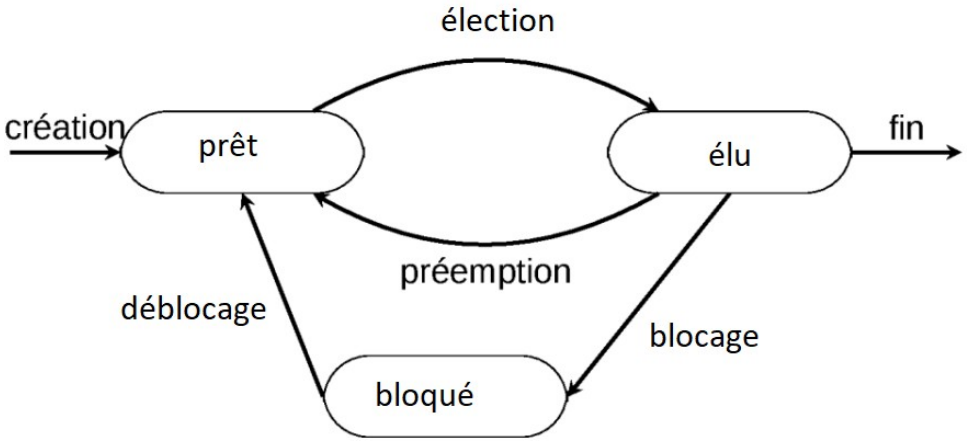
26-NSIJ2AN1

Correction

Exercice 1

Partie A

Q.1



Q.2

P1 prêt (il n'a pas besoin d'accéder à l'interface utilisateur)
P2 bloqué
P3 élu
/ autre réponse peut-être acceptable : P1 bloqué (en attente de l'interface)

Q.3

Un interblocage (*deadlock*) intervient par exemple quand un processus accède à une ressource, par exemple la carte Wifi et attend qu'une autre ressource, par exemple le disque dur, se libère et au même moment un autre processus accède au disque dur et attend que la carte Wifi se libère.

Q.4

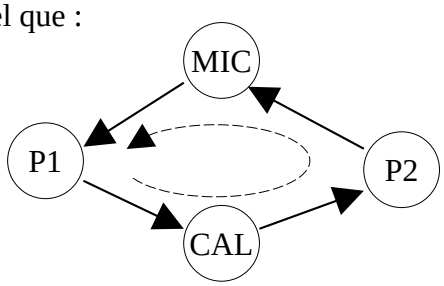
P1 accède à MIC et attend que CAL se libère mais P2 accède déjà à CAL et attend que MIC se libère. Il y a interblocage.

On peut aussi représenter la situation par un graphe orienté tel que :

R → P ressource attribuée.

P → R ressource demandée

La présence d'un cycle montre qu'il y a interblocage.



Q.5

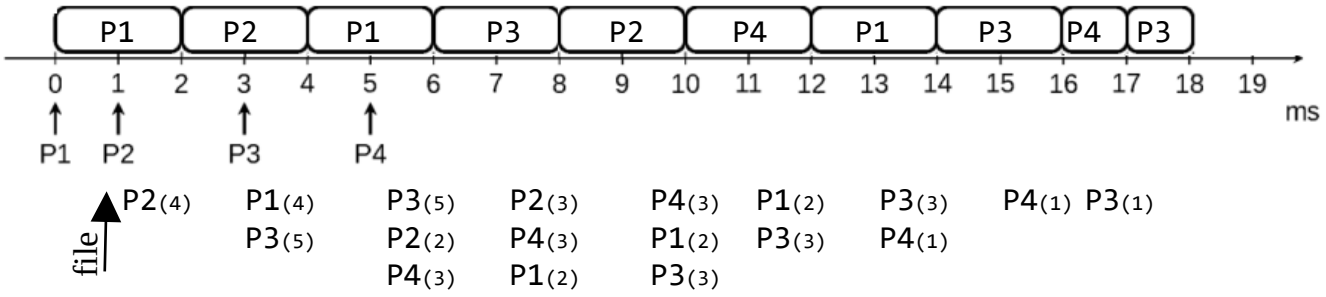
Une machine avec plusieurs processeurs permettra à plusieurs processus de s'exécuter en même temps.

Q.6

Les systèmes sur puce (SoC pour *System on a Chip*) ont l'avantage d'être plus intégrés, donc consomment moins d'énergie pour fonctionner sur batterie. Par contre ils ont le désavantage de ne pas permettre de mettre à jour le matériel ou de remplacer un composant en cas de panne.

Partie B

Q.7



Q.8

```
f = creer_file_vide()
enfiler(f, P1)
enfiler(f, P2)
enfiler(f, P3)
enfiler(f, P4)
```

Q.9

```
1 def execute_un_processus(file_d_attente, t):
2     processus = defiler(file_d_attente)
3     if processus['temps'] > quantum:
4         processus['temps'] = processus['temps'] - quantum
5         enfiler(file_d_attente, processus)
6         return t + quantum
7     else:
8         return t + processus['temps']
```

Q.10

```
1 def execute_tous_processus(file_d_attente):
2     t = 0
3     while not est_vide(file_d_attente):
4         t = execute_un_processus(file_d_attente, t)
5     return t
```

Exercice 2

Partie A

Q.1

```
pegula = Joueuse('Pegula', 'Jessica', 'USA', 29', 6101)
```

Q.2

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

Q.3

```
swiatek.ajouter_victoire(paloni)
```

Q.4

Le tri par insertion a un coût quadratique, en $O(n^2)$ dans le pire des cas.

Q.5

Étape	Contenu de `liste_joueuses`
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]

Q.6

```
1 def resultat_match(self, score):
2     self.score = score
3     nb_set_joueuse1 = 0
4     nb_set_joueuse2 = 0
5     for set in score:
6         if set[0] > set[1]:
7             nb_set_joueuse1 += 1
8         else:
9             nb_set_joueuse2 += 1
10    if  nb_set_joueuse1 > nb_set_joueuse2:
12        self.gagnante = self.joueuse1
13        self.perdante = self.joueuse2
14    else:
15        self.gagnante = self.joueuse2
16        self.perdante = self.joueuse1
17    self.gagnante.ajouter_victoire(self.perdante)
```

Partie B

Q.7

Un tournoi de tennis se modélise naturellement par un arbre binaire car chaque match oppose exactement 2 joueuses, et produit exactement 1 gagnante. C'est la définition d'un nœud dans un arbre binaire : un nœud a au plus 2 fils et 1 parent.

Q.8

```
N11 = Arbre(Q1, None, None)
N12 = Arbre(Q2, None, None)
N23 = Arbre(Q3, None, None)
N24 = Arbre(Q4, None, None)
N1 = Arbre(D1, N11, N12)
N2 = Arbre(D2, N23, N24)
tournoi = Arbre(F, N1, N2)
```

Q.9

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

Q.10

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

Q.11

```
1 def mise_a_jour(self):
2     """Met à jour les matchs de l'arbre"""
3     if self.racine.joueuse1 is None:
4         if self.gauche is not None:
5             # mise à jour si gagnante à gauche
6             if self.gauche.racine.gagnante is not None:
7                 self.racine.joueuse1 = self.gauche.racine.gagnante
8             else:
9                 self.gauche.mise_a_jour()
10    if self.racine.joueuse2 is None:
11        if self.droit is not None:
12            # mise à jour si gagnante à droite
13            if self.droit.racine.gagnante is not None:
14                self.racine.joueuse2 = self.droit.racine.gagnante
15            else:
16                self.droit.mise_a_jour()
```

Exercice 3

Partie A

Q.1

Chaque coureur a un numéro de dossard unique, `num_dossard` permet donc d'identifier chaque ligne de la table `coureur` de façon unique, il peut donc être utilisé comme clé primaire.

Q.2

La requête donne les noms et prénoms de tous les coureurs, triés par ordre alphabétique de leur nom. Sur l'extrait de la table, on obtinet :

```
BODIANE    Lola
BRELET     Sandra
DA SILVA   José
HANG LI    Léo
```

Q.3

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

Q.4

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

Q.5

En supposant que le numéro de dossard 42 n'a pas encore été donné :

```
INSERT INTO coureur
VALUES (42, 'REMY', 'Patrice', 1973, 'H', 1, 0) ;
```

Q.6

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

Q.7

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

Q.8

```
SELECT num_dossard, nom, prenom, temps
FROM coureur
WHERE id_epreuve = 2 AND sexe = 'F' AND annee <= 1986 AND temps > 0
ORDER BY temps ;
```

Partie B

Q.9

```
1000
```

Q.10

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

Q.11

```
def scratch(dico, annee):
    meilleur_temps = None
    for temps in dico[annee]:
         if meilleur_temps is None or temps <  meilleur_temps:
             meilleur_temps = temps
    return meilleur_temps
```

Q.12

2

Q.13

"local variable 'i_cat' referenced before assignement"

Q.14

```
assert cat in categories, "catégorie inexistante"
```

Q.15

1000 (= (900+1100+1000+1000+1000) / 5)

Q.16

```
def records(dico):
    liste_meilleurs_temps = []
    categories = ['JH','JF','SH','SF','MH','MF']
    for i in range(len(categories)):
          meilleurs_temps = 24 * 60 * 60
          for annee in dico.keys():
              if dico[annee][i] <  meilleurs_temps:
                     meilleurs_temps = dico[annee][i]
          liste_meilleurs_temps.append(meilleurs_temps)
    return liste_meilleurs_temps
```