INSTITUT Mines-Télécom

TELECOM SudParis

CI6 - Communication entre processus : tubes et signaux

CSC 3102

Introduction aux systèmes d'exploitation

Gaël Thomas

13/10/2015

Présentation du cours

- Contexte : comprendre comment les processus interagissent entre eux

- Objectifs :
  - Savoir utiliser les tubes et les signaux
  - Comprendre les notions de section critique et d'exclusion mutuelle

- Notions clés :
  - Signaux, tubes, fichier partagé, exclusion mutuelle, section critique

Pourquoi les processus communiquent ?

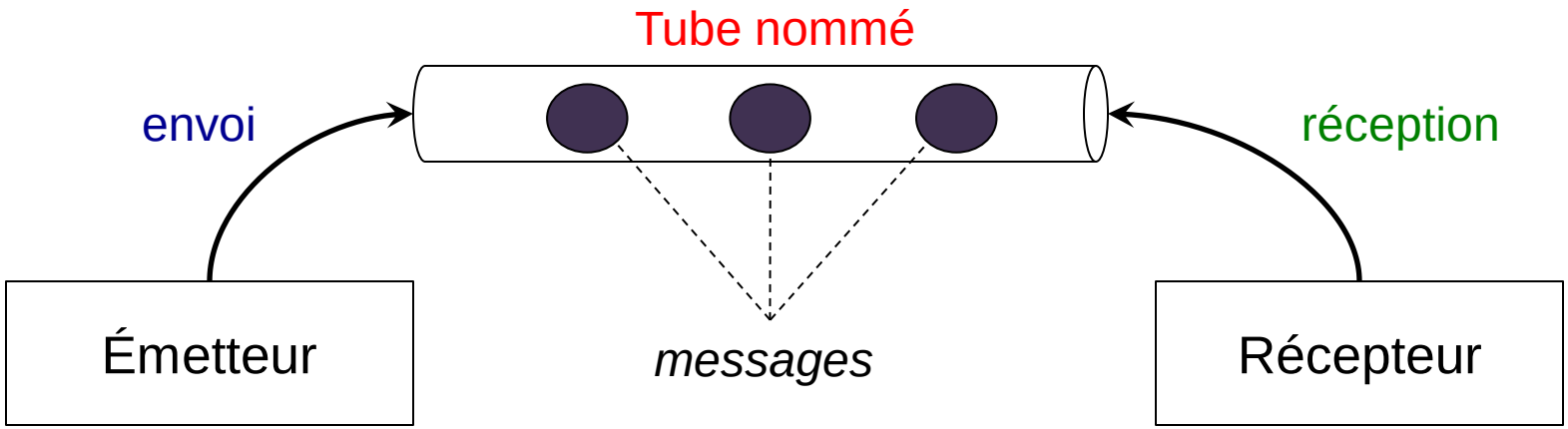
- Principalement pour rendre de nouveaux services
  - Une base de données + un serveur Web
  - Un processus d'affichage de notifications pour les autres processus
- Pour pouvoir répartir physiquement les processus
  - Un processus d'affichage et un moteur de jeux
    (moteur distant en mode jeu multi-joueur, moteur local en mode déconnecté)
- Pour gérer la vie des processus
  - Pour attendre la fin d'un processus, pour arrêter un processus

Mécanismes étudiés dans le cours

- Les tubes
  - Communication par envoi de message via un canal
  - Message = donnée quelconque
  - Pas de perte de message, réception dans l'ordre d'émission
- Les signaux
  - Communication par envoi de message direct
  - Message = entier entre 1 et 31
  - Perte de message, pas d'ordre chez le récepteur
- Les fichiers partagés
  - Deux processus accèdent à un fichier partagé

Tubes nommés

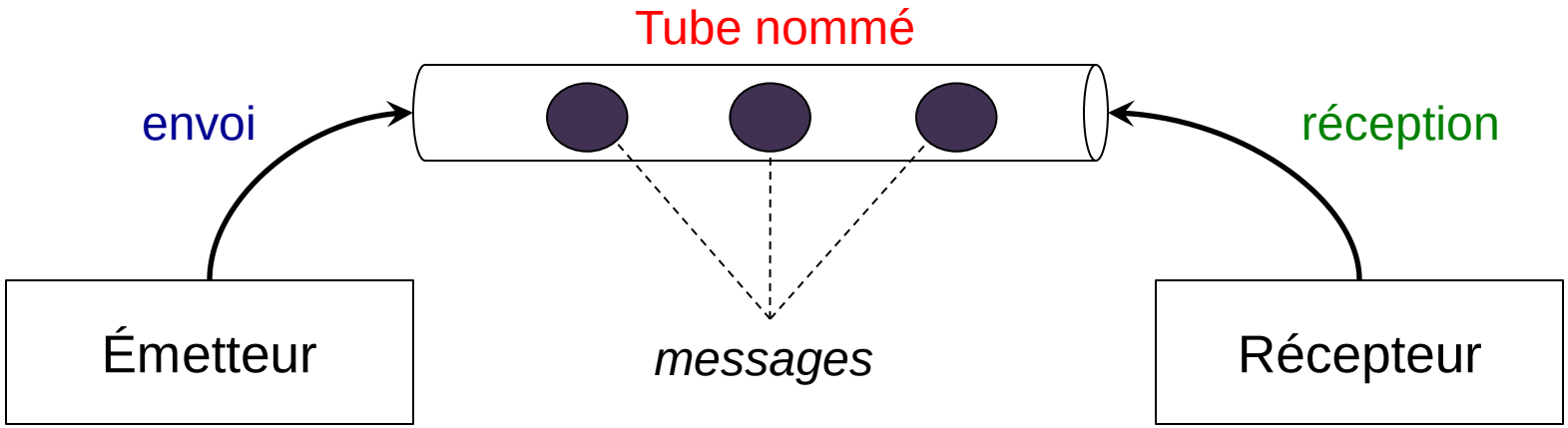
- Canal de communication = fichier spécial appelé un tube



- Principe :
  - Un tube est un fichier dans le système de fichiers
  - L'émetteur écrit dans le tube
  - Le récepteur lit dans le tube

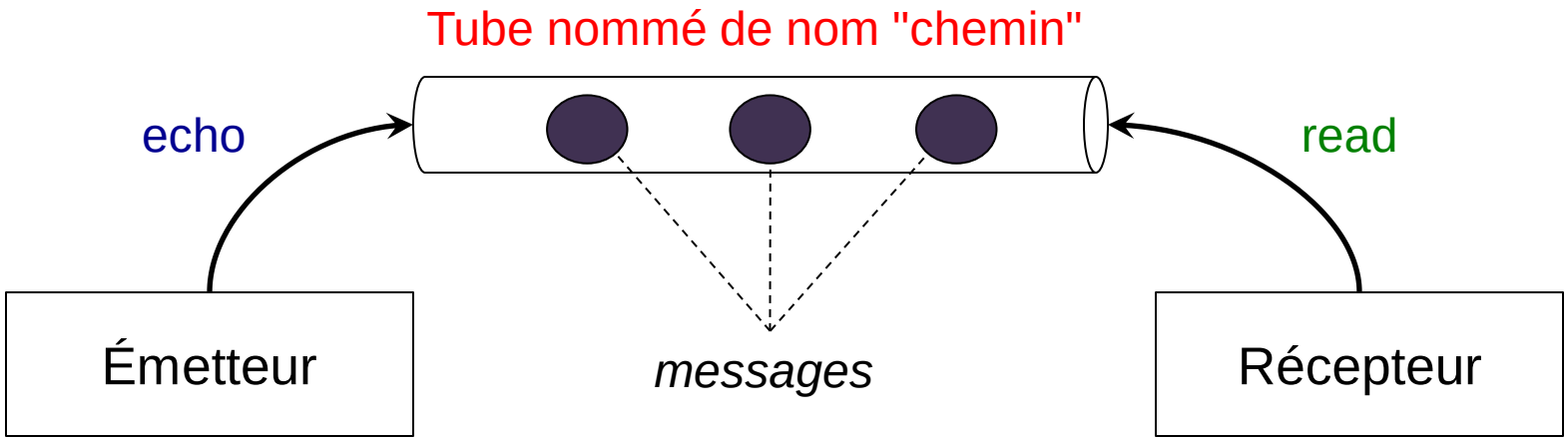
Tubes nommés

- Canal de communication = fichier spécial appelé un tube



- Nota bene :
  - Les messages arrivent dans l'ordre d'émission
  - Emetteur bloque tant qu'il n'existe pas de récepteur
  - Récepteur bloque tant qu'il n'existe pas d'émetteur

Tubes nommés



Mise en œuvre :

- `mkfifo chemin` : crée un tube nommé (visible avec `ls`)
- `echo message > chemin` : écrit dans le tube nommé
- `read message < chemin` : lit à partir du tube nommé
- `rm chemin` : détruit le tube nommé

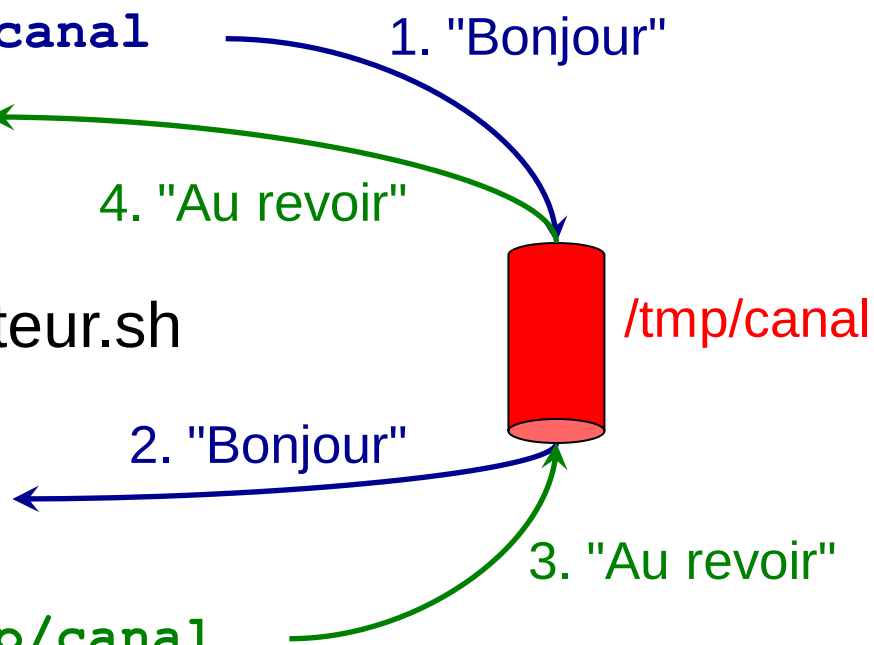
Tubes nommés – exemple

pipe_emetteur.sh

```
#!/bin/bash
mkfifo /tmp/canal
echo "Bonjour" > /tmp/canal
read a < /tmp/canal
echo "Reçoit $a"
```

pipe_recepteur.sh

```
#!/bin/bash
read a < /tmp/canal
echo "Reçoit $a"
echo "Au revoir" > /tmp/canal
```



Tubes anonymes

- Tube anonyme : comme tube nommé, mais sans nom

- Le « | » entre deux processus shell crée un tube anonyme

```
cmd_gauche | cmd_droite
```

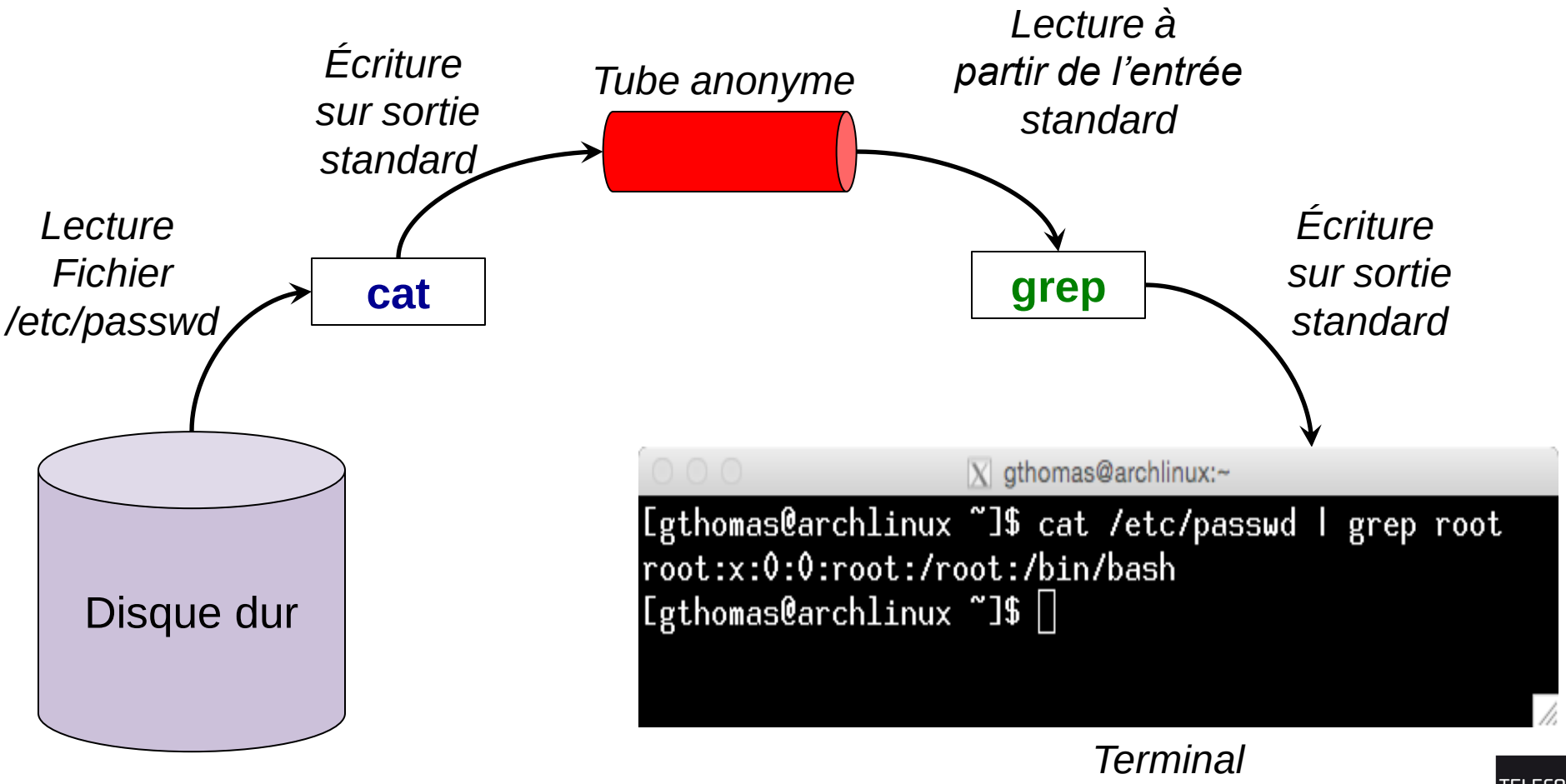
  - Sortie standard de `cmd_gauche` connectée au tube
  - Entrée standard de `cmd_droite` connectée au tube

- Équivalent à :

```
mkfifo tube-avec-nom
cmd_gauche > tube-avec-nom &
cmd_droite < tube-avec-nom
```

Tubes anonymes – exemple

```
cat /etc/passwd | grep root
```

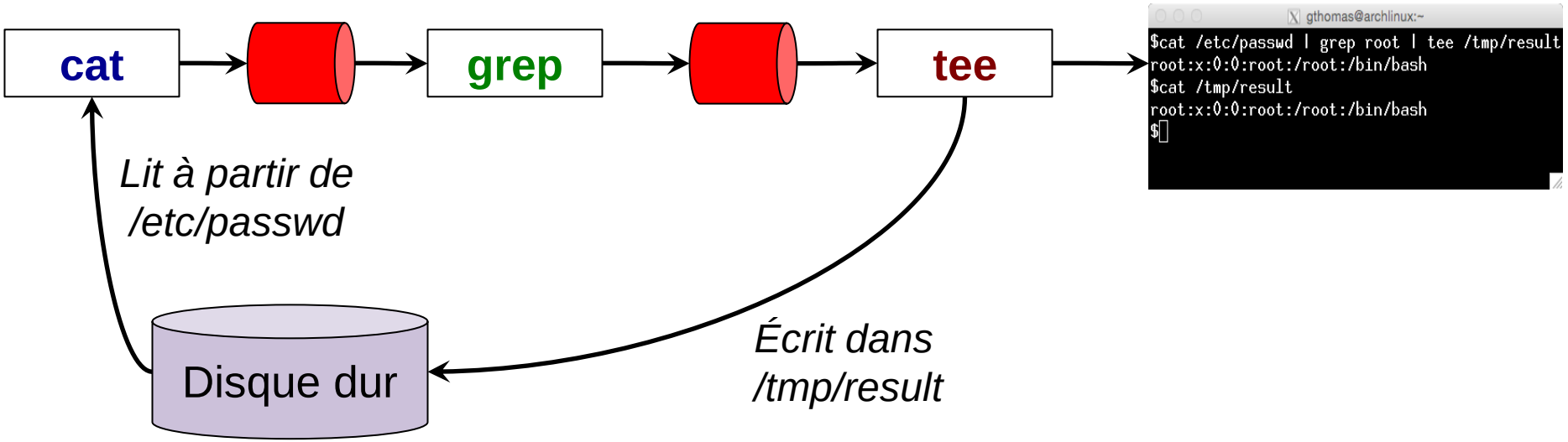


Terminal

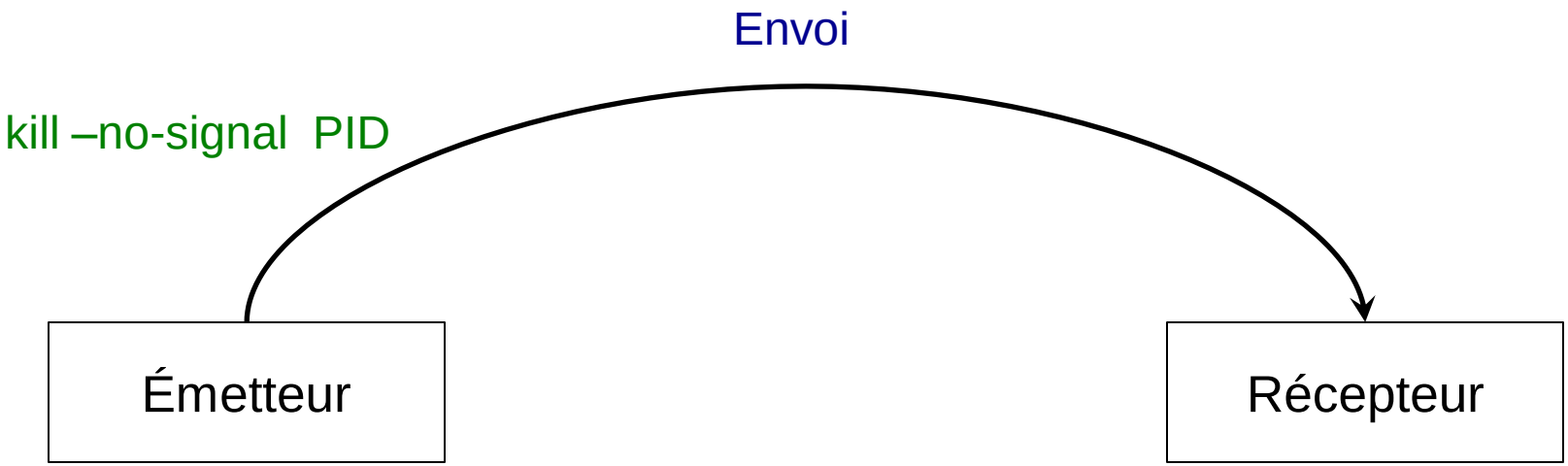
Commande utile : `tee`

- Écrit les données lues à partir de l'entrée standard
  - Sur la sortie standard
  - Et dans un fichier passé en argument

- Exemple : **`cat`** `/etc/passwd | ` **`grep`** `root | ` **`tee /tmp/result`**



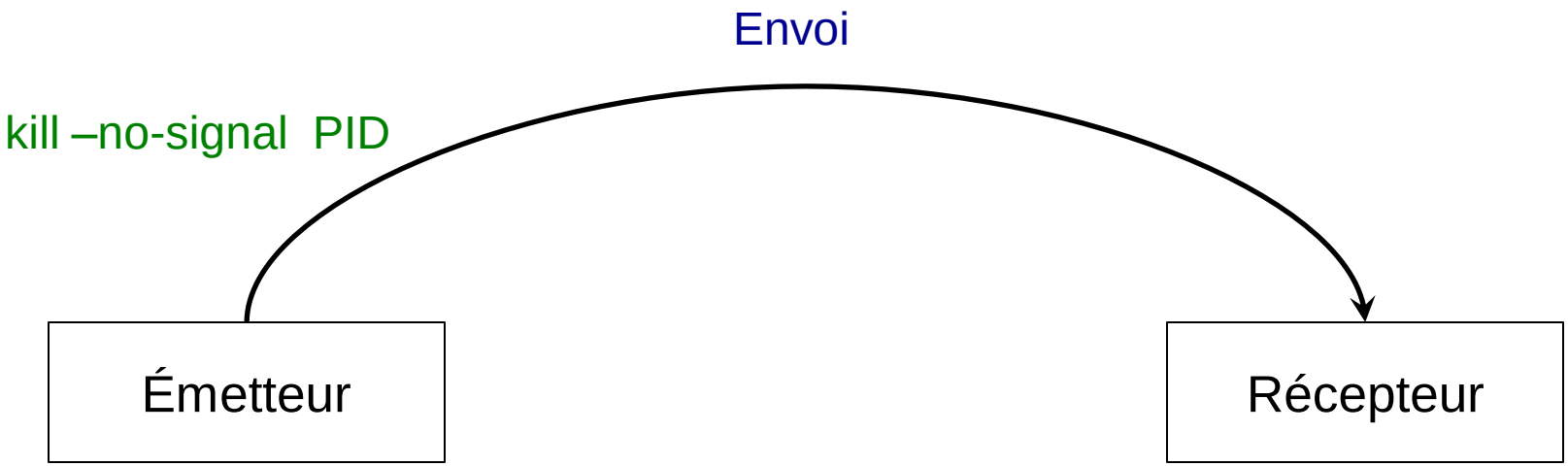
Signaux



- Principe :
  - Émetteur envoie un message à un proc. identifié par son PID
  - Une routine de réception est automatiquement invoquée chez le récepteur dès que le signal arrive
  - Par défaut, cette routine tue le récepteur

    (sauf dans le cas du SIGCHLD)

Signaux



- Nota bene :
  - Un message est limité à un nombre compris entre 1 et 31
  - Les messages peuvent se perdre

    (Cas de l'émission de deux signaux identiques avant réception)
  - Les messages arrivent dans n'importe quel ordre

Les signaux

Quelques exemples (man 7 signal) :

- SIGHUP (1) : fermeture terminal ⇒ à tous les processus attachés
- SIGINT (2) : Ctl-C dans un terminal ⇒ au processus premier plan
- SIGQUIT (3) : souvent Ctl-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) : via la commande kill, 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

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 :
  - `Ctrl-Z` : envoie un `SIGSTP` au processus au premier plan
  - `bg` : envoie un `SIGCONT` au processus stoppé

Les signaux

- Un processus peut attacher un gestionnaire de signal à un numéro de signal

```
trap commande signal
```

- Exemple : quitte après la réception de 3 SIGINT

```
#! /bin/bash
i=0
trap 'echo hello $i; i=$(expr $i + 1)' SIGINT
while [ $i -lt 3 ]; do sleep 1; done
```

- Envoi du signal avec `kill -INT PID`, où `PID` est l'identifiant du destinataire

Les signaux

- Attention, par défaut, un processus lancé en tâche de fond n'attrape pas les signaux avec la commande `trap`

- Pour quand même attraper les signaux, il faut utiliser

```
set -m
```

Attente de la fin d'un processus

- Commandes d'attente active de terminaison des fils directs :
  - `wait` : attendre la fin de tous les fils directs
  - `wait pid1 pid2...` : attendre la fin de pid1 et pid2 et …
- Le noyau du système d'exploitation envoie aussi un signal `SIGCHLD` au père quand un de ses fils se termine
  - Par défaut, bash masque les signaux `SIGCHLD`
  - Possibilité de les recevoir avec `set -o monitor`

Attente de la fin d'un processus

```
#!/bin/bash

emacs &          # lance emacs en tâche de fond
p=$!             # $!: pid du dernier processus lancé
echo "$p démarre"

wait $p          # attend fin de $p
echo "Fin de $p, avec code de retour $?
        # le $? est l'argument du exit de emacs
        # (ou du return de son main)
```

Notions clés

- Tubes (nommés et anonymes)
  - Fichier spécial dans le système de fichier
  - Envoi de messages de taille quelconque, ordre de réception = ordre d'émission, pas de perte de message
  - `mkfifo nom-tube` : crée un tube nommé mon-tube
- Signaux
  - Mécanisme de communication à base de messages
  - Message = nombre entre 1 et 31, ordre quelconque, perte
  - `kill -sig pid` : envoie un signal
  - `trap expr sig` : associe expr à la réception d'un signal sig
- Attente de la fin d'un processus
  - `wait`

À vous de jouer!