



1. Notion de processus

Un [processus](#) (en anglais, process), en informatique, est un programme en cours d'exécution par un ordinateur. De façon plus précise, il peut être défini comme :

- Un ensemble d'instructions à exécuter, pouvant être dans la mémoire morte (DD, SSD, clé USB...), mais le plus souvent chargé vers la mémoire vive (RAM).
- Des données de travail (stockées temporairement dans la mémoire vive)
- des ressources permettant des entrées-sorties de données, comme des ports réseau.

Chaque processus est identifié par :

- un **PID** : numéro de processus (Process Identification)
- un **PPID** : numéro du processus père
- un **UID** ou **USER** : identifiant de l'utilisateur qui a démarré le processus

Un même programme exécuté plusieurs fois générera plusieurs processus.

Exemple des processus sous Windows dans le gestionnaire des tâches (Ctrl+Alt+Suppr)

The screenshot shows the Windows Task Manager window with the 'Processus' tab selected. The 'Plus de détails' button is highlighted with a red box. A red arrow points to the 'Pro' column header in the process list table.

Nom	Statut	7%	74%	1%	0%	Pro
		Processeur	Mémoire	Disque	Réseau	
Applications (5)						
> Client PRONOTE.exe		0%	341,7 Mo	0 Mo/s	0,1 Mbits/s	
> Explorateur Windows (3)		0%	47,7 Mo	0 Mo/s	0 Mbits/s	
> Firefox (32 bits) (6)		0%	340,3 Mo	0 Mo/s	0 Mbits/s	
> Gestionnaire des tâches		0,5%	22,6 Mo	0,1 Mo/s	0 Mbits/s	
> Microsoft Office Word (32 bits) ...		0%	33,5 Mo	0 Mo/s	0 Mbits/s	

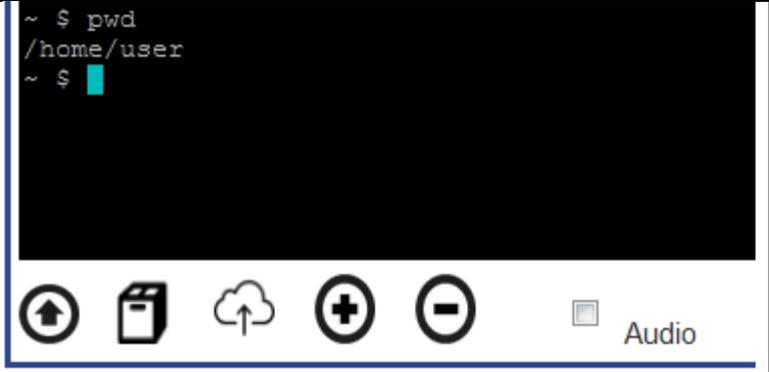
2. Accéder à la liste des processus sous Linux

2.1 Afin de prendre en main facilement Linux de chez vous, installer l’extension Firefox Xlinux comme indiqué dans l’annexe 2.

2.2 Démarrer une session Xlinux dans une nouvelle fenêtre du navigateur.

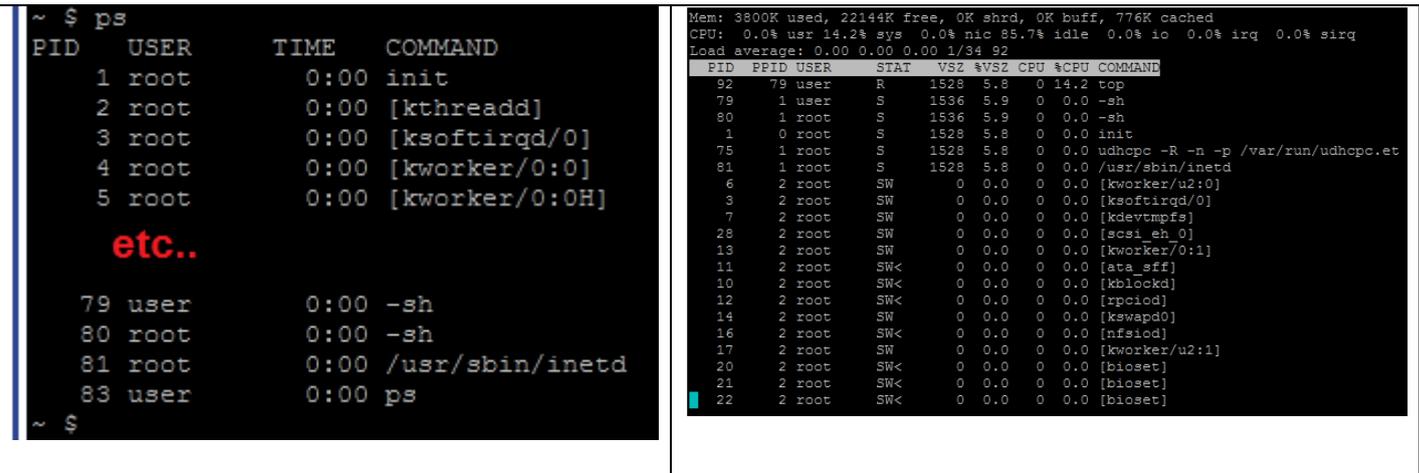
Rappel :

La commande `pwd` indique le répertoire courant



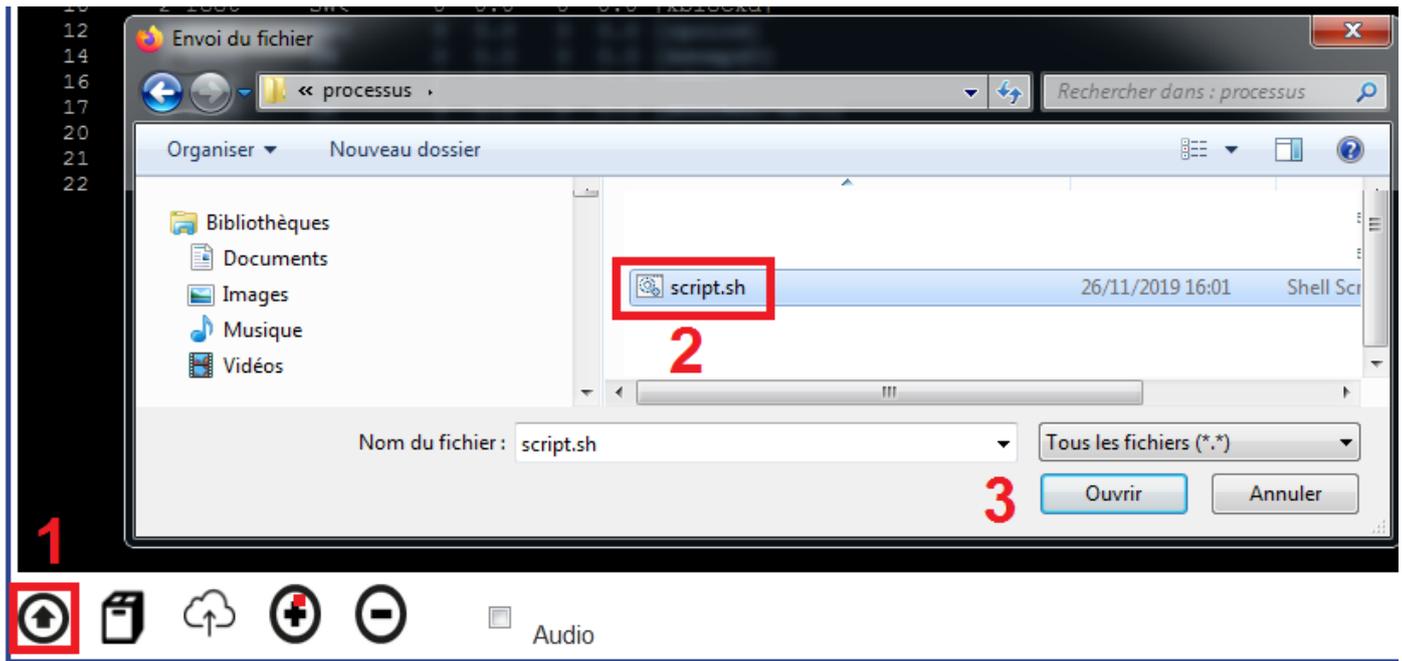
2.3 Exécuter la commande `ps` puis la commande `top` afin de lister les processus en cours d’exécution

Remarque : pour sortir de la commande TOP, taper `ctrl+c`



Repérer les colonnes PID, PPID et USER

2.4 Charger le fichier script.sh dans Xlinux (étapes 1,2 et 3)



2.5 Afficher le contenu du fichier script.sh

Vérifier que le fichier est bien chargé avec la commande <code>ls</code>	<pre>~ \$ ls script.sh ~ \$ cat script.sh #!/bin/bash</pre>
Affichage du contenu du fichier <code>script.sh</code> avec la commande <code>cat script.sh</code>	<pre>while true do r=1 done ~ \$</pre>

Le fichier script.sh contient un programme de boucle infinie

2.6 Exécuter le fichier script.sh

Commande <code>bash script.sh</code>	<pre>~ \$ bash script.sh</pre>
Puis touches <code>Ctrl+c</code> pour sortir	<pre>Ctrl+C pour sortir ^C~ \$</pre>

Cette commande exécute le script, mais ne permet pas de voir le PID du processus `script.sh`, car l'utilisateur n'a plus la « main » sur la console.

2.7 Exécuter le fichier script.sh mais cette fois ci en ajoutant le caractère & en fin de commande.

Commande <code>bash script.sh &</code>	<pre>~ \$ bash script.sh & ~ \$</pre>
--	---

2.8 Afficher la liste des PID et PPID avec la commande `top`.

PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
112	79	user	R	2312	8.8	0	45.7	bash script.sh
113	79	user	R	1528	5.8	0	0.3	top
79	1	user	S	1536	5.9	0	0.0	-sh
80	1	root	S	1536	5.9	0	0.0	-sh
1	0	root	S	1528	5.8	0	0.0	init

Quel est le PID de `script.sh` ?

112

Quel est le PPID de `script.sh` ? A quel processus père cela correspond ?

79, au processus `-sh`

Rappel :

Chaque processus est identifié par un entier unique appelé **PID** (Process IDentifier).

Lorsqu'un processus s'exécute, il peut être en **mode noyau** ou en **mode utilisateur**. Un processus passe en **mode noyau** dès qu'il manipule des données importantes pour le bon fonctionnement de l'ordinateur, il ne peut alors plus être interrompu afin de garantir l'intégrité des données manipulées.

Exemple : Un charpentier répare un trou dans la coque d'un navire, interrompre son travail ne serait pas une bonne idée.

Tous les processus créés le sont de la même façon :

- Un processus **père** est cloné à sa propre demande.
- Le clone se voit assigner un nouveau **PID**.
- Tout processus se voit aussi assigner un **PPID** (Parent Process IDentifier), c'est-à-dire le **PID** de son processus **père**, cela permet de garder un lien entre les 2 processus.

Exemple : Si l'un de vos parents vous demande de lui apporter quelque chose, il est important pour vous de vous rappeler lequel de vos parents vous a fait cette demande.

2.9 Stopper le processus `script.sh`

Commande `kill {PID_de_script.sh}`

```
~ $ kill 112
~ $
```

2.10 Afficher la liste des processus sous forme d'arbre.

<p>Commande ps tree</p>	<pre>~ \$ ps tree init-+-inetd _-sh _-sh---ps tree `-_udhcpd [1]+ Terminated bash script.sh ~ \$</pre>
--------------------------------	--

2.11 Si un processus est créé à partir d'un autre processus, comment est créé le tout premier processus ?

<p>Sous un système d'exploitation comme Linux, au moment du démarrage de l'ordinateur un tout premier processus (appelé processus 0) est créé à partir de "rien" (il n'est le fils d'aucun processus). Remarquer que les PID s'incrémentent au fur et à mesure du fonctionnement du système.</p>	<table border="1"> <thead> <tr> <th>PID</th> <th>PPID</th> <th>USER</th> <th>STAT</th> <th>VSZ</th> <th>%VSZ</th> <th>CPU</th> <th>%CPU</th> <th>COMMAND</th> </tr> </thead> <tbody> <tr> <td>127</td> <td>79</td> <td>user</td> <td>R</td> <td>2312</td> <td>8.8</td> <td>0</td> <td>87.5</td> <td>bash script.sh</td> </tr> <tr> <td>128</td> <td>79</td> <td>user</td> <td>R</td> <td>1528</td> <td>5.8</td> <td>0</td> <td>12.5</td> <td>top</td> </tr> <tr> <td>79</td> <td>1</td> <td>user</td> <td>S</td> <td>1536</td> <td>5.9</td> <td>0</td> <td>0.0</td> <td>-sh</td> </tr> <tr> <td>80</td> <td>1</td> <td>root</td> <td>S</td> <td>1536</td> <td>5.9</td> <td>0</td> <td>0.0</td> <td>-sh</td> </tr> <tr style="border: 2px solid red;"> <td>1</td> <td>0</td> <td>root</td> <td>S</td> <td>1528</td> <td>5.8</td> <td>0</td> <td>0.0</td> <td>init</td> </tr> <tr> <td>75</td> <td>1</td> <td>root</td> <td>S</td> <td>1528</td> <td>5.8</td> <td>0</td> <td>0.0</td> <td>udhcpd -R -n -p</td> </tr> <tr> <td>81</td> <td>1</td> <td>root</td> <td>S</td> <td>1528</td> <td>5.8</td> <td>0</td> <td>0.0</td> <td>/usr/sbin/inetd</td> </tr> <tr> <td>6</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[kworker/u2:0]</td> </tr> <tr> <td>3</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[ksoftirqd/0]</td> </tr> <tr> <td>7</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[kdevtmpfs]</td> </tr> <tr> <td>28</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[sosi_ah_0]</td> </tr> <tr> <td>13</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[kworker/0:1]</td> </tr> <tr> <td>11</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[ata_sff]</td> </tr> <tr> <td>10</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[kblockd]</td> </tr> <tr> <td>12</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[rpciod]</td> </tr> <tr> <td>16</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[nfsiod]</td> </tr> <tr> <td>17</td> <td>2</td> <td>root</td> <td>SW</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[kworker/u2:1]</td> </tr> <tr> <td>20</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[bioset]</td> </tr> <tr> <td>21</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[bioset]</td> </tr> <tr> <td>22</td> <td>2</td> <td>root</td> <td>SW<</td> <td>0</td> <td>0.0</td> <td>0</td> <td>0.0</td> <td>[bioset]</td> </tr> </tbody> </table>	PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND	127	79	user	R	2312	8.8	0	87.5	bash script.sh	128	79	user	R	1528	5.8	0	12.5	top	79	1	user	S	1536	5.9	0	0.0	-sh	80	1	root	S	1536	5.9	0	0.0	-sh	1	0	root	S	1528	5.8	0	0.0	init	75	1	root	S	1528	5.8	0	0.0	udhcpd -R -n -p	81	1	root	S	1528	5.8	0	0.0	/usr/sbin/inetd	6	2	root	SW	0	0.0	0	0.0	[kworker/u2:0]	3	2	root	SW	0	0.0	0	0.0	[ksoftirqd/0]	7	2	root	SW	0	0.0	0	0.0	[kdevtmpfs]	28	2	root	SW	0	0.0	0	0.0	[sosi_ah_0]	13	2	root	SW	0	0.0	0	0.0	[kworker/0:1]	11	2	root	SW<	0	0.0	0	0.0	[ata_sff]	10	2	root	SW<	0	0.0	0	0.0	[kblockd]	12	2	root	SW<	0	0.0	0	0.0	[rpciod]	16	2	root	SW<	0	0.0	0	0.0	[nfsiod]	17	2	root	SW	0	0.0	0	0.0	[kworker/u2:1]	20	2	root	SW<	0	0.0	0	0.0	[bioset]	21	2	root	SW<	0	0.0	0	0.0	[bioset]	22	2	root	SW<	0	0.0	0	0.0	[bioset]
PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND																																																																																																																																																																																						
127	79	user	R	2312	8.8	0	87.5	bash script.sh																																																																																																																																																																																						
128	79	user	R	1528	5.8	0	12.5	top																																																																																																																																																																																						
79	1	user	S	1536	5.9	0	0.0	-sh																																																																																																																																																																																						
80	1	root	S	1536	5.9	0	0.0	-sh																																																																																																																																																																																						
1	0	root	S	1528	5.8	0	0.0	init																																																																																																																																																																																						
75	1	root	S	1528	5.8	0	0.0	udhcpd -R -n -p																																																																																																																																																																																						
81	1	root	S	1528	5.8	0	0.0	/usr/sbin/inetd																																																																																																																																																																																						
6	2	root	SW	0	0.0	0	0.0	[kworker/u2:0]																																																																																																																																																																																						
3	2	root	SW	0	0.0	0	0.0	[ksoftirqd/0]																																																																																																																																																																																						
7	2	root	SW	0	0.0	0	0.0	[kdevtmpfs]																																																																																																																																																																																						
28	2	root	SW	0	0.0	0	0.0	[sosi_ah_0]																																																																																																																																																																																						
13	2	root	SW	0	0.0	0	0.0	[kworker/0:1]																																																																																																																																																																																						
11	2	root	SW<	0	0.0	0	0.0	[ata_sff]																																																																																																																																																																																						
10	2	root	SW<	0	0.0	0	0.0	[kblockd]																																																																																																																																																																																						
12	2	root	SW<	0	0.0	0	0.0	[rpciod]																																																																																																																																																																																						
16	2	root	SW<	0	0.0	0	0.0	[nfsiod]																																																																																																																																																																																						
17	2	root	SW	0	0.0	0	0.0	[kworker/u2:1]																																																																																																																																																																																						
20	2	root	SW<	0	0.0	0	0.0	[bioset]																																																																																																																																																																																						
21	2	root	SW<	0	0.0	0	0.0	[bioset]																																																																																																																																																																																						
22	2	root	SW<	0	0.0	0	0.0	[bioset]																																																																																																																																																																																						

Remarque :

L'arborescence sur une « vraie » machine Linux est beaucoup plus volumineuse que sur le simulateur Xlinux.

3. Rappel du rôle du Système d'Exploitation

3.1 Rappel de 1ere

<p>Le Système d'Exploitation (S.E.) gère l'exécution des différents programmes (un programme en cours d'exécution est appelé processus). Sachant que :</p> <ul style="list-style-type: none"> • Des processus peuvent être exécutés « simultanément » (temps partagé) et peuvent vouloir accéder aux mêmes ressources matérielles (périphériques, fichiers...) • Les processus peuvent avoir des priorités d'exécutions différentes. <p>Ainsi le Système d'Exploitation assure 4 grandes fonctions :</p> <ul style="list-style-type: none"> • Gestion des processus (ordre d'exécution, ordonnancement) • Gestion de l'accès à la mémoire (répartition et accès) • Gestion des périphériques d'entrées/sorties. • Gestion du système de fichiers. <p>Le système d'Exploitation peut présenter une interface graphique facilitant l'utilisation de l'ordinateur.</p>	
---	--

3.2 Les défis du système d'exploitation

Le Nombre d'utilisateurs de « l'ordinateur » :

- **Mono utilisateur** : exemple votre téléphone, votre calculatrice. On peut considérer votre ordinateur individuel, utilisé par une seule personne à la fois. (Même si plusieurs sessions peuvent être ouvertes simultanément)
- **Multi utilisateurs** : Cas des [serveurs dédiés](#).

Nombre de processus à exécuter :

- **Mono tâche** : votre calculatrice effectue une seule tâche (un seul traitement à la fois)
- **Multitâche** : votre téléphone et votre ordinateur vous permettent de lire une vidéo en même temps que vous consultez vos mails etc.

Le système d'exploitation doit gérer le **partage des ressources** (CPU, mémoire, réseau, périphériques, etc..) entre les processus et aussi entre les différents utilisateurs. On parle aussi d'**accès concurrents** aux ressources.

3.3 Notion de contexte d'exécution d'un programme

Lorsqu'un programme qui a été traduit en instructions machines s'exécute, le processeur central lui fournit toutes ses ressources (registres internes, place en mémoire centrale, données, code, ...), on appelle cet ensemble de ressources mises à disposition d'un programme "son **contexte d'exécution**".

4. Etats d'un processus

4.1 Généralités

Un processus est l'image en mémoire centrale d'un programme s'exécutant avec son contexte d'exécution.

Un ordinateur possède un ou plusieurs processeurs, qui sont eux-mêmes constitués de plusieurs unités de calcul, les cœurs et comme nous l'avons vu précédemment un ordinateur peut exécuter une multitude de processus (bien supérieurs au nombre d'unités de calcul d'un ordinateur)

- Comment contrôler l'ordre de passage des processus sur une unité de calcul (processeur) ?
- Comment contrôler la répartition du temps d'exécution entre les processus ?

Réponse :

C'est l'**ordonnanceur**, un composant du système d'exploitation qui va donner à un processus l'accès à une unité de calcul (processeur) selon un algorithme d'ordonnancement.

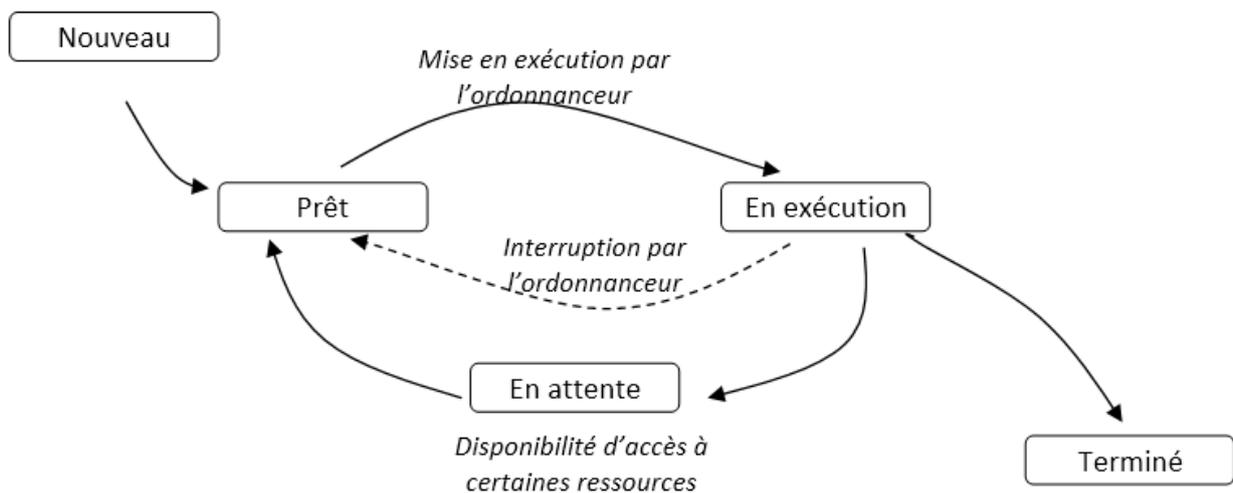
4.2 Exemple

Analyse du script Python suivant :

```
message = input ("Saisissez une phrase")
print(" Votre message est : ", message)
```

- Tant que l'utilisateur ne saisit rien ce script Python (ou plus exactement le processus correspondant) ne pourra pas avancer. L'ordinateur pourrait se retrouver bloqué à attendre ce processus.
- Le système d'exploitation par le biais d'un algorithme d'ordonnancement va mettre en attente ce processus permettant ainsi aux autres processus de l'exécuter.
- Dès que l'utilisateur aura effectué une saisie, l'ordonnanceur pourra réveiller ce processus.

4.3 Cycle de vie d'un processus



- **Nouveau** : état d'un processus en cours de création. Le système d'exploitation vient de copier le code exécutable du programme en mémoire.
- **Prêt** : Le processus peut être le prochain à s'exécuter. Il est dans une **file d'attente**
- **En exécution** ou **élu** (actif) : Le processus est en train de s'exécuter.
- **En attente** ou **bloqué** : Le processus est interrompu et en attente d'un événement externe (entrée/sortie, allocation mémoire etc...)
- **Terminé** : Le processus s'est terminé, le système d'exploitation est en train de désallouer **les ressources** que le processus utilisait.
- Un processus peut également être interrompu par l'ordonnanceur (afin de libérer l'unité de calcul et permettre à d'autres processus de s'exécuter, par exemple si son temps d'exécution est trop long)

5. Algorithmes d'ordonnancement

5.1 Principaux algorithmes d'ordonnancement

Tous les processus « prêts » sont présents dans une **file d'attente**. Tout nouveau processus sera ajouté à cette file.

Lorsqu'une unité de calcul est libre, l'**ordonnanceur** va **déterminer un nouveau processus à affecter** à l'unité de calcul (processeur).

Pour cela il existe plusieurs algorithmes d'ordonnancement :

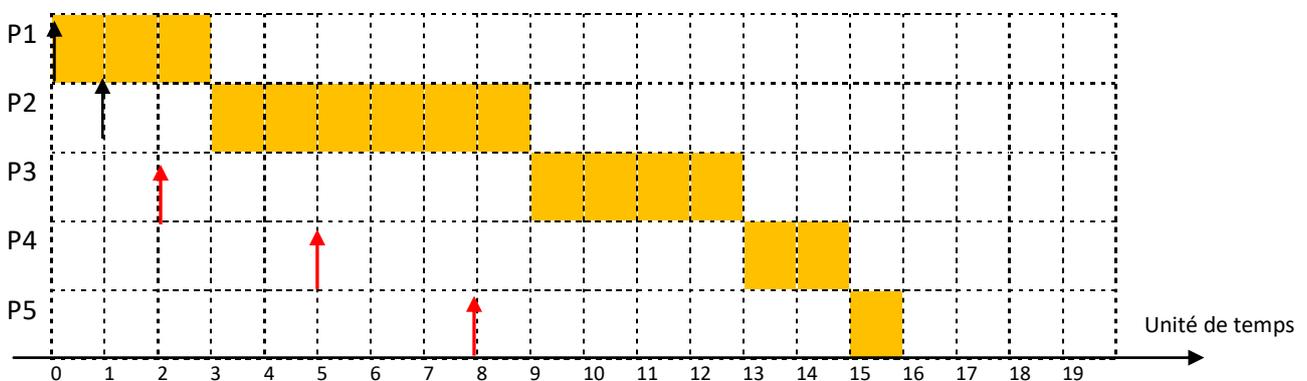
- Selon l'**ordre d'arrivée** ou **First Come First Served (FCFS)**. Les processus sont traités dans l'ordre de leur apparition dans la file d'attente (type FIFO)
- Selon la **durée de calcul** : **Shortest Job First (SJF)** ou Shortest job next (**SJN**). Le processus le « plus court » de la file d'attente est traité en premier par l'unité de calcul.
- En **temps partagé** sans notion de priorité (ou méthode du **tourniquet**) **Round Robin (RR)**: Les processus sont traités par bloc d'instructions à tour de rôle pendant un quantum de temps d'en général 20 à 30 ms. *Si un processus n'est pas terminé à la fin du quantum de temps, il repart à la fin de la liste d'attente.*
- Selon la **durée de calcul restante** : **Shortest Remaining Time (SRT)**. Le processus dont le temps restant est le « plus court » de la file d'attente est traité en premier par l'unité de calcul.
- Il existe d'autres algorithmes d'ordonnancement, comme par exemple le modèle **Priorité**, où chaque processus dispose d'une valeur de priorité et le processus de plus forte priorité est choisi à chaque fois.

5.2 Représentation d'un ordonnancement selon l'algorithme [FCFS](#) ou [FIFO](#)

Lors de leur apparition les processus sont stockés dans une file d'attente et **traités selon leur ordre d'arrivée**.

Processus	P1	P2	P3	P4	P5
Date d'arrivée ⁽¹⁾	0	1	2	5	8
Durée d'exécution ⁽¹⁾⁽²⁾	3	6	4	2	1
Temps d'attente ⁽¹⁾	0	2	7	8	7
Temps de rotation ⁽¹⁾	3	8	11	10	8
Rendement	1	0.75	0.36	0.2	0.125

(1) en unité de temps (2) temps d'utilisation de l'unité de calcul



✗ **Indiquer** par une flèche ↑ sur la représentation graphique ci-dessus la « date d'arrivée » des différents processus P1 à P5 (en suivant l'exemple fourni des processus P1 et P2)

✎ **Indiquer** le moment d'exécution de chaque processus **en grisant** les cases correspondantes.
(L'unité de calcul ne peut exécuter qu'un seul processus à la fois)

📖 Le **temps de rotation** d'un processus ou **temps de séjour** ou **temps d'exécution global**, correspond à la différence entre la **date d'arrivée** et l'**instant de terminaison** (ou la somme du temps d'attente et la durée d'exécution)

📖 Le **temps d'attente** d'un processus ou **durée d'attente** correspond à la différence entre le **temps de rotation** et la **durée d'exécution** par l'unité de calcul (temps de calcul) du processus.

✎ **Compléter** dans le tableau ci-dessus le temps d'attente et le temps de rotation de chaque processus.

📖 **Rendement** = $\frac{\text{durée d'exécution}}{\text{temps de rotation}}$

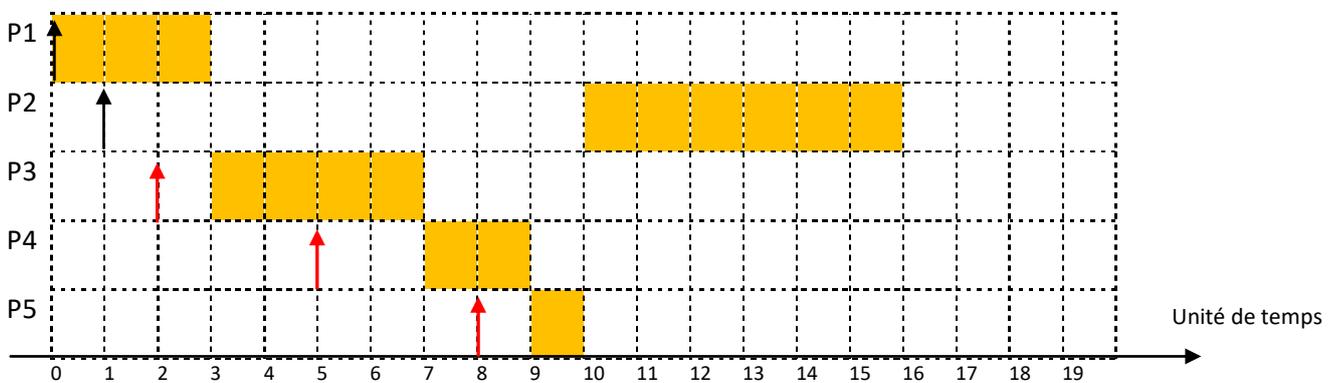
✎ **Calculer et reporter** dans le tableau précédent le rendement de chaque processus.

5.3 Représentation d'un ordonnancement selon l'algorithme [SJF](#)

Lors de leur apparition les processus sont stockés dans une file d'attente et **le processus le plus court de la file est traité en premier.**

Processus	P1	P2	P3	P4	P5
Date d'arrivée ⁽¹⁾	0	1	2	5	8
Durée d'exécution ⁽¹⁾⁽²⁾	3	6	4	2	1
Temps d'attente ⁽¹⁾	0	9	1	2	1
Temps de rotation ⁽¹⁾	3	15	5	4	2
Rendement	1	0.4	0.8	0.5	0.5

(1) en unité de temps (2) temps d'utilisation de l'unité de calcul



✎ **Indiquer** par une flèche ↑ sur la représentation graphique ci-dessus la « date d'arrivée » des différents processus P1 à P5 (en suivant l'exemple fourni des processus P1 et P2)

✎ **Indiquer** le moment d'exécution de chaque processus **en grisant** les cases correspondantes.
(L'unité de calcul ne peut exécuter qu'un seul processus à la fois)

✎ **Compléter** dans le tableau ci-dessus le temps d'attente et le temps de rotation de chaque processus.

✎ **Calculer et reporter** dans le tableau ci-dessus le rendement de chaque processus.

5.3 Représentation d'un ordonnancement selon l'algorithme Round Robin (RR)

Lors de leur apparition les processus sont stockés dans une file d'attente et sont traités par bloc d'instructions **à tour de rôle** pendant un **quantum de temps** maximum.

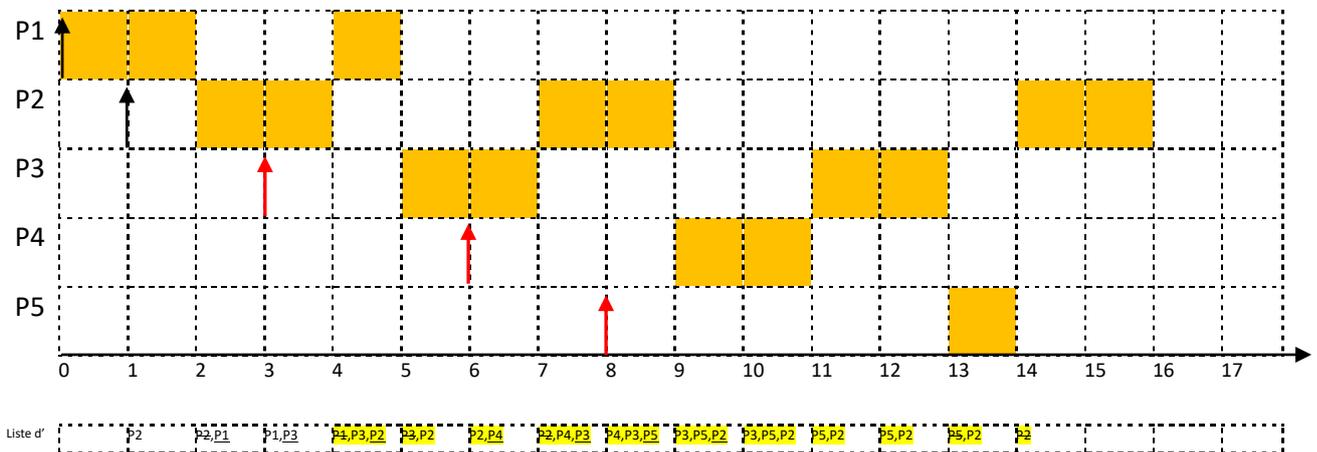
Le 1^{er} processus de la liste d'attente sera exécuté pendant au maximum un quantum de temps puis sera interrompu et replacé à la fin de la liste d'attente et ainsi de suite.

Interrompre un processus nécessite de sauvegarder son contexte afin qu'il puisse reprendre plus tard.

Dans l'exemple suivant nous prendrons un **quantum** de 2 unités de temps. (si au bout de 2 unités de temps un processus n'est pas terminé, il est interrompu et replacé à la fin de la liste d'attente)

Processus	P1	P2	P3	P4	P5
Date d'arrivée ⁽¹⁾	0	1	<u>3</u>	<u>6</u>	8
Durée d'exécution ⁽¹⁾⁽²⁾	3	6	4	2	1
Temps d'attente ⁽¹⁾	2	9	6	3	5
Temps de rotation ⁽¹⁾	5	15	10	5	6
Rendement	0.6	0.4	0.4	0.4	0.16

⁽¹⁾ en unité de temps ⁽²⁾ temps d'utilisation de l'unité de calcul



En dessous de la représentation graphique, nous avons représenté l'évolution de la liste d'attente.

A t = 1 : Le processus P1 étant en cours d'exécution, le processus P2 est placé dans la liste d'attente.

A t = 2 : Le Quantum de 2 unités de temps accordé à P1 est écoulé, son exécution est interrompue. P1 est replacé dans la liste d'attente. Le premier processus de la liste (ici P2) est exécuté.

A t = 3 : Le nouveau processus P3 est ajouté à la fin de la liste d'attente.

- ✎ **Indiquer** par une flèche ↑ sur la représentation graphique ci-dessus la « date d'arrivée » des différents processus P1 à P5 (Attention, valeurs différentes des exemples précédents)
- ✎ **Indiquer** le ou les moments d'exécution de chaque processus en grisant les cases correspondantes. (L'unité de calcul ne peut exécuter qu'un seul processus à la fois) et compléter simultanément l'évolution de la liste d'attente.
- ✎ **Compléter** dans le tableau ci-dessus le temps d'attente et le temps de rotation de chaque processus.
- ✎ **Calculer et reporter** dans le tableau ci-dessus le rendement de chaque processus.

Les ordinateurs classiques utilisent un ordonnancement de type temps partagé qui est proche du Round Robin en tenant compte de priorités.

 Pour s'entraîner : un [Simulateur sur Android](#) ou sur [PC](#) (FCFS et JSF seulement)

6. Accès concurrent aux ressources

6.1 Programmation concurrente en Python

Lors de son exécution un processus a besoin d'accéder à des ressources. Comme nous l'avons déjà indiqué les ressources peuvent être matérielles (processeur, mémoire, périphériques, fichiers...) ou logicielles (variables). Dans certaines situations plusieurs processus peuvent vouloir accéder aux mêmes ressources (fichiers, périphériques ou variables).

Afin de pouvoir montrer la problématique d'interblocage de processus, nous allons introduire la notion de programmation multithread en Python. Un thread est un sous processus démarré à partir d'un processus et s'exécutant de manière concurrente avec le reste du programme.

6.2 Programmation concurrente en Python

Ce script comporte 3 fonctions qui affichent chacune 50 messages. Au lieu de faire un appel classique de ces 3 fonctions qui ferait qu'elles s'exécutent les unes après les autres, nous réalisons un appel sous la forme de Thread (processus ou tâche).

Décrire le fonctionnement obtenu. Quels éléments nous permettent de dire que ces 3 fonctions se sont exécutées sous la forme de 3 processus distincts ? (Faire plusieurs essais)

Réponse :

Les affichages produits par les 3 fonctions s'entrecroisent, montrant que les 3 fonctions s'exécutent « simultanément » et pas chronologiquement à leur appel.

```
import threading
from time import sleep

def afficheBonjour():
    for i in range(50):
        print("Bonjour_" + str(i) + " ")
        sleep(0.0001)
    print("fin du thread Bonjour")

def afficheHello():
    for i in range(50):
        print("Hello_" + str(i) + " ")
        sleep(0.002)
    print("fin du thread Hello")

def afficheHola():
    for i in range(50):
        print("Hola_" + str(i) + " ")
        sleep(0.001)
    print("fin du thread Hola")

t1=threading.Thread(target=afficheBonjour)
t2=threading.Thread(target=afficheHello)
t3=threading.Thread(target=afficheHola)

t1.start()
t2.start()
t3.start()
```

6.3 Garantir la cohérence des données partagées

La fonction `incr()` est appelée 3 fois. Celle-ci incrémente de 1 000 000 la valeur de la variable globale `compteur`. Donc la variable `compteur` devrait avoir la valeur 3 000 000 en fin d'exécution. Tester le fonctionnement du programme. Indiquer si la valeur de la variable `compteur` est conforme à vos attentes ?

Réponse :

La valeur du compteur est inférieure à 3 000 000

```
import threading
compteur=0

def incr():
    global compteur
    for c in range(1000000):
        v=compteur
        compteur = v + 1

t1 = threading.Thread(target=incr)
t2 = threading.Thread(target=incr)
t3 = threading.Thread(target=incr)
t1.start()
t2.start()
t3.start()
t1.join()
t2.join()
t3.join()
print("valeur finale", compteur)
```

Remarque :

La méthode `join()` permet d'attendre que le processus associé se termine (Wait until the thread terminates)

Pourquoi le résultat obtenu n'est pas conforme à nos attentes ?

Les trois appels de la fonction `incr()` se font sous la forme de 3 processus distincts qui se partagent cette même variable globale `compteur`. Les 3 processus s'exécutent « simultanément » (en temps partagé par la méthode du tourniquet ou Round Robin).

Ainsi le processus x peut être interrompu à n'importe quel moment pour laisser la place au processus y.



Si le processus x est interrompu juste après sa ligne 9 de programme par le processus y, il ne prendra pas en compte la modification apportée à la variable `compteur` par le processus y. Le processus x a lu et stocké la valeur `compteur` dans sa variable locale `v` avant d'être interrompu et reprendra cette valeur pour la suite de son exécution, occultant ainsi le traitement réalisé par le processus y.

6.4 Notion de Section critique et de verrou (ou sémaphore)

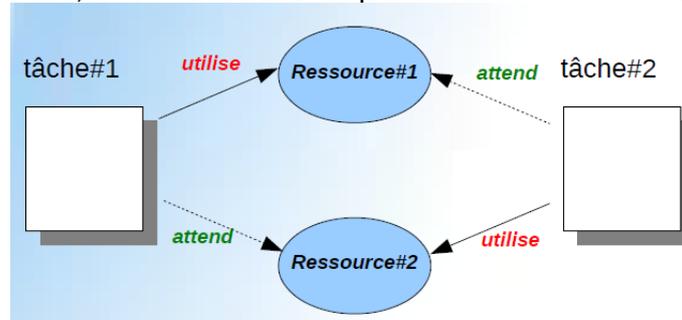
<p>Une section critique de notre code intervient lors de l'incrément de la variable <code>compteur</code>, si l'un des processus est interrompu par l'ordonnanceur entre les deux lignes ci-contre.</p>	<pre> 9 v=compteur 10 compteur = v +1 </pre>
<p>L'ajout d'un verrou permet de garantir l'exclusivité de l'accès à la variable <code>compteur</code> pendant cette section critique du programme.</p> <p>Modifier son script afin de faire apparaître les 3 lignes permettant la mise en place d'un verrou (sémaphore).</p> <p>Indiquer si le fonctionnement alors obtenu est conforme.</p>	<pre> verrou = threading.Lock() compteur=0 def incr(): global compteur for c in range(1000000): verrou.acquire() v=compteur compteur = v +1 verrou.release() </pre>
<p>Réponse :</p> <p>La variable <code>compteur</code> atteint la valeur 3 000 000 comme souhaité.</p>	

7. Problème d'interblocage ou DeadLock

7.1 Mise en situation

Considérons deux élèves comme étant deux processus. Leur tâche : l'élève 1 tracer un cercle, l'élève 2 tracer un rectangle au tableau. Pour réaliser leur traitement, ils ont besoin chacun de deux ressources : l'accès au tableau et l'accès au marqueur.

L'ordonnanceur donne accès au marqueur à un élève et l'accès au tableau à l'autre élève. Les deux élèves vont se trouver dans une situation d'attente interminable puisque le premier élève attend que la ressource feutre soit libre, et le second attend que la ressource feutre soit libre.



Dans un ordinateur le même phénomène peut se produire entre plusieurs processus, c'est l'interblocage.

7.2 Exemple de script utilisant les « [threads](#) »

Saisir ce script Python sous le nom `interblocage.py` et tester son fonctionnement.

Les deux fonctions sont-elles intégralement exécutées ?

Réponse :

4 messages devraient s'afficher, mais suivant la répartition du temps par l'ordonnanceur du système d'exploitation entre les deux processus f1 et f2, une situation d'interblocage (DeadLock en anglais) peut apparaître.

Aucun des deux processus ne s'exécute intégralement.

```
import threading
import time

verrou1 = threading.Lock()
verrou2 = threading.Lock()

def f1():
    verrou1.acquire()
    print("section critique 1.1")
    time.sleep(1)
    verrou2.acquire()
    print("section critique 1.2")
    verrou2.release()
    verrou1.release()

def f2():
    verrou2.acquire()
    print("section critique 2.1")
    time.sleep(1)
    verrou1.acquire()
    print("section critique 2.2")
    verrou1.release()
    verrou2.release()

t1 = threading.Thread(target=f1)
t2 = threading.Thread(target=f2)

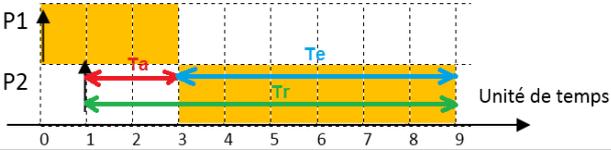
t1.start()
t2.start()
```

Cette situation d'interblocage a été théorisée par l'informaticien Edward Coffman (1934-) qui a énoncé quatre conditions (appelées conditions de Coffman) menant à l'interblocage :

1. **Exclusion mutuelle** : au moins une des ressources du système doit être en accès exclusif. (Les ressources ne sont pas partageables, un seul processus à la fois peut utiliser la ressource).
2. **Rétention des ressources** : un processus détient au moins une ressource et requiert une autre ressource détenue par un autre processus
3. **Non préemption** : Seul le détenteur d'une ressource peut la libérer. Les ressources ne sont pas préemptibles c'est-à-dire que les libérations sont faites volontairement par les processus. On ne peut pas forcer un processus à rendre une ressource.
4. **Attente circulaire** : Chaque processus attend une ressource détenue par un autre processus. P_1 attend une ressource détenue par P_2 qui à son tour attend une ressource détenue par P_3 etc.... qui attend une ressource détenue par P_1 ce qui clôt la boucle.

Processus et ordonnancement, d'après un document de E. Bansière- P. Jonin

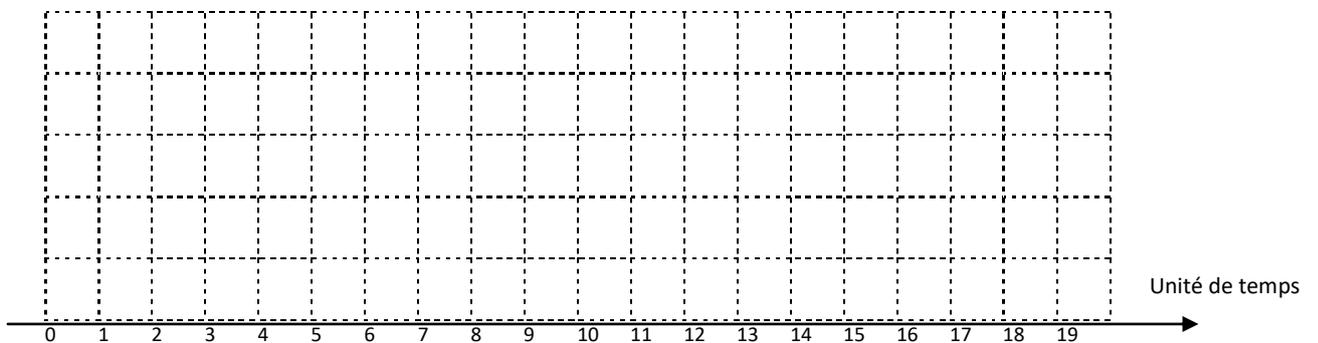
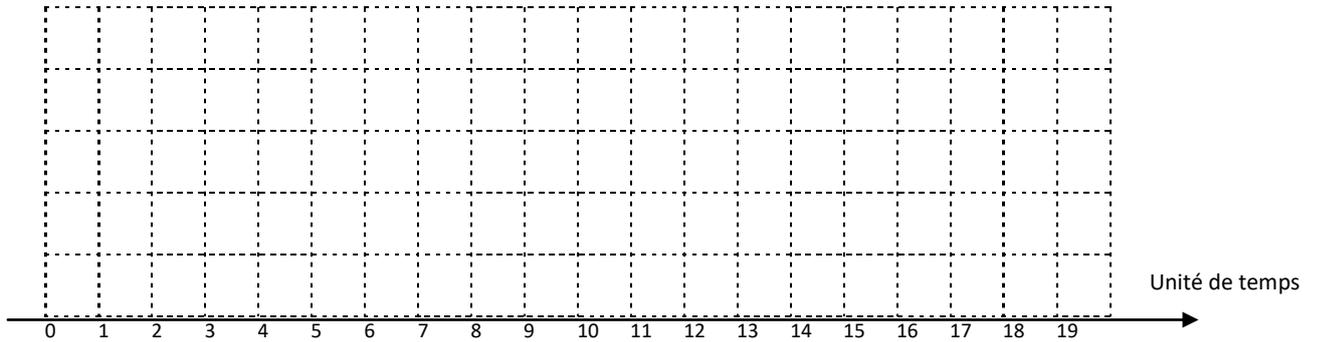
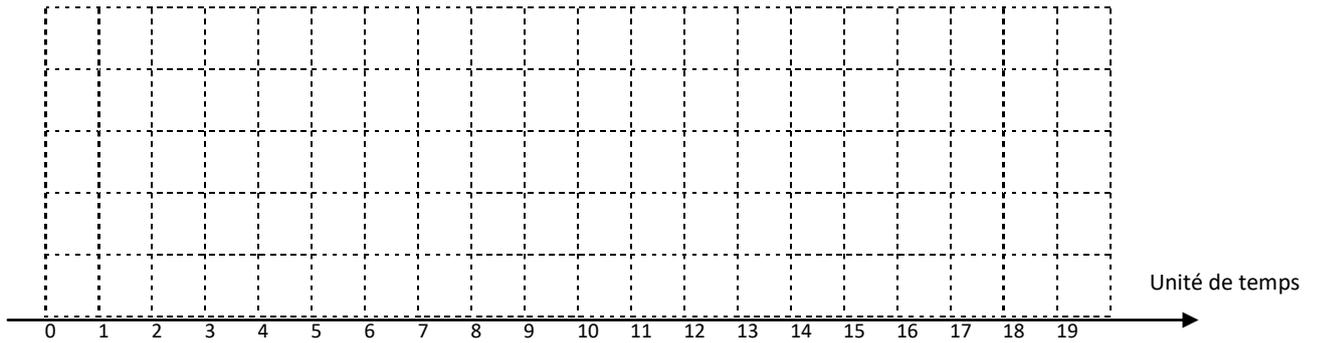
Annexe 1 : Lexique (A connaitre par cœur)

Processus (Tâche=thread)	Programme en cours d'exécution par un ordinateur. Etats d'un processus : Prêt, en exécution ou élu, en attente ou bloqué, terminé.
PID	Numéro de processus (Process Identification).
PPID	Numéro du processus père.
Commande ps	La commande ps liste les processus en cours d'exécution.
Commande kill pid	La commande kill pid , stoppe le processus identifié par son PID.
Système d'Exploitation	Gère l'exécution des différents programmes (processus). Les processus peuvent être exécutés « simultanément ».
Mono tâche	Ne peut exécuter qu'un processus à la fois.
Multitâche	Permet d'exécuter plusieurs processus en « même temps »
Accès concurrents	Partage des ressources (CPU, mémoire, réseau, périphériques, etc..) entre les processus.
Ordonnanceur	Dans les systèmes d'exploitation, l'ordonnanceur désigne le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur les processeurs d'un ordinateur.
Algorithme First Come First Served (FCFS)	Les processus sont traités dans l'ordre de leur apparition dans la file d'attente.
Algorithme Shortest Job First (SJF)	Le processus le « plus court » de la file d'attente est traité en premier par l'unité de calcul.
Round Robin (RR)	En temps partagé, l'ordonnanceur parcourt une file et alloue un temps processeur à chacun des processus pendant un quantum de temps.
Temps de rotation d'un processus	Somme du temps d'attente T_a et de la durée d'exécution T_e 
Le temps d'attente d'un processus	Différence entre le temps de rotation T_r et la durée d'exécution T_e par l'unité de calcul (temps de calcul) du processus.
Rendement	Rendement = $\frac{\text{durée d'exécution}}{\text{temps de rotation}}$
Interblocage (DeadLock)	L'interblocage se produit lorsque des processus concurrents s'attendent mutuellement.

Annexe 2 : Grilles d'entrainement

Processus	P1	P2	P3	P4	P5
Date d'arrivée ⁽¹⁾					
Durée d'exécution ⁽¹⁾⁽²⁾					
Temps d'attente ⁽¹⁾					
Temps de rotation ⁽¹⁾					
Rendement					

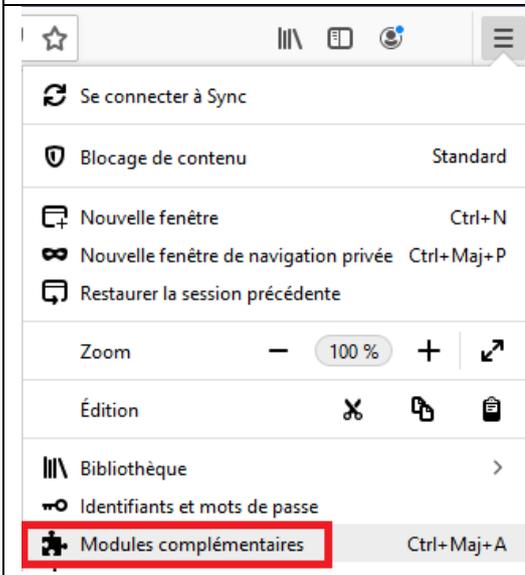
⁽¹⁾ en unité de temps ⁽²⁾ temps d'utilisation de l'unité de calcul



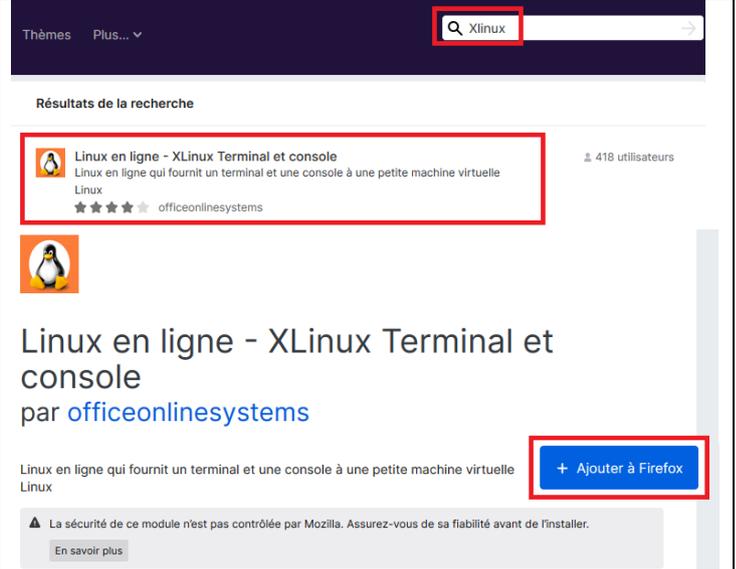
Annexe 3

Installation du plugin Xlinux dans le navigateur Firefox

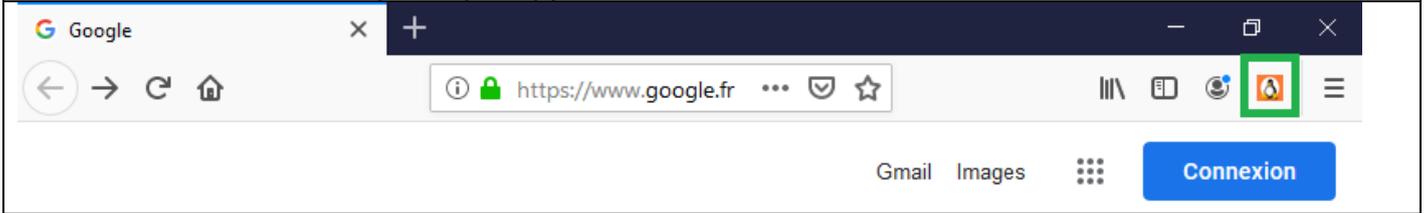
1 Installer un module complémentaire



2 Faire une recherche sur le mot clé « Xlinux », puis installer le plug-in



3 Maintenant une icône spécifique apparaît en haut à droite de la fenêtre.



4 Exécuter Xlinux. Une nouvelle fenêtre apparaît simulant un environnement Linux.

