

Chapitre 15 - Processus

Objectifs :

- ▷ Décrire la création d'un processus, l'ordonnancement de plusieurs processus par le système.
- ▷ Mettre en évidence le risque de l'interblocage (*deadlock*).

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 y restait jusqu'à ce que celui-ci "plante" ou se termine.

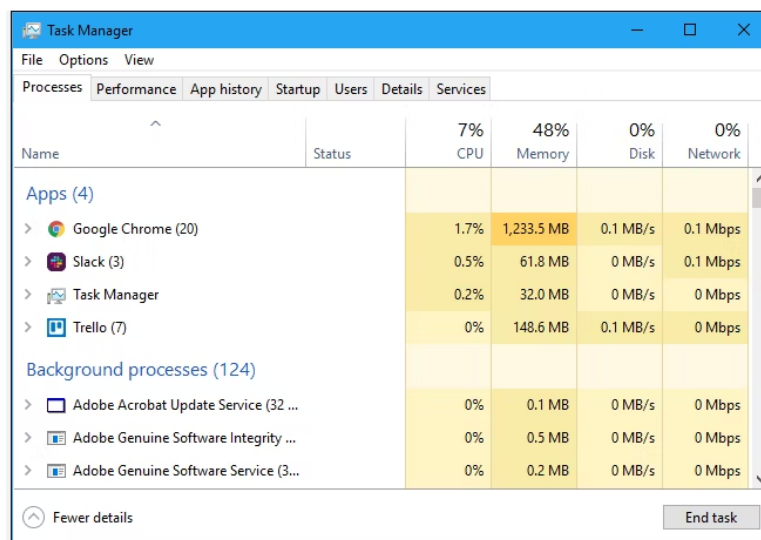
Les systèmes d'exploitation récents (*Windows*, *Linux* ou *MacOS* par exemple) permettent d'exécuter plusieurs tâches simultanément - ou en tous cas, donner l'impression que celles-ci s'exécutent en même temps. C'est ce que l'on appelle le **pseudo-parallélisme**.

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**.

Nous avons tous été confrontés à la problématique de la gestion des processus dans un système d'exploitation, en tant qu'utilisateur :

- quand nous cliquons sur l'icône d'un programme, nous provoquons la naissance d'un ou plusieurs processus liés au programme que nous lançons
- quand un programme ne répond plus, il nous arrive de lancer le gestionnaire de tâches pour tuer le processus en défaut



Name	Status	7% CPU	48% Memory	0% Disk	0% Network
Apps (4)					
Google Chrome (20)		1.7%	1,233.5 MB	0.1 MB/s	0.1 Mbps
Slack (3)		0.5%	61.8 MB	0 MB/s	0.1 Mbps
Task Manager		0.2%	32.0 MB	0 MB/s	0 Mbps
Trello (7)		0%	148.6 MB	0.1 MB/s	0 Mbps
Background processes (124)					
Adobe Acrobat Update Service (32 ...)		0%	0.1 MB	0 MB/s	0 Mbps
Adobe Genuine Software Integrity ...		0%	0.5 MB	0 MB/s	0 Mbps
Adobe Genuine Software Service (3...		0%	0.2 MB	0 MB/s	0 Mbps

Nous allons voir en détails dans ce cours comment les processus sont gérés dans les systèmes d'exploitation.

Un processus peut déclencher d'autres processus. On dira que ce sont des processus **parent** et **fil**s.

Certains fonctionnent en permanence en **arrière-plan** (comme un antivirus) et ceux qui s'exécutent dès le démarrage de la machine (les **services** sous *Windows* ou les **daemons** sous *Linux*).

2 Programme \neq Processus

On peut comparer un programme à une recette de cuisine listant les ingrédients et décrivant la méthode à utiliser pour la réaliser.

Le processus serait alors quelqu'un qui a récupéré la recette, acheté les ingrédients et qui serait en train de réaliser la recette.

De plus à un instant donné, on peut avoir plusieurs personnes dans la même cuisine. Chacun s'occupe d'une recette (la même ou pas...), chacun est sur son propre exemplaire de recette et dans un état d'avancement différent.

A retenir !

Un **programme** est une suite figée d'instructions, un ensemble statique.

Un **processus** est une exécution (ou une 'instance') d'un programme sur un ordinateur.

Il est caractérisé par :

- ▷ Un **ensemble d'instructions à exécuter** - souvent stockées dans un fichier exécutable sur lequel on clique pour lancer un programme. (ex. firefox.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 Firefox, chacune travaillera indépendamment de l'autre avec ses propres données. C'est ce qu'on appelle le contexte d'un processus.
- ▷ Des **ressources matérielles** : processeur, entrées-sorties ...

La gestion des processus offerte par le système d'exploitation permet d'agir de façon individuelle sur les différentes tâches sans impacter les autres.

On peut par exemple :

- en mettre certaine en pause.
- en terminer d'autres.
- gérer les tâches en fonction de priorités.

PROGRAM VS PROCESS		
Basis For Comparision	PROGRAM	PROCESS
Basic	Program is a set of Instruction	When a program is execute , it is known as process
Nature	Passive	Active
Lifespan	Longer	Limited
Required Resources	Program is stored on disk in some file and does not require any other resources.	Process holds resources such as CPU, Memory address , disk, I/O etc.

Faire les exercices 1 et 2.

3 Création d'un processus

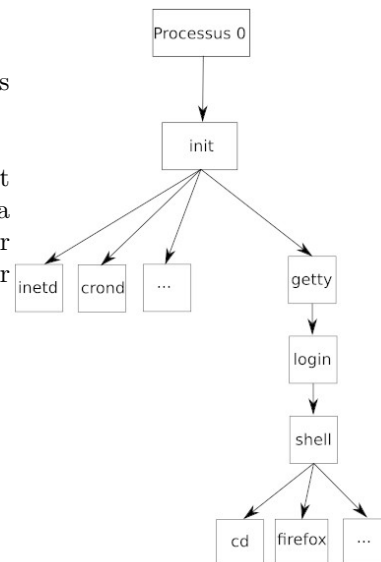
A retenir !

La création d'un processus peut intervenir :

- ▷ au **démarrage du système**
- ▷ par un **appel d'un autre processus**
- ▷ par une **action d'un utilisateur** (lancement d'application)

L'illustration ci-contre montre le graphe d'appels des processus sous *Linux*.

On comprend que tous les processus et applications sont en fait démarrés par le **Processus 0**, qui est lancé au démarrage de la machine ; celui-ci lance ensuite le processus fils **init** qui va lancer les processus systèmes (**daemons**) qui vont eux-mêmes lancer d'autres processus, ..., *etc.*



Lors de sa création, un processus sera identifié par :

- Un numéro (PID : *Process Identification*)
- Un PPID (le numéro du processus parent)
- Un UID, l'identifiant de l'utilisateur qui l'a lancé

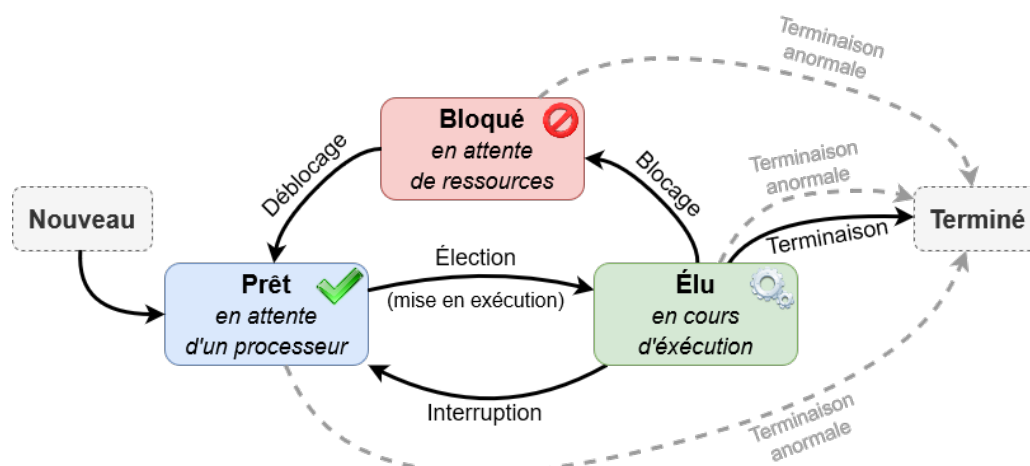
4 Etat du processus

Dans un système multitâches, plusieurs processus peuvent se trouver simultanément en cours d'exécution : ils se partagent l'accès aux ressources de la machine (Entrée-sortie, CPU, mémoire...).

A retenir !

Un processus peut prendre 3 états :

- ▷ **État actif** ou **élu** (*running*) : le processus utilise le processeur.
- ▷ **État prêt** ou **éligible** (*ready*) : le processus pourrait utiliser le processeur s'il était libre (et si c'était son tour).
- ▷ **État en attente** ou **bloqué** : le processus attend une ressource (ex : fin d'une entrée-sortie, par exemple il est en pause car il attend que l'utilisateur fasse une entrée clavier...).



Tout processus qui se bloque en attente d'un événement, passe dans l'état bloqué tant que l'événement attendu n'est pas arrivé.

Lors de l'occurrence de cet événement, le processus passe dans l'état prêt. Il sera alors susceptible de se voir attribuer le processeur pour continuer ses activités.

A retenir !

Le changement d'état d'un processus peut être provoqué par :

- un **autre processus** (qui lui a envoyé un signal, par exemple)
- le **processus lui-même** (appel à une fonction d'entrée-sortie bloquante),
- une **interruption** (fin de quantum, terminaison d'entrée-sortie, ..., etc.)

La sortie de l'état actif pour passer à l'état prêt se produit dans le cas des ordonnancements lorsqu'un processus plus prioritaire que le processus actif courant devient prêt.

5 Ordonnancement

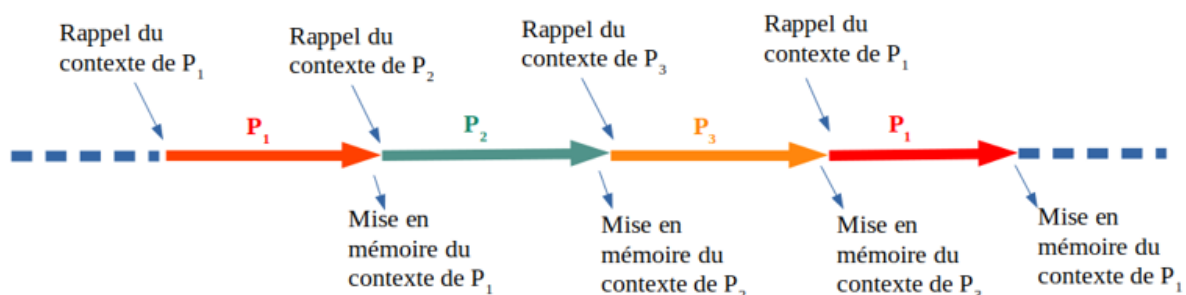
Si le processus est en cours d'exécution, le contexte est mis en mémoire dans les registres du processeur.

Si le processus est mis en pause pour qu'un autre s'exécute, le contexte est mis en mémoire quelque part et sera rappelé dans les registres du processeur lorsque le processus aura de nouveau la main.

Voici un schéma qui illustre l'**ordonnancement** de 3 processus (P1, P2, et P3) avec un seul processeur.

Chacun des processus occupe le processeur pendant un certain temps.

La **mise en mémoire** et le **rappel des contextes** est très important pour que l'exécution d'un processus n'interfère pas avec les autres.



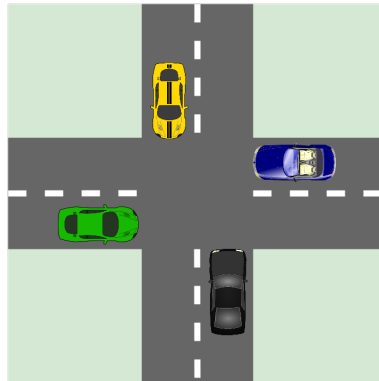
Il existe plusieurs types d'ordonnancement :

1. **FIFO** (*First In, First Out*) : par exemple la file d'impression dans une imprimante. Ici le premier processus à arriver va utiliser la ressource jusqu'à ce qu'il ait terminé.
2. **Round Robin** : la ressource est affectée à chaque processus à tour de rôle. On définit une durée fixe pendant laquelle un processus est exécuté sur le processeur
3. **Avec priorité** : l'ordre est déterminé par la priorité du processus ; si un processus plus prioritaire arrive, c'est lui qui prend la main.
4. **Le plus court d'abord** : ici on s'intéresse à la durée d'exécution de chaque processus. Cet ordonnancement est très efficace pour satisfaire au mieux les utilisateurs ; mais il n'est pas toujours facile d'évaluer la durée d'une tâche avant son début.

Faire les exercices 3 et 4.

6 Interblocage (*deadlock*)

Les **interblocages** sont des situations de la vie quotidienne. Un exemple est celui du carrefour avec priorité à droite où chaque véhicule est bloqué car il doit laisser le passage au véhicule à sa droite.



En informatique également, l'interblocage peut se produire lorsque des processus concurrents s'attendent mutuellement. Les processus bloqués dans cet état le sont définitivement. Ce scénario catastrophique peut se produire dans un environnement où des ressources nécessaires sont partagées entre plusieurs processus et l'un d'entre eux détient indéfiniment une ressource nécessaire à l'autre.

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

A retenir !

1. **Exclusion mutuelle** : au moins une des ressources du système doit être en accès exclusif, seule un processus peut l'utiliser à la fois
2. **Rétention des ressources** (*Hold and wait*) : 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.
4. **Attente circulaire** : chaque processus attend une ressource détenue par un autre processus.
5. P1 attend une ressource détenue par P2 qui à son tour attend une ressource détenue par P3, ..., etc., qui attend une ressource détenue par P1 ce qui clos la boucle.

Pour illustrer l'interblocage en Python, faire l'**exercice 5** sur *Collab*.



7 Exercices

Exercice 1 : Task manager

Ouvrir le gestionnaire de tâches de Windows en faisant **CTRL+ALT+Suppr** ou en tapant la commande **taskmgr** dans la console, puis en choisissant *Task Manager*. Aller dans l'onglet *Détails*

1. Quel processus utilise le plus de mémoire ?
2. Combien de pourcentage du temps le processeur est-il en attente (*System Idle Process*) ?
3. Quel processus utilise le plus le CPU ?
4. Combien y a-t-il d'utilisateurs ?

Faire un clic droit sur un des processus. Les actions disponibles sont :

- *End task* pour interrompre immédiatement le processus
- *End process tree* pour interrompre immédiatement le processus et son arborescence (les processus fils qu'il a démarré)
- *Set Priority* pour changer la priorité de ce processus par rapport aux autres

5. Ouvrir l'onglet *Services*. Il permet de voir les processus systèmes qui sont exécutés en arrière plan.

Vous pouvez observer l'utilisation des ressources (mémoire, disque, réseau) dans l'onglet *Performances*. L'utilisation du CPU est donnée soit de manière globale soit séparément par coeur. Pour voir l'un ou l'autre des affichages, faire un clic droit et choisir *Change graph to*.

6. Aller maintenant sur *Cocalc* et saisir la commande **top**. Il s'agit du gestionnaire de processus de *Linux*.
7. Quelles sont les colonnes communes avec l'onglet *Détails* du *Gestionnaire de tâches Windows* ?

Exercice 2 : Console Windows

1. Faire une recherche pour savoir comment démarrer depuis la console *Windows* le programme **msedge** (*Microsoft Edge*) en fournissant l'adresse du site du lycée.
2. Combien de processus ont été démarrés ? Quels sont leurs PID ? Quelle est leur relation ?
3. Identifier le PID du processus qui consomme le plus de mémoire.
4. Que permet de faire la commande **tasklist** dans une console *Windows* ?
5. Chercher de l'aide sur la commande **taskkill** et utilisez-la pour terminer le processus **msedge** qui consomme le plus de mémoire. Que se passe-t-il ?
6. Utiliser l'option **/F** pour forcer à terminer ce processus.
7. Terminer le processus **msedge** qui consomme le moins de mémoire.
8. Expliquer ce que vous observez.

Exercice 3 : Ordonnancement et chronogramme

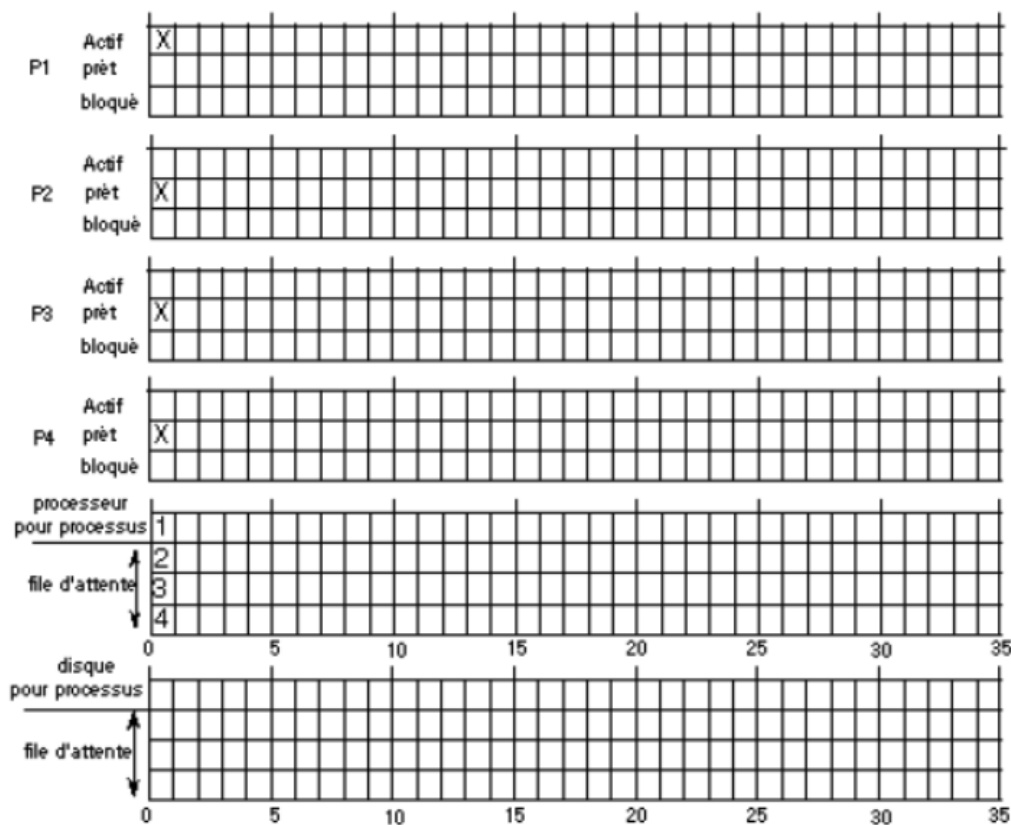
On considère un système **monoprocesseur** et les 4 processus **P1**, **P2**, **P3** et **P4** qui effectuent du calcul et des entrées/sorties avec un disque selon les temps donnés ci-dessous :

Processus P1	Processus P2
Calcul : 3 unités de temps	Calcul : 4 unités de temps
E/S : 7 unités de temps	E/S : 2 unités de temps
Calcul : 2 unités de temps	Calcul : 3 unités de temps
E/S : 1 unité de temps	E/S : 1 unité de temps
Calcul : 1 unité de temps	Calcul : 1 unité de temps
Processus P3	Processus P4
Calcul : 2 unités de temps	Calcul : 7 unités de temps
E/S : 3 unités de temps	
Calcul : 2 unités de temps	

On considère que l'ordonnancement sur le processeur se fait selon une politique FIFO : le processus élu à un instant t est celui qui est le plus anciennement dans l'état prêt. Initialement, l'ordre de soumission des processus est P1, puis P2, puis P3, puis P4.

De même, on considère que l'ordre de services des requêtes d'E/S pour le disque se fait selon une politique FIFO.

Sur graphe suivant, donnez le chronogramme d'exécution des 4 processus P1, P2, P3 et P4 :



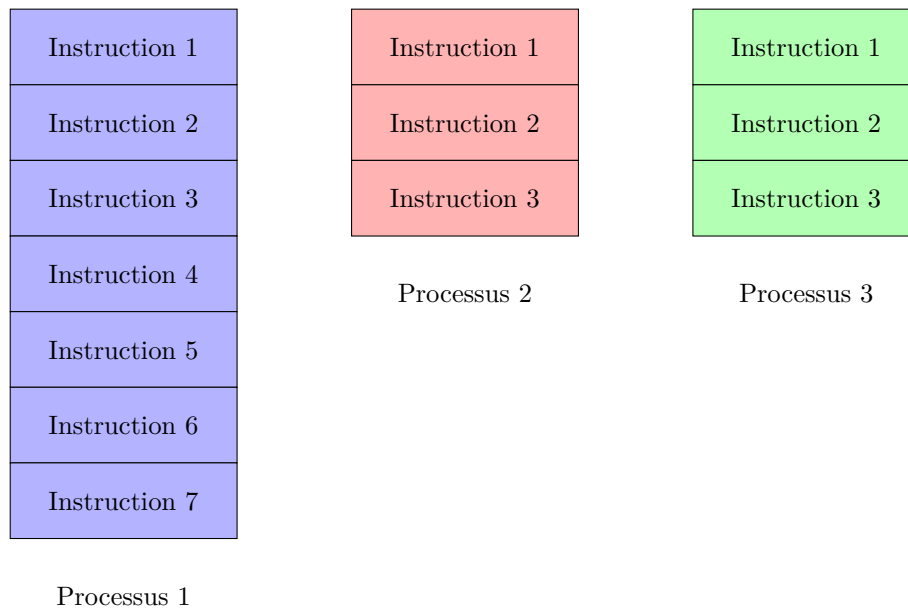
Vous distinguerez les états des processus : *Prêt*, *Actif* et *Bloqué* et vous indiquerez le contenu des files d'attente des processus (attente processeur et attente du disque, rappelons que le disque ne peut exécuter qu'une seule opération à la fois).

Pour vous guider, la première unité de temps est déjà portée sur le chronogramme.

Justifier votre raisonnement, en expliquant la gestion des files d'attentes et les transitions des processus. Donnez le temps de réponse moyen obtenu.

Exercice 4 : Ordonnancement de 3 processus

Les trois processus suivants doivent être exécutés simultanément sur un ordinateur à un processeur :



1. L'ordonnanceur du système d'exploitation utilise l'ordonnancement du **plus court d'abord**. Schématiser l'ordre de traitement des instructions des trois processus.
2. Schématiser l'ordre de traitement des instructions des trois processus pour un ordonnancement **Round Robin**.

Exercice 5 : Illustration de l'interblocage en Python

Sur *Collab*, suivre les consignes du [document suivant](#).

Exercice 6 : Exercice type BAC

Cet exercice est l'exercice n°2 du sujet n°2 du BAC 2021 métropole en candidat libre.

Les états possibles d'un processus sont : prêt, élu, terminé et bloqué.

1. Expliquer à quoi correspond l'état élu.
2. Proposer un schéma illustrant les passages entre les différents états.

On suppose que quatre processus C_1 , C_2 , C_3 et C_4 sont créés sur un ordinateur, et qu'aucun autre processus n'est lancé sur celui-ci, ni préalablement ni pendant l'exécution des quatre processus.

L'ordonnanceur, pour exécuter les différents processus prêts, les place dans une structure de données de type file. Un processus prêt est enfilé et un processus élu est défilé.

Parmi les propositions suivantes, recopier celle qui décrit le fonctionnement des entrées/sorties dans une file :

Premier entré, dernier sorti

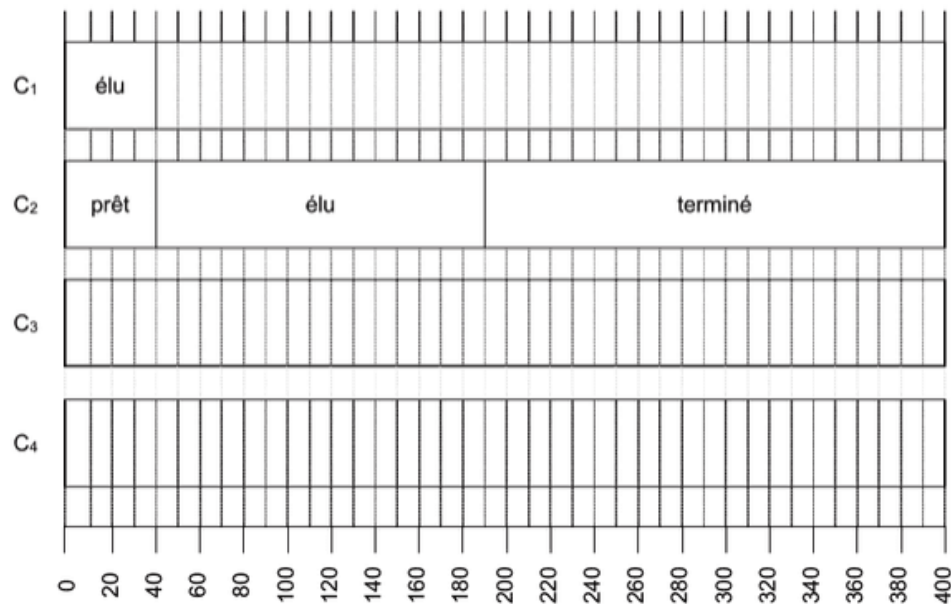
Premier entré, premier sorti

Dernier entré, premier sorti

3. On suppose que les quatre processus arrivent dans la file et y sont placés dans l'ordre C_1 , C_2 , C_3 et C_4 .
- ▷ Les temps d'exécution totaux de C_1 , C_2 , C_3 et C_4 sont respectivement 100 ms, 150 ms, 80 ms et 60 ms.
 - ▷ Après 40 ms d'exécution, le processus C_1 demande une opération d'écriture disque, opération qui dure 200 ms. Pendant cette opération d'écriture, le processus C_1 passe à l'état bloqué.
 - ▷ Après 20 ms d'exécution, le processus C_3 demande une opération d'écriture disque, opération qui dure 10 ms. Pendant cette opération d'écriture, le processus C_3 passe à l'état bloqué.

Sur la frise chronologique ci-dessous donnée en annexe, les états du processus C_2 sont donnés.

Compléter la frise avec les états des processus C_1 , C_3 et C_4 .



On trouvera ci-dessous deux programmes rédigés en pseudo-code.

Verrouiller un fichier signifie que le programme demande un accès exclusif au fichier et l'obtient si le fichier est disponible.

Programme 1	Programme 2
Verrouiller fichier_1 Calculs sur fichier_1 Verrouiller fichier_2 Calculs sur fichier_1 Calculs sur fichier_2 Calculs sur fichier_1 Déverrouiller fichier_2 Déverrouiller fichier_1	Verrouiller fichier_2 Verrouiller fichier_1 Calculs sur fichier_1 Calculs sur fichier_2 Déverrouiller fichier_1 Déverrouiller fichier_2

4. En supposant que les processus correspondant à ces programmes s'exécutent simultanément (exécution concurrente), expliquer le problème qui peut être rencontré.
5. Proposer une modification du programme 2 permettant d'éviter ce problème.

Exercice 7 : Exercice type BAC

Cet exercice est la première partie de l'exercice 3 du sujet 1 candidat libre Métropole 2021.

La commande UNIX `ps` présente un cliché instantané des processus en cours d'exécution.

Avec l'option `-eo pid,ppid,stat,command`, cette commande affiche dans l'ordre l'identifiant du processus PID (*process identifier*), le PPID (*parent process identifier*), l'état STAT et le nom de la commande à l'origine du processus.

Les valeurs du champ STAT indique l'état des processus :

- ▷ R : processus en cours d'exécution
- ▷ S : processus endormi

Sur un ordinateur, on exécute la commande `ps -eo pid,ppid,stat,command` et on obtient un affichage dont on donne ci-dessous un extrait.

```
$ ps -eo pid,ppid,stat,command
PID  PPID  STAT  COMMAND
1     0     Ss    /sbin/init
....  ....  ..    ...
1912  1908  Ss    Bash
2014  1912  Ss    Bash
1920  1747  Sl    Gedit
2013  1912  Ss    Bash
2091  1593  Sl    /usr/lib/firefox/firefox
5437  1912  Sl    python programme1.py
5440  2013  R     python programme2.py
5450  1912  R+    ps -eo pid,ppid,stat,command
```

À l'aide de cet affichage, répondre aux questions ci-dessous.

1. Quel est le nom de la première commande exécutée par le système d'exploitation lors du démarrage ?
2. Quels sont les identifiants des processus actifs sur cet ordinateur au moment de l'appel de la commande `ps` ? Justifier la réponse.
3. Depuis quelle application a-t-on exécuté la commande `ps` ? Donner les autres commandes qui ont été exécutées à partir de cette application.
4. Expliquer l'ordre dans lequel les deux commandes `python programme1.py` et `python programme2.py` ont été exécutées.
5. Peut-on prédire que l'une des deux commandes `python programme1.py` et `python programme2.py` finira avant l'autre ?