Gestion des processus et des ressources par un système d'exploitation

I./ Notion de processus :

Pour permettre le bon fonctionnement d'un système informatique, de nombreuses applications ou tâches doivent pouvoir être exécutées simultanément. Certaines applications peuvent même être exécutées plusieurs fois, et plusieurs applications peuvent accéder aux mêmes ressources physiques (mémoire, périphériques, …) en même temps. Afin de permettre cela et d'éviter les conflits, le *système d'exploitation* doit donc générer de nombreux *processus*, puis doit gérer leur exécution.

Attention : **« processus » ≠ « programme ». Le même programme exécuté plusieurs fois générera plusieurs processus : un processus est donc une instance particulière d'un programme, ainsi que son contexte d'exécution (ressources utilisées, paramètres d'exécution, …).**

II./ Ordonnancement :

Une des tâches principales du système d'exploitation est de *partager le temps d'exécution* entre les différents processus s'exécutant sur la machine : pendant un certain temps, seul un processus peut avoir accès aux ressources (processeur, mémoire, périphériques). Lorsque ce temps est fini, le processus se met en pause, libère les ressources qu'il utilisait, et donne sa place à un nouveau processus, etc… Pour l'utilisateur, tout cela est transparent et il a l'impression que tous les programmes fonctionnent simultanément. La partie du système d'exploitation se chargeant de cette tâche s'appelle *l'ordonnanceur*.

Il y a plusieurs modèles d'ordonnancement possibles. Voici les plus répandus :

1./ Le modèle tourniquet (ou « Round Robin ») :

Soit 3 processus P1, P2 et P3. Dans ce modèle, l'ordonnanceur va d'abord exécuter la première instruction du processus P1, puis la première instruction du processus P2, puis la première instruction du processus P3. Il va ensuite passer à la deuxième instruction du processus P1, et ainsi de suite. Le temps d'exécution est ainsi équitablement partagé entre les trois processus, donnant à l'utilisateur l'impression que les trois tâches se déroulent simultanément.

2./ Le modèle du premier entré, premier sorti (ou « FIFO » : First In, First Out) :

Soit trois processus P1, P2 et P3, créés à des moments différents, dans cet ordre. Dans ce modèle, l'ordonnanceur va d'abord exécuter le processus créé en premier (ici P1), puis le deuxième (P2), et enfin le troisième (P3). Ce modèle fonctionne comme la file d'impression des documents sur une imprimante.

3./ Le modèle du plus court d'abord (ou « SJF » : Shortest job first) :

Dans ce modèle, l'ordonnanceur va exécuter les processus du plus court, au plus long. Cette méthode n'est pas toujours utilisable car il est parfois difficile d'évaluer le temps d'exécution d'une tâche avant de l'avoir lancée.

4./ Le modèle du système de priorités :

Dans ce modèle, on affecte un ordre de priorité aux différentes tâches, qui seront alors exécutées en fonction de cet ordre. Pour que ce système soit réellement équitable, il faut que la répartition du niveau de priorité des différents processus soit réalisée de manière objective.

Pour vous entrainer, cliquez sur le lien suivant (https://view.genial.ly/5f5408e829c2060cf8fd3afa/) afin de réaliser un exercice interactif portant sur les trois premiers modèles d'ordonnancement cités.

III./ États d'un processus :

Afin de pouvoir réaliser ce partage des tâches, l'ordonnanceur va attribuer un *état* à chacun des processus :

1./ L'état « élu » (ou « exécution ») :

Un processus est dans *l'état « élu »* lorsqu'il est en train d'être exécuté par le processeur. Un seul processus peut se trouver dans l'état élu à un instant donné.

2./ L'état « bloqué » (ou « endormi » :

Lorsqu'un processus étant dans l'état élu demande l'accès à une ressource qui n'est pas disponible instantanément (car utilisée par un autre processus), il passe à *l'état « bloqué »* et se met à attendre la ressource convoitée. C'est alors un autre processus, auparavant en attente, qui passera alors à l'état élu.

3./ L'état « prêt » (ou « en attente ») :

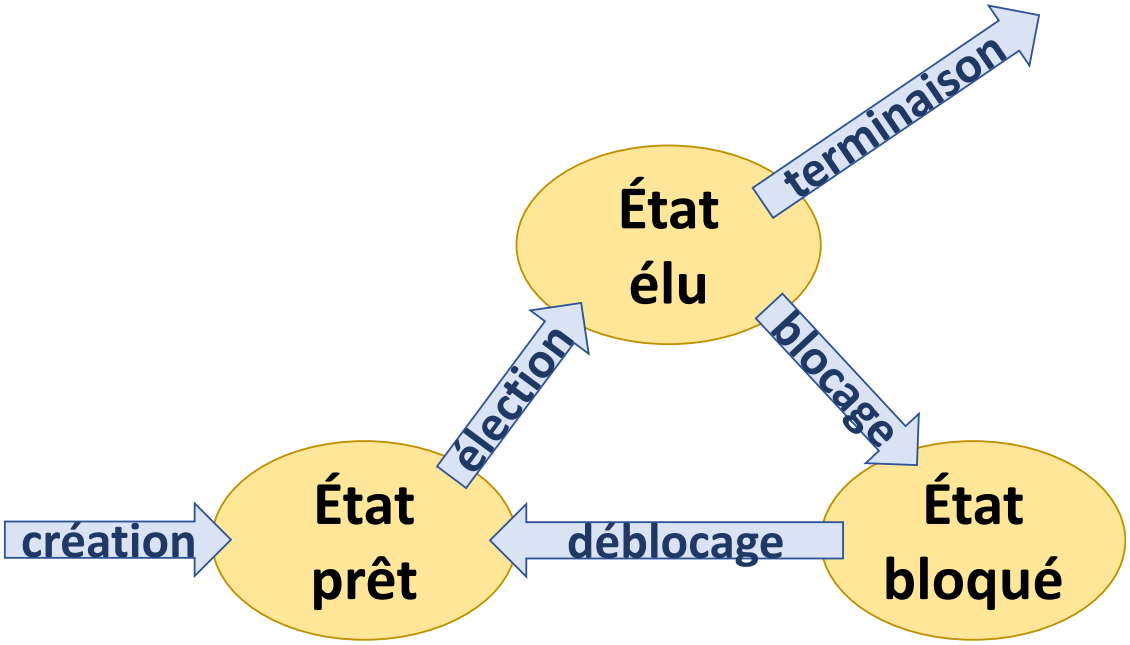
Un processus dans l'état bloqué qui finit par obtenir la ressource qu'il attendait ne peut pas forcément reprendre son exécution immédiatement car un autre processus est alors dans l'état élu. Le processus passe alors de l'état bloqué à *l'état « prêt »*. Il est alors susceptible de repasser à l'état élu lorsque « la place se libérera ».

4./ Remarques :

Un processus est toujours créé dans l'état prêt. Le passage de l'état prêt à l'état élu s'appelle « *l'élection* ». Le passage de l'état élu à l'état bloqué s'appelle le « *blocage* ». Un processus qui a fini d'utiliser une ressource doit la « *libérer* » afin que d'autres processus puissent l'utiliser à leur tour. Afin de libérer une ressource, un processus doit obligatoirement être dans un état élu. De la même façon, seul un processus dans l'état élu peut se terminer.

5./ Synthèse :

Voici un schéma récapitulant les points précédents :



Ces trois états sont les états principaux pour un processus, mais il en existe d'autres. Pour plus d'informations :
https://fr.wikipedia.org/wiki/Processus_(informatique)

IV./ Création et caractéristiques des processus :

1./ Création des processus :

Le tout premier processus à être exécuté sur l'ordinateur est créé au démarrage du système d'exploitation. C'est ce premier processus (appelé *processus 0*) qui va créer les processus suivants, qui a leur tour vont en créer d'autres, jusqu'à ce que tous les processus nécessaires au fonctionnement du système d'exploitation soient lancés. Lorsqu'un processus A crée un processus B, on dit que A est le père de B, et B est le fils de A. Le processus B pourra à son tour créer un processus C, etc… Une autre façon de créer un processus est l'action d'un utilisateur, par exemple le lancement d'une application.

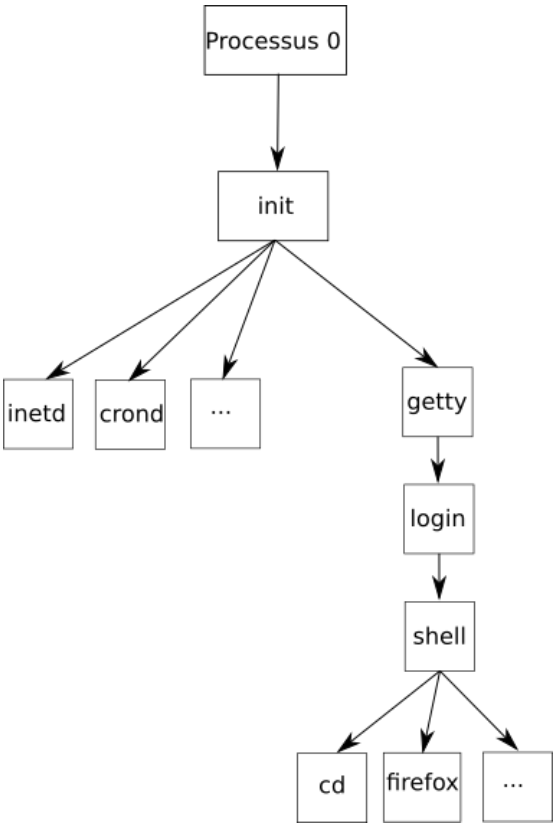
2./ Caractéristiques des processus :

Pour différencier les différents processus en cours d'exécution, le système d'exploitation leur donne à chacun un *nombre d'identification unique*, appelé *PID* (**P**rocess **ID**entifier). Le tout premier processus créé aura un PID égal à 0, le deuxième un PID de 1, etc… A chaque création de processus, le PID est incrémenté de 1.

En plus du PID, chaque processus possède d'autres caractéristiques :

- Le PPID (**P**arent **P**rocess **ID**entifier) : c'est le PID du processus parent. Tous les processus ont un PPID, mis à part le tout premier processus de PID 0. En effet le processus 0 est créé à partir de rien, et il n'est le fils d'aucun processus.
- Le propriétaire (User ou UID) qui correspond à l'utilisateur ayant créé le processus.
- Le niveau de priorité.
- L'état du processus.
- Le temps écoulé depuis la création.
- Etc…

3./ Exemple : organisation des processus sous Linux :



Sous Linux, le premier processus créé (processus 0), appelé aussi « *swapper* », va créer un processus appelé « init », qui aura donc un PID égal à 1. Ce processus « init » va ensuite créer un certain nombre de processus, comme « inetd », « crond », « getty », etc… Chacun des « fils » du processus « init » vont à leur tour créer d'autres processus, etc…

Le schéma ci-dessus montre un schéma simplifié des relations entre les processus d'un système Linux, qu'on peut représenter par une structure d'arbre.

En vous basant sur le schéma précédent, déterminez le PID et le PPID du processus « getty », en admettant que ce dernier est créé juste après « init ».

V./ Expérimentations sous Linux :

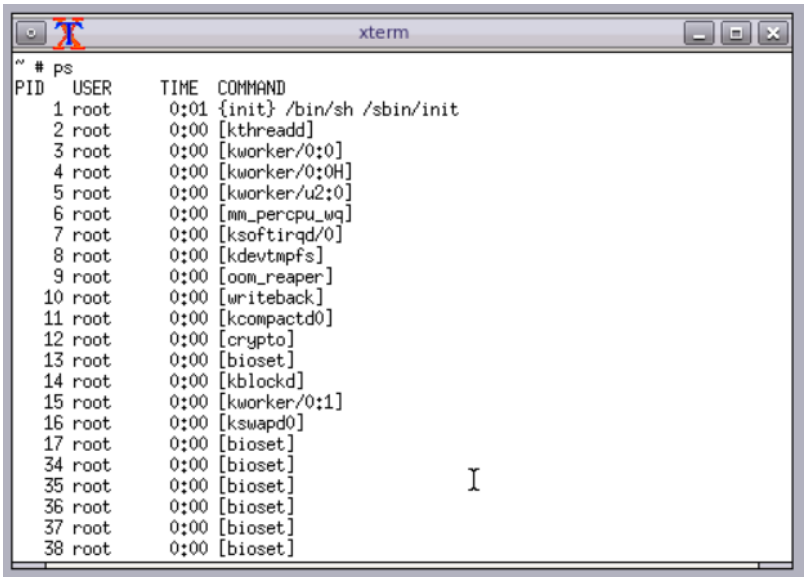
Nous allons utiliser un émulateur de système Linux tournant directement dans le navigateur. Cet émulateur tournant en mode graphique, son chargement sera un peu plus lent que les émulateurs en mode texte préalablement utilisés : il faudra donc patienter un peu pendant le chargement !

Pour accéder à l'émulateur, utilisez le lien suivant :
https://bellard.org/jslinux/vm.html?url=alpine-x86-xwin.cfg&mem=256&graphic=1

Après le chargement du système, l'interface graphique se charge, et on se retrouve devant un bureau vide (fond gris). La première chose à faire est de configurer le clavier en AZERTY, car le système démarre en QWERTY. Pour cela, il suffit de faire un clic-droit sur le bureau de la machine émulée afin de faire apparaître le menu, puis de choisir « Keyboard mapping », puis « French » : (pensez à bien refaire cette étape si vous relancez la machine virtuelle après un plantage, en actualisant le navigateur).

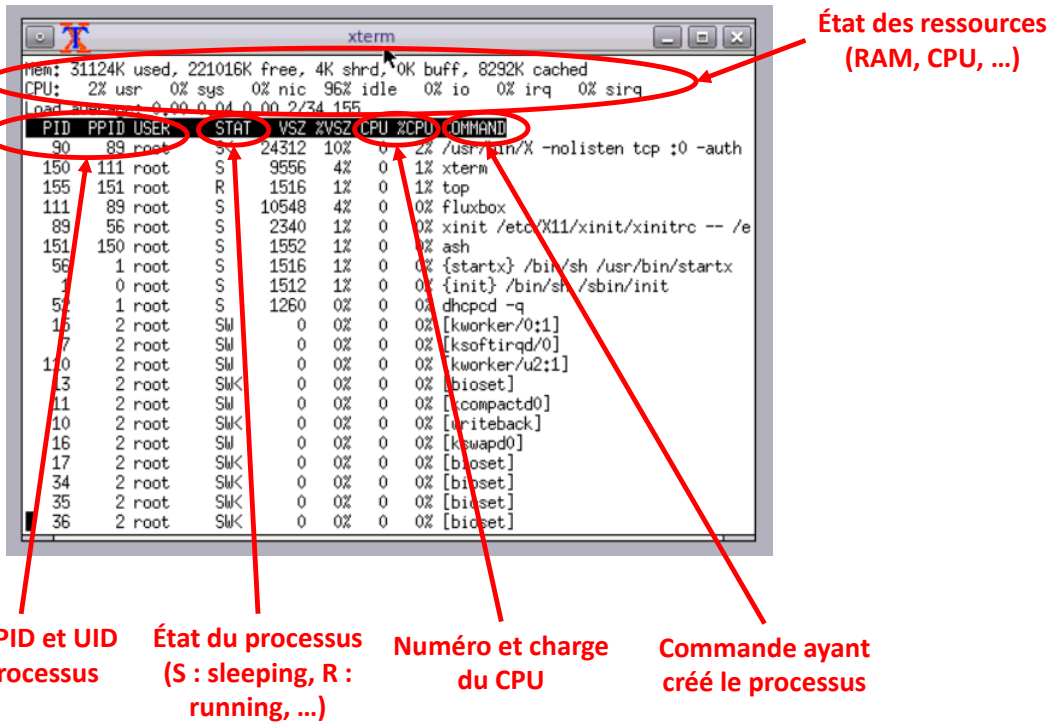
Ceci étant fait, nous allons pouvoir ouvrir une fenêtre « Terminal » (ligne de commande), en faisant clic-droit sur le bureau, puis « Terminal ». Une fenêtre nommée « xterm » s'ouvre alors (soyez patient, je vous le rappelle !).

Pour obtenir la liste des processus en cours d'exécution sur la machine, il suffit de taper la commande « ps » (comme **P**rocess **S**tatus) dans la fenêtre « xterm ». On obtient notamment le PID de chaque processus (on remarquera que le processus 0 n'apparait pas), ainsi que la commande ayant lancé le processus :



Une autre commande permet de lister les processus principaux en montrant leurs dépendances sous forme d'arbre. Il s'agit de la commande « pstree » (ou « pstree -p » pour afficher en plus les PID des processus). Testez cette commande.

Ces deux commandes ne donnent que peu d'informations sur les processus, et de plus elles ne montrent qu'un instant donné. Une troisième commande permet de suivre l'évolution des processus en temps réel, la commande « top ». Dans la fenêtre « xterm » (qu'on appellera « fenêtre n°1 »), tapez « top », pour obtenir quelque chose de ce type :



Vous devriez notamment avoir une ligne qui correspond au processus « xterm » que vous avez lancé. Notez le PID et le PPID de ce processus :

PID #1 : …… PPID #1 : ……

On remarque que l'affichage est réactualisé régulièrement. Pour s'en convaincre, lancez une autre fenêtre « xterm » (qu'on appellera « fenêtre n°2 ») et vérifiez que le processus correspondant apparaît bien dans la fenêtre faisant tourner la commande « top ». Notez alors le PID et le PPID de cette nouvelle fenêtre « xterm » :

PID #2 : …… PPID #2 : ……

Lancez une troisième fenêtre « xterm » (qu'on appellera « fenêtre n°3 »), et notez à nouveau le PID et le PPID correspondant :

PID #3 : …… PPID #3 : ……

On va maintenant lancer une quatrième application, appelée « Qemacs » (clic-droit puis « Qemacs »), un éditeur de textes. On obtient une nouvelle fenêtre appelée « qe » (que l'on appellera « fenêtre n°4 »), et une ligne « xterm -e qe » apparait dans la fenêtre n°1 (ceci correspond à la commande permettant de lancer Qemacs). Notez le PID et le PPID correspondant :

PID #4 : …… PPID #4 : ……

Que remarque-t-on si on compare les PPID de #1 à #4 ? Ce résultat vous semble-t-il logique ? Quel est le nom de l'application ayant le PID correspondant ? Recherchez sur Internet à quoi correspond cette application.

……
……
……
……
……

Lorsqu'un processus se termine de lui-même, à la fin de l'exécution de sa dernière instruction, il libère les ressources qu'il utilisait puis il est détruit par le système d'exploitation. Mais on peut aussi terminer un processus volontairement, à l'aide de la combinaison de touche (CTRL-C) ou en lui envoyant un signal.

Le signal permettant de terminer un processus proprement (c'est-à-dire en libérant les ressources utilisées) est le signal n°15 de terminaison (SIGTERM). Pour envoyer ce signal au processus que l'on veut terminer, on utilise la commande « kill ».

Pour tester cette commande, placez-vous dans la fenêtre n°2, et tapez « kill PID#4 » (***ATTENTION :*** à la place de « PID#4 », il faut taper la valeur correspondante, que vous avez notée précédemment !). Qu'arrive-t-il à la fenêtre n°4 ?
……
……
……

Toujours dans cette fenêtre n°2, tapez « kill PID#3 ». Qu'arrive-t-il à la fenêtre n°3 ?
……
……
……

Quelle commande faut-il taper pour fermer la fenêtre n°1 ? Testez votre hypothèse.
……
……
……

De la même façon, fermez la dernière fenêtre ouverte, la fenêtre n°2. Quelle commande faut-il utiliser ?
……
……
……

Dans certains cas (plantage, bug, …), un processus peut refuser de se terminer correctement. On peut alors lui envoyer le signal n°9 de destruction (SIGKILL) pour forcer sa terminaison inconditionnelle et immédiate. La commande correspondante est : « kill -9 PID_processus ».

Il existe d'autres signaux permettant de communiquer avec les processus. Plus d'informations sur ce lien : https://fr.wikipedia.org/wiki/Signal_(informatique)

VI./ Expérimentations sous Windows :

Avec un système Windows, on peut gérer les processus (appelés plutôt « services » dans le monde Microsoft) grâce au gestionnaire des tâches que l'on peut lancer de trois manières différentes :

- Par la combinaison de touches : CTRL-SHIFT-ECHAP,
- Par la combinaison de touches : CTRL-ALT-SUPPR, puis « Gestionnaire des tâches »,
- Par la commande « taskmgr » en ligne de commande.

Les sessions sur les PC du lycée ne nous donnant pas les droits nécessaires pour utiliser le gestionnaire des tâches, vous pourrez tester cet outil chez vous si vous avez une machine Windows. ***ATTENTION : une fausse manipulation avec ce programme peut entraîner un plantage de l'ordinateur. Pensez à sauvegarder tout fichier ouvert avant de faire des essais !***

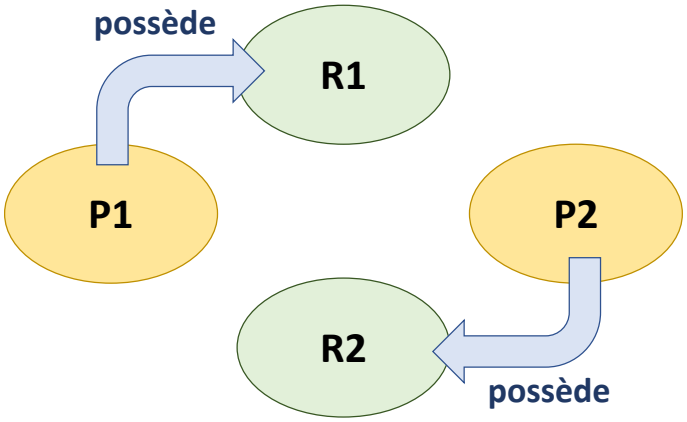
Pour les plus courageux (et surtout les plus patients), vous pouvez tester un émulateur de Windows 2000 en ligne, mais c'est extrêmement lent : https://bellard.org/jslinux/vm.html?url=win2k.cfg&mem=192&graphic=1&w=1024&h=768

Le gestionnaire des tâches est apparu dans Windows dès la version 3 (sous le nom de « Liste des tâches » à l'époque) et il a grandement évolué. C'est pourquoi sa présentation et ses fonctionnalités peuvent être assez différentes d'une version de Windows à l'autre.

VII./ Interblocage :

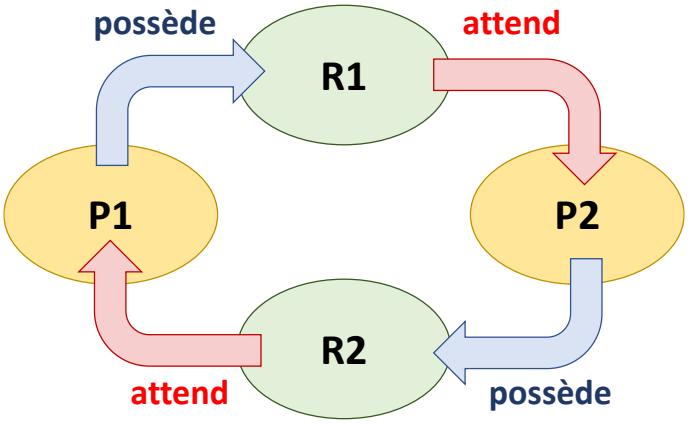
Nous avons vu qu'un processus ne pouvait accéder à une ressource que si celle-ci était disponible, c'est-à-dire si elle n'était pas déjà utilisée par un autre processus. Cela implique que chaque processus doit libérer les ressources qu'il n'utilise plus. Dans certains cas, il peut donc se produire des situations problématiques. Voyons un exemple :

Soient deux processus P1 et P2, et deux ressources appelées R1 et R2. On considère que dans l'état initial le processus P1 détient la ressource R1, et le processus P2 détient la ressource R2.



On suppose qu'à l'état initial, le processus P1 est dans l'état élu, alors que le processus P2 est dans l'état prêt. Le processus P1 s'exécute et, à un moment donné, il a besoin d'avoir accès à la ressource R2. Cette dernière n'étant pas disponible, car possédée par P2, le processus P1 ne peut pas continuer, il passe donc de l'état élu à l'état bloqué. Le processus P2 peut donc démarrer à son tour, et passe de l'état prêt à l'état élu. Il s'exécute, jusqu'au moment où il a besoin d'accéder à la ressource R1. Il ne peut pas le faire (car R1 est possédée par P1) et passe donc dans l'état bloqué.

On se trouve alors dans une situation où P1 et P2 ne peuvent plus être débloqués, car ils attendent une ressource qui ne peut pas être libérée, car possédée par un processus bloqué :



Cette situation s'appelle *l'interblocage* (ou deadlock). Il faut éviter ce cas de figure car d'une part des processus bloqués en permanence ne servent à rien, et en plus les ressources qu'ils monopolisent deviennent inaccessibles pour les autres processus.

Pour éviter un interblocage, on peut essayer de le détecter avant qu'il ne se produise, mais cela n'est pas toujours possible. Une autre solution est de mettre fin à l'un des deux processus, afin de libérer les ressources qu'il détient pour débloquer la situation.

Plus d'informations ici : https://fr.wikipedia.org/wiki/Interblocage