Les Signaux

Version 1

Pascal Malingrey

Académie Strasbourg

26 juin 2019

STOP ou ENCORE

Vous disposez dans votre dossier d'un fichier **ex1.py** (son contenu n'a pas d'importance pour le moment.) Dans un terminal vous exécuter la commande suivante :

terminal

```
nsi@lin$ python3 ex1.py
```

1. Que constatez vous ?

STOP ou ENCORE

Vous disposez dans votre dossier d'un fichier **ex1.py** (son contenu n'a pas d'importance pour le moment.) Dans un terminal vous exécuter la commande suivante :

terminal

```
nsi@lin$ python3 ex1.py
```

1. Que constatez vous ?
2. Comment interrompre le programme , tout en gardant le terminal actif ?

STOP ou ENCORE

Vous disposez dans votre dossier d'un fichier **ex1.py** (son contenu n'a pas d'importance pour le moment.) Dans un terminal vous exécuter la commande suivante :

terminal

```
nsi@lin$ python3 ex1.py
```

1. Que constatez vous ?
2. Comment interrompre le programme , tout en gardant le terminal actif ?
3. On va appuyer sur une combinaison de touche `Ctrl+C`

Que s'est-il passé ?

La combinaison de touche `Ctrl+C` a interrompu l' exécution du programme. Cela signifie deux choses :

- "quelque chose" est à l'écoute du clavier
- le programme traite l'information de la combinaison `Ctrl+C`

Commande shell

La commande `cat` est utilisée qu'à titre d'exemple, sa fonctionnalité n'est pas importante ici. Par contre les suivantes sont utiles :

- `ps` : liste les processus dans le terminal
- `^z` : Ctrl+z suspend un processus
- `fg` : reprend un processus

Dans un terminal

Les commandes

terminal

```
nsi@lin$ cat
nsi@lin$ ^z  #Appui Ctrl+z
nsi@lin$ ps
```

Dans un terminal

Les commandes

terminal

```
nsi@lin$ cat
nsi@lin$ ^z  #Appui Ctrl+z
nsi@lin$ ps
```

Le résultat

terminal

```
PID TTY          TIME CMD
25280 pts/0    00:00:00 bash
26623 pts/0    00:00:00 cat
26624 pts/0    00:00:00 ps
```

Dans un terminal

Les commandes

terminal

```
nsi@lin$ cat
nsi@lin$ ^z  #Appui Ctrl+z
nsi@lin$ ps
```

Le résultat

terminal

```
PID TTY          TIME CMD
25280 pts/0    00:00:00 bash
26623 pts/0    00:00:00 cat
26624 pts/0    00:00:00 ps
```

On voit trois processus

- le bash (qui correspond à notre terminal)
- cat (qu'on a lancé)
- ps (qui liste nos processus)

Remarque : Vous constatez que les numéros de PID ne sont pas les mêmes chez vous.

Stoppons le processus cat

Nous allons envoyer un signal de terminaison (comparable au Ctrl-C) au processus **cat** par le biais de la commande **killall**. killall envoie un signal aux processus dont le nom est indiqué avec le numéro du signal.
Regardons ce qu'il en est des processus

Les commandes

terminal

```
nsi@lin$ killall -2 cat
nsi@lin$ ps
```

Stoppons le processus cat

Nous allons envoyer un signal de terminaison (comparable au Ctrl-C) au processus **cat** par le biais de la commande **killall**. killall envoie un signal aux processus dont le nom est indiqué avec le numéro du signal.
Regardons ce qu'il en est des processus

Les commandes

terminal

```
nsi@lin$ killall -2 cat
nsi@lin$ ps
```

Le résultat

terminal

```
PID TTY          TIME CMD
25280 pts/0    00:00:00 bash
26623 pts/0    00:00:00 cat
26624 pts/0    00:00:00 ps
```

Aucune différence !!

Reprise du processus

On va demander une reprise du processus avec la commande `fg`

Les commandes

terminal

```
nsi@lin$ fg
nsi@lin$ ps
```

Reprise du processus

On va demander une reprise du processus avec la commande `fg`

Les commandes

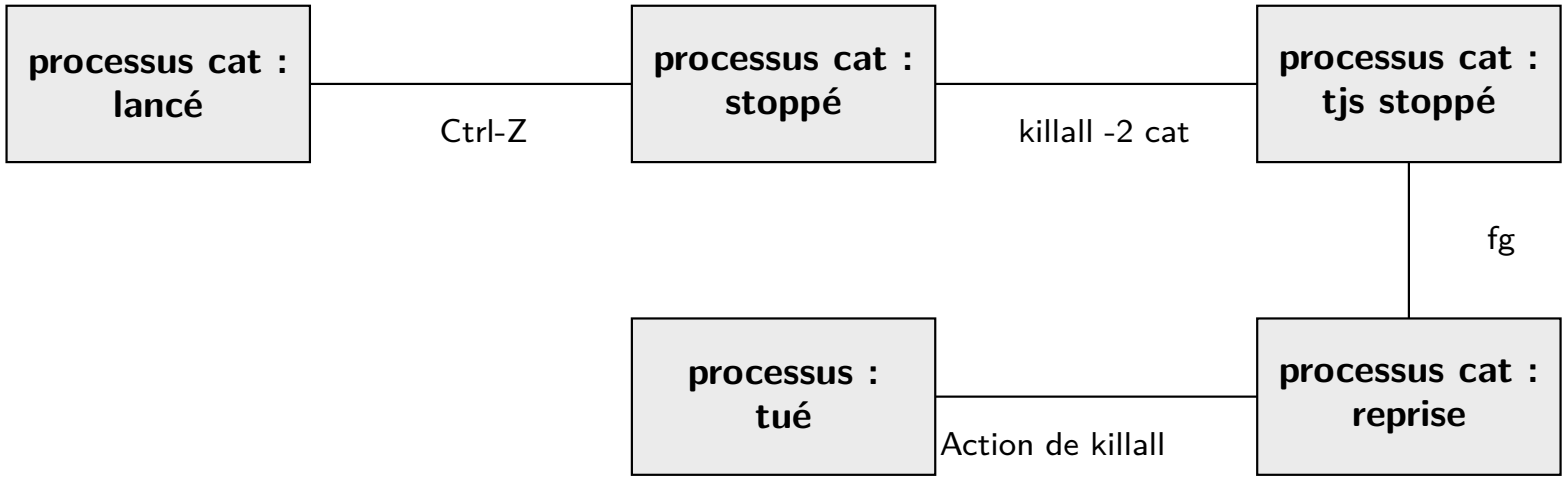
terminal

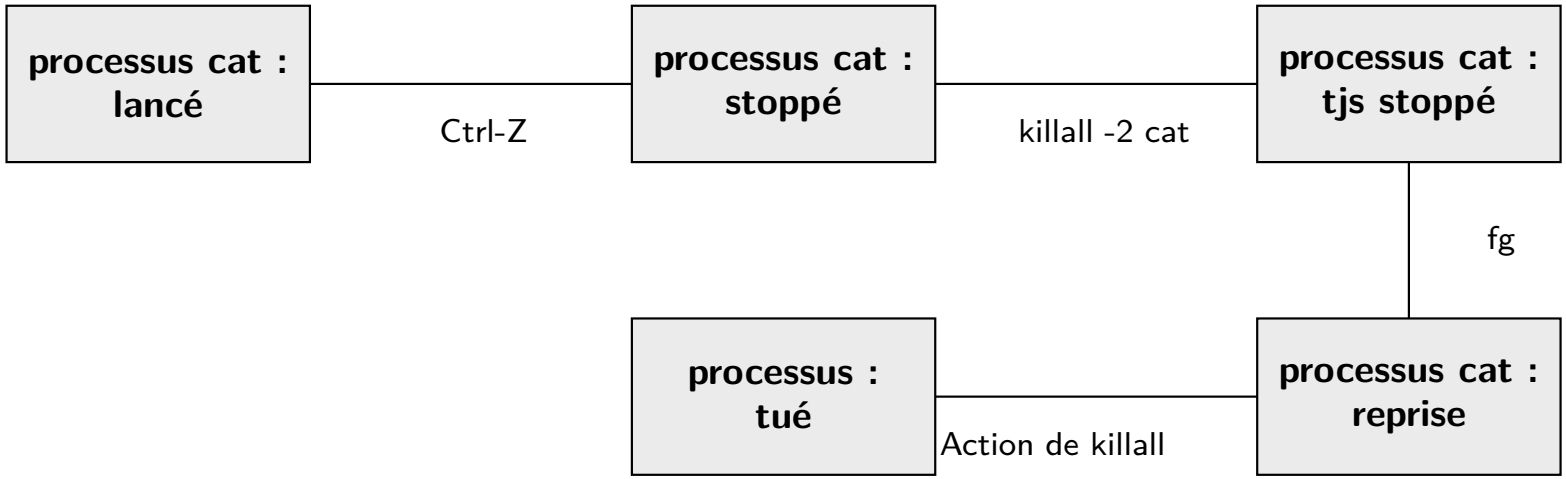
```
nsi@lin$ fg
nsi@lin$ ps
```

Le résultat

terminal

```
PID TTY          TIME CMD
25280 pts/0    00:00:00 bash
26624 pts/0    00:00:00 ps
```


processus cat : lancé
Ctrl-Z
processus cat : stoppé
killall -2 cat
processus cat : tjs stoppé
fg
processus cat : reprise
Action de killall
processus : tué



Conclusion

Suite à la reprise du processus, le signal **Ctrl-C** a été exécuté.
Le signal d'interruption n'est traité que lorsque le processus cat redevient actif, c'est donc bien le processus qui traite l'information du signal **2** , dont il est destinataire.
Cela signifie également que le signal **2** a été mémorisé par le système.

Définition d'un signal

Definition 1

Un signal est :

- un message envoyé par le noyau de manière *asynchrone* à :
  - un processus ; ou
  - un groupe de processus
- pour indiquer un événement système important

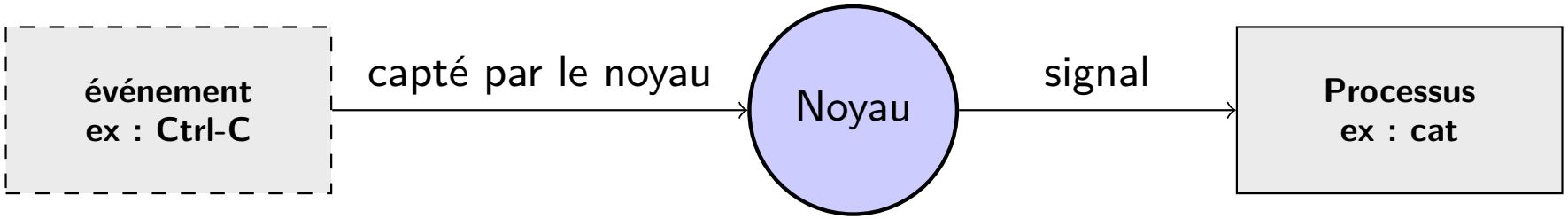
Définition d'un signal

Definition 1

Un signal est :

- un message envoyé par le noyau de manière *asynchrone* à :
  - un processus ; ou
  - un groupe de processus
- pour indiquer un événement système important

Le message peut être à l'initiative d'un autre processus. Cette communication limitée **(seul le numéro du signal est envoyé)** entre les processus. La norme POSIX définit un certain nombre de signaux, environ une vingtaine. (voir 3.3)



Quelques événements possibles

- frappe de caractère dans un terminal

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`
  - `Ctrl+Z` etc.

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`
  - `Ctrl+Z` etc.
- terminaison d'un processus

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`
  - `Ctrl+Z` etc.
- terminaison d'un processus
- un problème matériel : division par zéro, problème d'adressage, défaillance d'alimentation électrique, etc.

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`
  - `Ctrl+Z` etc.
- terminaison d'un processus
- un problème matériel : division par zéro, problème d'adressage, défaillance d'alimentation électrique, etc.
- l'expiration de délai préprogrammé (fonction alarm())

Quelques événements possibles

- frappe de caractère dans un terminal
  - `Ctrl+C`
  - `Ctrl+Z` etc.
- terminaison d'un processus
- un problème matériel : division par zéro, problème d'adressage, défaillance d'alimentation électrique, etc.
- l'expiration de délai préprogrammé (fonction alarm())
- ...

Lister les signaux

Exécuter la commande ci-dessous pour avoir la liste des signaux disponibles

terminal

```
nsi@lin$ killall -l
```

La norme POSIX distingue 32 signaux dont les principaux sont :

Numéro	Nom	Signification	Comportement
1	SIGHUP	Hang-up (fin de connexion)	T(erminaison)
2	SIGINT	Interruption (Ctrl-C)	T
	SIGQUIT	Interruption forte (Ctrl-\)	T + core
	SIGFPE	Erreur arithmétique	T + core
9	SIGKILL	Interruption immédiate et absolue	T + core
	SIGSEGV	Violation des protections mémoire	T + core
	SIGPIPE	Écriture sur un pipe sans lecteurs	T
20	SIGTSTP	Arrêt temporaire(Ctrl-Z)	Suspension
18	SIGCONT	Redémarrage d'un fils arrêté	Ignoré
	SIGCHLD	un des fils est mort ou arrêté	Ignoré
14	SIGALRM	Interruption d'horloge	Ignoré
19	SIGSTOP	Arrêt temporaire	Suspension
	SIGUSR1	Émis par un processus utilisateur	T
	SIGUSR2	Émis par un processus utilisateur	T

T : terminaison du processus ; core : création d'un fichier d'image mémoire

Remarque

Quelle différence entre SIGSTP et SIGSTOP, de même entre SIGINT et SIGKILL ?

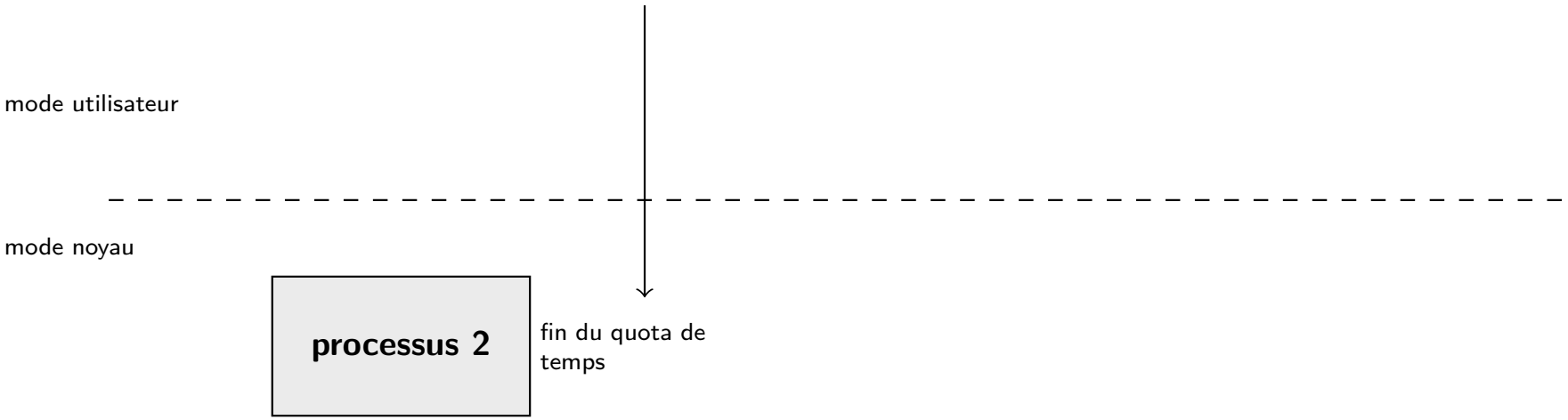
Certains signaux ne peuvent être bloqués ou ignorés par le processus, c'est le cas de de SIGSTOP et SIGKILL.

- SIGINT/SIGSTP laisse la possibilité au processus d'ignorer et/ou de contrôler le signal
- SIGKILL/SIGSTOP aucun moyen de passer outre la demande d'interruption.

Pour la liste de toutes les actions des signaux :
http://www.linux-france.org/article/man-fr/man7/signal-7.html

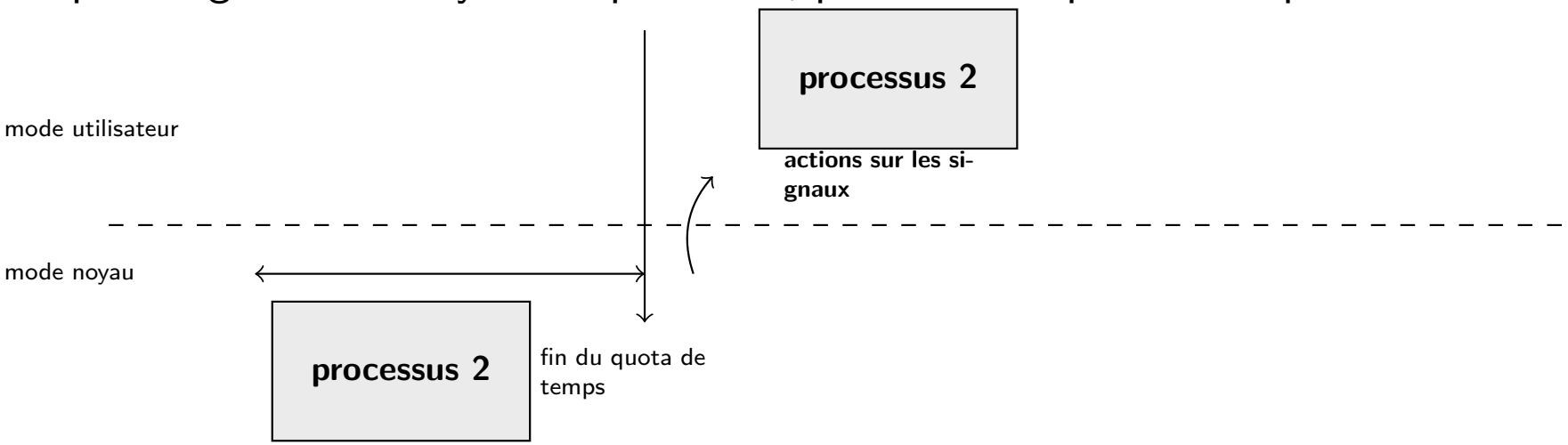
Résumé

La prise en compte d'un signal (on parle de **délivrance**) ne peut avoir lieu que dans une circonstance bien particulière : la bascule du mode système au mode utilisateur.
Lorsqu'un signal est envoyé à un processus, plusieurs cas peuvent se produire :



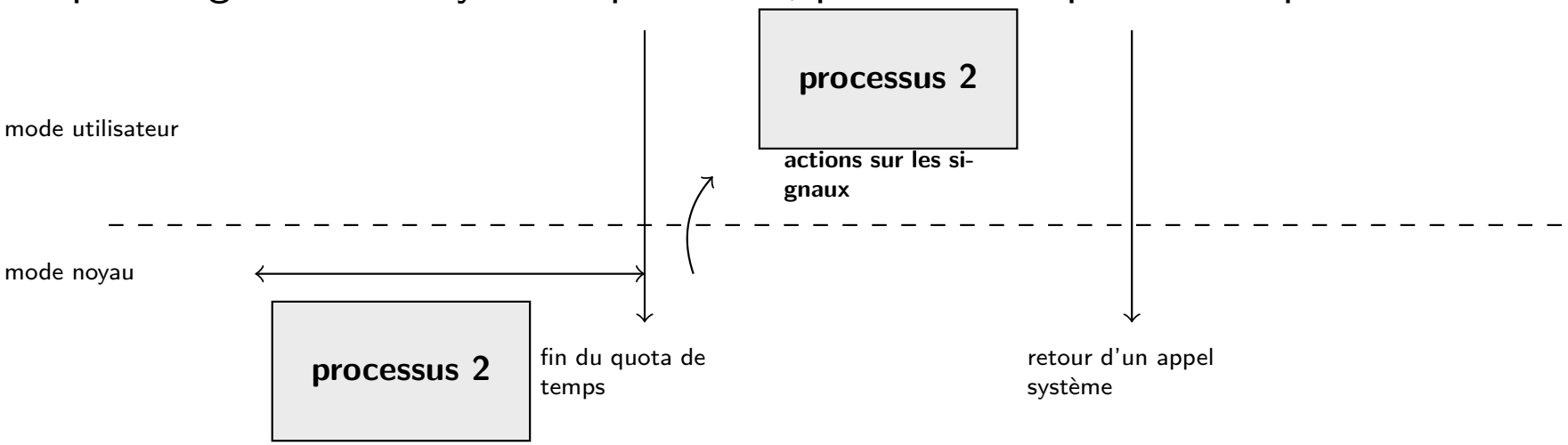
Résumé

La prise en compte d'un signal (on parle de **délivrance**) ne peut avoir lieu que dans une circonstance bien particulière : la bascule du mode système au mode utilisateur.
Lorsqu'un signal est envoyé à un processus, plusieurs cas peuvent se produire :



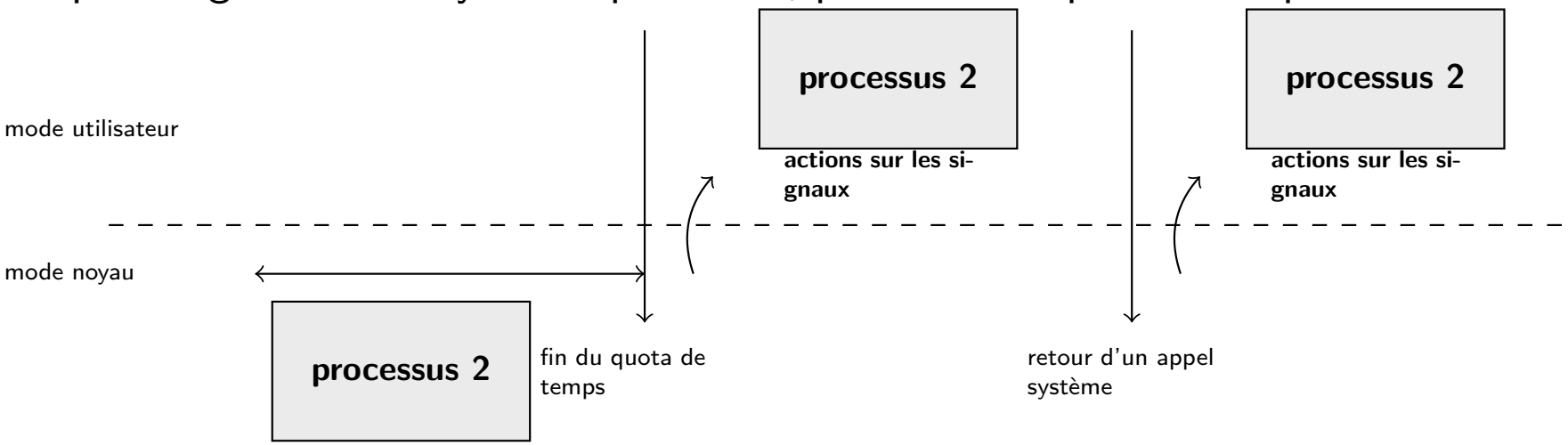
Résumé

La prise en compte d'un signal (on parle de **délivrance**) ne peut avoir lieu que dans une circonstance bien particulière : la bascule du mode système au mode utilisateur.
Lorsqu'un signal est envoyé à un processus, plusieurs cas peuvent se produire :



Résumé

La prise en compte d'un signal (on parle de **délivrance**) ne peut avoir lieu que dans une circonstance bien particulière : la bascule du mode système au mode utilisateur.
Lorsqu'un signal est envoyé à un processus, plusieurs cas peuvent se produire :



Résumé

- Le processus est en mode utilisateur. La délivrance devra alors attendre d'abord le passage du processus en mode noyau puis son retour au mode utilisateur. Pendant tout ce temps, le signal sera **pendant**, c'est à dire en attente de délivrance.

Résumé

- Le processus est en mode utilisateur. La délivrance devra alors attendre d'abord le passage du processus en mode noyau puis son retour au mode utilisateur. Pendant tout ce temps, le signal sera **pendant**, c'est à dire en attente de délivrance.
- Le processus est en mode noyau, par définition non interruptible. Le signal sera délivré dès que le processus reviendra au mode utilisateur : le signal est donc **pendant**. Lorsque le signal est délivré, la procédure qui lui est associée (son gestionnaire ou **handler**) est appelée. L'action du signal peut être :
  1. le comportement par défaut (par exemple l'interruption)
  2. l'ignorance
  3. le traitement personnalisé
  4. le masquage (blocage)

Etats d'un signal

Definition 2

Donc les différents états d'un signal sont :

généré/émis : L'événement associé au signal s'est produit

Etats d'un signal

Definition 2

Donc les différents états d'un signal sont :

généré/émis : L'événement associé au signal s'est produit

délivré : L'action associée au signal a été exécutée

Etats d'un signal

Definition 2

Donc les différents états d'un signal sont :

- **généré/émis :** L'événement associé au signal s'est produit
- **délivré :** L'action associée au signal a été exécutée
- **pendant :** Le signal émis n'a pas encore été pris en compte

Etats d'un signal

Definition 2

Donc les différents états d'un signal sont :

- **généré/émis :** L'événement associé au signal s'est produit
- **délivré :** L'action associée au signal a été exécutée
- **pendant :** Le signal émis n'a pas encore été pris en compte
- **bloqué/masqué :** La prise en compte du signal est volontairement différée.

Compléments

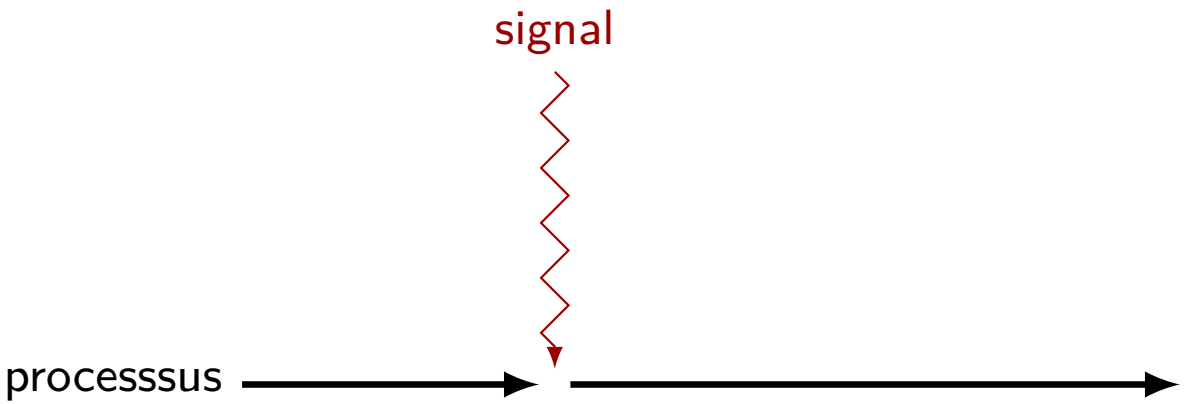
La structure de données interne (une par processus) gérant les signaux est un vecteur indexé sur les numéros de signaux et dont chaque case comporte 3 informations :

- Un **booléen** indiquant si le signal est pendant.Cette information est un booléen unique. Ce qui signifie que si un processus a déjà un signal d'un certain type pendant, il est inutile de lui envoyer à nouveau un signal du même type, celui-ci sera ignoré.
- Un **booléen** indiquant si les signaux de ce type sont bloqués.
- Un pointeur désignant le gestionnaire.

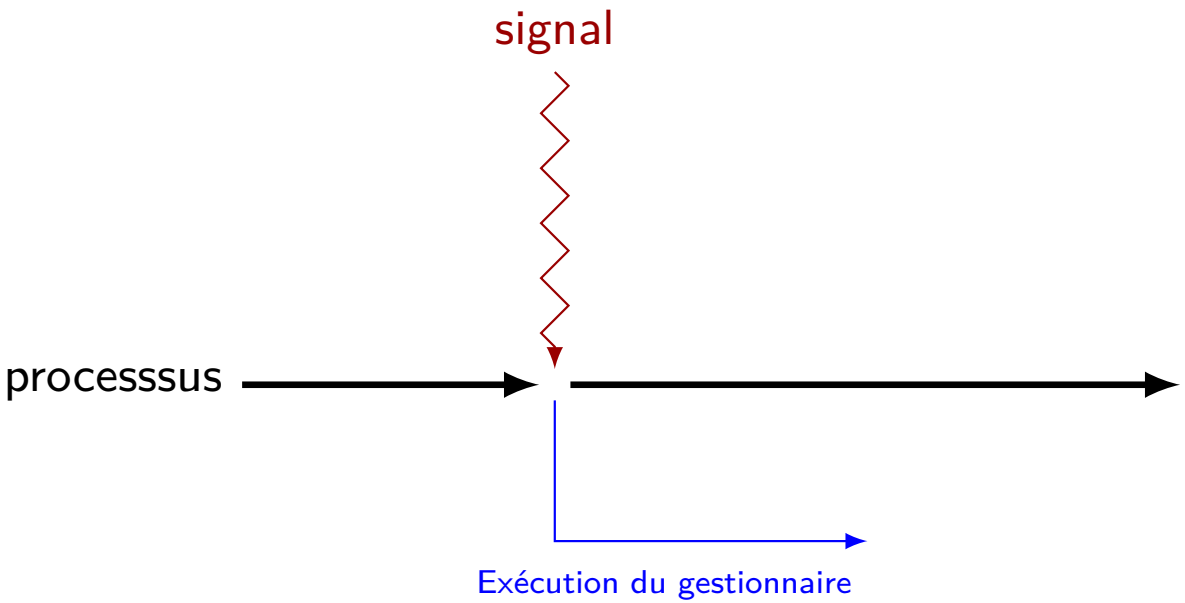
Le gestionnaire

processsus 

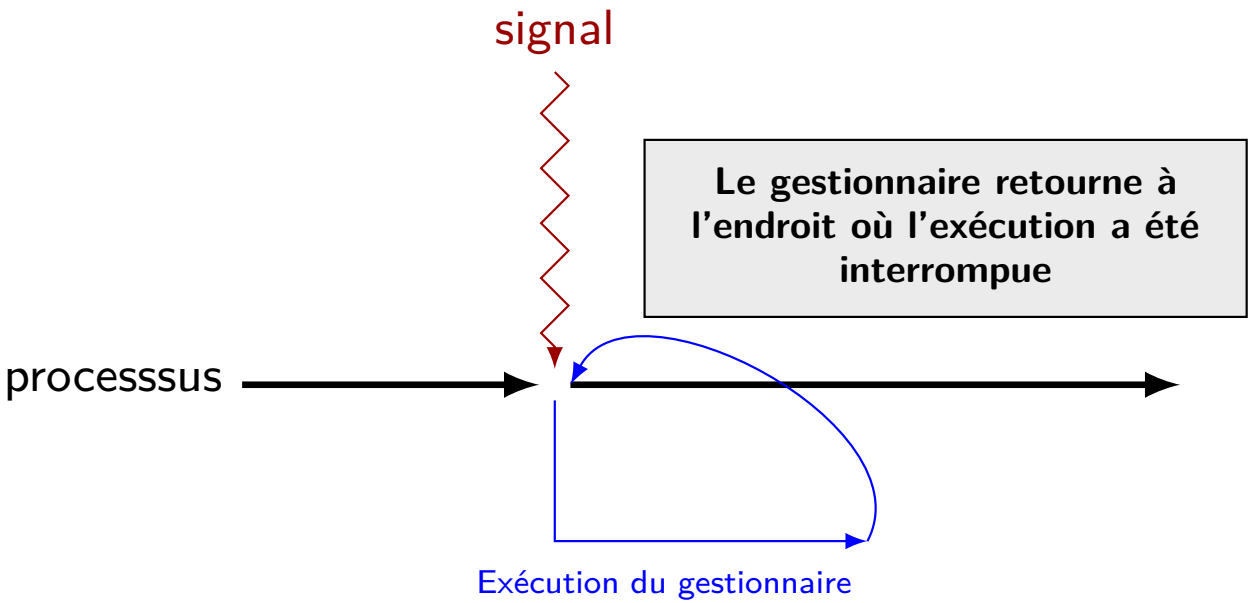
Le gestionnaire



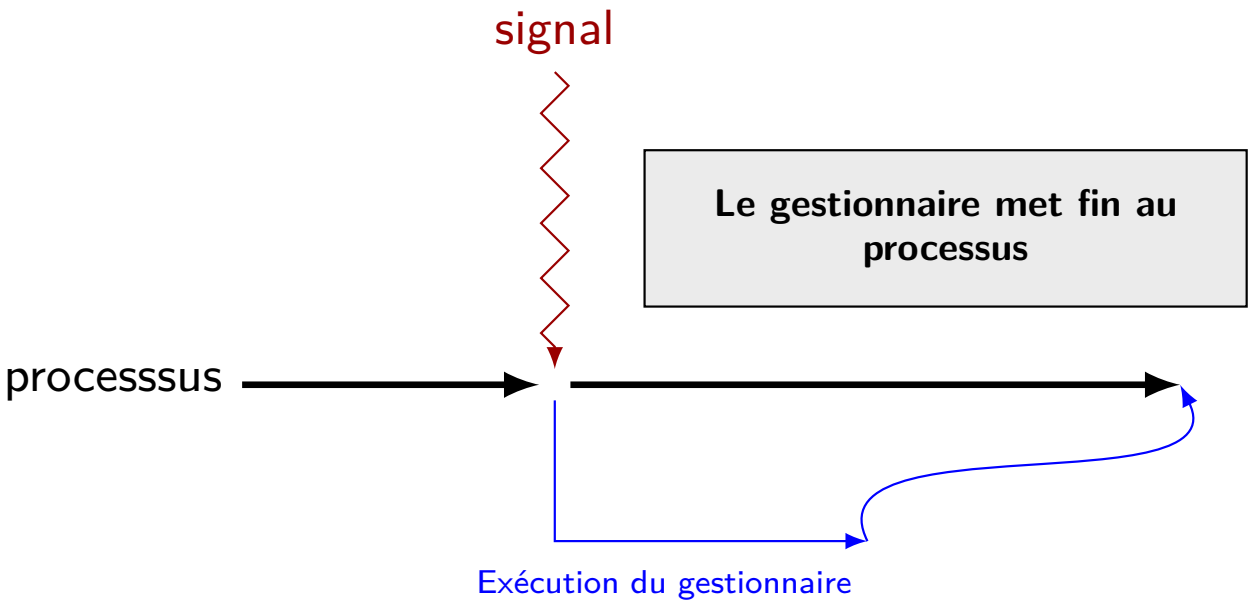
Le gestionnaire



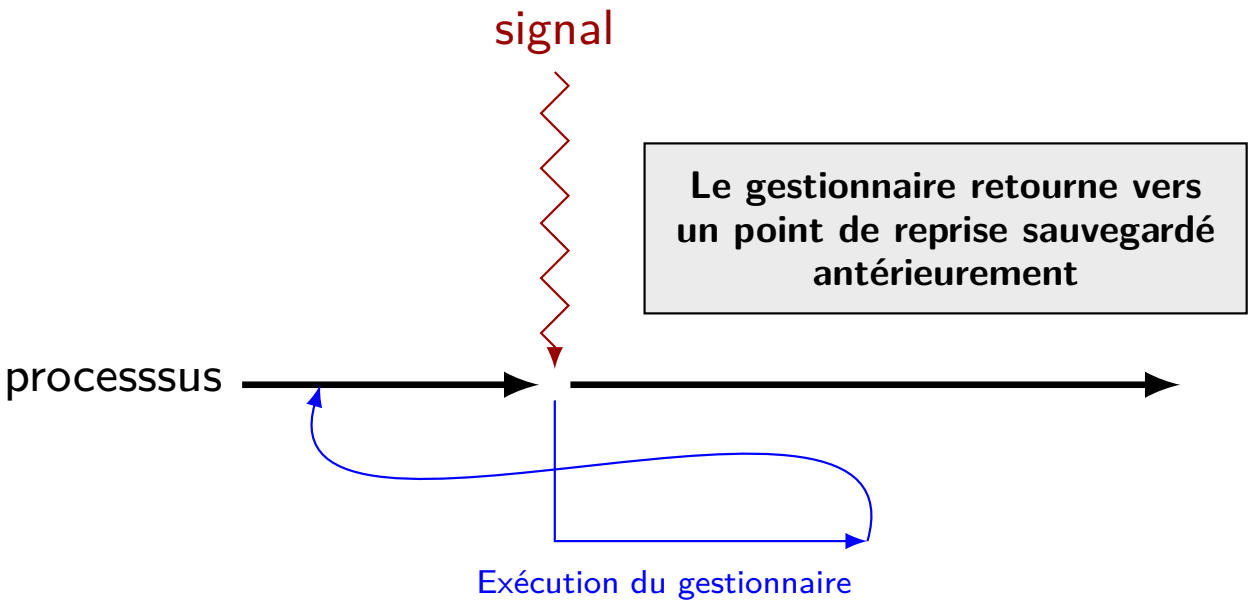
Le gestionnaire



Le gestionnaire

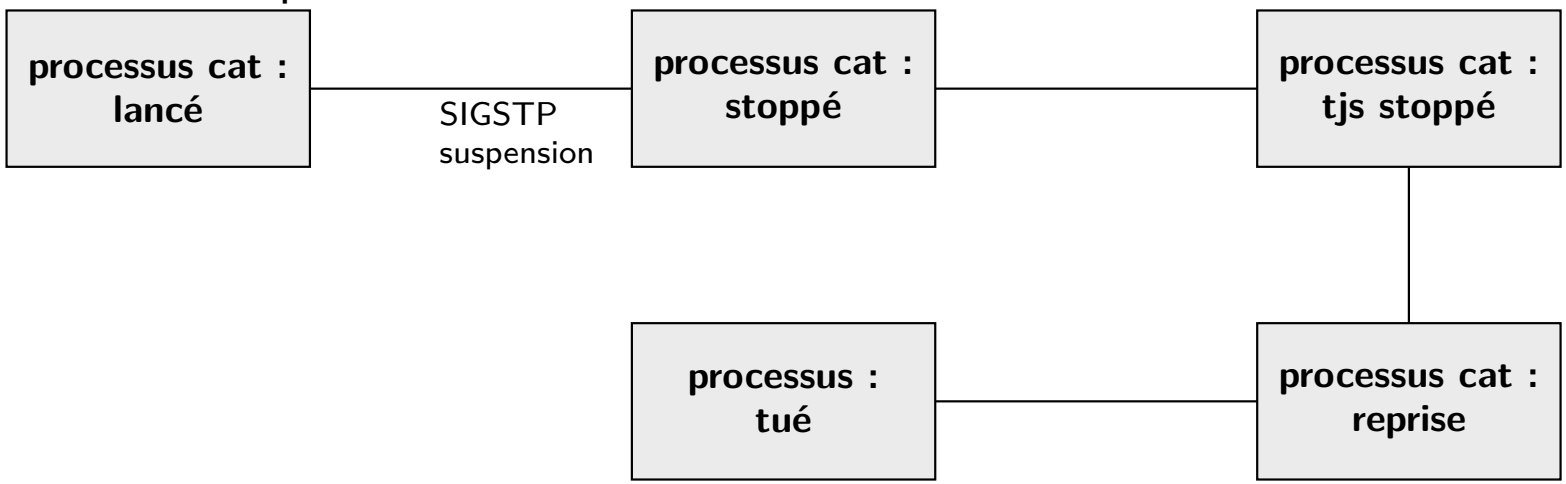


Le gestionnaire



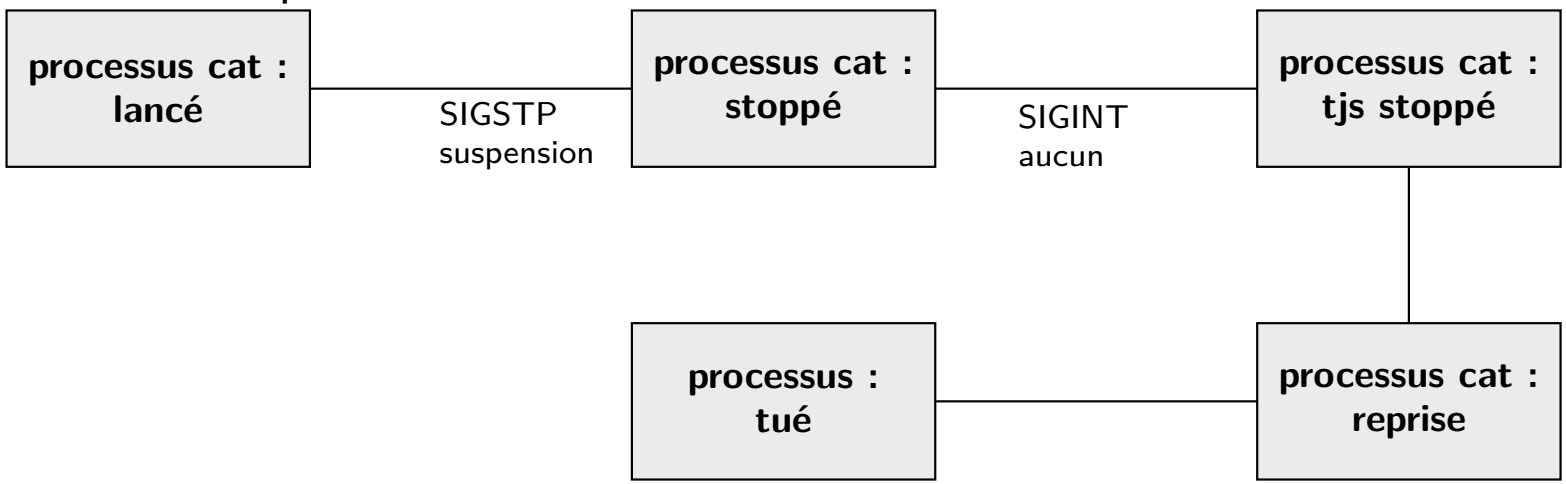
Application

Reprenons l'approche 2.
Indiquez le nom (dans la nomemclature POSIX) des signaux envoyés et les actions entre chacune des phases.



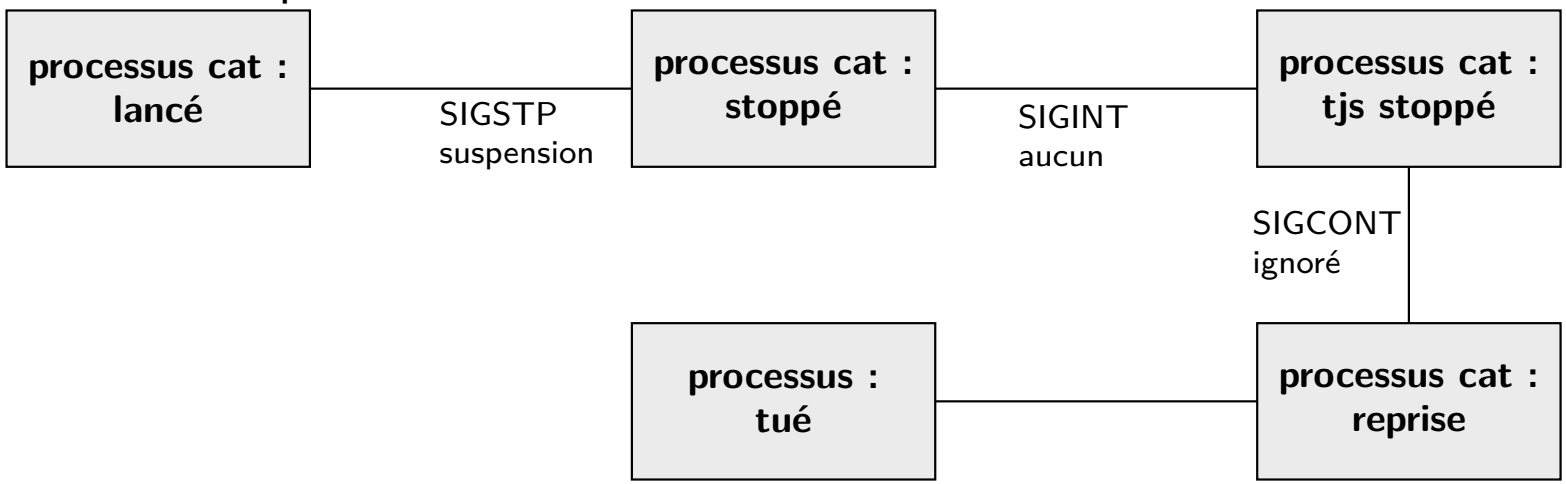
Application

Reprenons l'approche 2.
Indiquez le nom (dans la nomemclature POSIX) des signaux envoyés et les actions entre chacune des phases.



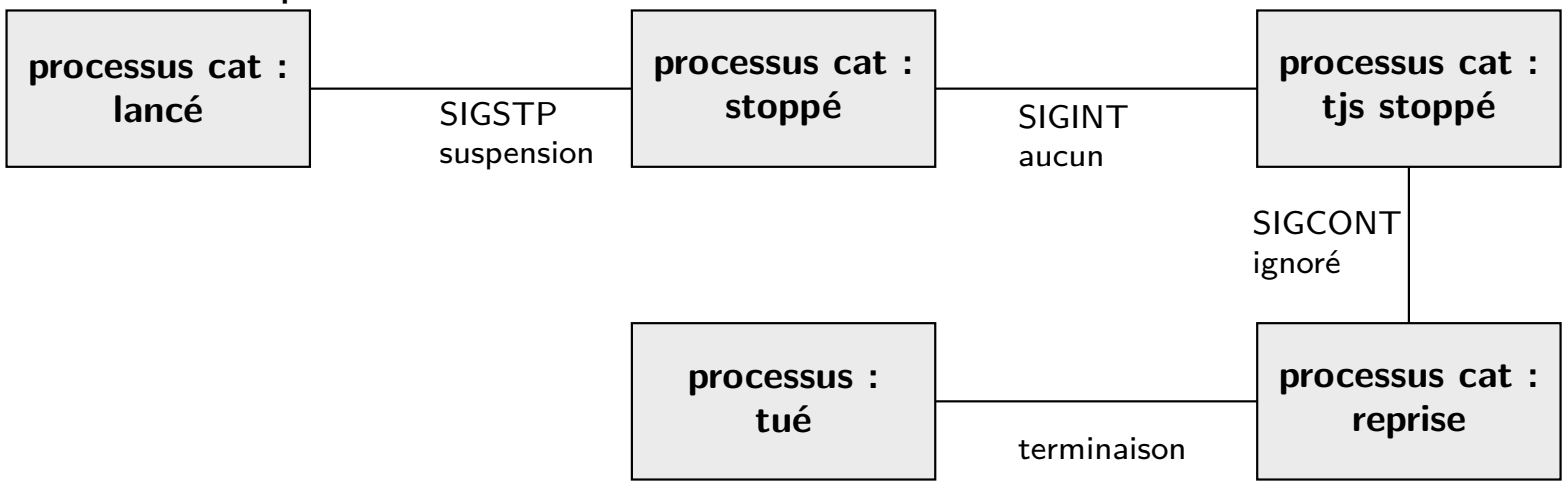
Application

Reprenons l'approche 2.
Indiquez le nom (dans la nomemclature POSIX) des signaux envoyés et les actions entre chacune des phases.



Application

Reprenons l'approche 2.
Indiquez le nom (dans la nomemclature POSIX) des signaux envoyés et les actions entre chacune des phases.



Retour à la programmation

Nous avons vu dans la première partie que certains signaux peuvent être interceptés par le processus.
Regardons comment faire à l'aide de Python

Les signaux avec Python

- **signal :** la librairie `signal` permet de travailler avec les signaux qui peuvent être modifiés (SIGINT, SIGTERM, SIGSTP, SIGALRM...)
- **signal.signal(monsignal, mafonction) :** permet d'exécuter la fonction **mafonction**, lors de l'apparition du signal **monsignal**. Il s'agit du gestionnaire (handler) du signal.
- **signal.alarm(t) :** envoi un signal alarm après un délai de *t* secondes.
- **signal.SIG... :** permet de faire référence au signal SIG... (en fait on récupère le numéro du signal).
  par exemple signal.SIGTERM fait réféfence au Ctrl-C.

Exerice 1

Exercice 1

*Reprenez le fichier **ex1.py** et le modifier pour que le signal SIGINT(Ctrl-C) ne puisse pas interrompre le programme.*

Exerice 1

Exercice 1

*Reprenez le fichier **ex1.py** et le modifier pour que le signal SIGINT(Ctrl-C) ne puisse pas interrompre le programme.*

ex1b.py

```
import signal
import time

def mongestionnaire(signum,stack):
    print("Ctrl-C est désactivé")

signal.signal(signal.SIGINT, mongestionnaire)

i = 1
while 1:
    print("itération :{}".format(i))
    time.sleep(1)
    i += 1
```

Exercice 2

Voici un programme :

```
import signal
import time

def mongestionnaire(signum, stack):
    print('Alarme :', time.ctime())

signal.signal(signal.SIGALRM, mongestionnaire)
signal.alarm(2)

print('Première:', time.ctime())
time.sleep(4)
print('Terminale :', time.ctime())
```

1. *Expliquer les commandes des lignes 6 et 7.*

Exercice 2

Voici un programme :

```
import signal
import time

def mongestionnaire(signum, stack):
    print('Alarme :', time.ctime())

signal.signal(signal.SIGALRM, mongestionnaire)
signal.alarm(2)

print('Première:', time.ctime())
time.sleep(4)
print('Terminale :', time.ctime())
```

1. *Expliquer les commandes des lignes 6 et 7.*
   Ligne 6 : on associe au signal SIGALRM, la fonction mongestionnaire comme handler.
   Ligne 7 : on envoie le signal SIGALRM au processus actuel avant un temps de 2 secondes d'attente.
2. *Que va afficher le programme, si l'heure de début est 12 :00 :00 (12h) ?*

Exercice 2

Voici un programme :

```
import signal
import time

def mongestionnaire(signum, stack):
    print('Alarme :', time.ctime())

signal.signal(signal.SIGALRM, mongestionnaire)
signal.alarm(2)

print('Première:', time.ctime())
time.sleep(4)
print('Terminale :', time.ctime())
```

1. *Expliquer les commandes des lignes 6 et 7.*
   Ligne 6 : on associe au signal SIGALRM, la fonction mongestionnaire comme handler.
   Ligne 7 : on envoie le signal SIGALRM au processus actuel avant un temps de 2 secondes d'attente.
2. *Que va afficher le programme, si l'heure de début est 12 :00 :00 (12h) ?*
   Première : 12 :00 :00
   Alarme : 12 :00 :02
   Terminale : 12 :00 :04

Exercice 3

*Notre programme possède une fonction **inconnue** qui peut prendre beaucoup de temps. Nous aimerions qu'au bout de 5s, celle-ci soit automatiquement interrompue. Apporter les modifications nécessaires, pour réaliser ce que l'on souhaite.*

La fonction inconnue

```
def inconnue(n):
    while n>0:
        print('je travaille encore  ',n)
        time.sleep(1)
        n -= 1
```

un corrigé

ex3.py

```
# ex3.py
import signal, time, sys

def mongestionnaire(signum,stack):
    print ("Arrêt du programme")
    sys.exit(0)

def inconnue(n):
    signal.signal(signal.SIGALRM,mongestionnaire)
    signal.alarm(5)
    while n>0:
        print('je travaille',n)
        time.sleep(1)
        n -= 1
```

Exercice 4

*Un programme attend une réponse de l'utilisateur suite à une commande **input**. On aimerait qu'au bout de 5 secondes un message s'affiche, pour demander de répondre dans les 5 prochaines secondes et si tel n'est pas le cas le programme s'interrompt.*

un corrigé

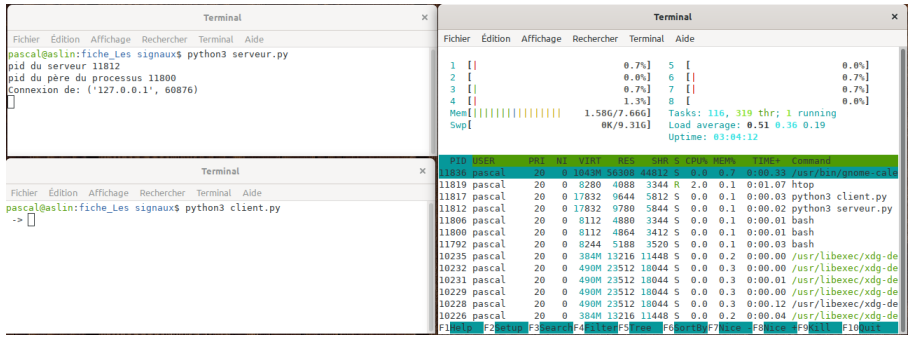
ex4.py

```
import signal

TIMEOUT = 5 # délai de rigueur
CPT =1

def mongestionnaire(signum,stack):
    global CPT
    if CPT == 1:
        print("Je suis généreux, je vous laisse encore 5 secondes")
        CPT += 1
        signal.alarm(TIMEOUT)
    else:
        print('Trop tard!')
        raise TimeoutError

signal.signal(signal.SIGALRM, mongestionnaire)

def input_delai():
    try:
        print ('Vous avez {} secondes pour répondre'.format(TIMEOUT))
        rep = input()
        return rep
    except TimeoutError: # temps dépassé
        return "temps depassé"

# le chrono tourne
signal.alarm(TIMEOUT)
s = input_delai()
# Ne pas oublier d'arrêter le chrono pour poursuivre
signal.alarm(0)
print( 'votre réponse est:',s)
```

Exercice 5

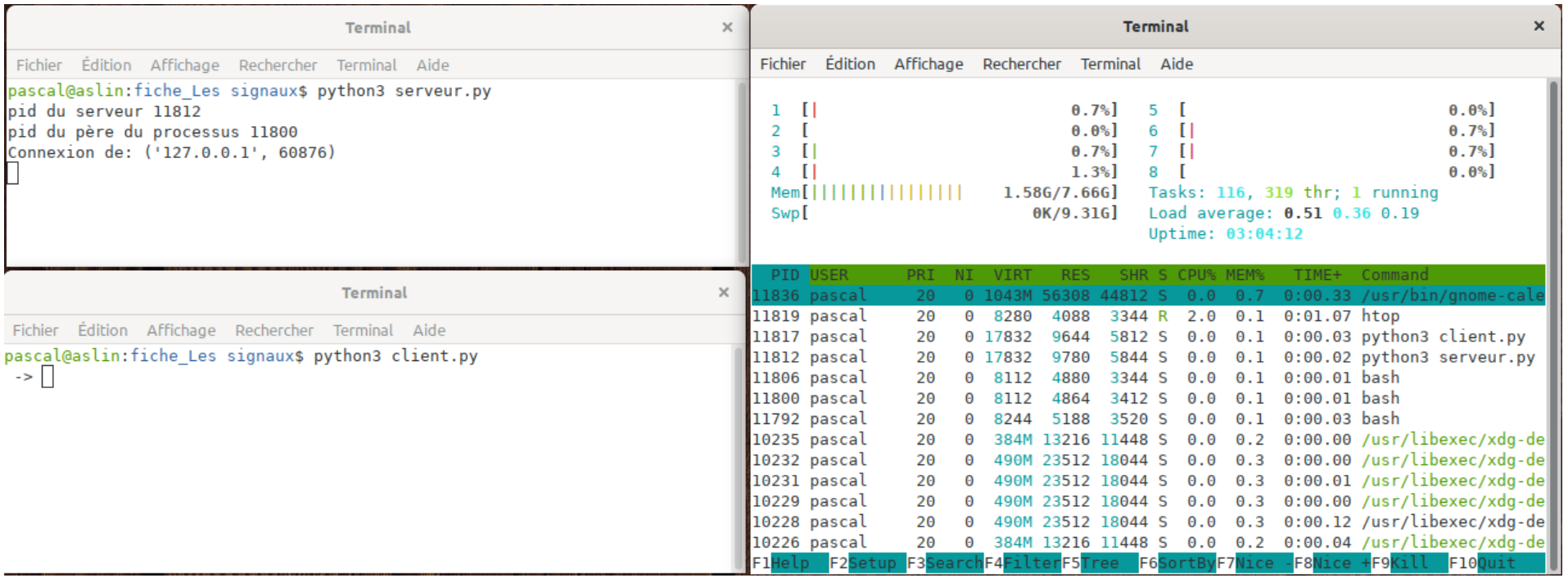
Nous avons deux programmes qui permettent de communiquer entre deux personnes (ex le jeu du juste prix). Le premier programme ***serveur.py*** *va créer le canal de communication et le client* ***client.py*** *va pouvoir s'y connecter.*

1. *Lancez dans deux shells différents le serveur avec* ***python3 serveur.py****, puis* ***python3 client.py*** *et testez la communication.*
2. *À l'aide d' une copie d'écran avec trois shell bash (et uniquement trois sont lancés),*



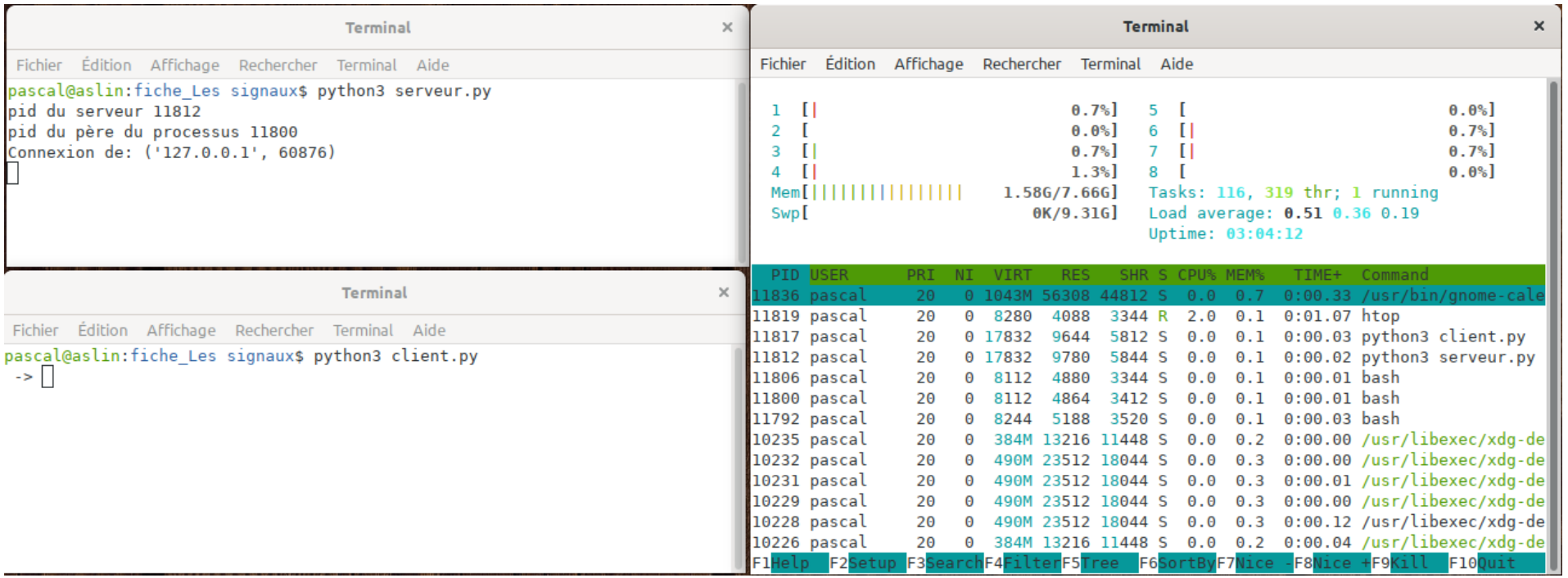
   répondez aux questions suivantes :
   - 2.1 *Quel est le numéro pid du processus lié à l'exécution de client.py ?*
   - 2.2 *Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?*
   - 2.3 *Comment mettre fin au processus associé serveur.py ? (donnez deux possibilités)*
3. *Tuez le serveur et continuez à envoyer des messages par le biais du client. Quelle message d'erreur apparaît ? En expliquer la raison. Aurait-on pu procéder différemment ?*
4. *Modifier le programme client pour que le programme prenne en considération cette erreur en demandant à l'utilisateur s'il veut continuer ou s'il veut mettre fin au programme.*

Corrigé exercice : question 2



2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

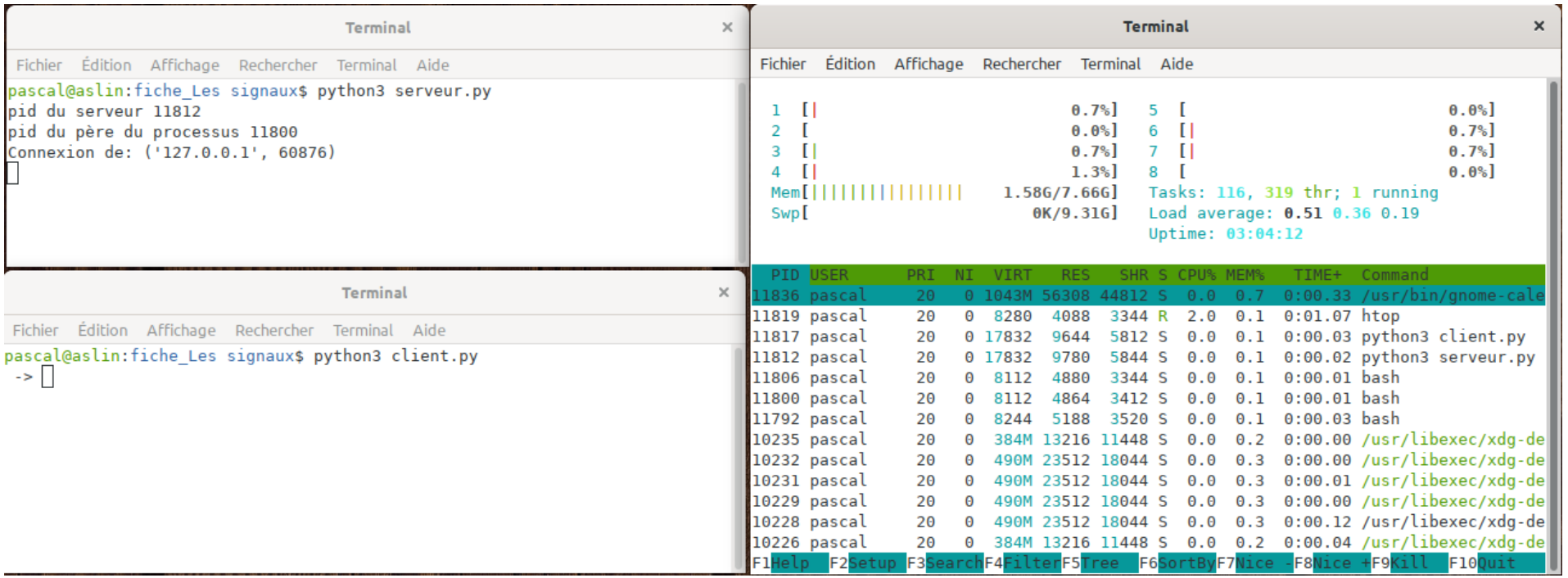
Corrigé exercice : question 2



2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

2.2 Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?

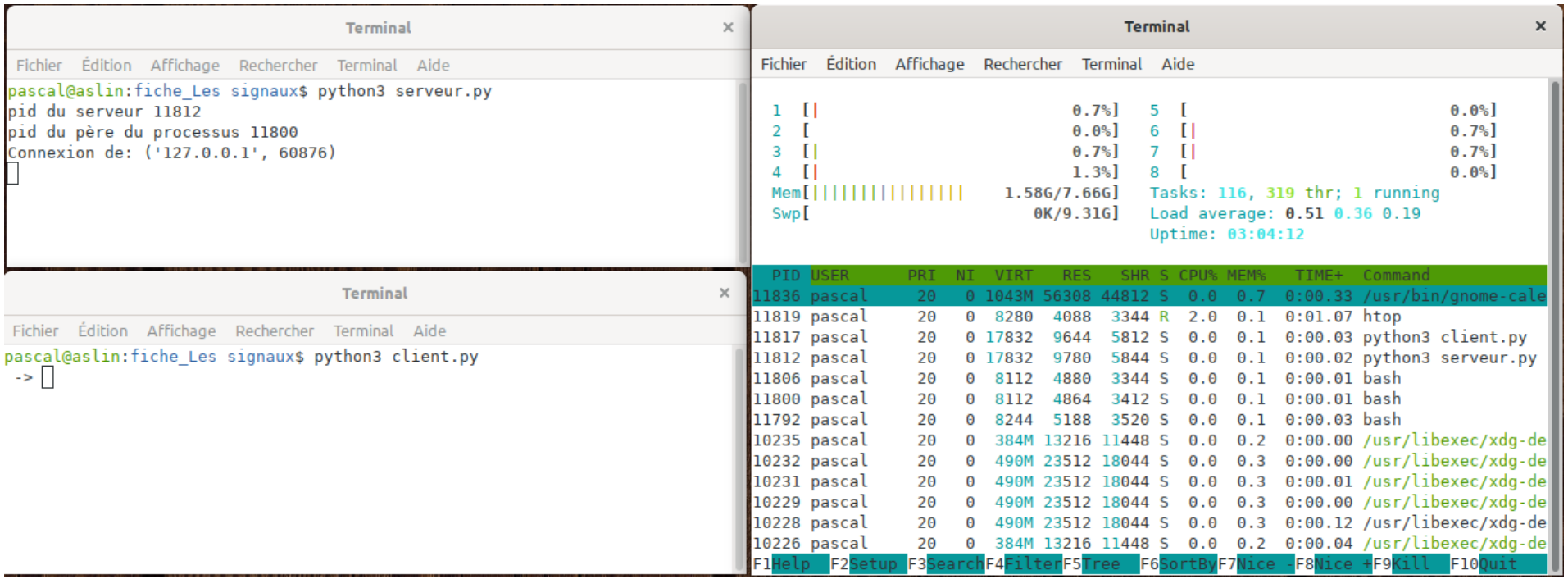
Corrigé exercice : question 2



2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

2.2 Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?

Corrigé exercice : question 2

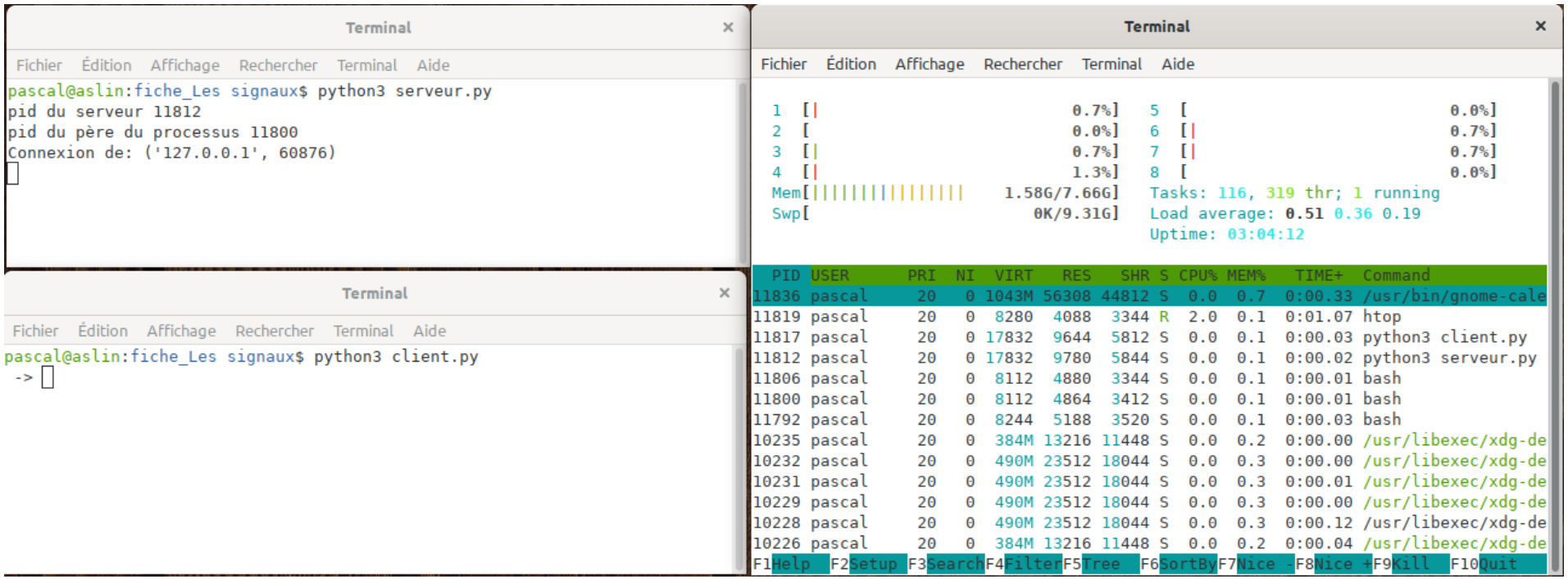


2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

2.2 Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?**Le parent est un programme *bash*, donc en utilisant les informations de *htop* on voit qu'il peut s'agir du pid 11806 ou 11792**

2.3 Comment mettre fin au processus associé serveur.py ? (donnez deux possibilités)

Corrigé exercice : question 2

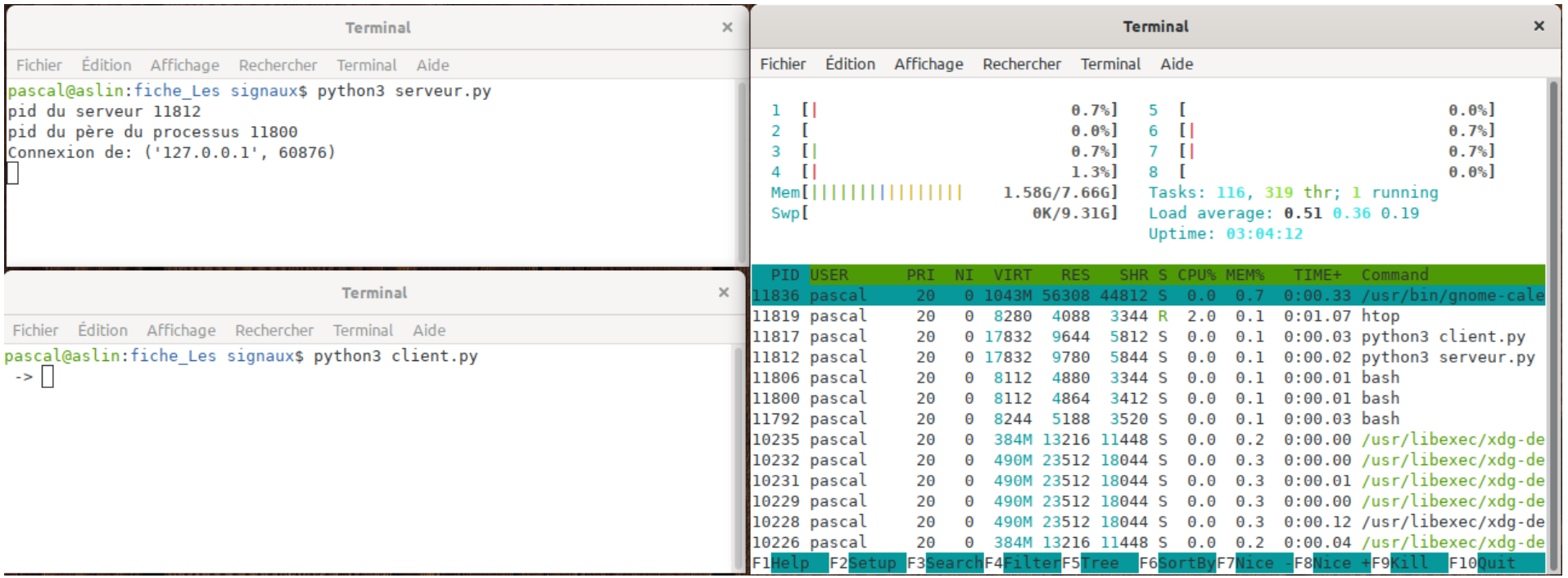


2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

2.2 Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?**Le parent est un programme *bash*, donc en utilisant les informations de *htop* on voit qu'il peut s'agir du pid 11806 ou 11792**

2.3 Comment mettre fin au processus associé serveur.py ? (donnez deux possibilités)

Corrigé exercice : question 2



2.1 Quel est le numéro pid du processus lié à l'exécution de client.py ? **La commande *htop* nous indique que le pid du client.py est 11817.**

2.2 Quel(s) pourrait(aient) être le pid du parent du processus à l'exécution de client.py ?**Le parent est un programme *bash*, donc en utilisant les informations de *htop* on voit qu'il peut s'agir du pid 11806 ou 11792**

2.3 Comment mettre fin au processus associé serveur.py ? (donnez deux possibilités)**En tapant Ctrl-C, avec la fenêtre du terminal correspondant actif, en cliquant sur la croix de la fenêtre appropriée ou en tapant dans une console kill -2 11812.**

Corrigé exercice : question 3

3. Tuez le serveur et continuez à envoyer des messages par le biais du client. Quelle message d'erreur apparaît ? En expliquer la raison. Aurait-on pu procéder différemment ?

Corrigé exercice : question 3

3. Tuez le serveur et continuez à envoyer des messages par le biais du client. Quelle message d'erreur apparaît ? En expliquer la raison. Aurait-on pu procéder différemment ?
   - **On a l'erreur suivante : BrokenPipeError : [Errno 32] Broken pipe.**
     En fermant le serveur, le canal de communication n'est plus disponible rendant impossible l'écriture des données dans le PIPE (canal, tuyau). Le programme client a reçu un signal SIGPIPE.

Corrigé exercice : question 3

3. Tuez le serveur et continuez à envoyer des messages par le biais du client. Quelle message d'erreur apparaît ? En expliquer la raison. Aurait-on pu procéder différemment ?
   - **On a l'erreur suivante : BrokenPipeError : [Errno 32] Broken pipe.**
     En fermant le serveur, le canal de communication n'est plus disponible rendant impossible l'écriture des données dans le PIPE (canal, tuyau). Le programme client a reçu un signal SIGPIPE.
   - **Il n'est pas nécessaire d'arrêter le serveur pour casser le canal de communication, on peut également exécuter la commande :** kill -13 pid (où pid est le numéro du processus à l'exécution du client).

Corrigé exercice : question 4

4. Modifier le programme client pour que le programme prenne en considération cette erreur en demandant à l'utilisateur s'il veut continuer ou s'il veut mettre fin au programme.

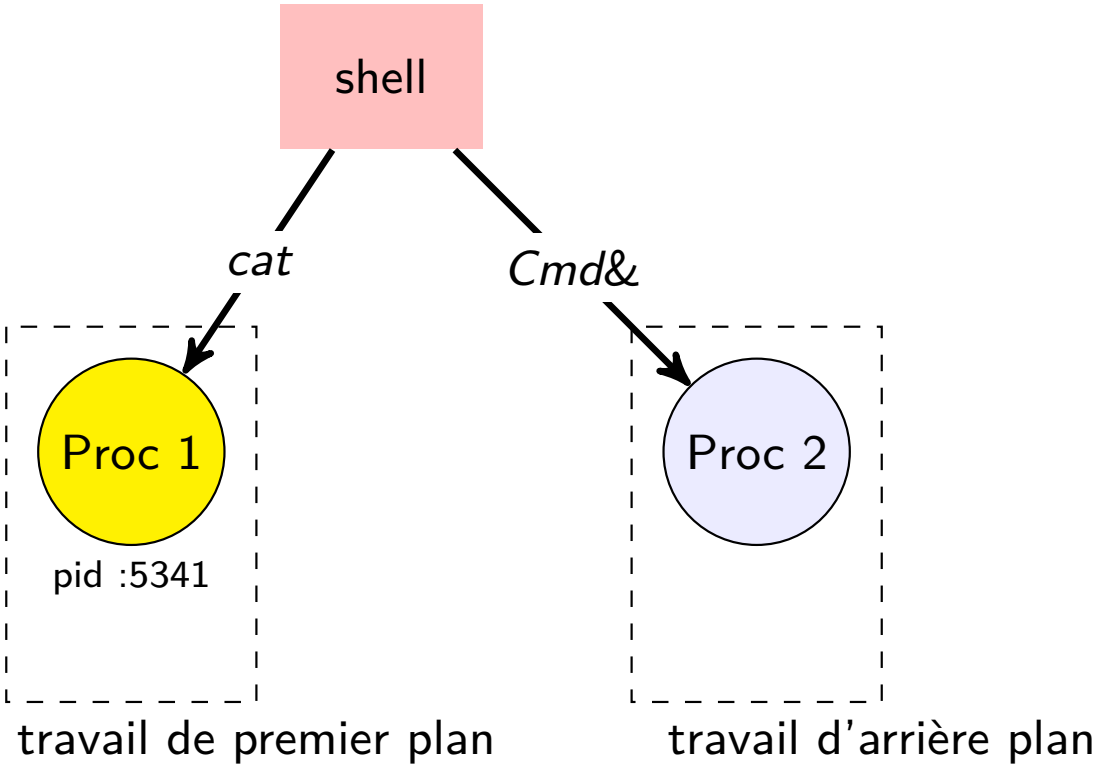
Corrigé exercice : question 4

client.py

```
import socket,signal,sys, os

def gestionnaire(signum,stack):
    r = input("Un problème de communication avec le serveur.\nVoulez vous attendre (o/n)?")
    if r=='n':
        sys.exit(0)
    else:
        raise BrokenPipeError

print("pid du client", os.getpid())
host, port = socket.gethostname(), 5000
signal.signal(signal.SIGPIPE, gestionnaire)

client_socket = socket.socket()
client_socket.connect((host, port))

try:
    message = input("Taper votre message -> ") #message à envoyer

    while message.lower().strip() != 'fin':
        client_socket.send(message.encode())  # envoi un  message
        data = client_socket.recv(1024).decode()  # reception  d'un message
        print('Reçu du serveur: ' + data)  # affiche le message reçu
        message = input(" -> ")  # nouveau message à envoyer
except BrokenPipeError:
    print("Le programme se poursuit")

finally:
    client_socket.close()
```

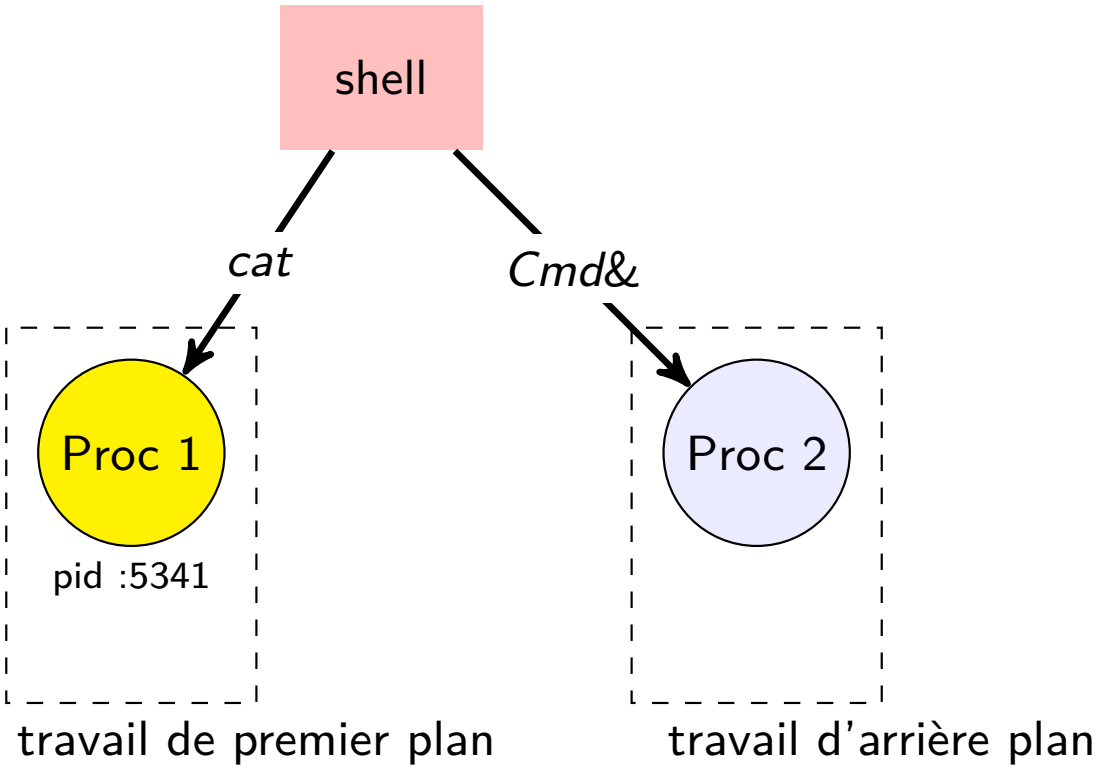
Description du fonctionnement CTRL-Z

Dans le terminal



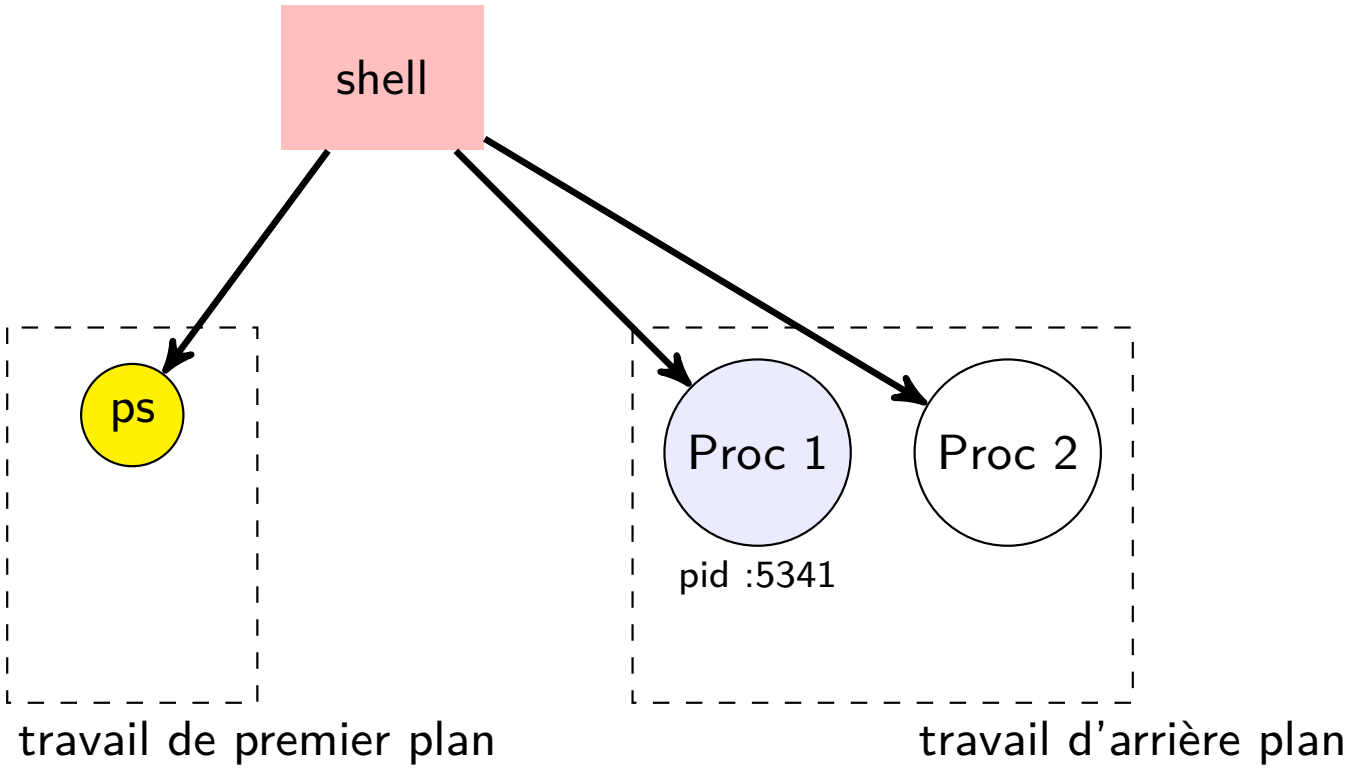
Description du fonctionnement CTRL-Z

Dans le terminal

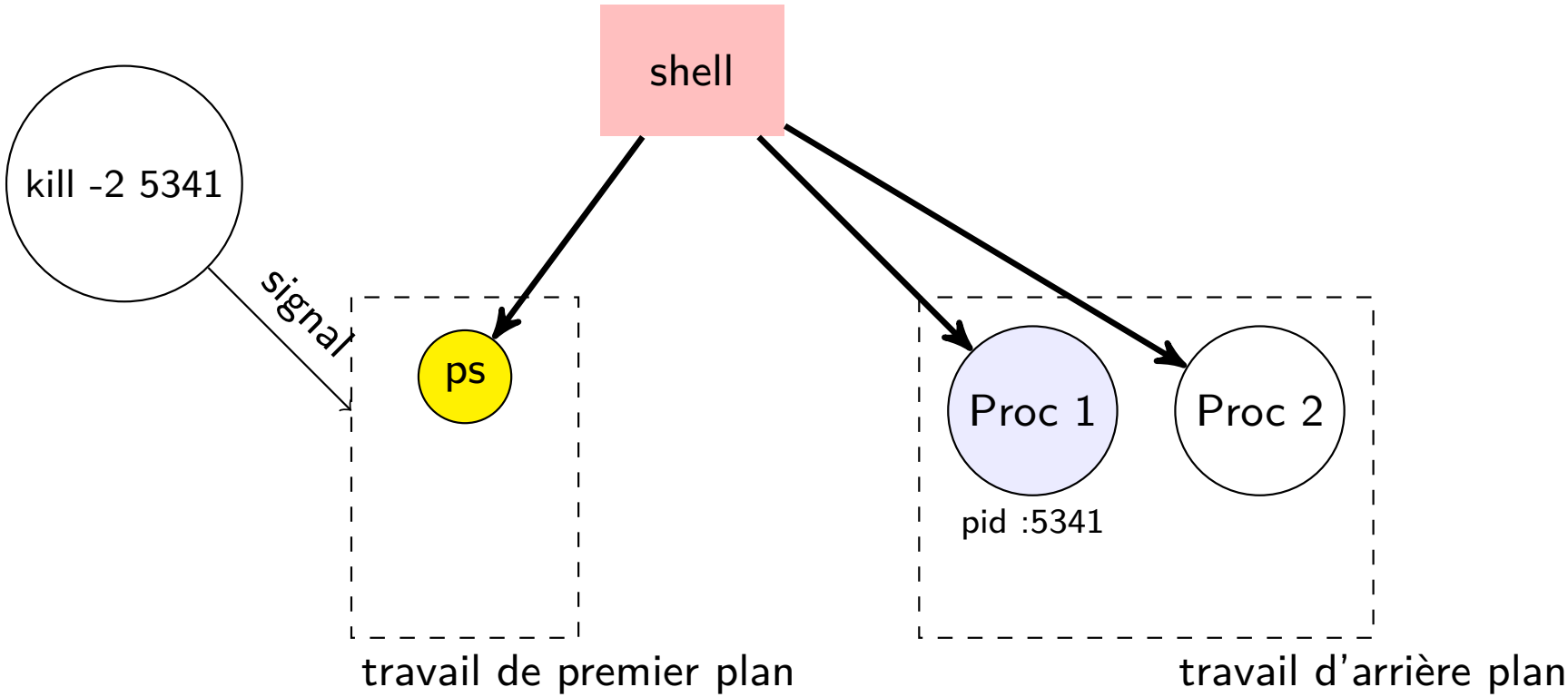


La commande **Ctrl-Z** va passer le processus **cat** dans les travaux en arrière plan et changer son état en **Stopped** . On aurait pu également taper la commande **kill SIGSTP 5341** (*si 5341 est le pid du processus de cat*). L'état **Stopped** du processus **cat** interdit au processus de s'exécuter. La reprise est assurée à la réception du signal SIGCONT.

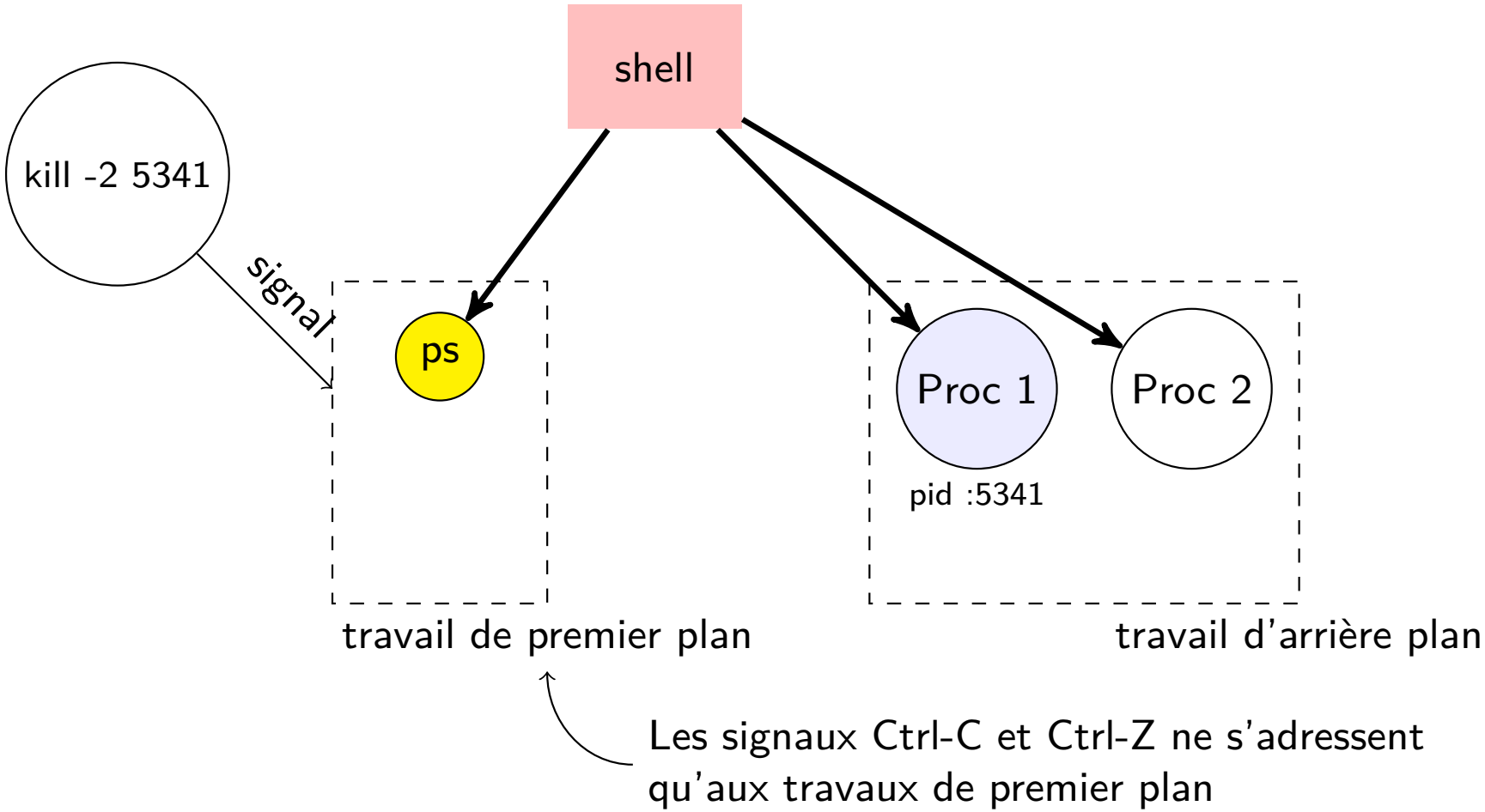
Description du fonctionnement CTRL-Z



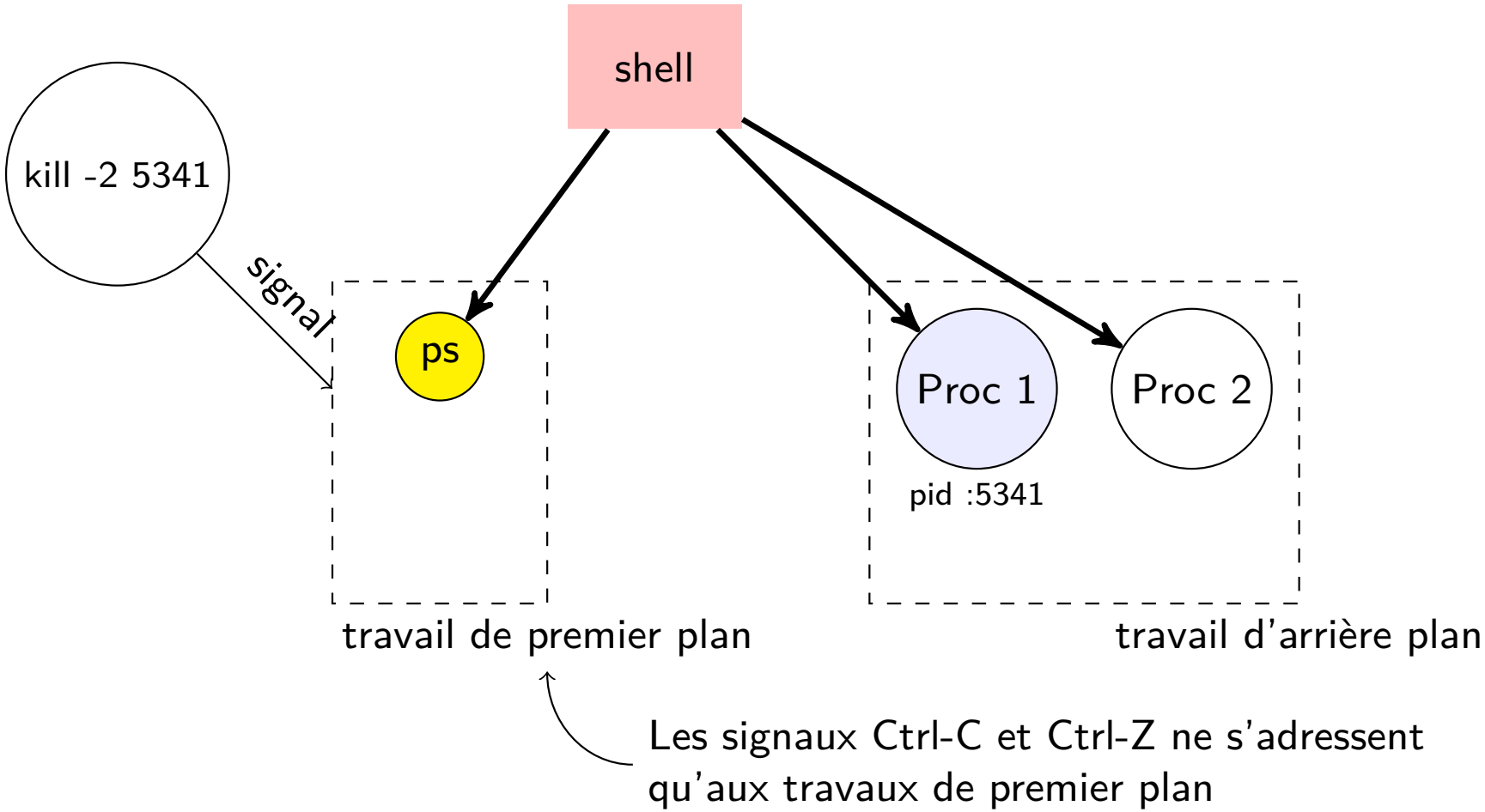
Description du fonctionnement CTRL-Z



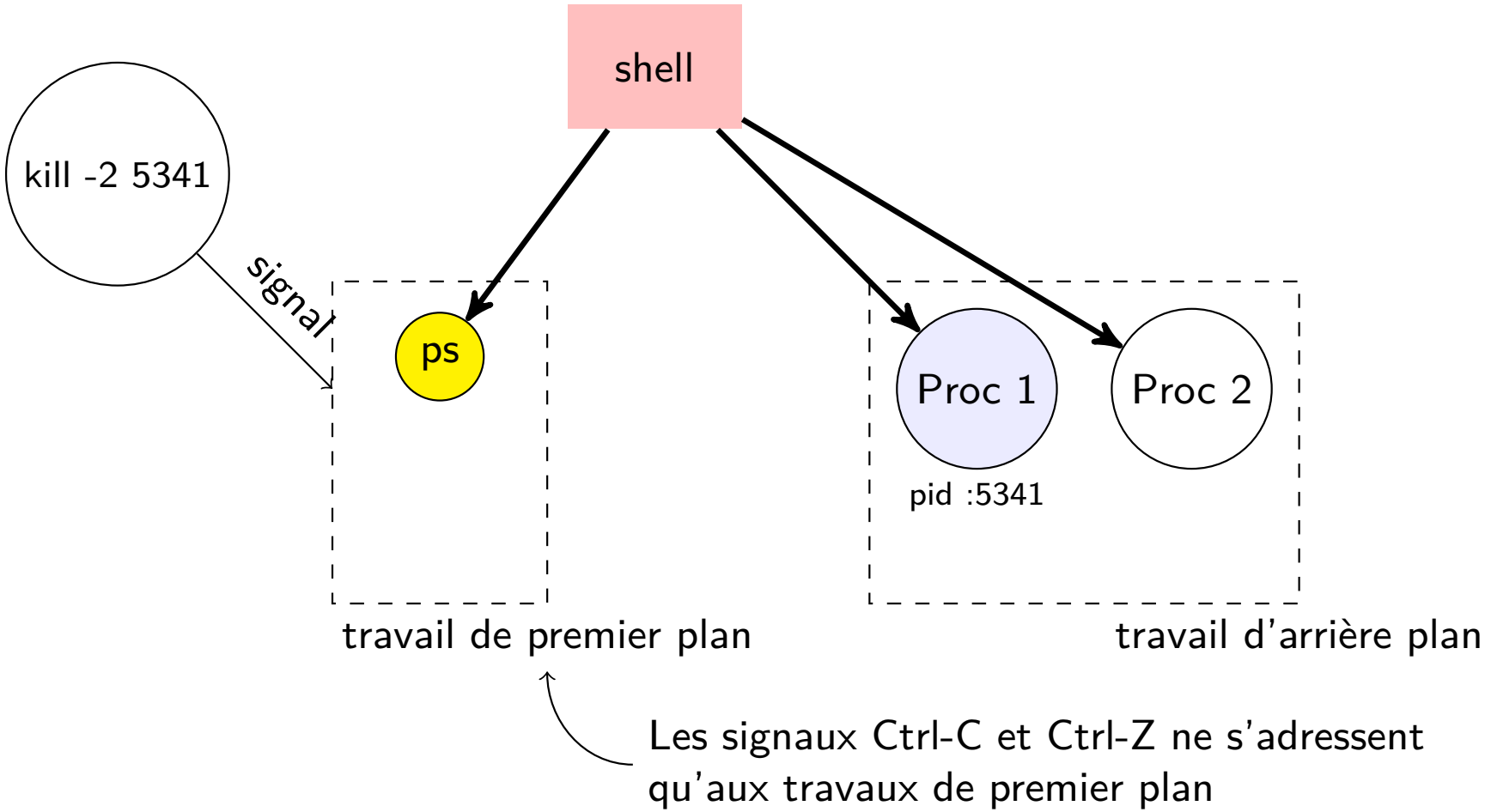
Description du fonctionnement CTRL-Z



Description du fonctionnement CTRL-Z

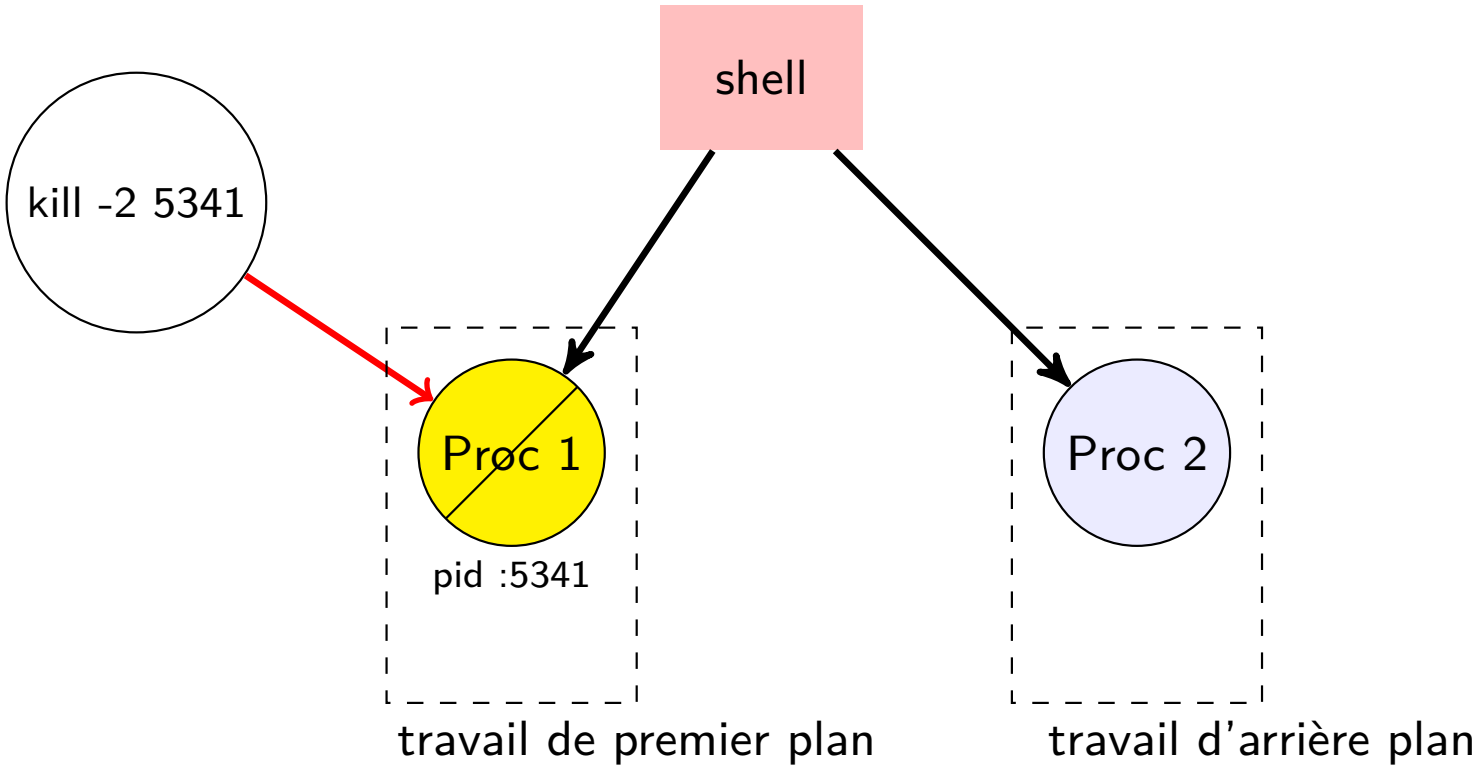


Description du fonctionnement CTRL-Z



La commande **killall -2 cat** n'a pas d'action sur le processus **cat** car il ne fait partie des processus qui sont au premier plan.
À l'aide de la commande **bg** (signal SIGCONT) le processus réintègre les processus de premier plan.

Description du fonctionnement CTRL-Z



Le signal SIGINT (kill -2) peut être délivré au processus destinataire, qui va se terminer, puisqu'il s'agit de l'action par défaut.