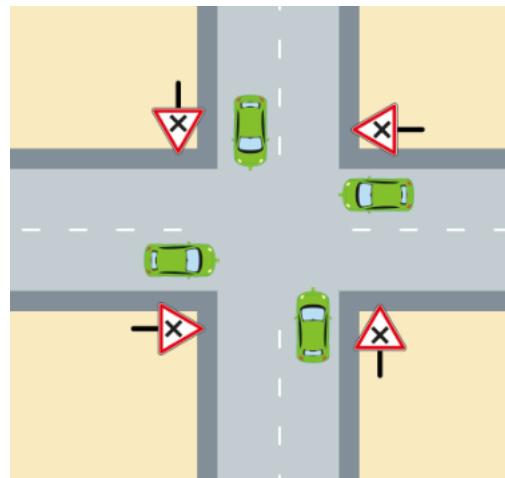


18.3

Les interblocages

NSI TERMINALE - JB DUTHOIT

Les interblocages sont des situations de la vie quotidienne !



Qui doit passer ?

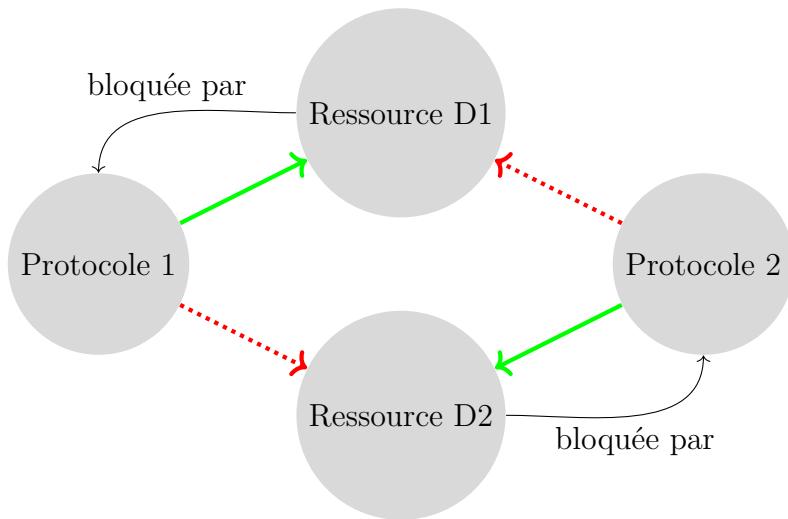
18.3.1 Quels interblocages en informatique ? Exemple de deux processus qui ont besoin de la même donnée de manière exclusive (pour la modifier par exemple)

Su le schéma ci-dessous, P1 et P2 ont tous les deux besoin des ressource D1 et D2
Voici un scénario possible :

- Le processus P1 commence son exécution (état élu), il demande la ressource D1. Il obtient satisfaction puisque D1 est libre
- P1 passe ensuite l'état "prêt".
- Pendant ce temps, le système a passé P2 en exécution : P2 commence son exécution et demande la ressource D2. Il obtient immédiatement D2 puisque cette ressource était libre.
- P2 poursuit son exécution
- P2 demande la ressource D1, il se retrouve dans un état bloqué puisque la ressource D1 a été attribuée à P1
- P1 est dans l'état prêt, il n'a pas eu l'occasion de libérer la ressource R1 puisqu'il n'a pas eu l'occasion d'utiliser R1
- P1 passe donc à l'état d'exécution . Afin de libérer D1, P1 a besoin de D2. Problème, car D2 n'est pas disponible ! Donc P1 ne peut pas libérer D1 et passe à bloqué.

Résumons la situation à cet instant : P1 possède la ressource D1 et se trouve dans l'état bloqué, P2 possède la ressource D2 et se trouve dans l'état bloqué (attente de D1)

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



Face à cette problématique, deux situations sont envisageables :

- Essayer d'éviter les interblocages.
- Déetecter quand un interblocage est apparu, et le solutionner

☞ La plupart des systèmes d'exploitation ont choisi de ne pas essayer de les éviter, mais plutôt de détecter et de les solutionner.

18.4

Programmation concurrente en Python

NSI TERMINALE - JB DUTHOIT

Un **tread** est un mini-processus démarré par un processus et s'exécutant de manière concurrente avec le reste du programme.

On utilise pour cela le module **threading** de la bibliothèque Python.

18.4.1 Un premier programme

Recopier et exécuter ce programme :

```
from time import *
from threading import *

def hello(n):
    for i in range(5):
        print(f"je suis le thread {n} et ma valeur est {i}")
    print(f'---fin du thread {n}')

th = []
for n in range(4):
    t = Thread(target = hello, args=[n])
    t.start() #On démarre le thread
    th.append(t)

for t in th:
    t.join() #On attend que le thread s'arrête
```

- ☛ Deux exécutions successives donnent deux affichages différents.
- ☛ L'ordre dans lequel sont démarré les threads ne donne pas d'indications sur l'ordre dans lequel ils se terminent...

18.4.2 Attention aux threads

Un premier programme sans surprise :

```
compteur = 0 # Variable globale
limite = 1000000

def calcul():
    global compteur
    for c in range(limite):
        temp = compteur
        # simule un traitement nécessitant des calculs
        compteur = temp + 1
print(compteur)
```

- ☛ Sans surprise, on a $\text{compteur} = 1000000\dots$

Utilisons maintenant le **threading** :

```

compteur = 0 # Variable globale
limite = 1000000

def calcul():
    global compteur
    for c in range(limite):
        temp = compteur
        # simule un traitement nécessitant des calculs
        compteur = temp + 1

th = []
compteur = 0
for i in range(4): # Lance en parallèle 4 exécutions du calcul
    p = Thread(target = calcul, args= [])
    p.start()          # Lance calcul dans un processus léger à part.
    th.append(p)

for t in th:
    t.join()

```

- On pourrait s'attendre à avoir `compteur = 4 × 1000000`, soit 4000000..mais non !! On obtient des résultats inférieurs à 4000000 et différents en fonction des exécutions...

Pour expliquer ce phénomène , supposons que t_0 soit en exécution avec `compteur = 42`. si t_0 est interrompu juste après avoir exécuté `c = compteur`, alors sa variable local `v` contient la valeur 42. C'est t_1 qui prend maintenant la main en exécutant tout d'abord `v = compteur` suivi de `compteur = v + 1`. La variable local `v` contient 42 et `compteur` contient donc 43. Si le thread t_0 reprend la main, il va continuer là où il s'est arrêté! t_0 exécute donc `compteur = v + 1`, où la valeur de `v` est 42 ! Le compteur repasse donc à 42 (au lieu de 44 ici) !

18.4.3 Les verrous

Pour corriger ce problème, on utilise des verrous pour garantir l'accès exclusif pour les deux lignes `v = compteur` et `compteur = v + 1`.

Un verrou est un objet que l'on peut essayer d'acquérir ! Pour l'acquérir, il suffit que personne ne détient le verrou.

 Si quelqu'un d'autre détient le verrou, alors on est bloqué :-(

```

compteur = 0 # Variable globale
limite = 1000000

verrou = Lock()

def calcul():
    global compteur
    for c in range(limite):
        verrou.acquire()

```

```

temp = compteur
# simule un traitement nécessitant des calculs
compteur = temp + 1
verrou.release()

th = []
compteur = 0
for i in range(4): # Lance en parallèle 4 exécutions de calcul
    p = Thread(target = calcul, args= [])
    p.start()          # Lance calcul dans un processus léger à part.
    th.append(p)

for t in th:
    t.join()

```

18.4.4 Interblocage

```

from threading import *

verrou1 = Lock()
verrou2 = Lock()

def f():
    for i in range(100):
        verrou1.acquire()
        print("section verrouillée f1")
        verrou2.acquire()
        print("section verrouillée f2")
        verrou2.release()
        verrou1.release()

def g():
    for i in range(100):
        verrou2.acquire()
        print("section verrouillée g2")
        verrou1.acquire()
        print("section verrouillée g1")
        verrou1.release()
        verrou2.release()

t1 = Thread(target = f, args=[])
t2 = Thread(target = g, args=[])
t1.start()
t2.start()
t1.join()
t2.join()

print("fin")

```

Exercice 18.181

► Exercice tiré du sujet Métropole 2021 (candidats libres)

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 S1 Gedit
2013 1912 Ss Bash
2091 1593 S1 /usr/lib/firefox/firefox
5437 1912 S1 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 ?

Exercice 18.182

► Exercice tiré du sujet Métropole 2021 (candidats libres)

Partie A :

Cette partie est un questionnaire à choix multiples (QCM). Pour chacune des questions, une seule des quatre réponses est exacte. Le candidat indiquera sur sa copie le numéro de la question et la lettre correspondant à la réponse exacte.

Aucune justification n'est demandée. Une réponse fausse ou une absence de réponse n'enlève aucun point.

1. Parmi les commandes ci-dessous, laquelle permet d'afficher les processus en cours d'exécution ?
 - a) dir
 - b) ps
 - c) man
 - d) ls
2. Quelle abréviation désigne l'identifiant d'un processus dans un système d'exploitation de type UNIX ?

- a) PIX
 b) SIG
 c) PID
 d) SID
3. Comment s'appelle la gestion du partage du processeur entre différents processus ?
 a) L'interblocage
 b) L'ordonnancement
 c) La planification
 d) La priorisation
4. Quelle commande permet d'interrompre un processus dans un système d'exploitation de type UNIX ?
 a) stop
 b) interrupt
 c) end
 d) kill

Partie B

1. Un processeur choisit à chaque cycle d'exécution le processus qui doit être exécuté. Le tableau ci-dessous donne pour trois processus P1, P2, P3 :
- la durée d'exécution (en nombre de cycles),
 - l'instant d'arrivée sur le processeur (exprimé en nombre de cycles à partir de 0),
 - le numéro de priorité.

Le numéro de priorité est d'autant plus petit que la priorité est grande. On suppose qu'à chaque instant, c'est le processus qui a le plus petit numéro de priorité qui est exécuté, ce qui peut provoquer la suspension d'un autre processus, lequel reprendra lorsqu'il sera le plus prioritaire.

Processus	Durée d'exécution	Instant d'arrivée	Numéro de priorité
P1	3	3	1
P2	3	2	2
P3	4	0	3

Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle.



2. On suppose maintenant que les trois processus précédents s'exécutent et utilisent une ou plusieurs ressources parmi R1, R2 et R3.
 Parmi les scénarios suivants, lequel provoque un interblocage ? Justifier

Scénario 1	Scénario 2	Scénario 3
P1 acquiert R1	P1 acquiert R1	P1 acquiert R1
P2 acquiert R2	P2 acquiert R3	P2 acquiert R2
P3 attend R1	P3 acquiert R2	P3 attend R2
P2 libère R2	P1 attend R2	P1 attend R2
P2 attend R1	P2 libère R3	P2 libère R2
P1 libère R1	P3 attend R1	P3 acquiert R2

Partie C :

Pour chiffrer un message, une méthode, dite du masque jetable, consiste à le combiner avec une chaîne de caractères de longueur comparable. Une implémentation possible utilise l'opérateur XOR (ou exclusif) dont voici la table de vérité :

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Dans la suite, les nombres écrits en binaire seront précédés du préfixe 0b.

1. Pour chiffrer un message, on convertit chacun de ses caractères en binaire (à l'aide du format Unicode), et on réalise l'opération XOR bit à bit avec la clé.

Après conversion en binaire, et avant que l'opération XOR bit à bit avec la clé n'ait été effectuée, Alice obtient le message suivant :

$$m = 0b\ 0110\ 0011\ 0100\ 0110$$

- a) Le message m correspond à deux caractères codés chacun sur 8 bits : déterminer quels sont ces caractères. On fournit pour cela la table ci-dessous qui associe à l'écriture hexadécimale d'un octet le caractère correspondant (figure 2).
Exemple de lecture : le caractère correspondant à l'octet codé 4A en hexadécimal est la lettre J.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- b) Pour chiffrer le message d'Alice, on réalise l'opération XOR bit à bit avec la clé suivante : $k = 0b\ 1110\ 1110\ 1111\ 0000$
Donner l'écriture binaire du message obtenu.
2. a) Dresser la table de vérité de l'expression booléenne suivante : $(a \text{ XOR } b) \text{ XOR } b$
b) Bob connaît la chaîne de caractères utilisée par Alice pour chiffrer le message. Quelle opération doit-il réaliser pour déchiffrer son message ?

Exercice 18.183

► Exercice tiré du sujet Métropole 2021 (sujet 2)

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.

- Les processus s'exécutent de manière concurrente. Justifier qu'une situation d'interblocage peut se produire.
- Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire. Aucune justification n'est attendue.
- 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 :
 - élu
 - bloqué
 - prêt
 - 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-quic --enable-tcp-fast-open --ppapi-flash-path=/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib/chromium-browser/chromium-browser-v7 --type=zygote
pi	6252	6211	0	09:07 ?		00:00:00	
pi	6254	6252	0	09:07 ?		00:00:00	
pi	6294	6211	4	09:07 ?		00:00:40	/usr/lib/chromium-browser/chromium-browser-v7 --type=gpu-process --field-trial-handle=10758
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.odt
pi	12253	831	1	09:13 ?		00:00:07	/usr/bin/python3 /usr/bin/sense_emu_gui
pi	20018	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 ?