

Cours de Réseau et communication Unix n°4

Edouard THIEL

Faculté des Sciences

Université d'Aix-Marseille (AMU)

Septembre 2016

Les transparents de ce cours sont téléchargeables ici :

<http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/rezo/>

Lien court : <http://j.mp/rezocom>

Plan du cours n°4

1. Les tubes nommés
2. Les sockets
3. Les sockets dans le domaine Unix

1 - Les tubes nommés

Points de communication du Système de Gestion de Fichiers.

Tubes

Tube anonyme

Pour communiquer par un tube anonyme, deux processus ont besoin d'une **relation de parenté** pour connaître les descripteurs.

Disparaît avec les processus

Tube nommé

Type spécial de fichier

- ▶ Persistant dans le SGF
- ▶ Plus de parenté nécessaire
- ▶ Droits avec `chmod`
- ▶ Le buffer est en mémoire, pas sur le disque

Étapes

Les étapes de l'existence d'un tube nommé :

1. Création
2. Ouverture
3. Lecture ou écriture
4. Fermeture
5. Suppression

Création d'un tube nommé en bash

Commande :

```
mkfifo [-m mode] fichier1 ...
```

Crée un tube nommé pour chacun des fichiers en argument

En option : les droits (syntaxe chmod)

Droits par défaut : 0666 - umask

Exemple :

```
<> mkfifo -m u=rw,go=r tube0
<> ls -l tube0
prw-r--r-- 1 thiel thiel 0 oct.  2 18:29 tube0|
```

Création d'un tube nommé en C

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);
```

Crée un tube nommé de nom `pathname`,
avec les droits en octal : `mode & ~umask`

Renvoie 0 succès, -1 erreur

Une fois créé, un tube nommé persiste tant que l'on ne l'efface pas explicitement (`rm` ou `unlink`).

Ouverture d'un tube nommé


Un tube nommé peut être ouvert en lecture ou écriture :

- ▶ une ou plusieurs fois ;
- ▶ par tout processus disposant des droits suffisants.

Ouverture par `open` :

```
int open(const char *pathname, O_RDONLY)    en lecture
                                     O_WRONLY) en écriture
```

Renvoie le descripteur ≥ 0 , sinon -1 erreur.

 `open` bloquant s'il n'y a pas au moins 1 lecteur et 1 écrivain
→ synchronisation

Rendre l'ouverture non bloquante :

```
open ( .., .. | O_NONBLOCK)
```


Lecture ou écriture

En théorie, les 2 extrémités d'un tubes doivent avoir été ouvertes avant de passer des données.

```
#include <unistd.h>
ssize_t read (int fd,          void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

Même comportement que pour les tubes anonymes :

- ▶ buffer de taille limitée, peut être vide ou plein ;
- ▶ caractères agglutinés ;
- ▶ write sur un tube sans lecteur → SIGPIPE et renvoie -1.
- ▶ read sur un tube vide sans écrivain → renvoie 0.

(En mode non bloquant, comportement différent)

Fermeture d'un tube nommé

Idem tube anonyme :

```
#include <unistd.h>
int close(int fd);
```

- Si le dernier lecteur fait `close`,
du côté écrivain : `write` → `SIGPIPE`, renvoie `-1`, `errno = EPIPE`
- Si le dernier écrivain fait `close`,
du côté lecteur :
 - ▶ `read` réussit tant que le buffer du tube n'est pas vide
 - ▶ tous les `read` suivants renvoient `0` (fin de tube)
 - ▶ si on fait `close`, `read` renvoie `-1` avec `errno = EBADF`

Exemple avec cat

Terminal 1	Terminal 2
\$ mkfifo tube0	
\$ cat > tube0	\$
	\$ cat < tube0
ez-draw	
	ez-draw
^D	
\$	\$
\$ cat > tube0	\$
helium	\$
^D	\$
	\$
\$	\$ cat < tube0
\$	helium
	\$

Suppression d'un tube nommé

- En bash : `rm pathname`
- En C :

```
#include <unistd.h>
int unlink(const char *pathname);
```

Renvoie 0 succès, -1 échec.

Effet commun :

- ▶ supprime ce nom dans le répertoire
- ▶ décrémente le compteur de liens matériels dans le i-node correspondant sur le disque.

→ Si le compteur de liens matériels est à 0 et si le compteur de référence dans TIM est à 0, alors le i-node est libéré.

Remarques

- Si 2 processus dialoguent via un tube nommé, la suppression du tube est sans effet sur le dialogue (Il disparaîtra physiquement après les `close`).
→ On peut supprimer un tube nommé temporaire après `open`
- Les tubes nommés ne sont pas bidirectionnels
→ Il faut 2 tubes pour dialoguer
- Si on ouvre un tube nommé plusieurs fois
 - ▶ en écriture : concaténation
 - ▶ en lecture : bagarre
- On peut utiliser `cat` pour déboguer un programme

2 - Les sockets

Descripteurs de fichiers spéciaux pour le réseau.

Introduction aux sockets

Littéralement "prises" (de courant)

Mécanisme général de communication entre processus

- ▶ locaux (sur une même machine)
- ▶ distants (sur machines différentes)

Introduit dans BSD 4.2 en 1981

- Entre processus locaux : sockets dans le "domaine Unix" via un nom de socket dans le SGF (à l'instar des tubes nommés).
- Entre processus distants : sockets dans le "domaine Internet" avec une adresse IP et un numéro de port.

Différences par rapport aux tubes nommés

Un tube nommé est "un point de rendez-vous", connu des 2 processus.

Une socket est une "adresse unique" : pour que 2 processus dialoguent,

- ▶ chacun crée sa propre socket,
- ▶ puis s'adresse à l'autre socket.

Par analogie : numéro de téléphone, ou adresse postale

Communication bidirectionnelle

Un dialogue = 2 sockets

Autres propriétés

Une socket est représentée par un descripteur

→ peut être hérité, dupliqué, redirigé, scruté, etc

On fait d'abord l'ouverture d'une socket, puis son nommage.

- ▶ l'ouverture crée un descripteur de type socket, sans adresse
- ▶ le nommage le rattache à une adresse

Domaines

Une socket se spécialise dans un domaine de communication.

Les domaines définis dans `<sys/socket.h>` pour Linux :

Nom	Description	Man page
AF_UNIX	Communications locales	unix
AF_INET	Internet IPv4	ip
AF_INET6	Internet IPv6	ipv6
AF_IPX	Novell IPX	
AF_NETLINK	Communication avec le noyau	netlink
AF_X25	X25 (minitel, etc ; désuet)	x25
AF_AX25	Radio amateur AX.25	
AF_ATMPVC	Raw ATM Permanent Virtual Conn.	
AF_APPLETALK	Appletalk	ddp
AF_PACKET	Interface bas niveau (root)	packet

Types de sockets

Il existe différents types de sockets, selon la nature des communications que l'on veut réaliser.

Les propriétés en jeu sont :

- P1 Fiabilité : tout message envoyé arrive au destinataire
- P2 Séquencement : les messages arrivent dans l'ordre d'envoi
- P3 Non duplication : un message envoyé n'arrive qu'une fois
- P4 Datagramme : un message de taille limitée, reçu en 1 fois, sans troncature ni concaténation (vs flux)
- P5 Connexion : nécessité d'établir une connexion
- P6 Priorité : possibilité d'envoyer des messages prioritaires

Principaux types de sockets

Constantes symboliques dans `<sys/socket.h>`

<code>SOCK_RAW</code>	datagrammes, bas niveau (root)
<code>SOCK_DGRAM</code>	datagrammes, non connecté, non fiable
<code>SOCK_STREAM</code>	flux, connecté, fiabilité max (P1+P2+P3)
<code>SOCK_RDM</code>	"Reliable Delivery Messages" : datagrammes, non connecté, fiabilité max
<code>SOCK_SEQPACKET</code>	datagrammes, connecté, fiabilité max
<code>SOCK_DCCP</code>	"Datagram Congestion Control Protocol" : bas niveau (linux only)

Création d'une socket

Pour créer une socket :

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

Renvoie ≥ 0 le descripteur de la socket, sinon -1 erreur

domain	AF_UNIX, AF_INET, etc
type	SOCK_DGRAM, SOCK_STREAM, etc
protocol	0 pour le protocole par défaut : <ul style="list-style-type: none">- protocole UDP pour SOCK_DGRAM- protocole TCP pour SOCK_STREAM

Fermeture d'une socket : avec `close`

Attachement à une adresse

Après sa création, une socket n'est qu'un descripteur, relié à aucune adresse, et connu que du processus.

Attachement à une adresse = étape nécessaire pour qu'un autre processus puisse dialoguer (il doit avoir sa propre socket).

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

Renvoie 0 succès, -1 erreur.

sockfd	la socket
addr	adresse d'un struct contenant l'adresse de la socket casté en (struct sockaddr*)
addrlen	taille du struct effectivement passé

3 - Les sockets dans le domaine Unix

Sockets identifiées par un nom dans le SGF.

Communication locale

Réservées à la communication sur une même machine.

Ne passe pas par la pile réseau du système → efficacité

1) Création : `int socket (AF_UNIX, int type, 0);`

2) Attachement avec `bind` à une adresse dans le SGF,
de type `struct sockaddr_un`

`bind` crée un fichier de type socket dans le SGF

Visible avec `ls -l` : 's' en premier

Suppression : par `rm` ou `unlink`

Type struct sockaddr_un

Le struct `sockaddr` est générique, il faut lui substituer un type d'adresse propre au domaine :

```
#include <sys/socket.h>
#include <sys/un.h>

#define UNIX_PATH_MAX    108           /* sous Linux */

struct sockaddr_un {
    sa_family_t sun_family;           /* AF_UNIX */
    char        sun_path[UNIX_PATH_MAX]; /* pathname */
};
```

Le champ `sun_path` contient le nom dans le SGF que l'on veut donner à la socket ; ne doit pas déjà exister.

Exemple complet

```
/* Inclure stdio.h, stdlib.h, unistd.h, string.h,  
    sys/types.h, sys/socket.h, sys/un.h */  
  
int sockfd; struct sockaddr_un adr;  
  
sockfd = socket (AF_UNIX, SOCK_DGRAM, 0);  
if (sockfd < 0) { perror ("socket"); exit (1); }  
  
adr.sun_family = AF_UNIX;  
strncpy (adr.sun_path, "/tmp/sock0", UNIX_PATH_MAX);  
adr.sun_path[UNIX_PATH_MAX-1] = 0;  
  
if (bind (sockfd, (struct sockaddr *) &adr, sizeof(adr)) < 0)  
{ perror ("bind"); exit (1); }  
  
/* ... */  
  
close (sockfd);  
unlink (adr.sun_path);
```

Liste des socket unix

La commande `netstat` affiche la liste des sockets du système et leur état.

```
netstat -x          Linux
netstat -f unix    MacOS
```

```
<> netstat -x | head -8
```

```
Sockets du domaine UNIX actives (sans serveurs)
```

Proto	Type	State	I-Node	Chemin
unix	DGRAM		1903	/var/run/wpa_supplicant/eth1
unix	DGRAM		8326	/dev/log
unix	STREAM	CONNECTE	70892	/tmp/.X11-unix/X0
unix	STREAM	CONNECTE	70225	
unix	STREAM	CONNECTE	68411	/tmp/dbus-Z7bOLyFUIN

Sous Windows

Sous Windows, l'équivalent des sockets existe sous le nom de **winsocks**.

```
#include <winsock2.h>

SOCKET WINAPI socket(
    _In_ int af,          // AF_INET, .. mais pas AF_UNIX
    _In_ int type,       // SOCK_DGRAM, SOCK_STREAM, ..
    _In_ int protocol    // 0
);

int bind(
    _In_ SOCKET s,
    _In_ const struct sockaddr *name, // à caster
    _In_ int namelen
);
```