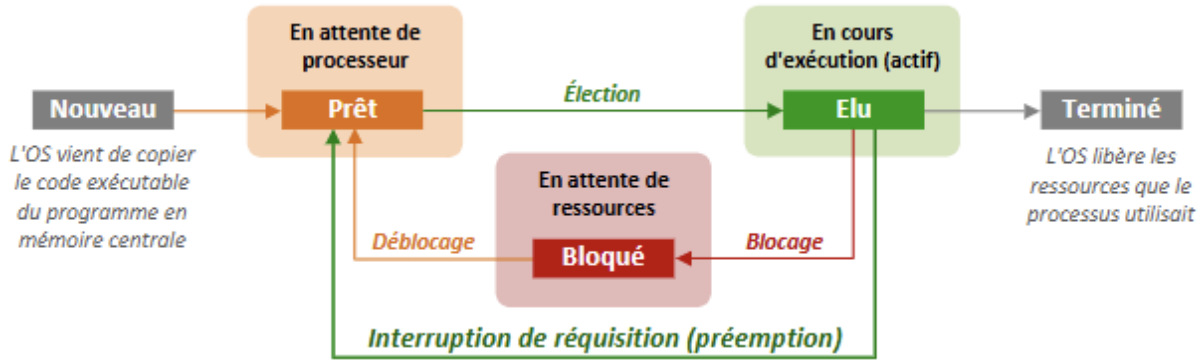
Correction 26-NSIJ2AN1

Exercice 1

Partie A

1. 

2. Dans cette situation, P1 est prêt, P2 est bloqué et P3 élu.

3. Un interblocage de processus peut se produire lorsque plusieurs processus sont en attente de ressources détenues par d'autres processus, créant ainsi une situation où aucun processus ne peut avancer. Cette situation peut survenir lorsque les conditions (nécessaires) de Coffman sont remplies : exclusion mutuelle, attente circulaire, non-préemption et rétention de ressources.

4. On peut bien tomber dans la situation décrite par Coffman, car, suivant l'ordre d'execution concurente des processus, il est possible que :
   - P1 détient MIC et attend CAL
   - P2 détient CAL et attend MIC

5. L'intérêt d'avoir plusieurs processeurs de disponibles est de pouvoir exécuter plusieurs processus en parallèle, ce qui peut améliorer les performances et réduire les temps d'attente. Cependant, cela n'empâche pas le risque d'interblocage !

6. Des avantages possibles pour les systèmes sur puces est qu'ils sont généralement plus compacts, consomment moins d'énergie et sont plus économiques que les systèmes à microprocesseurs traditionnels. Cependant, ils peuvent être moins puissants et moins flexibles que les systèmes à microprocesseurs, ce qui peut limiter leur utilisation dans certaines applications.

Partie B

7. Par ms : P1, P1, P2, P2, P3, P3, P4, P4, P1, P1, P2, P2, P3, P3, P4, P1, P1, P3.

8. ```
   fp = creer_file_vide()
   enfiler(fp, P1)
   enfiler(fp, P2)
   enfiler(fp, P3)
   enfiler(fp, P4)
   ```

9. ```
   def execute_un_processus(file_d_attente, t):
       processus = defiler(file_d_attente)
       if processus['temps'] > quantum:
           processus['temps'] = processus['temps'] - quantum
           enfiler(file_d_attente, processus)
           return t + quantum
       else:
           return t + processus['temps']
   ```

10.
```
def execute_tous_processus(file_d_attente):
    t = 0
    while not est_vide(file_d_attente):
        t = execute_un_processus(file_d_attente, t)
    return t
```

Exercice 2

Partie A

1. `pegula = Joueuse("Pegula", "Jessica", "USA", 29, 6101)`

2.
```
    def ajouter_victoire(self, adversaire):
        self.victoire += 1
        adversaire.defaite += 1
```

3. `swiatek.ajouter_victoire(paloni)`

4. Le coût temporel est quadratique : $O(n^2)$.

5.

Etape	Contenu de `liste_joueuses`
0	`[swiatek, gauff, paloni, sabalenka, pegula]`
1	`[gauff, swiatek, paloni, sabalenka, pegula]`
2	`[paloni, gauff, swiatek, sabalenka, pegula]`

6.
```
    def resultat_match(self, score):
        self.score = score
        nb_set_joueuse1 = 0
        nb_set_joueuse2 = 0
        for s1, s2 in score:
            if s1 > s2:
                nb_set_joueuse1 += 1
            else:
                nb_set_joueuse2 += 1
        if nb_set_joueuse1 > nb_set_joueuse2:
            self.gagnante = self.joueuse1
            self.perdante = self.joueuse2
        else:
            self.gagnante = self.joueuse2
            self.perdante = self.joueuse1
        self.gagnante.ajouter_victoire(self.perdante)
```

Partie B

7. Le tournoi de tennis peut être modélisé par un arbre binaire, car chaque match oppose deux joueuses et la gagnante avance à l'étape suivante.

8. `tournoi = Arbre(F, Arbre(D1, Arbre(Q1, None, None), Arbre(Q2, None, None)), Arbre(D2, Arbre(Q3, None, None), Arbre(Q4, None, None)))`

9. `tournoi.gauche.racine.joueuse1 = gauff`

10. Un programme récursif est un programme qui s'appelle lui-même, directement ou indirectement, pour résoudre un problème en le décomposant en sous-problèmes plus simples. Il est généralement composé d'un cas de base qui arrête la récursion et d'un cas récursif qui divise le problème en sous-problèmes et les résout en appelant la fonction elle-même.

Antoine Dequay

11.
```python
    def mise_a_jour(self):
        """Met à jour les matchs de l'arbre"""
        if self.racine.joueuse1 is None:
            if self.gauche is not None:
                # mise à jour si gagnante à gauche
                if self.gauche.racine.gagnante is not None:
                    self.racine.joueuse1 = self.gauche.racine.gagnante
                else:
                    self.gauche.mise_a_jour()
        if self.racine.joueuse2 is None:
            if self.droit is not None:
                # mise à jour si gagnante à droite
                if self.droit.racine.gagnante is not None:
                    self.racine.joueuse2 = self.droit.racine.gagnante
                else:
                    self.droit.mise_a_jour()
```

Exercice 3

Partie A

1. `num_dossard` est le numéro de dossard du coureur, il est donc unique pour chaque coureur. Par conséquent, il peut être utilisé comme clé primaire.

2. La requête renvoie le nom et le prénom des coureurs, triés dans l'ordre alphabétique des noms, dans une table. Il renvoie donc :

Nom	Prénom
BODIANE	Lola
BRELET	Sandra
DA SILVA	José
HANG LI	Léo

3.
```sql
SELECT nom, prenom
FROM coureur
WHERE sexe = 'F';
```

4.
```sql
SELECT COUNT(*)
FROM coureur;
```

5.
```sql
INSERT INTO coureur (nom, prenom, annee, sexe, id_epreuve, temps)
VALUES ('RENY', 'Patrice', 1973, 'M', 1, 0);
```

6.
```sql
DELETE FROM coureur
WHERE num_dossard = 137;
```

7.
```sql
SELECT distance, horaire
FROM epreuve JOIN coureur ON epreuve.id_epreuve = coureur.id_epreuve
WHERE coureur.num_dossard = 256;
```

8. On suppose que la nouvelle table s'appelle , on a alors :

```sql
SELECT dossard, nom, prenom, temps
FROM resultats JOIN coureur ON dossard = num_dossard
ORDER BY temps;
```

Si le temps a directement été mis à jour dans la table , alors il suffit de faire :

```sql
SELECT num_dossard, nom, prenom, temps
FROM coureur
ORDER BY temps;
```

Partie B

9. On trouve `1000`.

10. `dict_perf_5km[2026] = [1004, 1016, 1000, 1140, 1023, 1024]`

11.
```
def scratch(dico, annee):
    min_val = dico[annee][0]
    for e in dico[annee]:
        if e < min_val:
            min_val = e
    return min_val
```

12. C'est la valeur 2.

13. On aura l'erreur `"local variable 'i_cat' referenced before assignment"`, car la variable `i_cat` n'est définie que si la catégorie `cat` existe dans la liste `categories`.

14. `assert cat in categories, "La catégorie doit être dans la liste des catégories."`

15. Cela donne la moyenne sur toutes les années des meilleurs temps au 5 km dans la catégorie SH : $\frac{900+1100+1000+1000+1000}{5} = 1000$.

16.
```
def records(dico):
    records = []
    for i in range(6):
        record = 60 * 60 * 24  # 24h en secondes
        for annee in dico.keys():
            if dico[annee][i] < record:
                record = dico[annee][i]
        records.append(record)
    return records
```