CORRIGÉ NSI 2026 Amérique du Nord – Jour 2

Exercice 1 (6 points)

Question 1

Compléter le schéma des états d'un processus.

Le modèle classique est :

```
Création → Prêt
Prêt --élection--> Élu
Élu --préemption--> Prêt
Élu --blocage--> Bloqué
Bloqué --déblocage--> Prêt
```

Réponse à placer :

```
gauche : prêt
droite : élu
bas : bloqué

flèche haute : élection
flèche droite-bas : blocage
flèche bas-gauche : déblocage
```

Question 2

Déterminer l'état de chaque processus.

P1

L'application est en arrière-plan.

Le processus d'interface graphique n'a rien à faire.

```
P1 : prêt
```

P2

Attend la carte WIFI.

Il attend une ressource.

```
P2 : bloqué
```

P3

Décode la musique.

Il travaille actuellement.

```
P3 : élu
```

Question 3

Quand un interblocage peut-il apparaître ?

Réponse :

```
Un interblocage survient lorsque plusieurs processus
attendent mutuellement une ressource détenue par un autre
processus et qu'aucun ne peut continuer.
```

Exemple :

```
P1 possède A et attend B
P2 possède B et attend A
```

Aucun des deux ne peut avancer.

Question 4

Justifier qu'un interblocage peut apparaître.

On observe :

P1

```
MIC puis CAL
```

P3

```
CAL puis MIC
```

Situation possible :

```
P1 obtient MIC
P3 obtient CAL
P1 attend CAL
P3 attend MIC
```

Donc :

```
interblocage
```

Question 5

Intérêt de plusieurs processeurs.

Réponse :

```
Plusieurs processus peuvent réellement
s'exécuter simultanément.

Les performances augmentent et le temps
d'attente diminue.
```

Question 6

Système sur puce (SoC).

Avantage

```
faible consommation énergétique
```

ou

```
gain de place
```

Inconvénient

```
réparation difficile
```

ou

```
évolution matérielle limitée
```

Partie B

Question 7

Ordonnancement Round Robin

Quantum :

```
2 ms
```

Données :

Processus	Arrivée	Temps
P1	0	6
P2	1	4
P3	3	5
P4	5	3

Chronogramme complet

```
0-2  : P1
2-4  : P2
4-6  : P1
6-8  : P3
8-10 : P2
10-12 : P4
12-14 : P1
14-16 : P3
16-17 : P4
17-18 : P3
```

Fin :

```
18 ms
```

Question 8

Créer la file.

```
fp = creer_file_vide()

enfiler(fp, P1)
enfiler(fp, P2)
enfiler(fp, P3)
enfiler(fp, P4)
```

Question 9

Compléter execute_un_processus.

Correction :

```
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']
```

Question 10

Compléter execute_tous_processus.

```
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 : Tennis – POO – Arbres binaires

Question 1

Instancier Pegula.

```
pegula = Joueuse(
    "Pegula",
    "Jessica",
    "USA",
    29,
```

```
    6101
)
```

Question 2

Méthode ajouter_victoire.

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

Question 3

Finale remportée par Swiatek.

`swiatek.ajouter_victoire(paloni)`

Question 4

Coût du tri par insertion.

Réponse :

$O(n^2)$

Question 5

Tri complet

Ordre croissant des points :

```
Paloni : 4843
Gauff : 6063
Pegula : 6101
Swiatek : 7470
Sabalenka : 10541
```

Étapes :

```
0
[swiatek, gauff, paloni, sabalenka, pegula]

1
[gauff, swiatek, paloni, sabalenka, pegula]

2
[gauff, paloni, swiatek, sabalenka, pegula]

3
[paloni, gauff, swiatek, sabalenka, pegula]

4
[paloni, gauff, swiatek, pegula, sabalenka]

5
[paloni, gauff, pegula, swiatek, sabalenka]
```

Question 6

Méthode resultat_match.

```
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)
```

Question 7

Pourquoi un arbre binaire ?

Chaque match dépend exactement :

`deux matchs précédents`

Chaque nœud possède donc :

`deux fils`

=> arbre binaire.

Question 8

Créer tournoi.

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

Question 9

Mettre à jour la demi-finale.

`tournoi.gauche.racine.joueuse1 = gauff`

Question 10

Définition récursivité.

```
Une fonction récursive est une fonction
qui s'appelle elle-même.
```

Question 11

Compléter mise_a_jour.

```
6  if self.gauche.racine.gagnante is not None

7  self.racine.joueuse1

13 if self.droit.racine.gagnante is not None

14 self.racine.joueuse2
```

```
16 self.droit.mise_a_jour()
```

EXERCICE 3 : SQL et dictionnaires

Question 1

Pourquoi num_dossard est une clé primaire ?

```
Chaque dossard est unique.
```

Il identifie un seul coureur.

Question 2

Requête SQL.

```
SELECT nom, prenom
FROM coureur
ORDER BY nom;
```

Renvoie :

```
noms et prénoms des coureurs
classés par ordre alphabétique du nom
```

Question 3

Femmes inscrites.

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

Question 4

Nombre total d'inscrits.

```
SELECT COUNT(*)
FROM coureur;
```

Question 5

Insertion de Patrice REMY.

```
INSERT INTO coureur
(nom, prenom, annee, sexe, id_epreuve, temps)
VALUES
('REMY','Patrice',1973,'H',1,0);
```

Question 6

Suppression du dossard 137.

```
DELETE FROM coureur
WHERE num_dossard=137;
```

Question 7

Distance et horaire.

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

Question 8

Classement Master Femme 10 km.

```
SELECT num_dossard,
       nom,
       prenom,
       temps
FROM coureur
WHERE sexe='F'
AND annee < 1986
AND id_epreuve = 2
ORDER BY temps;
```

Question 9

Valeur de :

```
dict_perf_5km[2025][2]
```

Réponse :

```
1000
```

Question 10

Ajout 2026.

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

Question 11

Fonction scratch.

```
def scratch(dico, annee):
    liste = dico[annee]
    meilleur = liste[0]
    for t in liste:
        if t < meilleur:
            meilleur = t
    return meilleur
```

Question 12

Valeur de i_cat.

```
SH
```

Position :

```
2
```

Question 13

Erreur obtenue.

```
local variable 'i_cat'
referenced before assignment
```

Question 14

Assertion.

```
assert cat in categories,\
"Catégorie inconnue"
```

Question 15

Résultat de

`mystere(dict_perf_5km,'SH')`

Calcul :

```
900
1100
1000
1000
```

Moyenne :

```
(900+1100+1000+1000)/4
= 1000
```

Réponse :

```
1000
```

Question 16

Fonction records.

```
def records(dico):

    resultats = []

    for categorie in range(6):

        meilleur = 24*3600

        for annee in dico:

            if dico[annee][categorie] < meilleur:
                meilleur = dico[annee][categorie]

        resultats.append(meilleur)

    return resultats
```

Appréciation de correcteur

Ce sujet Jour 2 est légèrement plus accessible que le Jour 1. Les principales difficultés se trouvent :

- dans l'ordonnancement Round Robin ;
- la méthode `resultat_match` ;
- la récursivité dans l'arbre du tournoi ;
- la fonction `records`.

Une copie avec des réponses SQL correctes et les programmes complets, permet raisonnablement d'obtenir une très bonne note.