BACCALAURÉAT GÉNÉRAL BLANC

NSI

Epreuve de spécialité

DURÉE DE L'ÉPREUVE : **3 h 30**

L'usage d'une calculatrice N´EST PAS autorisé

Exercice 1 : COMMANDES UNIX, PROCESSUS (7 POINTS)

Exercice 2 : CROISIERES EN BATEAU (6 POINTS)

Exercice 3 : ARBRES BINAIRES DE RECHERCHE (7 POINTS)

EXERCICE 1 : COMMANDES UNIX, PROCESSUS (7 points)

Cet exercice porte sur les systèmes d'exploitation, les commandes UNIX, les structures de données (de type LIFO et FIFO) et les processus.

"Linux ou GNU/Linux est une famille de systèmes d'exploitation open source de type Unix fondée sur le noyau Linux, créé en 1991 par Linus Torvalds. De nombreuses distributions Linux ont depuis vu le jour et constituent un important vecteur de popularisation du mouvement du logiciel libre."

Source : Wikipédia, extrait de l'article consacré à GNU/Linux.

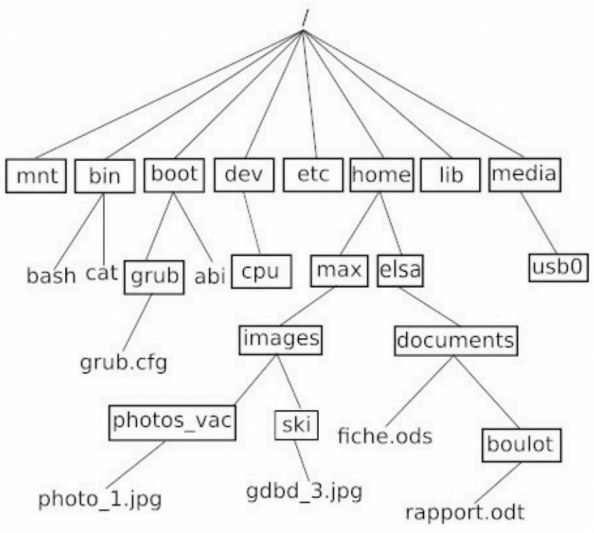
"*Windows est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC. Windows est un système d'exploitation propriétaire."*

Source : Wikipédia, extrait de l'article consacré à Windows.

1. Expliquer succinctement les différences entre les logiciels libres et les logiciels propriétaires.

2. Expliquer le rôle d'un système d'exploitation.

On donne ci-dessous une arborescence de fichiers sur un système GNU/Linux (les noms encadrés représentent des répertoires, les noms non encadrés représentent des fichiers, / correspond à la racine du système de fichiers) :



3. Indiquer le chemin absolu du fichier `rapport.odt`.

On suppose que le répertoire courant est le répertoire `elsa`.

4. Indiquer le chemin relatif du fichier `photo_1.jpg`.

L'utilisatrice Elsa ouvre une console (aussi parfois appelée terminal), le répertoire courant étant toujours le répertoire `elsa`. On fournit ci-dessous un extrait du manuel de la commande UNIX `cp` :

```
NOM
     cp - copie un fichier

UTILISATION
     cp fichier_source fichier_destination
```

5. Déterminer le contenu du répertoire `documents` et du répertoire `boulot` après avoir exécuté la commande suivante dans la console :

```
cp documents/fiche.ods documents/boulot
```

“Un système d’exploitation est multitâche (en anglais : multitasking) s’il permet d’exécuter, de façon apparemment simultanée, plusieurs programmes informatiques. GNU/Linux, comme tous les systèmes d’exploitation modernes, gère le multitâche.”

“Dans le cas de l’utilisation d’un monoprocesseur, la simultanéité apparente est le résultat de l’alternance rapide d’exécution des processus présents en mémoire.”

Source : Wikipédia, extraits de l’article consacré au Multitâche.

Dans la suite de l’exercice, on s’intéresse aux processus. On considère qu’un monoprocesseur est utilisé. On rappelle qu’un processus est un programme en cours d’exécution. Un processus est soit élu, soit bloqué, soit prêt.

6. Recopier et compléter le schéma ci-dessous avec les termes suivants :

élu, bloqué, prêt, élection, blocage, déblocage.

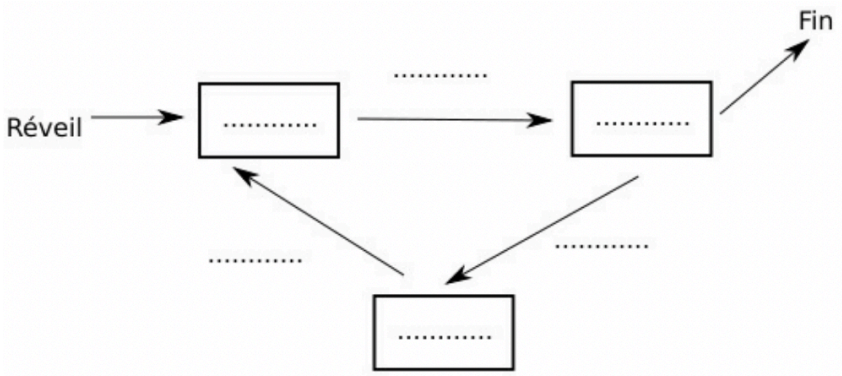


Figure 2. Schéma processus

7. Donner l’exemple d’une situation qui contraint un processus à passer de l’état élu à l’état bloqué.

“Dans les systèmes d’exploitation, l’ordonnanceur est le composant du noyau du système d’exploitation choisissant l’ordre d’exécution des processus sur le processeur d’un ordinateur.”

Source : Wikipédia, extrait de l’article consacré à l’ordonnancement.

L’ordonnanceur peut utiliser plusieurs types d’algorithmes pour gérer les processus. L’algorithme d’ordonnancement par “ordre de soumission” est un algorithme de type FIFO (First In First Out), il utilise donc une file.

8. Nommer une structure de données linéaire de type LIFO (Last In First Out).

À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).

Par exemple, l'exécution d'un processus P4 qui a un instant d'arrivée égal à 7 et une durée d'exécution égale à 2 peut être représentée par le schéma suivant :

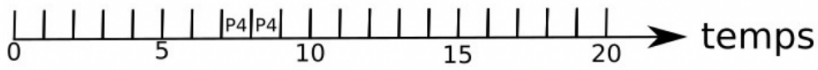


Figure 3. Utilisation du processeur

L'ordonnanceur place les processus qui ont besoin d'un accès au processeur dans une file, en respectant leur ordre d'arrivée (le premier arrivé étant placé en tête de file). Dès qu'un processus a terminé son exécution, l'ordonnanceur donne l'accès au processus suivant dans la file.

Le tableau suivant présente les instants d'arrivées et les durées d'exécution de cinq processus :

5 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	3
P2	1	6
P3	4	4
P4	6	2
P5	7	1

9. Recopier et compléter le schéma ci-dessous avec les processus P1 à P5 en utilisant les informations présentes dans le tableau ci-dessus et l'algorithme d'ordonnancement "*par ordre de soumission*".

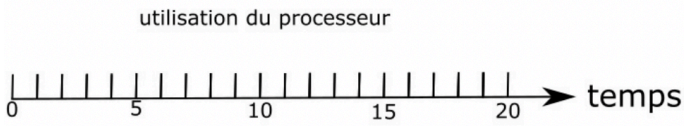


Figure 4. Utilisation du processeur

On utilise maintenant un autre algorithme d'ordonnancement : l'algorithme d'ordonnancement "par tourniquet". Dans cet algorithme, la durée d'exécution d'un processus ne peut pas dépasser une durée Q appelée quantum et fixée à l'avance.

Si ce processus a besoin de plus de temps pour terminer son exécution, il doit retourner dans la file et attendre son tour pour poursuivre son exécution.

Par exemple, si un processus P1 a une durée d'exécution de 3 et que la valeur de Q a été fixée à 2, P1 s'exécutera pendant deux unités de temps avant de retourner à la fin de la file pour attendre son tour ; une fois à nouveau élu, il pourra terminer de s'exécuter pendant sa troisième et dernière unité de temps d'exécution.

10. Recopier et compléter le schéma ci-dessous, en utilisant l'algorithme d'ordonnancement "par tourniquet" et les mêmes données que pour la question 9, en supposant que le quantum Q est fixé 2.

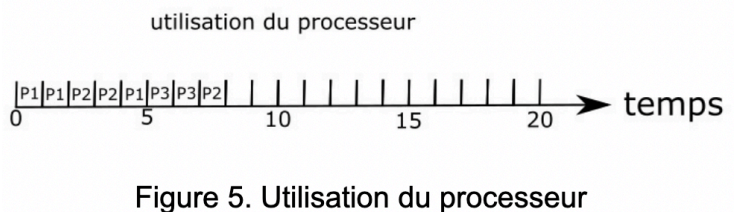


Figure 5. Utilisation du processeur

On considère deux processus P1 et P2, et deux ressources R1 et R2.

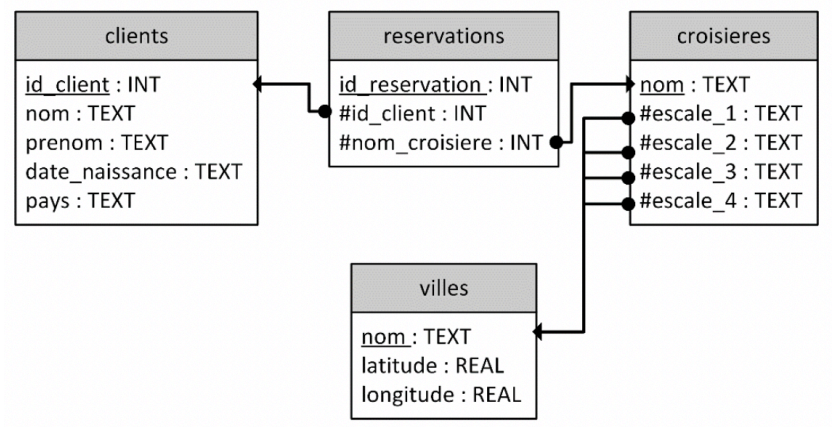
11. Décrire une situation qui conduit les deux processus P1 et P2 en situation d'interblocage.

EXERCICE 2 : CROISIERES EN BATEAU (6 points)

Cet exercice traite de protocoles de routage, de sécurité des communications et de base de données relationnelle.

Une agence de voyage propose des croisières en bateau. Chaque croisière a un nom unique et passe par quatre escales correspondant à des villes qui ont elles aussi des noms différents.

Pour gérer les réservations de ses clients, l'agence utilise une base de données. Voici la description des trois relations de cette base dont les clés primaires ont été soulignées et les clés étrangères indiquées par un # :



Remarque : l'énoncé de cet exercice utilise tout ou une partie des mots suivants du langage SQL : `SELECT, FROM, WHERE, JOIN ON, INSERT INTO, VALUES, UPDATE SET, OR, AND.`

Partie A

L'agence de voyage possède deux bureaux distincts.

Elle passe par un prestataire de service qui héberge sa base de données et utilise um système de gestion de base de données relationnelle.

Vous trouverez ci-après un schéma du réseau entre les deux bureaux de l'agence de voyage et le prestataire. On peut y voir les différents routeurs (nommés de A à I) ainsi que le coût des liaisons entre eux.

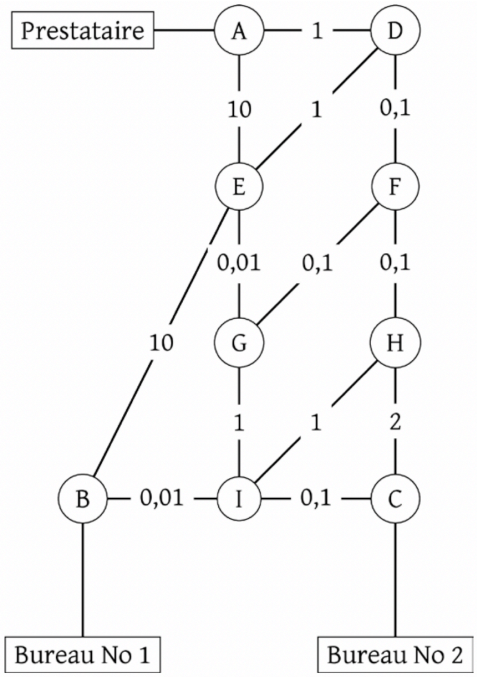


Figure 1. Topologie du réseau

1. Donner deux services rendus par un système de gestion de bases de données relationnelles.

Le protocole RIP (Routing Information Protocol) est un protocole de routage qui minimise le nombre de routeurs par lesquels les paquets transitent.
Le protocole OSPF (Open Shortest Path First) est un protocole de routage qui minimise le coût du transit des paquets.

2. Donner la route suivie par une requête issue du bureau numéro 1 jusqu'au prestataire si on utilise le protocole RIP.

3. Donner les deux routes que pourrait suivre une requête issue du bureau numéro 2 jusqu'au prestataire si on utilise le protocole OSPF. Donner le coût de chaque route.

Partie B

4. Expliquer pourquoi l'attribut `id_client` a été choisi comme clé primaire dans la relation `clients`.

5. Définir ce qu'est une clé étrangère. Donner la ou les clés étrangères de chaque relation qui en a en précisant la clé primaire qu'elles référencent.

L'agence a obtenu l'autorisation de faire escale dans quatre nouvelles villes : *Puerto saibo, Puerto kifecho, Puerto kifebo* et *Puerto repo*. Elle souhaite créer une nouvelle croisière qui passera par ces quatre villes. Un stagiaire de l'agence demande de l'aide à une Intelligence Artificielle (IA) :

Pour ajouter la nouvelle croisière nommée 'Croisière Puerto' avec ses escales correspondantes, vous pouvez utiliser la requête suivante :

sql Copy code

```
INSERT INTO croisieres (nom, escale_1, escale_2, escale_3,
escale_4)
VALUES ('Croisière Puerto',
'Puerto sebo',
'Puerto kifecho',
'Puerto kifebo',
'Puerto repo');
```

Figure 2 - Réponse de l'IA

Il tape alors la requête proposée mais obtient le message d'erreur suivant du SGBD (Systèmes de Gestion de Bases de Données) : `FOREIGN KEY constraint failed.`

6. Expliquer l'erreur commise et proposer une solution.

Partie C

7. Jean Barc, un allemand né le 29 juin 1972, demande un geste commercial en raison de sa fidélité à l'agence. Expliquer les requêtes SQL suivantes saisies par le gestionnaire :

```
SELECT id FROM clients
WHERE nom = 'Barc' AND prenom = 'Jean' AND date_naissance =
'1972/06/29' AND pays = 'Allemagne';
```

```
SELECT id_reservation FROM reservations
WHERE id client = 1243;
```

8. Écrire les deux requêtes de la **question 7** sous la forme d'une requête unique.

9. Un client souhaite modifier sa réservation d'identifiant `20456`. Il souhaite remplacer la croisière de cette réservation par la toute dernière offre de l'agence : la *Croisière Puerto*. Écrire une requête SQL qui permet de mettre à jour la base de données pour lui donner satisfaction.

10. Donner une requête SQL permettant d'obtenir les noms, prénoms et dates de naissance des clients ayant choisi la croisière nommée *Croisière Piano* ou celle nommée *Croisière Puerto*.

EXERCICE 3 : ARBRES BINAIRES DE RECHERCHE (7 points)

Cet exercice porte sur les arbres binaires de recherche, la POO et la récursivité.

Nous disposons d'une classe `ABR` pour les arbres binaires de recherche dont les clés (ou valeurs) sont des entiers :

```
1  class ABR():
2      def __init__(self) :
3          # Initialise une instance d'ABR vide.
4
5      def cle(self):
6          # Renvoie la clé de la racine de l'instance d'ABR.
7
8      def sad(self):
9          # Renvoie le sous-arbre droit de l'instance d'ABR.
10
11     def sag(self):
12         # Renvoie le sous-arbre gauche de l'instance d'ABR.
13
14     def est_vide(self):
15         # Renvoie True si l'instance d'ABR est vide et False
sinon.
16
17     def inserer(self, cle_a_inserer):
18         # Insère cle_a_inserer à sa place dans l'instance d'ABR.
```

Considérons ci-dessous trois arbres binaires de recherche :

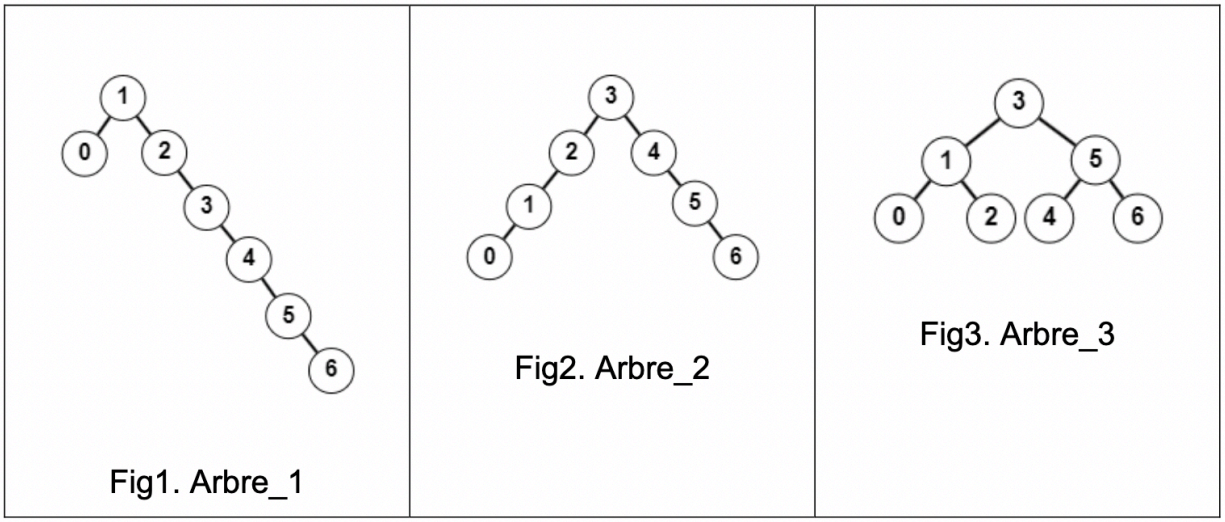


Fig1. Arbre_1

Fig2. Arbre_2

Fig3. Arbre_3

Dans tout l'exercice, nous ferons référence à ces trois arbres binaires de recherche et utiliserons la classe `ABR` et ses méthodes.

Partie A

1. Un arbre est une structure de données hiérarchique dont chaque élément est un nœud.

Recopier et compléter le texte ci-dessous en choisissant des expressions parmi **au maximum, au minimum, exactement, feuille, racine, sous-arbre gauche** et **sous-arbre droit** :

– Le nœud initial est appelé ....

– Un nœud qui n'a pas de fils est appelé ....

– Un arbre binaire est un arbre dans lequel chaque nœud a ... deux fils.

– Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est :

- supérieure à chaque clé de tous les nœuds de son ...
- inférieure à chaque clé de tous les nœuds de son ....

2. Donner dans l'ordre les clés obtenues lors du parcours préfixe de l'arbre no 1.

3. Donner dans l'ordre, les clés obtenues lors du parcours suffixe, également appelé postfixe, de l'arbre no 2.

4. Donner dans l'ordre, les clés obtenues lors du parcours infixe de l'arbre no 3.

5. Recopier et compléter les instructions ci-dessous afin de définir puis de construire, en y insérant les clés dans un ordre correct (il y a plusieurs possibilités, on en demande une), les trois instances de la classe `ABR` qui correspondent aux trois arbres binaires de recherche représentés plus haut.

```
arbre_no1 = ABR()
arbre_no2 = ...
arbre_no3 = ...
for cle_a_inserer in [..., ..., ..., ..., ..., ..., ...]:
    arbre_no1....
for cle_a_inserer in [..., ..., ..., ..., ..., ..., ...]:
    arbre_no2....
for cle_a_inserer in [..., ..., ..., ..., ..., ..., ...]:
    arbre_no3....
```

6. Voici le code de la méthode `hauteur` de la classe `ABR` qui renvoie la hauteur d'une instance d'`ABR` :

```
    def hauteur(self):
        if self.est_vide() :
            return -1
        else :
            return 1 + max(self.sag().hauteur(),
                           self.sad().hauteur())
```

Donner, en utilisant cette méthode, la hauteur des trois instances `arbre_no1`, `arbre_no2` et `arbre_no3` de la classe `ABR` définies plus haut et qui correspondent aux trois arbres représentés plus haut.

7. Recopier et compléter le code de la méthode `est_presente` ci-dessous qui renvoie `True` si la clé `cle_a_rechercher` est présente dans l'instance d'`ABR` et `False` sinon :

```
1     def est_present(self, cle_a_rechercher):
2         if self.est_vide() :
3             return ...
4         elif cle_a_rechercher == self.cle() :
5             return ...
6         elif cle_a_rechercher < self.cle() :
7             return ...
8         else :
9             return ...
```

8. Expliquer quelle instruction, parmi les trois ci-dessous, nécessitera le moins d'appels récursifs avant de renvoyer son résultat :

- `arbre_no1.est_present(7)`
- `arbre_no2.est_present(7)`
- `arbre_no3.est_present(7)`

Partie B

9. On rappelle que la fonction `abs(x)` renvoie la valeur absolue de x. Par exemple :

```
>>> abs (3)
3
>>> abs (-2)
2
```

On donne la méthode `est_partiellement_equilibre(self)` de la classe `ABR`. Cette méthode renvoie `True` si l'instance de la classe `ABR` est l'implémentation d'un arbre partiellement équilibré et `False` sinon :

```
1.    def est_partiellement_equilibre(self) :
2.        if self.est_vide() :
3.            return True
4.        return abs(self.sag().hauteur() -
self.sad().hauteur() ) <= 1)
```

Expliquer ce qu'on appelle ici un arbre partiellement équilibré.

Un arbre binaire est équilibré s'il est partiellement équilibré et si ses deux sous-arbres, droit et gauche, sont eux-mêmes équilibrés. Un arbre vide est considéré comme équilibré.

10. Justifier que, parmi les trois arbres définis plus haut, deux sont partiellement équilibrés.

11. Justifier que, parmi les trois arbres définis plus haut, un seul est équilibré.

12. Définir et coder la méthode récursive `est_equilibre` de la classe `ABR` qui renvoie `True` si l'instance de la classe `ABR` est l'implémentation d'un arbre équilibré et `False` sinon.