Prérequis

1. Système d'exploitation : Linux
2. Les notions de base sur le fonctionnement du noyau et les processus.
3. Utilisation du terminal (shell)
4. Python :
   — écriture d'une fonction
   — avoir vu *try except* est un plus (gestion des erreurs avec Python)

Organisation

— La séance est prévue pour être faite une partie en classe entière ou idéalement en salle informatique et la séance d'exercices en salle informatique. (1h+1h).
— matériels : poste sous Linux avec Python3 installé *(les fichiers ex1.py client.py et serveur.py sont à donner aux élèves)*
— La feuille de cours est à compléter (il s'agit de la partie les notions).
— Un diaporama accompagne les fiches afin d'apporter des précisions sur les notions abordées.

Compétences visées

— Comprendre comment les processus échangent des informations entre eux à l'aide des signaux.
— Utilisation de la commande kill.
— Implémentation des signaux avec Python, écriture des gestionnaires de signaux.

Table des matières

1 Première approche

1.1 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 ?
   On va appuyer sur une combinaison de touche `Ctrl+C`

1.2 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
— un programme traite l'information de la combinaison `Ctrl+C`

2 Deuxième approche

2.1 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

2.2 Dans un terminal

1. Liste des commandes : *à taper*

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

2. résultat : *à compléter*

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
```

3. Stoppons le processus cat
   — Nous allons envoyer un signal de terminaison au processus cat par le biais de la commande **killall** qui envoie un signal aux processus dont le nom est indiqué avec le numéro du signal.

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

   — Regardons ce qui s'est passé

terminal
```
nsi@lin$ ps
```

   — Que remarque-t-on ?

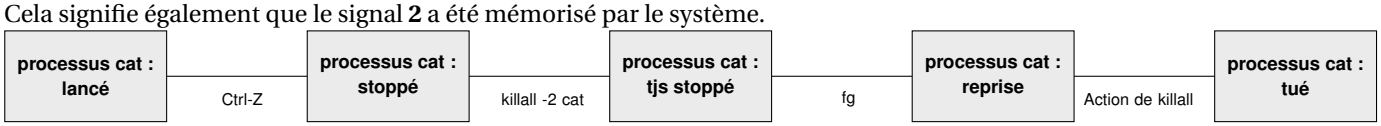
   Le processus `cat` est toujours présent. En fait il a été suspendu par la commande `^z`.
   — On va demander une reprise du processus avec la commande `fg`

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

   — Que remarque-t-on ?

2.3 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.



En fait les choses sont un peu plus compliquées : cf annexe

Commande

- **ps :** liste les processus actifs attachés au terminal, les processus sont identifiés par leur PID.
- **kill :** commande permettant d'envoyer un signal à un processus identifié par son PID avec le numéro du signal.
- **killall :** commande permettant d'envoyer un signal aux processus dont le nom est indiqué avec le numéro du signal.

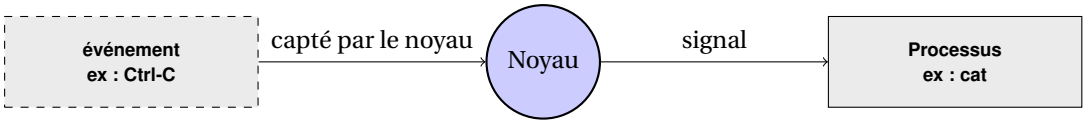
3 Les notions

3.1 Définition d'un signal

Définition 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)

3.2 Schéma 1



Quelques événements possibles
- frappe de caractère dans un terminal *(c'est interpréteur de commande qui envoie un signal)*
  - `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())
- ...

3.3 Les différents signaux

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

terminal
```
nsi@lin$ kill -l
```

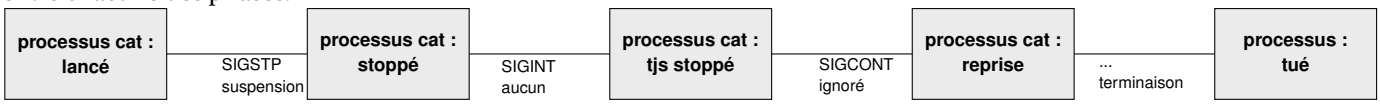
La norme POSIX distingue 32 signaux dont les principaux sont : *compléter le tableau*

Numéro	Nom	Signification	Comportement
1	SIGHUP	Hang-up (fin de connexion)	T(erminaison)
2	SIGINT	Interruption (Ctrl-C)	T
3	SIGQUIT	Interruption forte (Ctrl-\)	T + core
8	SIGFPE	Erreur arithmétique	T + core
9	SIGKILL	Interruption immédiate et absolue	T + core
11	SIGSEGV	Violation des protections mémoire	T + core
13	SIGPIPE	Écriture sur un pipe sans lecteurs	T
20	SIGTSTP	Arrêt temporaire(Ctrl-Z)	Suspension
18	SIGCONT	Redémarrage dun fils arrêté	Ignoré
17	SIGCHLD	un des fils est mort ou arrêté	Ignoré
14	SIGALRM	Interruption d'horloge	Ignoré
19	SIGSTOP	Arrêt temporaire	Suspension
10	SIGUSR1	Émis par un processus utilisateur	T
12	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 laisse la possibilité au processus d'ignorer et/ou de contrôler le signal par contre avec SIGKILL aucun moyen de passer outre la demande d'interruption. De même pour SIGSTOP, le signal ne peut pas être ignoré et modifié.
Pour la liste de toutes les actions des signaux : http://www.linux-france.org/article/man-fr/man7/signal-7.html

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

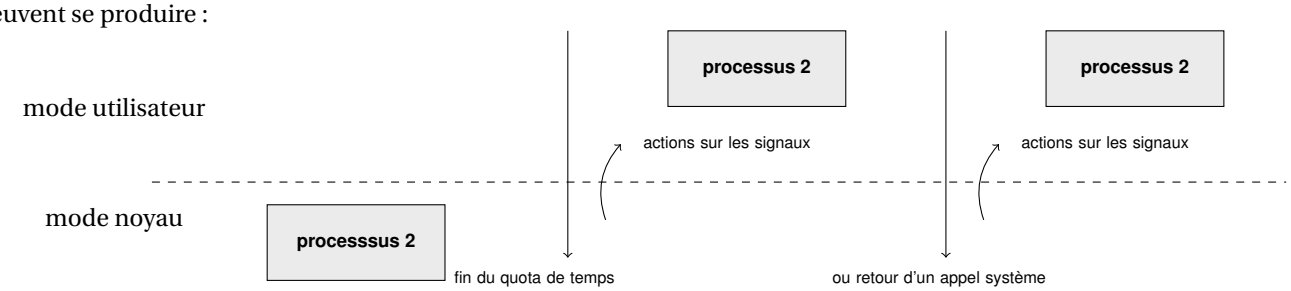


3.4 Que se passe-t-il à la réception du signal?

— Terminaison de l'exécution.
— Suspension de l'exécution (le processus père est prévenu).
— Rien : le signal est ignoré.
— Exécution d'une fonction définie par l'utilisateur.

3.5 Résumé et un peu plus : Gestion des signaux

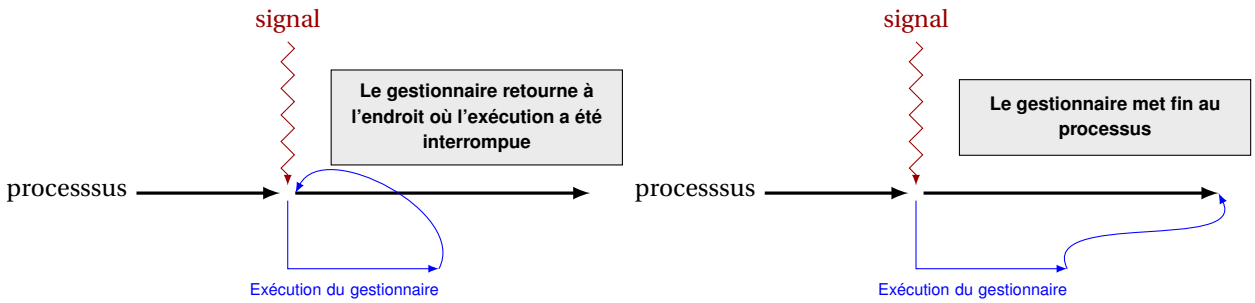
La prise en compte d'un signal (on parle de **délivrance**) ne peut avoir lieu que dans une circonstance bien particulière : la bascule du mode noyau au mode utilisateur. Lorsqu'un signal est envoyé à un processus, plusieurs cas peuvent se produire :

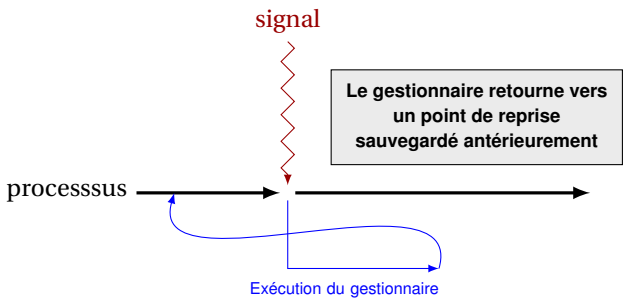


— Le processus est en mode utilisateur :
La délivrance devra alors attendre d'abord le passage du processus en mode noyau puis son retour au mode utilisateur. Pendant tout ce temps, le signal sera **pendant**, c'est à dire en attente de **délivrance**.
— Le processus est en mode noyau, par définition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

Le gestionnaire ou handler





Attention : Le signal SIGCONT ne peut être pris en charge par un **gestionnaire** car il s'adresse à un processus qui est en endormi (SIGSTOP).

— Le signal peut être **bloqué** (on dit également **masqué**). Ceci signifie que la délivrance du signal est différée jusqu'à ce que le blocage soit levé. Ceci n'est possible que pour certains signaux.
Lorsqu'un signal bloqué arrive sur le processus, il n'est pas supprimé, il est juste mis en attente : il devient donc pendant.

Définition 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.*

L'ensemble des informations des signaux pour un processus est regroupé en un vecteur (un pour chaque processus) 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 (handler).

3.6 Description détaillée des signaux

Voir la deuxième page de l'annexe

4 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.

Exercice 1 Reprenez le fichier **ex1.py** et le modifier pour que le signal SIGINT(Ctrl-C) ne puisse pas interrompre le programme.
Quelle commande permet de mettre fin au programme?

ex1b.py

```
# ex1b.py
import signal, 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
```

La commande **kill -2 pid**(SIGINT) ne permet plus d'arrêter le programme, il faut donc envoyer un signal d'interruption qui ne peut pas être intercepté ou modifi', il s'agit **kill - 9 pid** (SIGKILL).

Exercice 2 Voici un programme :

```
1  import signal, time
2
3  def mongestionnaire(signum, stack):
4      print('Alarme :', time.ctime())
5
6  signal.signal(signal.SIGALRM, mongestionnaire)
7  signal.alarm(2)
8
9  print('Première:', time.ctime())
10 time.sleep(4)
11 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 :02

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.

La fonction inconnue

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

Apporter les modifications nécessaires, pour réaliser ce que l'on souhaite.

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
```

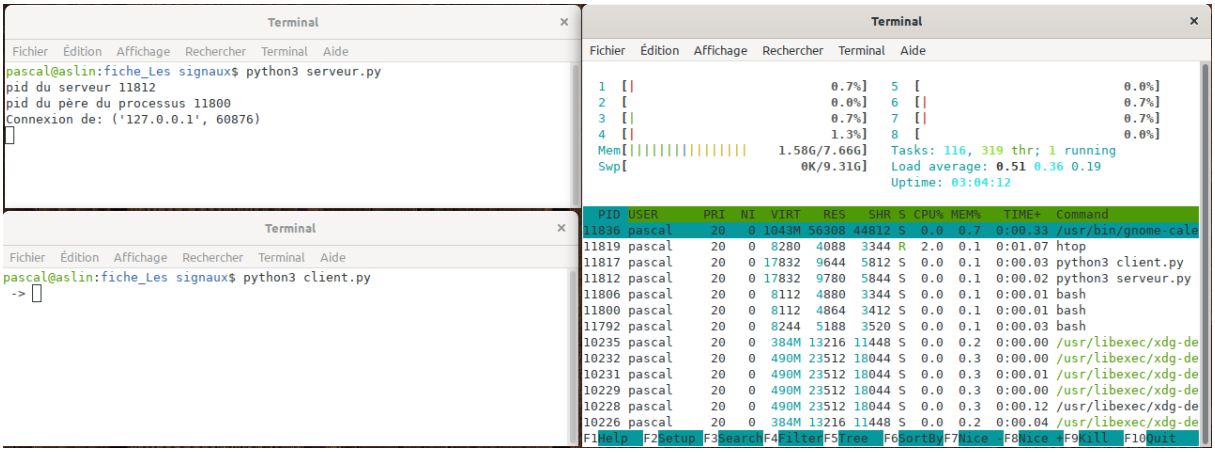
Exercice 4 Un programme attend une réponse de l'utilisateur suite à une commande **input**. On aimerait qu'au bout de 5 secondes un message s'affiche, pour demander de répondre dans les 5 prochaines secondes et si tel n'est pas le cas le programme s'interrompt.

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 None

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

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 :



   (a) 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.

   (b) 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

   (c) 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.

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).
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.

client.py

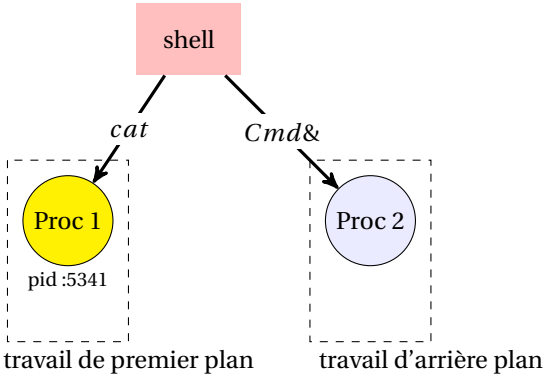
```
# client.py
import socket,signal,sys, os

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()
```

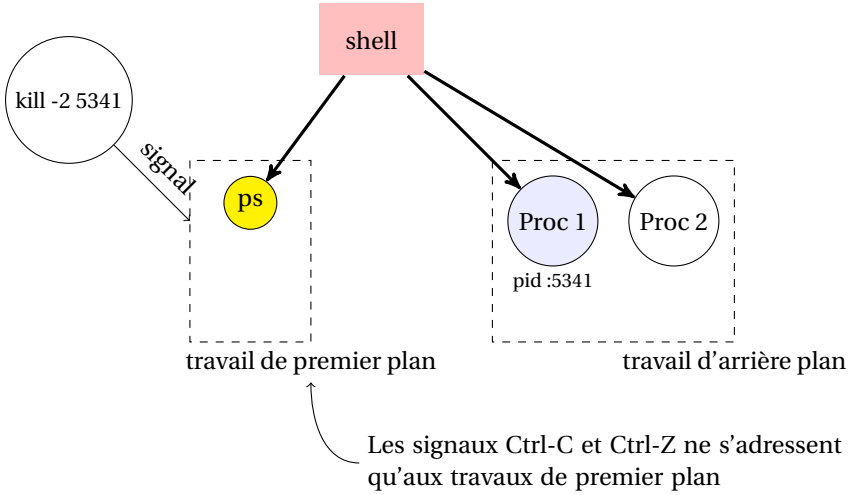
5 Annexes

5.1 Description du fonctionnement CTRL-Z

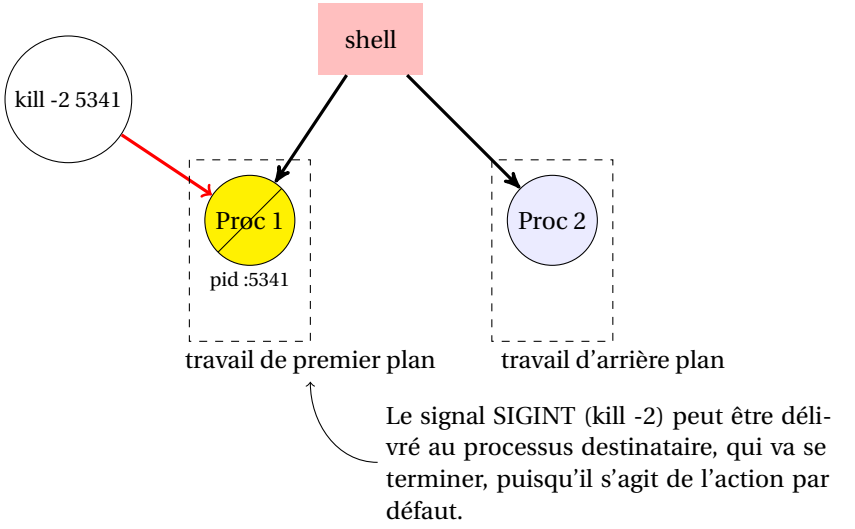
Dans le terminal



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.



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 **fg** (signal SIGCONT) le processus réintègre les processus de premier plan.



5.2 Signaux provoqués par une exception matérielle

source : http://mat.free.free.fr/downloads/unix/signaux.pdf

SIGFPE Floating-point exception : envoyé lors d'un problème avec une opération en virgule flottante mais il est aussi utilisé pour l'erreur de division entière par zéro. L'action par défaut est la mise à mort du processus en créant un fichier d'image mémoire core. Ce fichier est alors utilisé par l'outil de "debugger" gdb.

SIGILL Illegal instruction : envoyé au processus qui tente d'exécuter une instruction illégale. Ceci ne doit jamais se produire dans un programme normal mais il se peut que le fichier exécutable soit corrompu (erreur lors du chargement en mémoire). L'action par défaut est la mise à mort du processus en créant un fichier d'image mémoire core.

SIGSEV Segmentation violation : envoyé lors d'un adressage mémoire invalide (emploi d'un pointeur mal initialisé). L'action par défaut est la mise à mort du processus en créant un fichier d'image mémoire core.

SIGBUS Bus error : envoyé lors d'une erreur d'alignement des adresses sur le bus. L'action par défaut est la mise à mort du processus en créant un fichier d'image mémoire core.

SIGTRAP Trace trap : envoyé après chaque instruction, il est utilisé par les programmes de mise au point (debug). L'action par défaut est la mise à mort du processus en créant un fichier

5.3 Signaux de terminaison ou d'interruption de processus

SIGHUP Hangup : envoyé à tous les processus attachés à un terminal lorsque celui-ci est déconnecté du système. Il est courant d'envoyer le signal à certains processus démons afin de leur demander de se réinitialiser. L'action par défaut est la miseà mort du processus.

SIGINT Interrupt : le plus souvent déclenché par [Ctrl-C]. L'action par défaut est la mise à mort du processus.

SIGQUIT Quit : similaire à SIGINT mais pour [Ctrl-\]. L'action par défaut est la mise à mort du processus en créant un fichier d'image mémoire core.

SIGIOT I/O trap instruction : émis en cas de problème hardware (I/O).

SIGABRT Abort : Arrêt immédiat du processus (erreur matérielle). à core

SIGKILL Kill : utilisé pour arrêter l'exécution d'un processus. L'action par défaut est non modifiable.

SIGTERM Software termination : C'est le signal qui par défaut est envoyé par la commande kill. L'action par défaut est la mise à mort du processus.

5.4 Signaux utilisateurs

SIGUSR1 User defined signal 1 : Définie par le programmeur. L'action par défaut est la mise à mort du processus.

SIGUSR2 User defined signal 2 : idem que SIGUSR1 .

5.5 Signal généré pour un tube

SIGPIPE Signal d'erreur d'écriture dans un tube sans processus lecteur. L'action par défaut est la mise à mort du processus.

5.6 Signaux liés au contrôle d'activité

SIGCLD Signal envoyé au processus parent lorsque les processus fils terminent. Aucune action par défaut. Signaux liés au contrôle d'activité

SIGCLD Signal envoyé au processus parent lorsque les processus fils terminent. Aucune action par défaut.

SIGCONT L'action par défaut est de reprendre l'exécution du processus s'il est stoppé. Aucune action si le processus n'est pas stoppé.

SIGSTOP Suspension du processus. L'action par défaut est la suspension (!! non modifiable).

SIGTSTP Emission à partir d'un terminal du signal de suspension [CTRL-Z]. Ce signal à le même effet que SIGSTOP mais celui-ci peut être capturé ou ignoré. L'action par défaut est la suspension.

SIGTTIN Est émis par le terminal en direction d'un processus en arrière-plan qui essaie de lire sur le terminal. L'action par défaut est la suspension.

SIGTTOU Est émis par le terminal en direction d'un processus en arrière-plan qui essaie d'écrire sur le terminal. L'action par défaut est la suspension.

5.7 Signal lié à la gestion des alarmes

SIGALRM Alarm clock : généré lors de la mise en route du système d'alarme par l'appel système alarm().

Humour : SIGTERM et SIGKILL ©Daniel Stori