ISS - Initiation aux Systèmes d'exploitation et au Shell
LU2IN020

TD 09 – Les signaux

Julien Sopena

décembre 2022

Le but de cette neuvième semaine est d'étudier la communication par signaux. On y étudiera l'envoi de signaux, mais aussi la redéfinition des handlers de signaux.

Exercice 1 : Le point sur la communication inter processus

Question 1

Pourquoi est-il complexe de faire communiquer deux processus ?

Question 2

Quels mécanismes de communication avons-nous déjà mis-en-oeuvre dans le cadre de cette UE ? Pour chacun d'eux, listez les avantages et les limites.

Exercice 2 : Premiers signaux

Le système d'exploitation offre un autre mécanisme de communication inter-processus (*IPC*), celui des **signaux**. Ce mécanisme est minimaliste, car on ne peut pas communiquer de valeur autre que le numéro du signal envoyé).

Cependant, il présente l'avantage de :
- ne pas poser de contraintes sur les processus communiquant, autre que l'appartenance au même utilisateur et la connaissance du *pid* du destinataire.
- de pouvoir interrompre le destinataire à tout moment. Ce dernier s'arrête alors en plein milieu de son code et exécute une fonction (dit **handler**) de traitement associée au signal reçu.

Il présente aussi certaines limites que nous allons voir au cours de cette séance de TD et de TP.

Question 1

Sachant que la commande `kill` permet d'envoyer un signal à un processus dont le *pid* est passé en paramètre, quel affichage est produit par l'exécution du script `pere.sh`.

pere_1.sh

```
#! /bin/bash

echo "Debut du père ($$)"

./fils_1.sh &
pid_fils=$!

for i in {1..4} ; do
  sleep 2
  kill -s SIGTERM $pid_fils
done

echo "Fin père"
```

fils_1.sh

```
#! /bin/bash

echo "Debut du fils ($$)"

for i in {1..10} ; do
  echo "Fils : $i"
  sleep 1
done

echo Fin du fils
```

Question 2

Voici une nouvelle version qui utilise la commande `trap` pour redéfinir la fonction à exécuter à la réception d'un signal, appelée *handler*. Quel affichage est produit par l'exécution du script `pere_2.sh` ?

pere_2.sh

```
#! /bin/bash

echo "Debut du père ($$)"

./fils_2.sh &
pid_fils=$!

for i in {1..4} ; do
  sleep 2
  kill -s SIGTERM $pid_fils
done

echo "Fin père"
```

fils_2.sh

```
#! /bin/bash

trap "echo Réception signal à $i" SIGTERM

echo "Debut du fils ($$)"

for i in {1..10} ; do
  echo "Fils : $i"
  sleep 1
done

echo Fin du fils
```

Question 3

Proposez une correction qui permette un affichage correct à la réception d'un signal `SIGTERM`.

Question 4

Proposez maintenant une nouvelle version qui restaure le handler par défaut (il suffit dans un `trap` de remplacer le handler par un `-`) après la réception du premier signal.

Question 5

On veut maintenant restaurer le handler par défaut après la réception de trois signaux, quelle que soit la date de réception.

Question 6

Que devient l'affichage de votre dernier script si l'on modifie le père en sortant la commande `sleep` de la boucle (sans modifier le fils) ?

pere_6.sh

```
#! /bin/bash

echo "Debut du père ($$)"

./fils_6.sh &
pid_fils=$!

sleep 2
for i in {1..4} ; do
  kill -s SIGTERM $pid_fils
done

echo "Fin père"
```

Exercice 3 : Ping-pong

Dans cet exercice, on veut implémenter en Shell un ping-pong entre deux processus. Ainsi, l'exécution du processus `ping.sh`, lancera le processus `pong.sh` puis les deux processus seront chargés respectivement d'afficher les `"ping"` et `"pong"`. Ces affichages devront se faire alternativement (`"ping"`, puis `"pong"`, puis `"ping"`, ...) et être espacés d'au moins 1 seconde.

Question 1

Implémentez ces deux scripts en utilisant l'envoi du signal `SIGUSR1` pour synchroniser les affichages. Dans cette première version, les affichages se feront de manière infinie.

Question 2

Modifiez vos scripts de façon à pouvoir les arrêter (tous les deux) en envoyant un signal `SIGUSR2` au processus exécutant `ping.sh`.