

# Programmes Processus Threads Ressources

## Résumé<sup>1</sup>

Un programme,..... Nous en avons tous entendu parler, mais que ce passe-t-il quand nous le 'lançons' sur une machine ?

Nous entrons dans le domaine des processus, des threads, des interblocages, des processus concurrents ou parallèles ...

Bienvenue sous le capot de nos ordinateurs ...

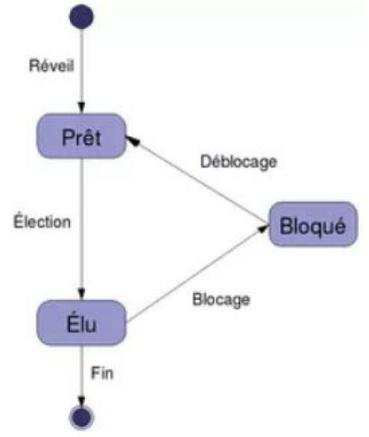


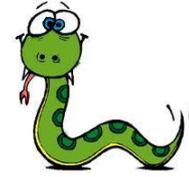
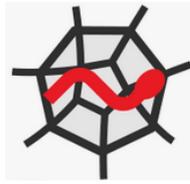
Diagramme d'état d'un processus

## Sommaire

<b>1</b>	<b>Programme, Processus, Threads</b> .....	<b>2</b>
1.1	<i>Programme Processus et Threads de quoi parle-t-on ?</i> .....	2
1.2	<i>Ressources et cycle de vie d'un processus</i> .....	3
1.3	<i>Interblocage des processus</i> .....	3
<b>2</b>	<b>L'Operating System ou OS</b> .....	<b>4</b>
2.1	<i>Brève description</i> .....	4
2.2	<i>Accès concurrent versus parallélisme</i> .....	5
2.3	<i>Le daemon</i> .....	5
2.4	<i>A la découverte du processeur</i> .....	5
<b>3</b>	<b>Les processus sous Linux</b> .....	<b>6</b>
<b>4</b>	<b>Une première mise en œuvre avec Python</b> .....	<b>7</b>
4.1	<i>La bibliothèque threading</i> .....	7
4.2	<i>Exemple de mise en œuvre</i> .....	7
<b>5</b>	<b>Sujet de Bac</b> .....	<b>9</b>
<b>6</b>	<b>Bibliographie</b> .....	<b>11</b>



<sup>1</sup> Source Wikipédia



# 1 Programme, Processus, Threads

---

## 1.1 Programme Processus et Threads de quoi parle-t-on ?

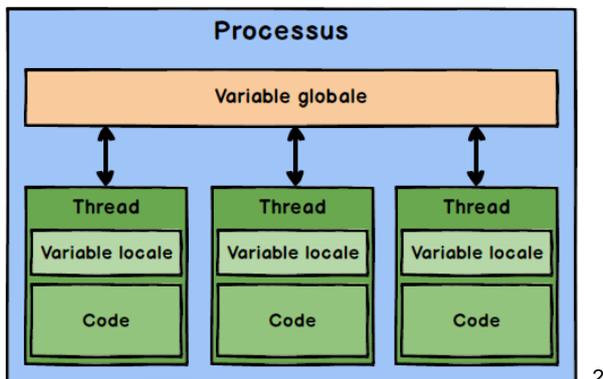
### Programme

Un programme est un fichier contenant une suite d'instructions qui lorsqu'elles sont exécutées modifient l'état du processeur et de la mémoire afin de réaliser une tâche donnée. Lancer l'exécution d'un programme revient à exécuter une instance de celui-ci. Plusieurs instances sont possibles si on lance plusieurs fois le même programme.

### Processus

Un programme en exécution est un processus. L'entité qui exécute la séquence d'instruction d'un programme est un thread. Il est possible, dans un processus, d'utiliser plusieurs threads. Chacun pouvant être exécuté simultanément par plusieurs cœurs.

Le processus est une instance en exécution d'un programme depuis son lancement jusqu'à sa fin. C'est le système d'exploitation qui gère les processus.



2

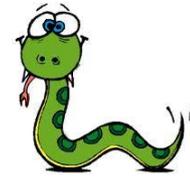
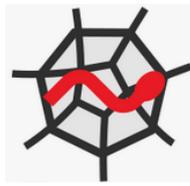
### Thread

Un thread est défini en informatique comme la plus petite unité pouvant être planifiée dans un système d'exploitation. Les threads sont normalement créés par un script ou d'un programme informatique (qui sont implémentées sur un seul processeur par le multitâche).



---

<sup>2</sup> <https://waytolearnx.com/2020/06/les-threads-en-python.html>  
<https://koor.fr/Python/CodeSamples/ThreadSample.wp>



## 1.2 Ressources et cycle de vie d'un processus

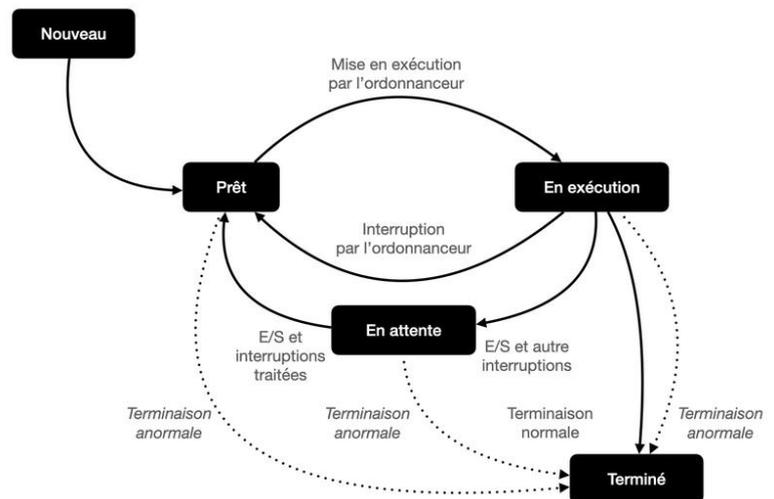
Un processus n'a pas seulement besoin d'accéder au processeur, mais il a souvent besoin d'accéder à des *ressources* autres comme:

- La mémoire vive: RAM,
- La mémoire de masse: disques durs, clés USB, mémoire flash...
- La lecture ou l'écriture d'un fichier...
- Les périphériques d'entrée et de sortie: clavier, souris, écran, imprimante...

Ces *ressources* externes étant beaucoup moins rapides que le processeur, elles **bloquent** les processus lors de leur exécution.

Ainsi, lors de la vie d'un processus, celui-ci peut passer par trois états:

- **PRÊT**: le processus est lancé et attend l'accès au processeur.
- **ÉLU**: le processus a obtenu l'accès au processeur: il est **en exécution**.
- **BLOQUÉ**: le processus est en cours d'exécution, mais attend une.



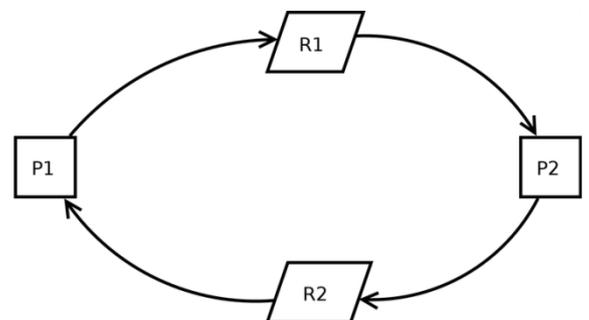
Tous les processus actifs à un instant donné dans un ordinateur sont tous concurrents pour l'accès au temps CPU et aux ressources de la machine.

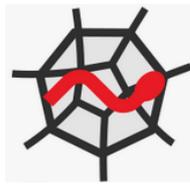
C'est le système d'exploitation ou OS qui réalise l'arbitrage.

## 1.3 Interblocage des processus

Un ensemble de processus est en interblocage si chaque processus attend la libération d'une ressource qui est allouée à un autre processus de l'ensemble.

Comme tous les processus sont en attente, aucun ne pourra s'exécuter et donc libérer les ressources demandées par les autres. Ils attendront tous indéfiniment.



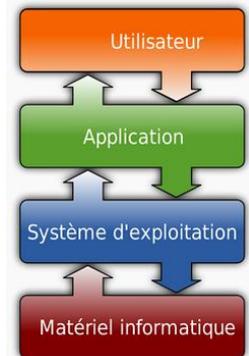


## 2 L'Operating System ou OS

### 2.1 Brève description

Le système d'exploitation a les responsabilités suivantes:

- Gérer le lancement des diverses applications et donner l'illusion que l'ordinateur est *multitâche*;
- identifier les utilisateurs;
- gérer l'organisation du disque dur et de ses fichiers;
- contrôler l'accès aux données du disque dur et ressources de l'ordinateur.



L'operating système organise les cycles de vie des processus. Il est lui-même un processus de plus bas niveau que nous appellerons noyau. Ce noyau gère les différents processus en les organisant dans une file d'attente.

**Il faut bien comprendre à ce stade que l'apparition des processus est totalement asynchrone. Les processus apparaissent à des instants non définis à l'avance.**

Il répartit les ressources, temps CPU, accès mémoire selon plusieurs stratégies d'ordonnancement en voilà quelques exemples :

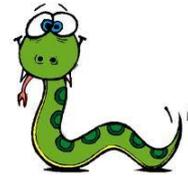
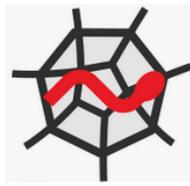
- Round-robin (ou méthode du tourniquet) : les processus se voient alloué une petite unité de temps, appelé quantum de temps. La file d'attente est gérée comme une file circulaire. L'ordonnanceur parcourt cette file et alloue un temps processeur à chacun des processus pour un intervalle de temps de l'ordre d'un quantum au maximum.
- FIFO : le premier processus arrivé dans la file est le premier servi.
- Shortest job first (SJF, ou SJN -*Shortest Job Next*-) : le processus le plus rapide à exécuter est servi en premier.
- LIFO : le dernier processus arrivé est servi en premier.

Q1. Comparer Round Robin et SJF, quel est le danger dans l'utilisation de SJF ? Et quel peut-être son avantage par rapport à Round Robin mais à quelle condition ?

Q2. Rappeler la signification de l'acronyme FIFO.

Q3. Rappeler la signification de l'acronyme LIFO.





## 2.2 Accès concurrent versus parallélisme

### Concurrence

Le système d'exploitation peut donner l'apparence que tous les programmes sont simultanément exécutés avec un seul processeur. Ce n'est qu'une apparence ils se partagent le temps CPU.

### Parallélisme

Si on dispose de plusieurs cœurs un vrai parallélisme est alors possible. Chacun des processus utilisant de manière indépendante un seul cœur.

## 2.3 Le daemon<sup>3</sup>

Un *daemon* mot anglais, souvent traduit erronément par *démon*, est un type de programme informatique, un processus ou un ensemble de processus qui s'exécute en arrière-plan plutôt que sous le contrôle direct d'un utilisateur.

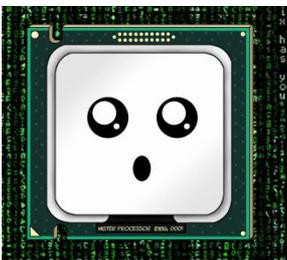
Le terme *daemon* semble être introduit en 1963 par les concepteurs de CTSS du MIT, en réponse au « dragon », terme employé par les concepteurs d'ITS. Le rétro-acronyme *Disk And Execution MONitor* (« moniteur de disque et d'exécution ») a été inventé pour justifier le terme *daemon* après qu'il fut devenu populaire.

Les *daemons* sont souvent démarrés lors du chargement du système d'exploitation (UNIX, LINUX) et servent en général à répondre à des requêtes du réseau, à l'activité du matériel ou à d'autres programmes en exécutant certaines tâches.

Sous Microsoft Windows, ces fonctions sont exécutées par des programmes appelés *services*.

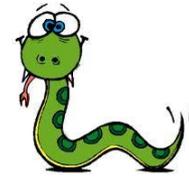
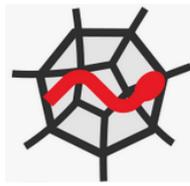
## 2.4 A la découverte du processeur

 VOYAGE\_DANS\_L\_INFINIMENT\_NUMERIQUE\_-\_LE\_PROCESSEUR\_1.mp4



---

<sup>3</sup> [https://fr.wikipedia.org/wiki/Daemon\\_\(informatique\)](https://fr.wikipedia.org/wiki/Daemon_(informatique))



### 3 Les processus sous Linux

La commande ps -ef sous linux permet de visualiser la liste des processus avec la signification suivante :

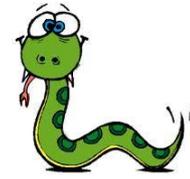
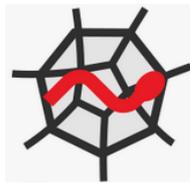
La documentation Linux donne la signification des différents champs :

- UID : identifiant utilisateur effectif ;
- PID : identifiant de processus ;
- PPID : PID du processus parent ;
- C : partie entière du pourcentage d'utilisation du processeur par rapport au temps de vie des processus ;
- STIME : l'heure de lancement du processus ;
- TTY : terminal de contrôle
- TIME : temps d'exécution
- CMD : nom de la commande du processus

UID	PID	PPID	C	STIME	TTY	TIME	CMD
...							
pi	6211	831	8	09:07	?	00:01:16	/usr/lib/chromium-browser/chromium-browser-v7 --disable-quit --enable-tcp-fast-open --p
pi	6252	6211	0	09:07	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi	6254	6252	0	09:07	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi	6294	6211	4	09:07	?	00:00:40	/usr/lib/chromium-browser/chromium-browser-v7 --type=gpu-process --field-trial-handle=1
pi	6300	6211	1	09:07	?	00:00:16	/usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi	6467	6254	1	09:07	?	00:00:11	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	11267	6254	2	09:12	?	00:00:15	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	12035	836	0	09:13	?	00:00:00	/usr/lib/libreoffice/program/oosplash --writer file:///home/pi/Desktop/mon_fichier.odt
pi	12073	12035	2	09:13	?	00:00:15	/usr/lib/libreoffice/program/soffice.bin --writer file:///home/pi/Desktop/mon_fichier.c
pi	12253	831	1	09:13	?	00:00:07	/usr/bin/python3 /usr/bin/sense_emu_gui
pi	20010	6211	1	09:21	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi	20029	6254	56	09:21	?	00:00:28	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20339	6254	4	09:21	?	00:00:01	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20343	6254	2	09:21	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20464	6211	17	09:22	?	00:00:00	/proc/self/exe --type=utility --field-trial-handle=1075863133478894917,6306120996223181
pi	20488	6254	14	09:22	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20510	676	0	09:22	pts/0	00:00:00	nc -l -p 4444

Nous voyons que chaque processus est identifié par un numéro d'ID. De plus un processus peut avoir un processus parent PPID.





## 4 Une première mise en œuvre avec Python

---

### 4.1 La bibliothèque `threading`<sup>4</sup>

`threading` — Parallélisme basé sur les fils d'exécution (*threads*)

#### Démarrage, arrêt, attente de terminaison des processus

`start()`

Lance l'activité du fil d'exécution.

`run()`

Méthode représentant l'activité du fil d'exécution.

`join(timeout=None)`

Attend que le fil d'exécution se termine. Ceci bloque le fil appelant jusqu'à ce que le fil dont la méthode `join()` est appelée se termine – soit normalement, soit par une exception non gérée – ou jusqu'à ce que le délai optionnel `timeout` soit atteint.

#### Réservation et relâchement de ressources

Exécuté de manière atomique.

`acquire(blocking=True, timeout=-1)`

Acquiert un verrou, bloquant ou non bloquant.

`release()`

Release a lock. This can be called from any thread, not only the thread which has acquired the lock.

`locked()`

Return `True` if the lock is acquired.

### 4.2 Exemple de mise en œuvre

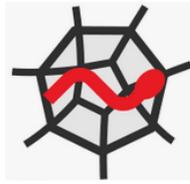


Processus\_1. Faire fonctionner puis documenter le script ci-dessous :  Script\_THREAD\_Demo\_1.py



---

<sup>4</sup> <https://docs.python.org/fr/3/library/threading.html>



## Exemple de résultats :

```
All threads are started
Thread 9 finished!
Thread 7 finished!
Thread 5 finished!
Thread 1 finished!
Thread 3 finished!
Thread 0 finished!
Thread 8 finished!
Thread 2 finished!
Thread 4 finished!
Thread 6 finished!
counter == 1000000
```

```
All threads are started
Thread 0 finished!
Thread 3 finished!
Thread 1 finished!
Thread 7 finished!
Thread 9 finished!
Thread 5 finished!
Thread 4 finished!
Thread 2 finished!
Thread 6 finished!
Thread 8 finished!
counter == 1000000
```

Q4. Commenter ces résultats.

Le code :

```
# https://kooor.fr/Python/CodeSamples/ThreadSample.wp
# Script_THREAD_Demo_1.py

from threading import Thread, Lock

class DemoThread( Thread ):

    counter = 0
    __lock = Lock()          # Try to remove

    def __init__(self, name):
        Thread.__init__(self, name=name)

    def run(self):
        for i in range(100000):
            try:
                DemoThread.__lock.acquire()          # Try to remove
                DemoThread.counter += 1
            finally:
                DemoThread.__lock.release()          # Try to remove

            print(self.name + " finished!")

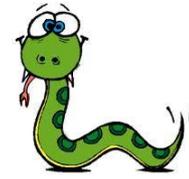
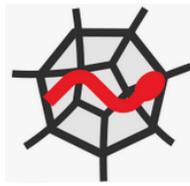
# Démarrage de 10 threads
threads = []
for i in range(10):
    thread = DemoThread("Thread " + str(i))
    thread.start()
    threads.append(thread)

print("All threads are started")

# On attend que les 10 threads aient terminés.
for thread in threads:
    thread.join()

# On vérifie si l'on a bien 1 000 000 dans DemoThread.counter
print("counter == " + str(DemoThread.counter))
```





## 5 Sujet de Bac

---

### EXERCICE 4 (4 points)

*Cet exercice porte sur les systèmes d'exploitation : gestion des processus et des ressources.*

Les parties A et B peuvent être traitées indépendamment.

#### Partie A :

Dans un bureau d'architectes, on dispose de certaines ressources qui ne peuvent être utilisées simultanément par plus d'un processus, comme l'imprimante, la table traçante, le modem. Chaque programme, lorsqu'il s'exécute, demande l'allocation des ressources qui lui sont nécessaires. Lorsqu'il a fini de s'exécuter, il libère ses ressources.

#### Programme 1

demander (table traçante)  
demander (modem)  
exécution  
libérer (modem)  
libérer (table traçante)

#### Programme 2

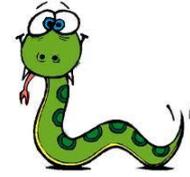
demander (modem)  
demander (imprimante)  
exécution  
libérer (imprimante)  
libérer (modem)

#### Programme 3

demander (imprimante)  
demander (table traçante)  
exécution  
libérer (table traçante)  
libérer (imprimante)

On appelle p1, p2 et p3 les processus associés respectivement aux programmes 1, 2 et 3.

1. Les processus s'exécutent de manière concurrente.  
Justifier qu'une situation d'interblocage peut se produire.
2. Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire. Aucune justification n'est attendue.
3. Supposons que le processus p1 demande la table traçante alors qu'elle est en cours d'utilisation par le processus p3. Parmi les états suivants, quel sera l'état du processus p1 tant que la table traçante n'est pas disponible :  
a) élu                      b) bloqué                      c) prêt                      d) terminé



**Partie B :**

Avec une ligne de commande dans un terminal sous Linux, on obtient l'affichage suivant :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
...							
pi	6211	831	8	09:07	?	00:01:16	/usr/lib/chromium-browser/chromium-browser-v7 --disable-quit --enable-tcp-fast-open --p
pi	6252	6211	0	09:07	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi	6254	6252	0	09:07	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi	6294	6211	4	09:07	?	00:00:40	/usr/lib/chromium-browser/chromium-browser-v7 --type=gpu-process --field-trial-handle=1
pi	6300	6211	1	09:07	?	00:00:16	/usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi	6467	6254	1	09:07	?	00:00:11	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	11267	6254	2	09:12	?	00:00:15	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	12035	836	0	09:13	?	00:00:00	/usr/lib/libreoffice/program/oosplash --writer file:///home/pi/Desktop/mon_fichier.odt
pi	12073	12035	2	09:13	?	00:00:15	/usr/lib/libreoffice/program/soffice.bin --writer file:///home/pi/Desktop/mon_fichier.c
pi	12253	831	1	09:13	?	00:00:07	/usr/bin/python3 /usr/bin/sense_emu_gui
pi	20010	6211	1	09:21	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi	20029	6254	56	09:21	?	00:00:28	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20339	6254	4	09:21	?	00:00:01	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20343	6254	2	09:21	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20464	6211	17	09:22	?	00:00:00	/proc/self/exe --type=utility --field-trial-handle=1075863133478894917,6306120996223181
pi	20488	6254	14	09:22	?	00:00:00	/usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=10758
pi	20519	676	0	09:22	pts/0	00:00:00	ps -ef

La documentation Linux donne la signification des différents champs :

- UID : identifiant utilisateur effectif ;
- PID : identifiant de processus ;
- PPID : PID du processus parent ;
- C : partie entière du pourcentage d'utilisation du processeur par rapport au temps de vie des processus ;
- STIME : l'heure de lancement du processus ;
- TTY : terminal de contrôle
- TIME : temps d'exécution
- CMD : nom de la commande du processus

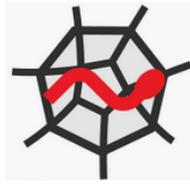
1. Parmi les quatre commandes suivantes, laquelle a permis cet affichage ?

- a) `ls -l`
- b) `ps -ef`
- c) `cd ..`
- d) `chmod 741 processus.txt`

2. Quel est l'identifiant du processus parent à l'origine de tous les processus concernant le navigateur Web (chromium-browser) ?

3. Quel est l'identifiant du processus dont le temps d'exécution est le plus long ?





## 6 Bibliographie

---

Quelques ressources pour la réalisation de ce cours

<https://www.lyceum.fr/tg/nsi/3-architectures-materielles-systemes-dexploitation-et-reseaux/2-gestion-des-processus-par-un-systeme-dexploitation/>

[http://lycee.educinfo.org/index.php?page=creation\\_thread&activite=processus](http://lycee.educinfo.org/index.php?page=creation_thread&activite=processus)

<https://dlatreyte.github.io/terminales-nsi/chap-12/3-processus/> TB avec exemples de code

[https://nsi4noobs.fr/IMG/pdf/c2\\_tnsi\\_processus.pdf](https://nsi4noobs.fr/IMG/pdf/c2_tnsi_processus.pdf)

