Cours de Réseau et communication Unix n°7

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°7

1. Notions de port et de service
2. Sockets Internet
3. Résolution de nom
4. Sockets IPv6

1 - Notions de port et de service

Propre au modèle internet.

Modèle en couches internet

Adresse IP = point d'entrée dans une machine (couche 3) par lequel passent les paquets IP encapsulant les différents protocoles (ICMP, UDP, TCP, etc) de la couche 4.

Ces paquets ne sont pas adressés à un processus en particulier car la notion de processus est propre à Unix, non portable.

Mécanisme général choisi : notions de service et de port

Notion de service

On s'adresse à un service disponible sur une machine donnée (service web, mail, ftp, ssh, etc).

Avantages :

- plusieurs services peuvent être assurés par un même processus ;
- un service peut être assuré par plusieurs processus ;
- on peut lancer ou relancer à la demande un serveur qui assurera un service demandé.

Notion de port

Solution retenue pour coder un service dans un datagramme UDP ou un segment TCP :

port = entier entre 1 et 65535

Ports de 1 à 1024 : réservés aux applications internet standard (nécessitent droits root niveau serveur)

21	ftp	80	http
22	ssh	443	https
23	telnet	389	ldap
25	smtp	873	rsync
110	pop3	995	pop3s
143	imap2	993	imaps

Correspondance service $\longleftrightarrow$ port : fichier ascii `/etc/services`

2 - Sockets Internet

Sockets du domaine `AF_INET`

Types de sockets

Protocole ICMP	$\rightarrow$ type	`SOCK_RAW`
UDP/IP		`SOCK_DGRAM`
TCP/IP		`SOCK_STREAM`

Tout ce qu'on a vu avec les sockets Unix en UDP et TCP reste valable pour les sockets internet :

squelettes de clients/serveurs identiques,

sauf pour ce qui concerne le domaine :

on remplace un fichier de type socket par un couple (adresse IP, numéro de port).

Squelettes de client/serveur

Serveur UDP/IP	`socket`
	`bind`
	`recvfrom`
	`sendto`
	`close`

Client UDP/IP	`socket`
	`bind`
	`sendto`
	`recvfrom`
	`close`

Serveur TCP/IP	`socket`
	`bind`
	`listen`
	`accept`
	`read`
	`write`
	`close`

Client TCP/IP	`socket`
	`bind`
	
	`connect`
	`write`
	`read`
	`close`

Format d'adresses

Le format d'adresses pour bind en IPv4 est :

```
#include <sys/socket.h>
#include <netinet/in.h>

struct sockaddr_in {
    sa_family_t    sin_family; // AF_INET
    in_port_t      sin_port;   // in network byte order
    struct in_addr sin_addr;   // internet address
};

struct in_addr {
    uint32_t       s_addr;     // in network byte order
};
```

Endianness (boutisme)

Terme provenant des Voyages de Gulliver de Jonathan Swift.

Les entiers peuvent être représentés de plusieurs façons :

- little endian (petit boutisme) : octet de poids faible en premier (à la plus petite adresse) ; processeurs Intel, Dec Alpha, etc
- big endian (grand boutisme) : octet de poids fort en premier (à la plus petite adresse) ; processeurs Motorola, Sparc, etc
- middle-endian (moyen-boutisme) : mélange des deux ordres ; PDP-11, certains processeurs ARM

Network byte order (ordre réseau) : big-endian

Conversions

On distingue l'ordre réseau (network)
de l'ordre local (host)

Dans un `struct sockaddr_in`, les champs `sin_port` et `sin_addr.s_addr` sont toujours dans l'ordre réseau.

Primitives de conversion :

```
#include <arpa/inet.h>

uint16_t htons (uint16_t hostshort);
uint16_t ntohs (uint16_t netshort);
uint32_t htonl (uint32_t hostlong);
uint32_t ntohl (uint32_t netlong);
```

Exemple

Pour demander le port 10000 :

```
struct sockaddr_in adr;
adr.sin_port = htons (10000);
```

Pour afficher le numéro de port :

```
printf ("Port = %d\n", ntohs (adr.sin_port));
```

Remarque : 10000 = 0x2710 ; 0x1027 = 4135

Demander un port libre

On ne peut pas attacher plusieurs sockets à un même port, ni parfois réutiliser tout de suite un port fermé.

On peut demander à `bind` un port libre par 0 :

```
struct sockaddr_in adr;
adr.sin_port = htons (0);
```

puis récupérer le port attribué par `bind` avec

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

par exemple (après `bind`) :

```
socklen_t addrlen = sizeof (struct sockaddr_in);
if (getsockname (soc, (struct sockaddr*) &addr,
                 &addrlen) < 0) exit (1);
printf ("Port attribué = %d\n", ntohs (adr.sin_port));
```

Adresse IP locale

Pour créer une socket locale, il faut spécifier l'adresse IP locale :

`INADDR_ANY`	0.0.0.0	: toutes les interfaces disponibles
`INADDR_LOOPBACK`	127.0.0.1	: localhost

On écrit donc

```
adr.sin_family = AF_INET;
adr.sin_port = htons (port);  // 0 pour port libre
adr.sin_addr.s_addr = htonl (INADDR_ANY);

if (bind (soc, (struct sockaddr*) &addr,
        sizeof (struct sockaddr_in)) < 0) {
   perror ("bind"); exit (1);
}
```

3 - Résolution de nom

Transformation d'un nom de machine en adresse IP.

Nom de machine

- Plus commode de désigner une machine par un nom que par une adresse IP.
- Permet de modifier l'adresse IP de manière transparente.
- Permet d'associer plusieurs adresses IP à un nom (load-balancing).

Nécessite

- un système d'annuaire : les serveurs de nom (DNS) ;
- un mécanisme pour les contacter ;
- un système de nommage souple.

Noms courts

Nom générique de la machine elle-même : `localhost`

Connaître le nom de la machine elle-même : commandes `hostname` ou `uname -n`

Spécifier le nom (selon système) : fichiers `/etc/hostname` et `/etc/hosts`

Noms complets

FQDN : Fully Qualified Domain Name

`nom.[sous-domaines.]domaine.TLD[.]`

Nom : une machine possède un nom principal, éventuellement des noms secondaires.

TLD : Top Level Domain : .com, .org, .fr, etc.

Les TLD sont gérés par des organismes différents pour attribuer les noms de domaine (pas d'organisme suprême) :

- INTERNIC : .com, .org, .net
- AFNIC : .fr
- etc (plus de 300 domaines)

Acheter un nom de domaine : passer par un registrar

Interroger les TLD

```
<> whois -H korben.info
...
Domain Name:KORBEN.INFO
Created On:08-Nov-2005 21:37:23 UTC
Last Updated On:29-Dec-2012 20:45:11 UTC
Expiration Date:08-Nov-2020 21:37:23 UTC
Sponsoring Registrar:Gandi SAS (R191-LRMS)
...
Registrant Name:Manuel Dorne
Registrant Organization:
Registrant Street1:Whois Protege / Obfuscated whois
Registrant Street2:Gandi, 15 place de la Nation
Registrant Street3:
Registrant City:Paris
...
Name Server:LUCY.NS.CLOUDFLARE.COM
Name Server:ANDY.NS.CLOUDFLARE.COM
...
```

Bases de noms

- Base de données locale : fichier `/etc/hosts`

- Serveurs de noms : DNS (RFC 882, 1983)
décentralisés, situés dans le monde entier, se mettent à jour en dialoguant avec d'autres DNS.

- Configuration locale (selon système) : `/etc/resolv.conf`
  - indique comment résoudre les noms
  - DNS principal, secondaire

Exemples de DNS : Google public 8.8.8.8 et 8.8.4.4 ; OpenDNS 208.67.222.222

Interroger des DNS : commandes `nslookup` et `dig`

Résolution d'adresse

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

Si l'adresse `name` est sous la forme "a.b.c.d", `gethostbyname` fait une simple conversion binaire.

Sinon, `gethostbyname` consulte des DNS (selon configuration système) pour demander l'adresse IP.

Renvoie un `struct hostent*` contenant l'adresse IP sous forme binaire, sinon `NULL` au bout d'un certain délai.

En cas d'erreur, ne pas utiliser `errno` et `perror` mais `extern int h_errno;` et `void herror(const char *s);`

Informations renvoyées par gethostbyname

```
#include <netdb.h>

struct hostent {
    char  *h_name;        // Nom officiel de l'hôte
    ...
    int    h_length;      // Longueur de l'adresse
    char **h_addr_list;   // Liste d'adresses dans
                          // l'ordre d'octets réseau
}
#define h_addr  h_addr_list[0] // pour compatibilité
```

Après `gethostbyname`, l'adresse IP recherchée est dans `h_addr`.

Recopie d'une adresse IP
de `struct hostent *hp` vers `struct sockaddr_in adr` :

```
memcpy (&adr.sin_addr.s_addr, hp->h_addr, hp->h_length);
```

Fonctions utilitaires

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr in);
```

Convertit une adresse IPv4 (dans l'ordre réseau) en chaîne de caractère `"a.b.c.d"` (en mémoire statique).

4 - Sockets IPv6

Nouveau type d'adresse, ne change pas la notion de port.

L'espace des ports IPv6 est partagé avec IPv4.

Différences avec IPv4

Domaine `AF_INET6` pour `socket`

Format d'adresses différent pour `bind` :

```
#include <sys/socket.h>
#include <netinet/in.h>

struct sockaddr_in6 {
    sa_family_t     sin6_family;   // AF_INET6
    in_port_t       sin6_port;     // port number
    uint32_t        sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    uint32_t        sin6_scope_id; // Scope ID
};
struct in6_addr {
    unsigned char   s6_addr[16];   // IPv6 address
};
```

`gethostbyname` : compatible IPv6, appel identique

Fonctions utilitaires

La fonction `inet_ntoa` est remplacée par :

```
#include <arpa/inet.h>

const char *inet_ntop (int domain, const void *src,
                       char *dst, socklen_t size);
```

Convertit une adresse IPv6 (dans l'ordre réseau) en chaîne de caractère `"a:b:c:d:e:f:g:h"`.

`src` pointe sur un `struct in6_addr`.

`dst` pointe sur un buffer de taille `size` $\geqslant$ `INET6_ADDRSTRLEN`.

Renvoie `dst` ou `NULL`.