TD 3

Gestion des processus Linux

Gestion des processus LINUX

Un processus (tâche) est une commande ou un programme en cours sur le système

Chaque processus est identifié par un numéro : le PID (Process IDentifier)

Linux utilise la fonction fork() pour créer un processus. Le processus d'origine est nommé **processus père** et le nouveau processus créé **processus fils.** Ce dernier possède un nouveau PID

Les deux ont le même code source, mais la valeur retournée par fork() nous permet de savoir si l'on est dans le processus père ou dans le processus fils. Il en résulte que **tous les processus ont un père et qu'ils ont zéro, un ou plusieurs processus fils**

Les deux processus père et fils disposent de leurs propres espaces de mémoires mais il peuvent partager un espace de mémoire pour communiquer : la mémoire partagée

Gestion des processus LINUX

- Deux processus ont un PPID (Parent Process Identifier) égal à 0 :
  - init (systemd) → qui a un PID égale à 1 est le père des processus dans le « **user mode** »
  - kthreadd qui a un PID égale à 2 est le père des processus dans le « **kernel mode** » (init n'est pas son père)
    - Kernel mode : espace réservé au noyau pour faire fonctionner les drivers, le réseau, la gestion de la mémoire, …
    - User mode : espace ou sont exécutés les programmes « pour l'utilisateur », systemd, X-windows, daemons, ….

- **L'arrêt du processus père force obligatoirement l'arrêt des processus fils**

- La fonction exec() est un autre appel système très utilisé: elle permet de « remplacer » le processus courant par un autre binaire

Gestion des processus LINUX

Au démarrage Boot Loader fait appel à la fonction **start_kernel()** du fichier **init/main.c**.

Cette fonction va créer le tout premier processus : le **swapper** (ou **Processus 0**, ou encore **sched pour scheduler**) qui occupera la première entrée de la table des processus.

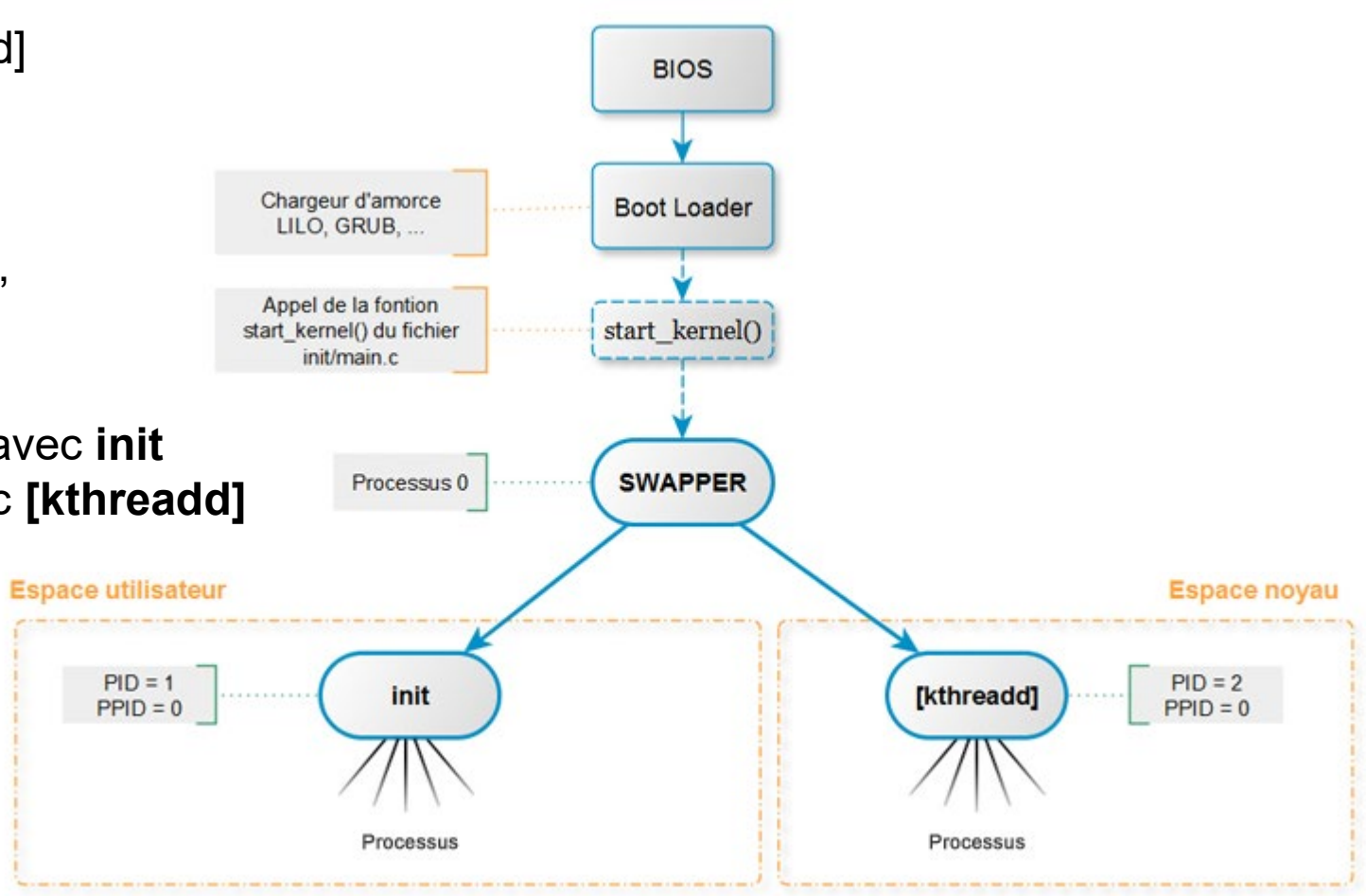
Le swapper va ensuite créer deux processus:

- le processus init
- le processus [kthreadd]

et va s'endormir.

À partir de ce moment-là,
2 espaces co-existent au
sein du système:

- un espace utilisateur avec **init**
- un espace noyau avec **[kthreadd]**



Gestion des processus LINUX

- Mode utilisateur : le processeur de la machine limite l'action des instructions exécutées

- Mode noyau : le processeur de la machine ne limite pas l'action des instructions exécutées

Quand un OS utilise ces deux modes en se réservant le mode noyau et en faisant fonctionner les programmes en mode utilisateur, les programmes font alors partie de l'espace utilisateur

Les états d’un processus

Un processus peut-être dans l'un des quatre états suivants :

- En exécution
- En attente d'exécution : dans cet état le processus a accès à toutes les ressources RAM qui lui sont nécessaires … il est prêt à être exécuté
- Bloqué : dans cet état le processus attend une ressource non encore disponible comme par exemple le cas d'un programme qui demande à l'utilisateur de saisir une valeur
- Zombi : le processus a terminé son exécution, il est prêt a mourir. A ce stade, l’ordonnanceur a pris en compte l’arrêt mais pas encore le processus père (il existe donc encore pour le système)

Ordonnanceur: au sein d’un OS, l’ordonnanceur choisit l’ordre d’exécution des processus d’une machine

Exercices - Gestion des processus

La commande ps permet d'afficher les processus en cours.

Listez :

- vos processus
- l'ensemble des processus du système
- tous les processus du système en faisant apparaître : PPID, USER,PID
- trouvez les processus avec le PPID de 0
- l'ensemble des processus en faisant apparaître la parenté…

Que remarquez vous pour le processus « ps » que vous venez de lancer ?

Que représente le résultat de la commande ps (sans paramètre)

Proposez un test pour valider que si on tue le processus père, on tue automatiquement le(s) fils

Priorité et ordonnancement

- Linux est dit multitâches : il peut exécuter plusieurs tâches en parallèle
- Pour exécuter plusieurs tâches en // avec un seul processeur, Linux affecte à chaque programme une tranche de temps (quantum). Les processus sont alors exécutés à tour de rôle
- L'affectation de cette tranche de temps et la gestion de l'ordre de passage est géré par l'ordonnanceur : on dit alors que l'ordonnanceur est « préemptif »
- L'ordonnanceur préemptif gère un accès équitable à la CPU. Pour chaque processus, après utilisation de son temps de calcul, l'ordonnanceur réévalue sa priorité et le replace dans la file d'attente
- Il existe différents types d'algorithmes de gestion de la préemption :
  - Le **multitâche coopératif** où le système laisse les applications gérer leur ordre de passage (Gamma 60 en 1958, Windows 3.11)
  - Le **multitâche préemptif** (W95, OS X, …) où le processeur indique à l'OS qu'il doit mettre en pause le processus en cours d'exécution

nice et renice

- L'utilisateur peut influer sur la priorité des processus à l'aide de deux commandes: nice et renice
- nice permet de changer le niveau de priorité par défaut d'un processus avant son démarrage

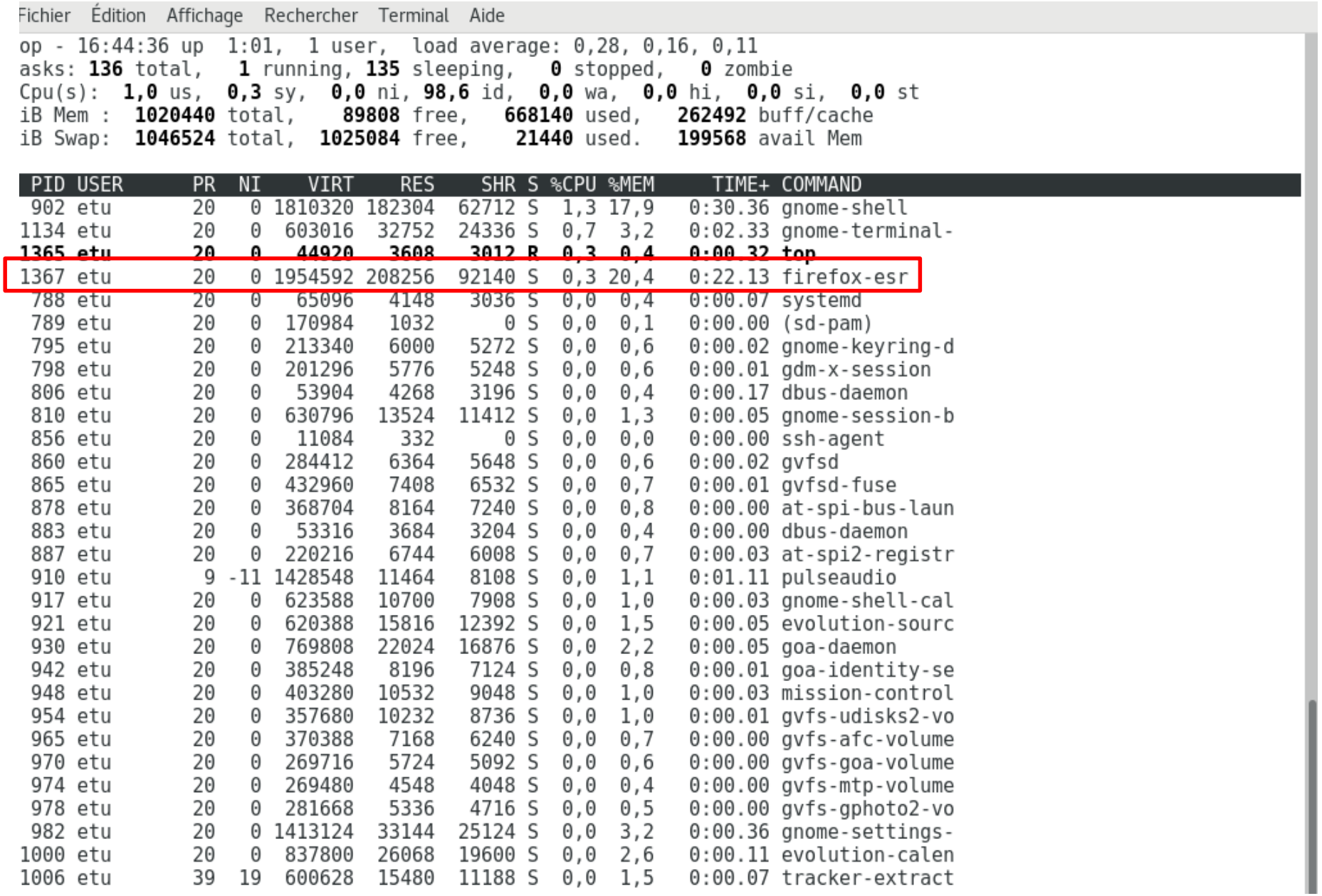
  #nice -n 12 firefox
- renice permet de changer le niveau de priorité par défaut d'un processus en cours d'exécution

  #top -u etu pour connaître la priorité d'exécution -NI-

  #renice -n 15 -p 1367)
- Les différents niveaux :
  - priorité la plus basse : +19
  - priorité la plus haute : -20 (uniquement attribuable par root)
  - par défaut : 0
- nice est utile quand plusieurs processus demandent plus de puissance CPU que le processeur ne peut en fournir

nice et renice

#top : renvoie la listes des processus en cours avec leur priorité - colonne NI



Fichier Édition Affichage Rechercher Terminal Aide

```
op - 16:44:36 up  1:01,  1 user,  load average: 0,28, 0,16, 0,11
asks: 136 total,   1 running, 135 sleeping,   0 stopped,   0 zombie
Cpu(s):  1,0 us,  0,3 sy,  0,0 ni, 98,6 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
iB Mem :  1020440 total,    89808 free,   668140 used,   262492 buff/cache
iB Swap:  1046524 total,  1025084 free,    21440 used.   199568 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  902 etu       20   0 1810320 182304  62712 S  1,3 17,9   0:30.36 gnome-shell
 1134 etu       20   0  603016  32752  24336 S  0,7  3,2   0:02.33 gnome-terminal-
 1365 etu       20   0   44920   3608   3012 R  0,3  0,4   0:00.32 top
 1367 etu       20   0 1954592 208256  92140 S  0,3 20,4   0:22.13 firefox-esr
  788 etu       20   0   65096   4148   3036 S  0,0  0,4   0:00.07 systemd
  789 etu       20   0  170984   1032      0 S  0,0  0,1   0:00.00 (sd-pam)
  795 etu       20   0  213340   6000   5272 S  0,0  0,6   0:00.02 gnome-keyring-d
  798 etu       20   0  201296   5776   5248 S  0,0  0,6   0:00.01 gdm-x-session
  806 etu       20   0   53904   4268   3196 S  0,0  0,4   0:00.17 dbus-daemon
  810 etu       20   0  630796  13524  11412 S  0,0  1,3   0:00.05 gnome-session-b
  856 etu       20   0   11084    332      0 S  0,0  0,0   0:00.00 ssh-agent
  860 etu       20   0  284412   6364   5648 S  0,0  0,6   0:00.02 gvfsd
  865 etu       20   0  432960   7408   6532 S  0,0  0,7   0:00.01 gvfsd-fuse
  878 etu       20   0  368704   8164   7240 S  0,0  0,8   0:00.00 at-spi-bus-laun
  883 etu       20   0   53316   3684   3204 S  0,0  0,4   0:00.00 dbus-daemon
  887 etu       20   0  220216   6744   6008 S  0,0  0,7   0:00.03 at-spi2-registr
  910 etu        9 -11 1428548  11464   8108 S  0,0  1,1   0:01.11 pulseaudio
  917 etu       20   0  623588  10700   7908 S  0,0  1,0   0:00.03 gnome-shell-cal
  921 etu       20   0  620388  15816  12392 S  0,0  1,5   0:00.05 evolution-sourc
  930 etu       20   0  769808  22024  16876 S  0,0  2,2   0:00.05 goa-daemon
  942 etu       20   0  385248   8196   7124 S  0,0  0,8   0:00.01 goa-identity-se
  948 etu       20   0  403280  10532   9048 S  0,0  1,0   0:00.03 mission-control
  954 etu       20   0  357680  10232   8736 S  0,0  1,0   0:00.01 gvfs-udisks2-vo
  965 etu       20   0  370388   7168   6240 S  0,0  0,7   0:00.00 gvfs-afc-volume
  970 etu       20   0  269716   5724   5092 S  0,0  0,6   0:00.00 gvfs-goa-volume
  974 etu       20   0  269480   4548   4048 S  0,0  0,4   0:00.00 gvfs-mtp-volume
  978 etu       20   0  281668   5336   4716 S  0,0  0,5   0:00.00 gvfs-gphoto2-vo
  982 etu       20   0 1413124  33144  25124 S  0,0  3,2   0:00.36 gnome-settings-
 1000 etu       20   0  837800  26068  19600 S  0,0  2,6   0:00.11 evolution-calen
 1006 etu       39  19  600628  15480  11188 S  0,0  1,5   0:00.07 tracker-extract
```

Les signaux

- Les signaux, ou interruptions logicielles, sont des événements externes qui changent le déroulement d'un programme de manière asynchrone, c'est-à-dire à n'importe quel instant lors de l'exécution du programme

- Les programmes peuvent traiter les signaux et donc réagir en fonction. Ils peuvent également choisir de l'ignorer.

- Par exemple
  - le signal d'arrêt sigint (kill -15 ou ctrl+c) est masquable
  - le signal « sigkill » (kill -9) est non masquable et va donc forcer l'arrêt dans tous les cas

Les signaux

Signaux fréquents sous linux (non exhaustif) :

SIGHUP (1)

- permet de recharger la configuration d'un démon
- d'envoyer un signal d'arrêt aux processus lancé depuis un terminal après sa fermeture)

SIGINT (2)

- (CTRL +C) par défaut arrêt du processus depuis son terminal de lancement

SIGKILL (9)

- arrêt brutal d'un processus: ce signal ne peut être ignoré

SIGTERM (15)

- arrêt propre (peut être ignoré par le processus)

SIGTSTP (20)

- demande de suspension depuis le clavier

SIGCONT (18)

- demande de reprise du processus

kill / killall / trap

Les commandes kill et killall permettent d'envoyer des signaux aux processus.

La différence est que killall se base sur le nom du processus et kill sur son PID.

#killall ping ➔ envoie un SIGTERM à tous les processus ping en cours

Commandes équivalentes (envoie le signal SIGTERM numéroté 15)

#kill 1343

#kill -15 1343

#kill -TERM 1343

#kill -9 -1 tue tous les processus possibles (attention c'est violent !)

La commande trap est une fonction qui permet d'intercepter les signaux en bash.

Elle permet d'émettre des signaux à l'aide de scripts

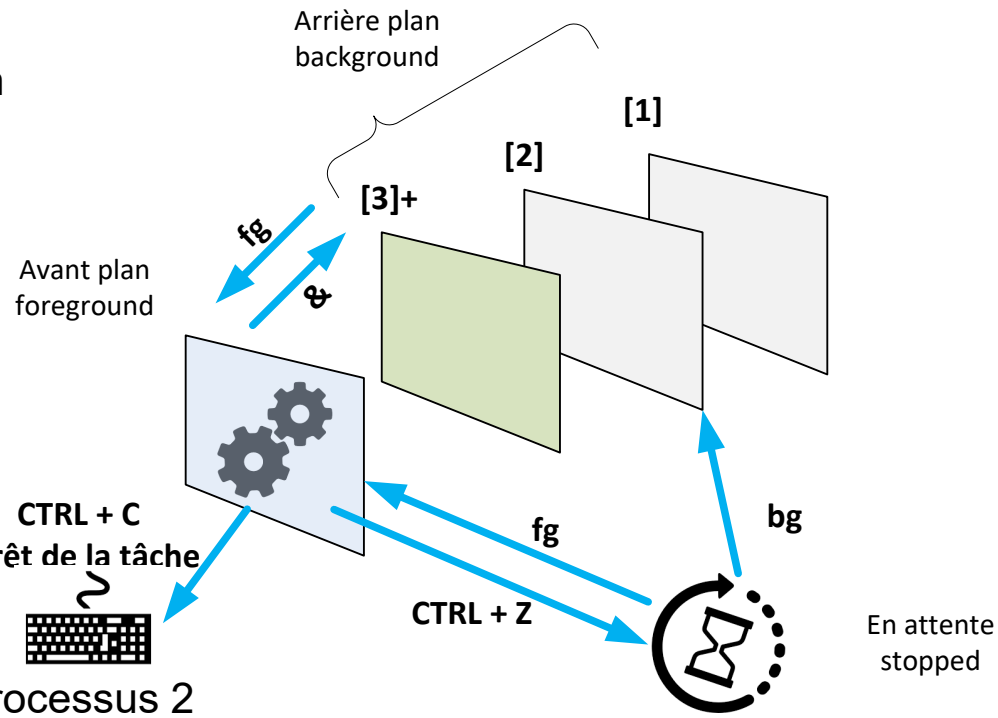
#trap « commande » <SIGNAL>

#trap « echo interception » 2 ➔ permet d'afficher « interception » lors d'un appui sur CTRL+C (signal SIGINT – n°2)

#trap '' 2 ➔ inhibe CTRL+C dans le terminal où elle a été lancée

Background / foreground
& / fg / bg / jobs / nohup

- Il faut considérer un terminal comme un empilage de terminaux « virtuels »
- Il ne peut y avoir qu'une tâche par terminal et il n'y a qu'un terminal directement accessible par l'utilisateur : le premier plan (foreground). Les programmes interactifs doivent être en premier plan. Quand on quitte un terminal, tous les programme qui lui sont liés ferment
- La commande **nohup** permet de détacher le processus du terminal: il devient indépendant du terminal
  - **#nohup mon_script**
- **CTRL+Z** permet de mettre pause une tâche : on peut alors soit la basculer en arrière-plan soit la ramener à l'avant
- Une tâche peut-être mise en pause puis passée en arrière-plan ou directement passée en arrière-plan
- **&** : permet de lancer une tâche en arrière-plan
  - **#ma_tache &**
- L'utilitaire **jobs** permet de lister les jobs en arrière-plan et en attente
- **fg** pour passer en avant-plan
  - **#fg 2** : passe la tâche de fond 2 en avant-plan
- **#bg 2 :** pour passer de pause à tâche de fond le processus 2



Exercices - Signaux

- Lancez deux terminaux
  - Dans le premier, lancez le script ./test.sh (détail page suivante)
  - Dans le second :
    - Repérez son pid et sa position dans la hiérarchie des processus.

      Comment expliquez-vous le positionnement du processus « sleep 2 » (une des commandes du script)
    - Arrêtez proprement le processus avec kill
- Dans votre terminal, envoyez la commande suivante : trap ‘’ 15
- Lancer le script ./test.sh
- Arrêtez-le proprement avec kill <PID>

  Que remarquez vous ? Pourquoi selon-vous ? Quelle peut-être l’utilité de cette fonction ?

Exercices - Tâches

- Dans un terminal, lancez le script ./test.sh
  - Mettez-le en pause puis passez-le en tâche de fond.

    Que remarquez vous une fois le script lancé en tâche de fond?
  - Mettez-le en pause puis ramenez-le au premier plan (foreground)
  - Fermez-le terminal sans arrêter le script, puis vérifiez son état avec ps
  - Comment faire pour ne pas arrêter le script en quittant le terminal ?

Exercices - Script test.sh

```
#! /bin/bash
i=0
while true
        do
                i=`expr $i + 1`
                echo "Valeur:" $i
                sleep 2
        done
```