Processus

I. Commandes Unix

1. Commandes `ps` et `top`

Nous allons utiliser les deux commandes `ps` et `top` qui permettent de lister les processus dans le terminal.

1. Dans un terminal : *à taper*

```
nsi@ordi$ gedit &
nsi@ordi$ ps
```

2. résultat : *à compléter*

```
PID TTY          TIME CMD
.
.
.
```

3. Dans le terminal : *à taper*

```
nsi@ordi$ ps -u
```

4. Que représente chacune des colonnes ? *à compléter*
   - `USER` indique
   - `PID` donne l'identifiant numérique du processus.
   - `%CPU` et `%MEM` indiquent respectivement
   - `STAT` indique l'état du processus, `S` pour *sleeping*, le processus est en attente et `R` pour *running*, le processus est dans l'état prêt ou élu.
   - `COMMAND` indique la commande utilisée pour lancer le programme.
   - `START` et `TIME` indiquent respectivement
5. Fermer l'application `gedit` et utiliser à nouveau la commande `ps` dans le terminal. Que constatez-vous ?
6. Dans le terminal : *à taper*

```
nsi@ordi$ top
```

   Quelles sont les principales différences avec la commande `ps` ?
   -
   -

2. Commandes `kill` et `killall`

La commande `cat` n'est utilisée ici qu'à titre d'exemple, sa fonctionnalité n'est pas importante ici.

1. Dans un premier terminal : *à taper*

   ```
   nsi@ordi$ cat
   ```

2. Nous allons envoyer un signal de terminaison au processus `cat` par le biais de la commande `killall` qui envoie un signal aux processus dont le nom est indiqué.
   Dans un deuxième terminal : *à taper*

   ```
   nsi@ordi$ killall cat
   ```

3. Que constatez-vous ?

4. Dans le premier terminal taper à nouveau `cat`, puis, dans le deuxième terminal, à l'aide de la commande `ps -u`, déterminer le PID du processus ainsi créé : *à compléter*

   ```
   USER     PID %CPU    %MEM    VSZ RSS TTY STAT    START   TIME    COMMAND
                0.0     0.0                 S+              0:00    cat
   ```

   Remarque On peut également utiliser la commande `pgrep` en tapant dans le terminal la commande `pgrep cat`.

5. dans le deuxième terminal, *à taper*
   en utilisant le PID de `cat` que vous avez obtenu :

   ```
   nsi@ordi$ kill 17369
   ```

6. Que constatez-vous ?

3. En résumé

ps : liste les processus actifs attachés au terminal, les processus sont identifiés par leur PID.

top : fournit une vue dynamique temps réel du système en cours d'exécution.

kill : interrompt le processus dont le PID est donné en paramètre.

killall : interrompt **les** processus dont le nom est donné en paramètre.

Tester les différentes options `-u`, `-a`, `-e` et `-f` de la commande `ps`.
Déterminer leur fonctionnement à l'aide de la commande `man ps`.

- `ps -u` :
- `ps -a` :
- `ps -e` :
- `ps -f` :

II. Les processus

1. Définition d'un processus

Définition Un **processus** est une instance d'exécution d'un programme.

- Un processus est décrit par :
  - la mémoire allouée par le système pour l'exécution du programme ;
  - les ressources utilisées par le programme ;
  - les valeurs stockées dans les registres du processeur.
- Un processus possède un numéro unique, le PID (*Process ID*).

Les notions de programmes et de processus sont différentes : le même programme exécuté plusieurs fois générera plusieurs processus.

2. Un exemple

1. Dans un terminal, exécuter la commande suivante :

   ```
   nsi@ordi$ gnome-system-monitor
   ```

2. Dans un deuxième terminal, faire en sorte d'obtenir le PID du processus comme nous l'avons vu précédemment.
3. Dans le deuxième terminal toujours : *à taper*
   En utilisant le PID de `gnome-system-monitor` que vous avez obtenu :

   ```
   nsi@ordi$ top -p 5963
   ```

4. Que constatez-vous comme changements dans l'affichage de la fonction `top` lorsque vous utilisez le programme `gnome-system-monitor` ?

3. Les différents états d'un processus

Définition Les principaux états d'un processus sont les suivants :

Prêt : le processus peut être le prochain a s'exécuter. Il est dans la file des processus qui attendent leur tour.

Élu (actif ou exécution) : les processus est entrain de s'exécuter.

Bloqué : le processus est interrompu et en attente d'un événement externe (entrée/sortie, allocation mémoire, etc.)

On peut rajouter à ces trois états deux états éphémères **nouveaux** et **terminé** qui correspondent respectivement à un processus en cours de création et au système d'exploitation qui désalloue les ressources attribués a un processus qui vient de se finir.

4. Le cycle de vie d'un processus

Après sa création un processus est mis dans l'état prêt. En temps normal un processus variera de entre *prêt*, *élu* et *bloqué*.

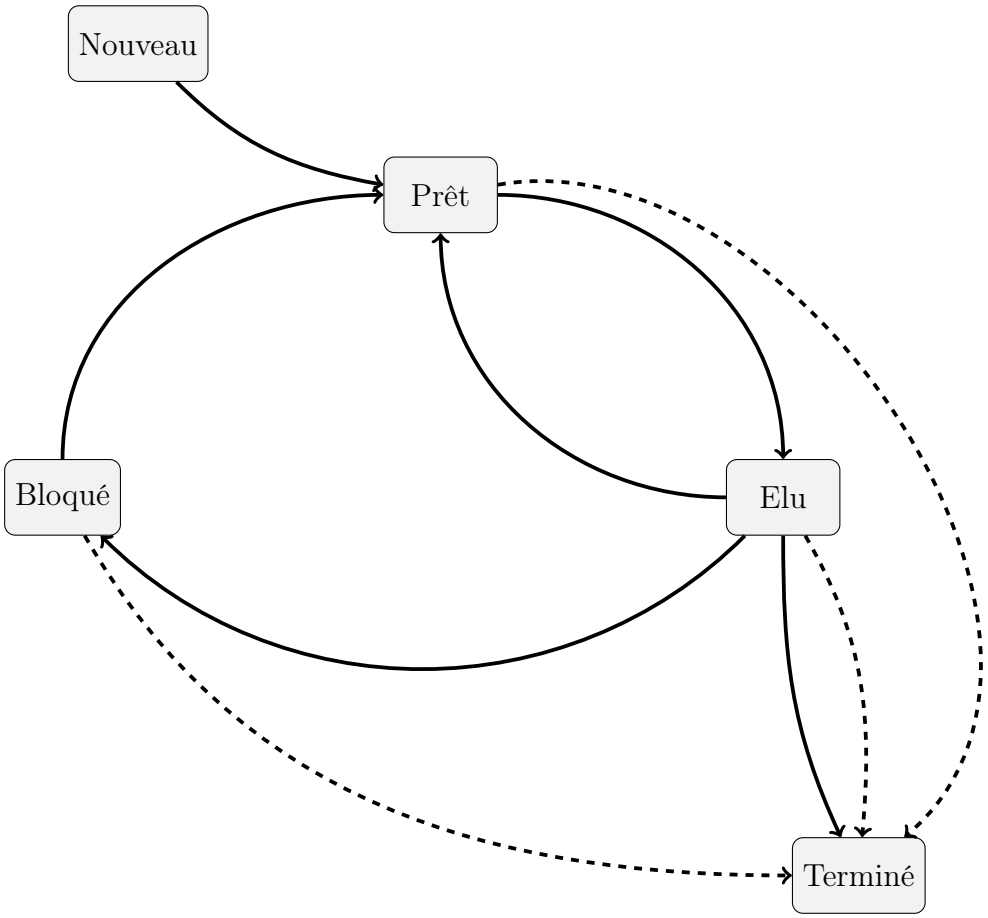
Plusieurs processus peuvent être dans l'état *prêt* mais un seul sera placé dans l'état *élu* ; c'est **l'ordonnanceur** (*scheduler*) du système d'exploitation qui est chargé de classer les processus dans une file. C'est lui qui décide quel processus est actif et a pour charge, de manière permanente et à une fréquence élevée, de désactiver le processus actif pour en activer un autre, faisant ainsi fonctionner alternativement les différents processus, donnant l'impression qu'ils fonctionnent en même temps.

Alors qu'il est élu, le processus peut avoir besoin d'attendre une ressource quelconque comme, par exemple, une ressource en mémoire. Il doit alors quitter momentanément le processeur, pour que ce dernier puisse être utilisé à d'autres tâches. Le processus passe donc dans l'état bloqué.

Une fois le processus terminé, il est placé dans l'état *terminé*, le système d'exploitation libère alors les ressources qui lui sont allouées. Notons que quel que soit l'état du processus, il peut se terminer de façon anormale (erreur dans un programme, problème matériel, interruption de l'utilisateur, etc.).

Le schéma ci-dessous résume le cycle de vie d'un processus. À chaque flèche du schéma (sauf celle partant de « Nouveau »), ajouter le numéro correspondant parmi ceux donnés ci-dessous (certains numéros peuvent apparaître plusieurs fois) :

1. *mise en exécution par l'ordonnanceur ;*
2. *interruption par l'ordonnanceur ;*
3. *terminaison normale ;*
4. *terminaison anormale ;*
5. *attente d'une ressource ;*
6. *obtention de la ressource.*



EXERCICE 3 (4 points)

Cet exercice traite du thème architecture matérielle, et plus particulièrement des processus et leur ordonnancement.

1. Avec la commande `ps -aef` on obtient l'affichage suivant :

```
PID  PPID  C STIME TTY          TIME CMD
8600     2  0 17:38 ?        00:00:00 [kworker/u2:0-fl]
8859     2  0 17:40 ?        00:00:00 [kworker/0:1-eve]
8866     2  0 17:40 ?        00:00:00 [kworker/0:10-ev]
8867     2  0 17:40 ?        00:00:00 [kworker/0:11-ev]
8887  6217  0 17:40 pts/0    00:00:00 bash
9562     2  0 17:45 ?        00:00:00 [kworker/u2:1-ev]
9594     2  0 17:45 ?        00:00:00 [kworker/0:0-eve]
9617  8887 21 17:46 pts/0    00:00:06 /usr/lib/firefox/firefox
9657  9617 17 17:46 pts/0    00:00:04 /usr/lib/firefox/firefox -contentproc -childID
9697  9617  4 17:46 pts/0    00:00:01 /usr/lib/firefox/firefox -contentproc -childID
9750  9617  3 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9794  9617 11 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9795  9794  0 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox
9802  7441  0 17:46 pts/2    00:00:00 ps -aef
```

On rappelle que : *PID* = Identifiant d'un processus (*Process Identification*)

PPID = Identifiant du processus parent d'un processus (*Parent Process Identification*)

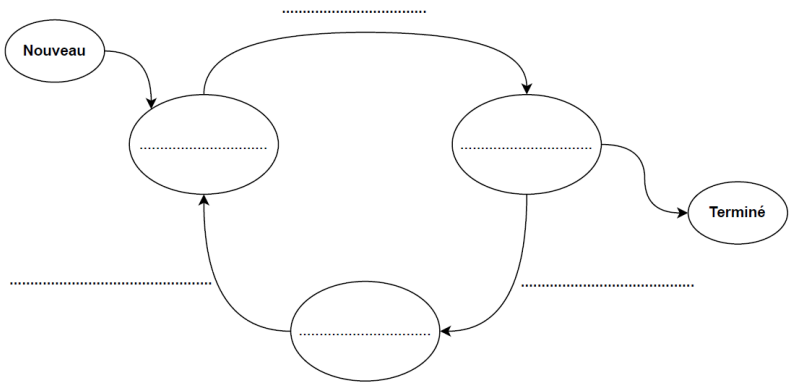
a. Donner sous forme d'un arbre de PID la hiérarchie des processus liés à *`firefox`*.

b. Indiquer la commande qui a lancé le premier processus de *`firefox`*.

c. La commande *`kill`* permet de supprimer un processus à l'aide de son *PID* (par exemple *`kill 8600`*). Indiquer la commande qui permettra de supprimer tous les processus liés à *`firefox`* et uniquement cela.

2.

a. Recopier et compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Élu, En attente, Prêt, Blocage, Déblocage, Mise en exécution*



On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en nombre de cycles du processeur.

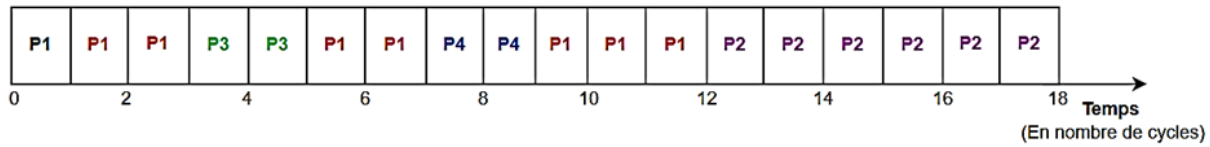
Processus	**P1**	**P2**	**P3**	**P4**
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une file d'attente en fonction de leur instant d'arrivée.

On se propose d'ordonnancer ces quatre processus avec la méthode suivante :

- Parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la durée **restante** est la plus courte ;
- Le processeur exécute un cycle de ce processus puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;
- En cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;
- Tout ceci jusqu'à épuisement des processus en liste d'attente.

On donne en exemple ci-dessous, l'ordonnancement des quatre processus de l'exemple précédent suivant l'algorithme ci-dessus.

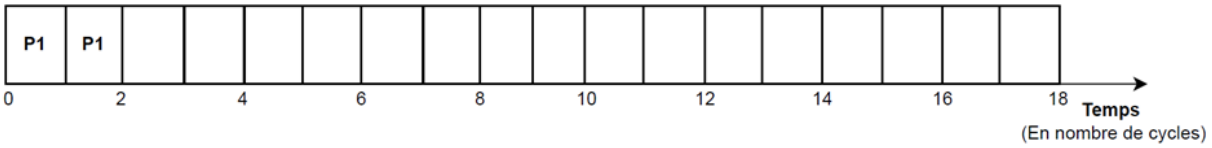


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

b. Calculer la moyenne des temps d'exécution des quatre processus.

On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté précédemment mais au lieu d'exécuter un seul cycle, le processeur exécutera à chaque fois deux cycles du processus choisi. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

c. Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



d. Calculer la nouvelle moyenne des temps d'exécution des quatre processus et indiquer si cet ordonnancement est plus performant que le précédent.

3. On se propose de programmer l'algorithme du premier ordonnanceur. Chaque processus sera représenté par une liste comportant autant d'éléments que de durées (en nombre de cycles). Pour simuler la date de création de chaque processus, on ajoutera en fin de liste de chaque processus autant de chaînes de caractères vides que la valeur de leur date de création.

```
1  p1 = ['1.8','1.7','1.6','1.5','1.4','1.3','1.2','1.1']
2  p2 = ['2.6','2.5','2.4','2.3','2.2','2.1','','']
3  p3 = ['3.2','3.1','','','']
4  p4 = ['4.2','4.1','','','','','','','']
5  liste_proc = [p1, p2, p3, p4]
```

La fonction `choix_processus` est chargée de sélectionner le processus dont le temps restant d'exécution est le plus court parmi les processus en liste d'attente.

a. Recopier sans les commentaires et compléter la fonction `choix_processus` ci-dessous. Le code peut contenir plusieurs lignes.

```
1  def choix_processus(liste_attente):
2      """Renvoie l'indice du processus le plus court parmi
3      ceux présents en liste d'attente liste_attente"""
4      if liste_attente != []:
5          mini = len(liste_attente[0])
6          indice = 0
7          ...
8          return indice
```

Une fonction `scrutation` (non étudiée) est chargée de parcourir la liste `liste_proc` de tous les processus et de renvoyer la liste d'attente des processus en fonction de leur arrivée. À chaque exécution de `scrutation`, les processus présents (sans chaînes de caractères vides en fin de liste) sont ajoutés à la liste d'attente. La fonction supprime pour les autres un élément de chaîne de caractères vides.

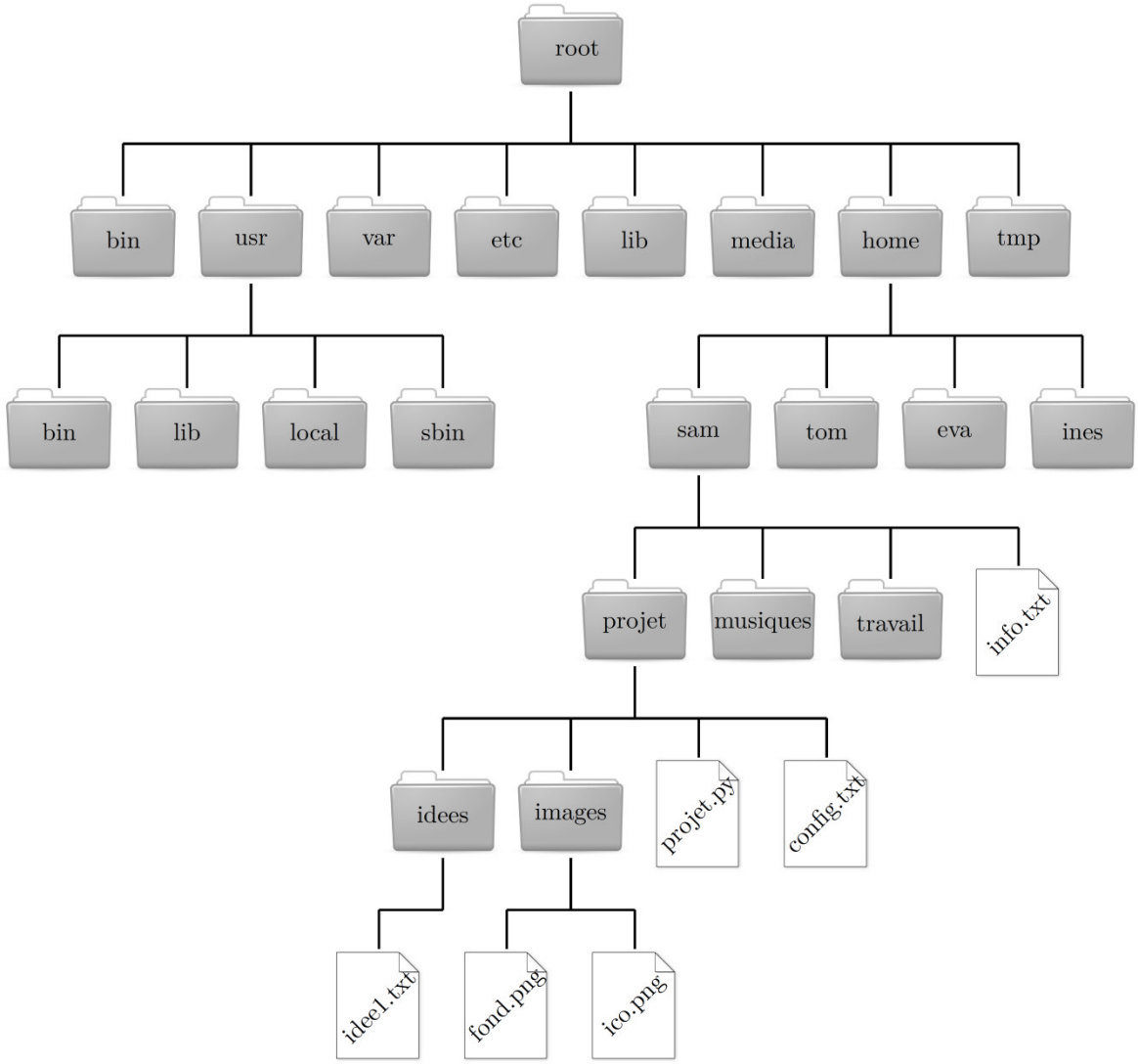
b. Recopier et compléter les différentes instructions de la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
1  def ordonancement(liste_proc):
2      """Exécute l'algorithme d'ordonnancement
3      liste_proc -- liste des processus
4      Renvoie la liste d'exécution des processus"""
5      execution = []
6      attente = scrutation(liste_proc, [])
7      while attente != []:
8          indice = choix_processus(attente)
9          ... # A FAIRE (plusieurs lignes de code) ...
10         attente = scrutation(liste_proc, attente)
11     return execution
```

Exercice 3

Thème abordé : système d'exploitation

Nous avons l'arborescence ci-dessous sous un environnement Linux.



Samuel a pour nom d'utilisateur `sam`. Il a ouvert un terminal et le répertoire courant est le répertoire `musiques`. Pour tout l'exercice, on pourra tirer parti de l'annexe 2 répertoriant différentes commandes du système d'exploitation.

1. Ecrivez la ou les commande(s) qui permet(tent) de se déplacer du répertoire actuel `musiques` au répertoire `projet` :
   a. en utilisant un chemin relatif.
   b. en utilisant un chemin absolu.

2. Le répertoire courant est à présent le répertoire `sam`
   a. Ecrire la commande qui permet de lister le contenu du répertoire `projet`.
   b. Le fichier `config.txt` est protégé en écriture pour tous les utilisateurs. On souhaite modifier ce droit afin que l'utilisateur `sam` et lui seul puisse

modifier le contenu du fichier. Ecrire la commande permettant d'effectuer ce changement.

3. Le répertoire courant est toujours `sam`. L'utilisateur souhaite supprimer le répertoire `projet` en tapant l'instruction :

```
rm projet
```

Il constate que cette instruction ne fonctionne pas car ce répertoire n'est pas vide. *Finalement, il tape l'instruction :*

```
rm –R projet
```

où « R » signifie « récursif ». Le répertoire est finalement supprimé.

a. Pourquoi cette instruction fonctionne-t-elle, contrairement à la précédente ?

Les fichiers et dossiers ont été effacés dans cet ordre :

- fichier `idee1.txt`
- dossier `idees`
- fichier `fond.png`
- fichier `ico.png`
- dossier `images`
- fichier `projet.py`
- fichier `config.txt`
- dossier `projet`

b. Quel type de parcours a été réalisé par le système d'exploitation ?

4. On considère la fonction récursive suivante en langage Python :

```
def nb_fichiers(list_fich, i) :
  if i == len(list_fich) :
    return 0
  elif list_fich [i][0] == 'b' :
    return 1 + nb_fichiers(list_fich, i+1)
  else :
    return nb_fichiers(list_fich, i+1)
```

où `list_fich` est une liste contenant des noms de fichiers.

Indiquer ce que renvoie l'appel suivant en expliquant les étapes :

```
nb_fichiers(['nsi.bmp','banana.mp3','job.txt','BoyerMoore.py'],0)
```

EXERCICE 1 (4 points)

Thèmes abordés : systèmes d'exploitation linux

L'entreprise capNSI gère les contrats de ses clients en créant pour chacun d'eux un sous-dossier dans le dossier `Contrats` sur leur ordinateur central. Le système d'exploitation de cet ordinateur est une distribution linux. Quelques commandes de bases pour ce système d'exploitation sont rappelées en annexe 1 en fin de sujet.

Dans la console représentée sur la figure ci-dessous, on peut visualiser les répertoires (ou dossiers) à la racine de l'ordinateur central avec l'instruction **`ls :`**

```
gestion@capNSI-ordinateur_central:~$ ls
Bureau      Documents      Modèles      Public
Téléchargements  Contrats     Images     Musique
Vidéos
```

1.

a. Donner le nom de l'utilisateur et le nom de l'ordinateur correspondant à la capture d'écran précédente.

b. Ecrire les instructions permettant d'afficher la liste des dossiers clients du répertoire `Contrats` en partant de la situation ci-dessous :

```
gestion@capNSI-ordinateur_central:~$
```

Après une campagne de démarchage, l'entreprise a gagné un nouveau client, Monsieur Alan Turing. Elle souhaite lui créer un sous-dossier nommé **`TURING_Alan`** dans le dossier **`Contrats`**. De plus, elle souhaite attribuer tous les droits à l'utilisateur et au groupe et seulement la permission en lecture pour tous les autres utilisateurs. La commande **`chmod`** permet de le faire.

2.

a. Ecrire les instructions permettant de créer le sous-dossier **`TURING_Alan`** à partir du répertoire racine.

b. Ecrire l'instruction permettant d'attribuer les bons droits au sous-dossier **`TURING_Alan`**.

En Python, le module `os` permet d'interagir avec le système d'exploitation. Il permet de gérer l'arborescence des fichiers, des dossiers, de fournir des informations sur le système d'exploitation. Par exemple, le code de la page suivante, exécuté dans la console, permet de créer le sous-dossier **`TURING_Alan`** précédent :

```
>>> import os
>>> os.mkdir("Contrats/TURING_Alan")
>>> os.chmod("Contrats/TURING_Alan", 774)
```

L’entreprise dispose d’un tableau de nouveaux clients :

```
tab_clients = [
  ('LOVELACE', 'Ada'),
  ('BOOLE', 'George'),
  ('VONNEUMANN', 'John'),
  ('SHANNON', 'Claude'),
  ('KNUTH', 'Donald')
]
```

Elle souhaite automatiser le formatage des tableaux des nouveaux clients. Elle souhaite également automatiser la création et l’attribution des droits des dossiers portant les noms des nouveaux clients.

3. Ecrire une fonction **`formatage(tab)`** qui prend en paramètre un tableau de tuplets (**`Nom, Prenom`**) des nouveaux clients et renvoie un tableau de chaines de caractères. Par exemple, **`formatage(tab_clients)`** renvoie `['LOVELACE_Ada', 'BOOLE_George', 'VONNEUMANN_John', 'SHANNON_Claude', 'KNUTH_Donald']`

4. Ecrire une fonction **`creation_dossiers(tab)`** qui prend en paramètre un tableau de chaînes de caractères et qui crée et modifie les droits des dossiers au nom de ces chaines de caractères avec les mêmes droits que le sous-dossier **`TURING_Alan`**.