

IF 110 – Les signaux



Joachim Bruneau-Queyreix

ENSEIRB-MATMECA

Bordeaux-INP

jbruneauqueyreix@enseirb-matmeca.fr

D'après le cours d'introduction aux systèmes d'exploitation de Télécom SudParis



Présentation du cours

- Contexte : comprendre un mécanisme de communication inter-processus

- Objectif : Savoir utiliser les signaux

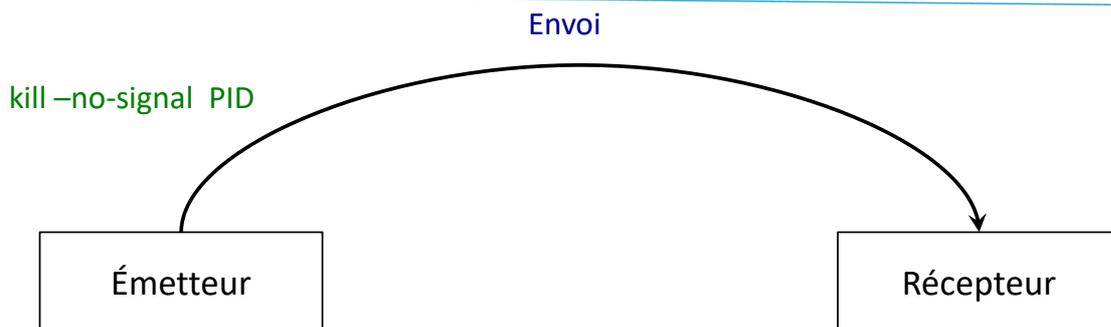
- Notion clé : Signaux (`kill`, `trap`)

Les signaux

- ❑ Signal = un mécanisme de communication inter-processus
 - Communication simple par envoi de message direct
 - Message = un entier entre 1 et 31
 - Perte de message possible
 - Ordre de réception aléatoire (différent de l'ordre d'émission)
- ❑ Souvent utilisé pour
 - Arrêter un processus (par exemple, `control-c`)
 - Notifier un processus lorsque sa configuration change
 - Prévenir un processus qu'il effectue une opération invalide (accès mémoire invalide, division par zéro...)

3

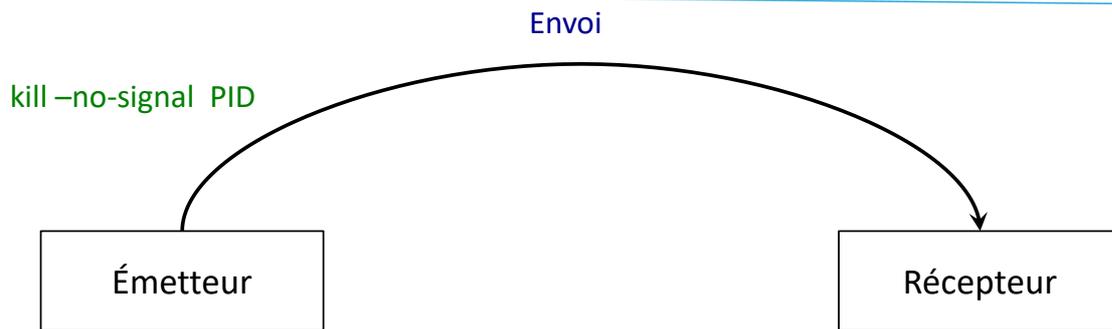
Principe de fonctionnement



- Émetteur envoie un message à un processus
- Une routine de réception est automatiquement invoquée chez le récepteur dès que le signal arrive
- Par défaut, cette routine tue le récepteur
(sauf pour les signaux SIGCHLD, SIGSTP, SIGCONT)

4

Principe de fonctionnement



- Nota bene :
 - Un message est limité à un nombre compris entre 1 et 31
 - Tout signal émis est délivré (sauf si le même numéro de signal est émis une seconde fois avant réception – dans ce cas le deuxième signal est perdu)
 - Ordre de réception aléatoire

5

Les signaux

Quelques exemples ([man 7 signal](#)) :

- **SIGHUP (1)** : fermeture terminal ⇒ à tous les processus attachés
- **SIGINT (2)** : control-c dans un terminal ⇒ au processus premier plan
- **SIGQUIT (3)** : souvent control-d, généré par un processus à lui-même
- **SIGILL (4)** : instruction illégale (envoyé par le noyau)
- **SIGFPE (8)** : division par 0 (envoyé par le noyau)
- **SIGKILL (9)** : pour terminer un processus
- **SIGSEGV (11)** : accès mémoire invalide (envoyé par le noyau)
- **SIGTERM (15)** : argument par défaut de la commande kill
- **SIGCHLD (17)** : envoyé par le noyau lors de la mort d'un fils
- **SIGSTP (18)** : suspend un processus
- **SIGCONT (19)** : redémarre un processus suspendu
- **SIGUSR1 (30)** : libre, sémantique définie pour chaque processus
- **SIGUSR2 (31)** : libre, sémantique définie pour chaque processus

6



Les signaux

- ❑ Deux signaux bien utiles
 - `SIGSTP` : demande au système de suspendre un processus
 - `SIGCONT` : demande au système de le redémarrer

- ❑ Bash utilise ces signaux :
 - `control-z` : envoie un `SIGSTP` au processus au premier plan
 - `bg` et `fg` : envoient un `SIGCONT` au processus stoppé (rappel : `bg` background, `fg` foreground)

7



Les signaux

- ❑ Un processus peut attacher un gestionnaire dit de signal avec
 - `trap expression sig`

- ❑ Un processus peut envoyer un signal à un destinataire avec
 - `kill -sig pid`

- ❑ Où
 - `expression` : expression quelconque bash
 - `sig` : numéro de signal (nombre ou symbole comme `USR1`)
 - `pid` : PID du processus destinataire

8



Principe de fonctionnement

```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh

```
$
```

Terminal 1

```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$
```

Terminal 2

9



Principe de fonctionnement

```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh

```
$
```

Terminal 1



```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$ ./recepteur.sh
```

Terminal 2

Terminal 2 : lancement de `recepteur.sh`

10



Principe de fonctionnement

```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$
```

Terminal 1

```
$ ./recepteur.sh
```

Terminal 2

recepteur.sh attache le gestionnaire 'echo coucou' à USR1



Principe de fonctionnement

```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$
```

Terminal 1

```
$ ./recepteur.sh  
PID: 52075
```

Terminal 2

recepteur.sh affiche son PID



Principe de fonctionnement

```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$
```

Terminal 1

```
$ ./recepteur.sh  
PID: 52075
```

Terminal 2

recepteur.sh exécute la boucle infinie



Principe de fonctionnement



```
#!/bin/bash  
  
kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash  
  
trap 'echo coucou' USR1  
  
echo "PID: $$"  
  
while true; do  
    sleep 1  
done
```

recepteur.sh

```
$ ./emetteur.sh 52075
```

Terminal 1

```
$ ./recepteur.sh  
PID: 52075
```

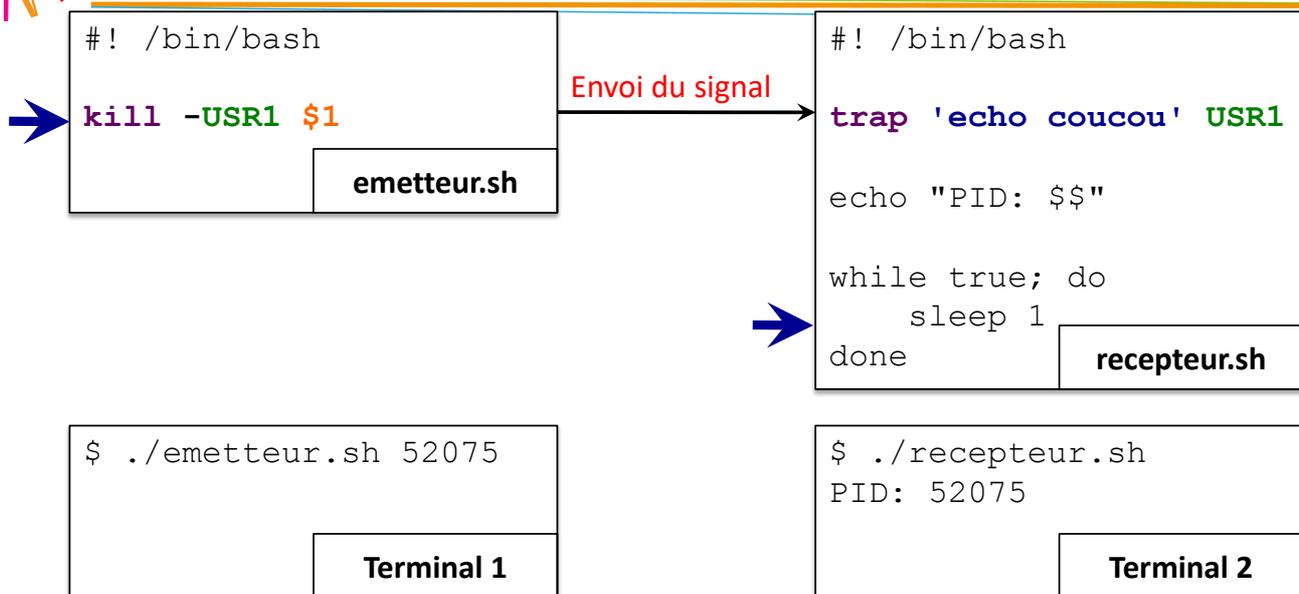
Terminal 2

Terminal 1 : lancement de emetteur.sh



Principe de fonctionnement

USR1 en attente



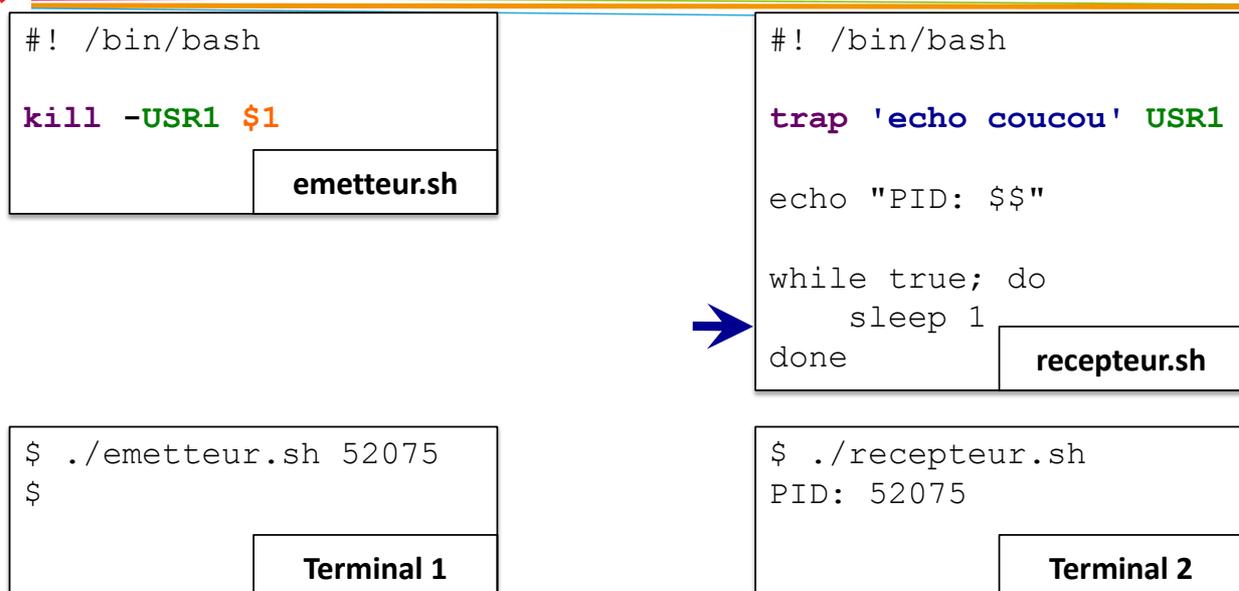
emetteur.sh envoie le signal USR1 à recepneur.sh

15



Principe de fonctionnement

USR1 en attente



emetteur.sh termine
(l'ordre entre emetteur.sh et recepneur.sh est aléatoire)

16



Principe de fonctionnement

```
#!/bin/bash

kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash

trap 'echo coucou' USR1

echo "PID: $$"

while true; do
  sleep 1
done
```

recepteur.sh



```
$ ./emetteur.sh 52075
$
```

Terminal 1

```
$ ./recepteur.sh
PID: 52075
coucou
```

Terminal 2

recepteur.sh reçoit le signal
⇒ le système déroute l'exécution de recepteur.sh vers le gestionnaire
⇒ affiche coucou



Principe de fonctionnement

```
#!/bin/bash

kill -USR1 $1
```

emetteur.sh



```
#!/bin/bash

trap 'echo coucou' USR1

echo "PID: $$"

while true; do
  sleep 1
done
```

recepteur.sh

```
$ ./emetteur.sh 52075
$
```

Terminal 1

```
$ ./recepteur.sh
PID: 52075
coucou
```

Terminal 2

À la fin du gestionnaire du signal, l'exécution reprend là où elle s'était arrêtée



Notions clés

❑ Les signaux

- Mécanisme de communication à base de messages
- Message = nombre entre 1 et 31
- Ordre de réception aléatoire
- Perte possible en cas d'envoi multiple du même numéro de signal
- `kill -sig pid` : envoie un signal `sig` à `pid`
- `trap expr sig` : associe `expr` à la réception d'un signal `sig`