Le but de cette séquence d'explique la gestion des processus dans un système d'exploitation.

1. **Introduction :**

Dans les années 1970, les ordinateurs personnels n'étaient pas capables d'exécuter plusieurs tâches à la fois : on lançait un programme et on attendait que celui-ci se termine.

Les systèmes d'exploitation récents (Windows, Linux ou osX par exemple) permettent d'exécuter plusieurs tâches simultanément (en tous cas, donner l'impression que celles-ci s'exécutent en même temps).

A un instant donné, il n'y a donc pas un mais plusieurs programmes qui sont en cours d'exécution sur un ordinateur : on les nomme **processus**.

Une des tâches du système d'exploitation est **d'allouer à chacun des processus les ressources** dont il **a besoin en termes de mémoire**, entrées-sorties ou temps processeur, et de s'assurer que les processus ne se gênent pas les uns les autres.

Seulement, nous avons tous déjà vécu la situation suivante :

Lorsqu'on clique sur l'icône d'un programme, cela provoque la naissance d'un ou plusieurs processus liés au programme lancé et lorsque le programme ne répond plus, il nous arrive de recliquer dessus (le processus sera lancé plusieurs fois), il nous arrive aussi de lancer le gestionnaire de taches pour tuer le processus en défaut.

1. **Qu'est-ce qu'un processus ?**

Il faut distinguer un programme de son exécution : le lancement d’un programme entraîne des lecture/écriture de registres et d’une partie de la mémoire.

D’ailleurs, un même programme peut être exécuté plusieurs fois sur une même machine au même moment en occupant des espaces mémoire différents.

**Un processus** représente **une instance d’exécution d’un programme dans une machine donnée**.

Il est caractérisé par :

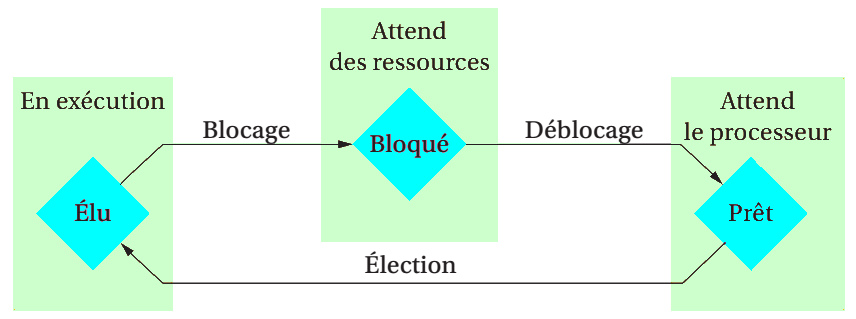
* un ensemble d'instructions à exécuter - souvent stockées dans un fichier sur lequel on clique pour lancer un programme (par exemple chrome.exe)
* un espace mémoire dédié à ce processus pour lui permettre de travailler sur des données qui lui sont propres : si vous lancez deux instances de chrome, chacune travaillera indépendamment de l'autre avec ses propres données.
* des ressources matérielles : processeur, entrées-sorties (accès à internet en utilisant la connexion Wifi).
* des ressources mémoires

Il ne faut donc pas confondre le fichier contenant un programme (portent souvent l'extension .exe sous windows) et le ou les processus qu'il engendre quand il est exécuté : Un programme est juste un fichier contenant une suite d'instructions, alors que les processus sont des instances de ce programme ainsi que les ressources nécessaires à leur exécution (plusieurs fenêtres de chrome ouvertes en même temps).

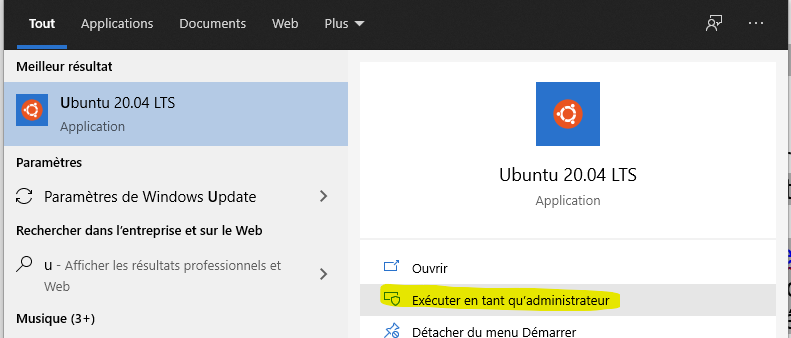
1. **Les états d'un processus :**

Au cours de son existence dans la machine, un même processus peut se retrouver dans différents états :

* Un processus est créé et se trouve alors dans **l’état prêt** : il attend de pouvoir avoir accès au processeur.
* Le processus obtient l’accès au processeur. Il passe alors dans **l’état élu**.
* 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 celui-ci puisse être utilisé à d’autres tâches (le processeur ne doit pas attendre!). Le processus passe donc dans **l’état bloqué**.
* Le processus a obtenu la ressource attendue mais s’est fait prendre sa place dans le processeur par un autre processus. Il se met donc en attente et repasse à **l’état prêt**.
* Un processus ne pourra terminer que s’il est déjà dans l’état élu sauf anomalie.



1. **Gérer les processus sur un système Linux**
2. Ouvrir Ubuntu 20.04LTS en tant qu'administrateur (installé en première).



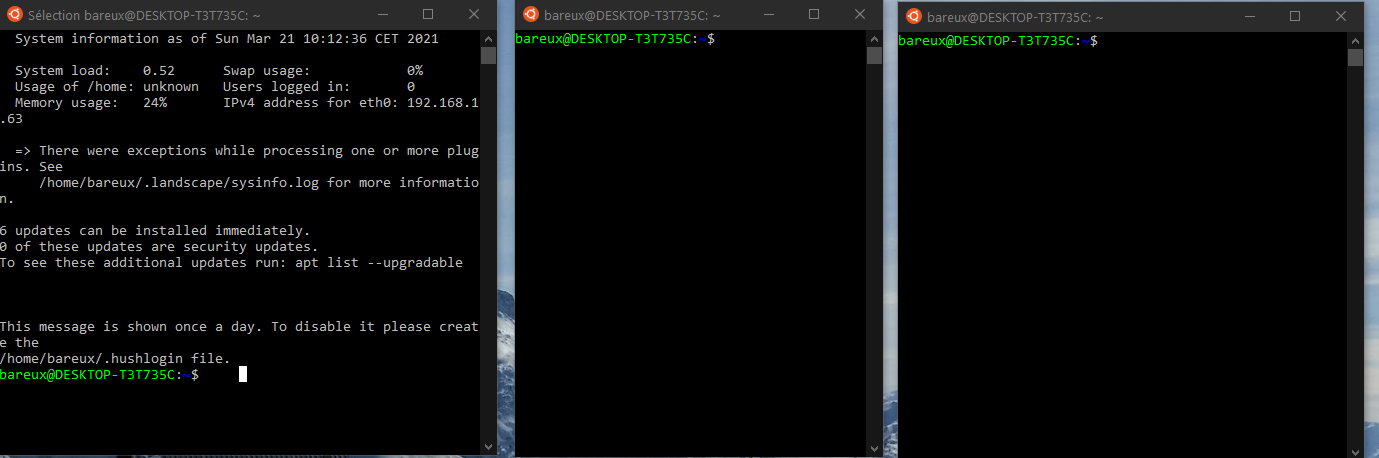
Si le terminal s'ouvre et qu'il affiche "Appuyer sur une touche pour continuer" cela signifie qu'il faut réactiver les fonctionnalités.

Pour cela, exécuter cmd en tant qu'administrateur et taper la ligne de code suivante :

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart



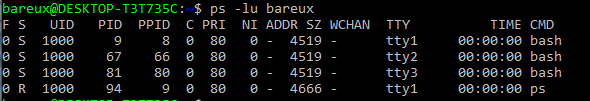
puis redémarrer l'ordinateur et relancer ubuntu. Relancer le 3 fois pour ouvrir 3 terminaux en même temps.



* 1. **La commande ps:**

La commande ps est utilisée pour afficher des informations sur un processus. L’option u permet de préciser le propriétaire, l’option l permet un affichage avec plus d’attributs (long).

1. Par exemple, pour connaître les processus appartenant à l’utilisateur moi: ps -lu bareux



* La colonne S indique l’état (state) du processus : S pour stopped, R pour running et Z pour zombie.
* PID est le processus identifier : un identifiant sous forme d’entier donné par le système.
* PPID est le parent processus identifier qui donne l’identifiant du parent qui a engendré le processus.
* CMD est le nom de la commande.
* TTY (tty) : terminal associé au processus.

Faire sur votre session en modifiant l'instruction en conséquence et capturer le résultat

1. Par exemple, pour visualiser les processus sous forme d'arbre : pstree, capturer votre arbre
2. A l'aide de ps -aef, quel est le PID du processus init de la racine ?
3. Quel est le PPID de ce init ?
4. Ce init possède-t-l un frère ?
5. Citer quelques descendants directs de init
   1. **La commande top :**

Une commande indispensable à connaître sous Linux pour inspecter les processus est la commande **top**.

L'affichage se rafraîchit en temps réel contrairement à ps qui fait un instantané.

Voici quelques options qui s'activent par des raccourcis clavier.

En voici quelques-uns :

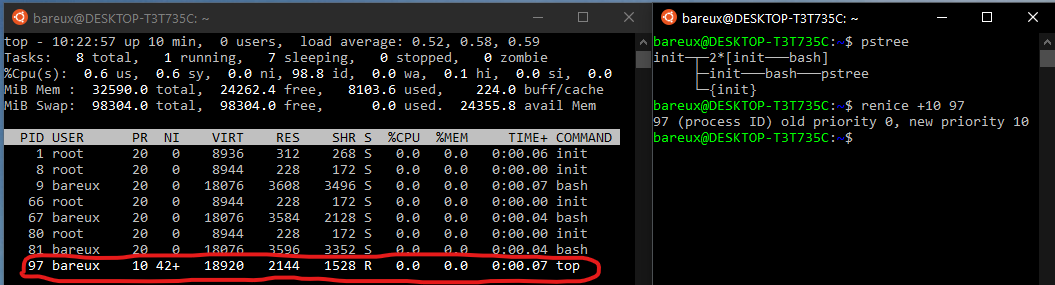
* h : affiche l'aide
* M : trie la liste par ordre décroissant d'occupation mémoire. Pratique pour repérer les processus trop gourmands
* P : trie la liste par ordre décroissant d'occupation processeur
* i : filtre les processus inactifs. Cela ne montre que ceux qui travaillent réellement.
* k : permet de tuer un processus - à condition d'en être le propriétaire. Essayez de tuer init ...
* V : permet d'avoir la vue arborescente sur les processus.
* q : permet de quitter top

1. Capturer le résultat de la commande top que vous laisserez actif dans ce terminal jusqu’à la fin du TP.

Le lien entre PR et NI est simple : PR = NI + 20 ce qui fait qu'une priorité PR de 0 équivaut à un niveau de priorité maximal.

1. Pour augmenter la priorité du processus top dont le PID est 97, il suffit de taper : renice +10 PID

Augmenter la priorité de 10 de processus top et faites une capture.



* 1. **La commande kill :**

Pour tuer un processus, on lui envoie un signal de terminaison.

1. Pour tuer top, vous lui enverrez la commande kill PID où PID désigne le numéro du processus à quitter, dans notre exemple le PID vaut 97 dans le deuxième terminal.
2. Relancer top dans un terminal.

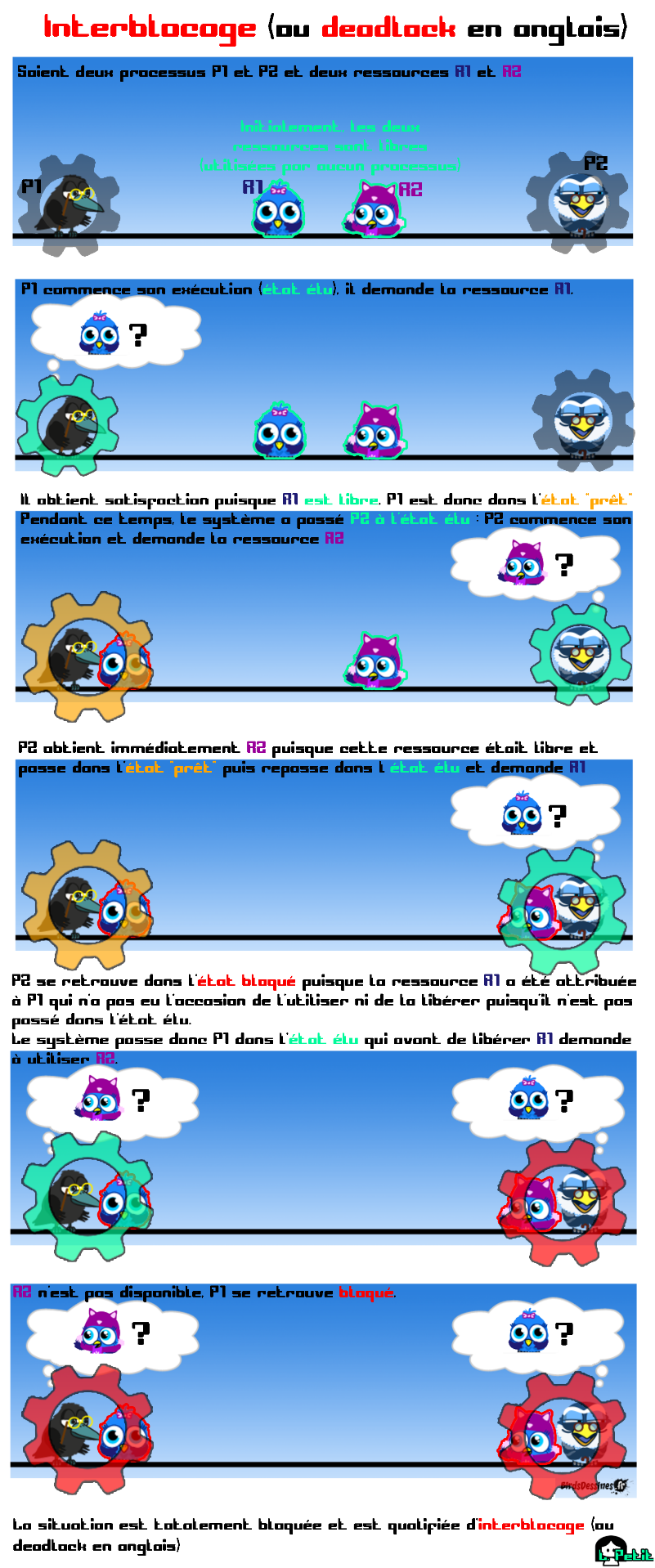
Dans l'autre, exécuter python3 à l'aide de la commande python3. (Si cela ne fonctionne pas, il faut l'installer avec l'instruction sudo apt-get pyton3 puis faire un upadte et upgrade).

Dans la console de python 3, exécuter les lignes de code suivantes :

a=2

while a<3:

print("toto")

1. Quel est le processus le plus gourmand ?
2. Quelle est la commande pour tuer ce processus ? Tester dans le 3ième terminal et capturer le résultat.
3. Essayer de tuer le processus init et capturer le résultat.
4. **Interblocage**

Plusieurs processus peuvent être dans l’état prêt, comment choisir celui qui sera élu ? L’ordonnanceur (scheduler) classe les processus prêts dans une file et le répartiteur (dispatcher) alloue quant à lui un processeur à l’élu dans le cas d’une architecture multiprocesseur.

Il existe plusieurs politiques d’ordonnancement dont le choix va dépendre des objectifs du système.

Exemples :

* Premier arrivé, premier servi : simple, mais peu adapté à la plupart des situations.
* Plus court d’abord : très efficace, mais il est la plupart du temps impossible de connaître à l’avance le temps d’exécution d’un processus.
* Priorité : le système alloue un niveau de priorité aux processus. Cependant des processus de faible priorité peuvent ne jamais être élus.
* Tourniquet : un quantum de temps est alloué à chaque processus. Si le processus n’est pas terminé au bout de ce temps, il est mis en bout de file en état prêt.
* Un système hybride entre tourniquet et priorité qu’on retrouve dans les systèmes Unix.

Qu'est-ce que l'interblocage ?

Soit 2 processus P1 et P2, soit 2 ressources R1 et R2.

Initialement, les 2 ressources sont "libres" (utilisées par aucun processus).

Le processus P1 commence son exécution (état élu), il demande la ressource R1. Il obtient satisfaction puisque R1 est libre, P1 est donc dans l'état "prêt".

Pendant ce temps, le système a passé P2 à l'état élu : P2 commence son exécution et demande la ressource R2. Il obtient immédiatement R2 puisque cette ressource était libre. P2 repasse immédiatement à l'état élu et poursuit son exécution (P1 lui est toujours dans l'état prêt).

P2 demande la ressource R1, il se retrouve dans un état bloqué puisque la ressource R1 a été attribuée à P1 : P1 est dans l'état prêt, il n'a pas eu l'occasion de libérer la ressource R1 puisqu'il n'a pas eu l'occasion d'utiliser R1 (pour utiliser R1, P1 doit être dans l'état élu).

P2 étant bloqué (en attente de R1), le système passe P1 dans l'état élu et avant de libérer R1, il demande à utiliser R2.

Problème : R2 n'a pas encore été libéré par P2, R2 n'est donc pas disponible, P1 se retrouve bloqué.

Résumons la situation à cet instant : P1 possède la ressource R1 et se trouve dans l'état bloqué (attente de R2), P2 possède la ressource R2 et se trouve dans l'état bloqué (attente de R1)

Pour que P1 puisse poursuivre son exécution, il faut que P2 libère la ressource R2, mais P2 ne peut pas poursuivre son exécution (et donc libérer R2) puisqu'il est bloqué dans l'attente de R1. Pour que P2 puisse poursuivre son exécution, il faut que P1 libère la ressource R1, mais P1 ne peut pas poursuivre son exécution (et donc libérer R1) puisqu'il est bloqué dans l'attente de R2. Bref, la situation est totalement bloquée !

Cette situation est qualifiée d'interblocage (deadlock en anglais).

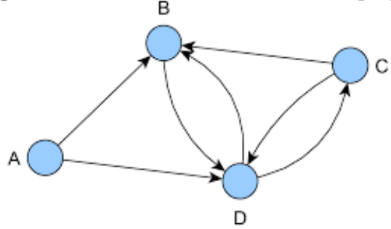
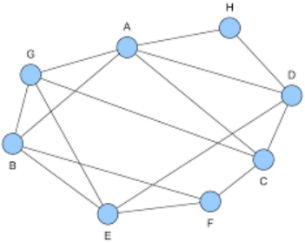
1. **Exercices :**

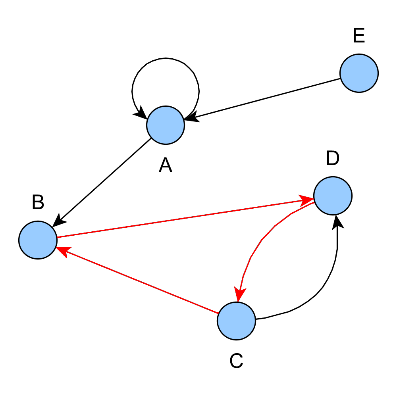
Dans un graphe, on distingue les objets appelés **sommets** et les relations entre sommets.

Il existe deux types de graphes :

Les graphes orientés Les graphes non orientés

où les relations sont appelées **arcs**. où les relations sont appelées **arêtes**.





On note **A→B** l’arc (A,B) d’un graphe orienté où A est le sommet de départ et B celui d’arrivée.

On appelle **chemin**, toute suite de sommets consécutifs reliés par des arcs, il est dit **élémentaire** si elle ne comporte pas plusieurs fois le même sommet.

On appelle **circuit**, un chemin dont le sommet de début et le sommet de fin sont identiques.

1. Sept processus Pi sont dans la situation suivante par rapport aux ressources Ri:

P1 a obtenu R1 et demande R2;

P2 demande R3 et n’a obtenu aucune ressource

P3 demande R2 et n’a obtenu aucune ressource

P4 a obtenu R2 et R4 et demande R3

P5 a obtenu R3 et demande R5

P6 a obtenu R6 et demande R2

P7 a obtenu R5 et demande R2.

On voudrait savoir s’il y a interblocage.

Construire un graphe orienté où les sommets sont les processus et les ressources, tels que :

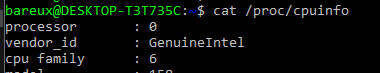
la présence de l’arc Ri → Pj signifie que le processus Pj a obtenu la ressource Ri;

la présence de l’arc Pj → Ri signifie que le processus Pj demande la ressource Ri.

Il y a interblocage lorsque des circuits sont présents dans le graphe. Chercher ces circuits afin de déterminer s’il y a bien interblocage ou non.

1. En utilisant la commande cat /proc/cpuinfo noter le nombre de processeurs disponibles sur votre machine

exemple : ici on a 16 processus car le premier a le numéro 0 et le dernier le 15.





Créer un programme python nommé infini.py contenant une boucle infinie à l'aide de la commande nano:

Exemple :

a=3

while a<5:

a=2

Créer un second programme test.py contenant

def bidon():

a = 0

for i in range(100000):

a += a\*\*3

Lancer un interpréteur python3 et noter son numéro de processus

Dans l'interpréteur python, tapez les commandes

>>> from timeit import timeit

>>> import test

>>> timeit(test.bidon, number = 100)

Cette commande va lancer 100 fois la fonction bidon et renvoyer le temps d'exécution moyen.

Taper la commande python3 infini.py & autant de fois qu'il y a de processeurs sur la machine le symbole & indique au shell de lancer le programme en arrière-plan. Nous allons donc monopoliser l'ensemble des ressources processeurs de la machine avec des boucles infinies. Le travail de l'ordonnanceur sera donc bien visible car les ressources processeur vont se raréfier.

Relancer timeit(test.bidon, number = 100) dans le shell python. Vous devriez noter un ralentissement par rapport à la première exécution. En effet, le processeur a moins de temps à consacrer à l'exécution de la fonction bidon.

Changer la priorité de l'interpréteur python en mettant un renice à +20.

Relancer timeit(test.bidon, number = 100) dans le shell python. Que constatez-vous ?

Tuer tous les processus des programmes infini.py en cours d'exécution.