Systèmes distribués

Louis Mandel
louis.mandel@lri.fr
Laboratoire de Recherche en Informatique
Université Paris-Sud 11

IFIPS
Cycle ingénieur de la filière étudiant
année 2007/2008

Insuffisance des tuyaux

- Communication locale à une machine.
  - Pas bien adaptée au modèle client-serveur :
    - Le serveur accède seul à une resource partagée
    - Les clients accèdent à cette resource par l'intermédiaire d'une connection avec le serveur.
    - Le serveur sérialise et réglemente l'accès à cette ressource.
  - On pourrait le faire avec des tuyaux nommés
    - le serveur crée un tuyau nommé à l'établissement du service et lit des demandes de connections sur celui-ci.
    - le client crée un tuyau nommé et fait une demande de connection au serveur en passant sur le tuyau de référence l'adresse de son propre tuyau.
    - problème de l'exclusion mutuelle entre les demandes de connection. Protocole réalisable mais complexe.

Les sockets (prises)

- Généralisation des tuyaux
  - Elles abstraient le modèle client-serveur.
  - Elles permettent la communication locale et distante (entre processus de différentes machines).
  - Elles permettent différents protocoles de transmmissions de façon transparente.
- Le protocole de branchement
  - Le serveur propose un service sur un port dont l'adresse est publique et se met en écoute sur celui-ci.
  - Le client fait une demande de branchement sur ce port. Si le branchement réussit, le client et le serveur reçoivent chacun une prise pour communiquer ensemble en privé.
  - Plusieurs clients reçoivent des prises différentes pour communiquer avec le même serveur.

Schéma général

- 1. Création d'une socket (`socket`) :
  - on choisi le domain (local v.s. distante)
  - on choisi le type de communication
- 2.(a). Branchement de la socket en mode server
  1. On lie la prise à une adresse (`bind`).
  2. On se met en écoute sur la prise (`listen`).
  3. On accepte une connection, ce qui, à chaque connection, retourne une nouvelle prise connectée avec le client (`accecpt`).
  4. On traite la connection puis on reprend à l'étape précédente.
  5. La fermeture de la prise ferme le service (`close`/`shutdown`).

Schéma général

- 2.(b). Branchement de la prise en mode client
  1. On demande à être connecté à une adresse (`connect`).
  2. On envoie les données et on écoute les réponses (`send`/`recv`).
  3. La fermeture de la prise ferme la connection (`close`/`shutdown`).

Différents types de communication

- Les types de sockets
  - `SOCK_STREAM` :
    - Transmission fiable, par flot d'octets.
    - Le plus répandue. Transmission d'une suite d'octets sans structure particulière. Ex. rsh, ssh, ftp etc.
  - `SOCK_SEQPACKET` :
    - Transmission fiable, par paquets
  - `SOCK_DGRAM` :
    - Transmission non fiable, par paquets.
    - La plus proche du réseau, la plus économique : type internet.
    - Données sans importance.
  - `SOCK_RAW` :
    - Accès aux couches basses du réseau.

Les domaines de communication

- Les dommaines
  - `AF_UNIX` :
    - Le domaine Unix : ne permet la communication qu'au sein d'une même machine
  - `AF_INET`
    - Le domaine Internet
- Dans la suite nous nous intéresserons aux sockets du domaine Internet

Création de Socket (1)

- `#include <sys/types.h>`
  `#include <sys/socket.h>`
  `int socket(int domain, int type, int protocol);`
  - `domain` : `AF_UNIX`, `AF_INET`, ...
  - `type` : `SOCK_STREAM`, `SOCK_DGRAM`, ...
  - `protocol` : `0` (protocole par default correspondant au type de communication)
- Réglages d'une socket
  - `getsockopt/setsockopt`
  - Temporisations des émissions/réceptions
  - Synchronisation des opérations
  - Taille des buffers
  - ...

Structure des adresses Internet

- Une adresse internet est représentée par une structure de type `struct in_addr`
-
```
struct in_addr {
  in_addr_t s_addr
} ;
```
- `in_addr_t` est un entier 32 bits (généralement `unsigned int`)
- Exemples : `10.200.0.142`, `127.0.0.1`

Machines

- Une adresse de hôte est représentée par une structure de type `struct hostent`
- `struct hostent {`

```
  char *h_name ;        // nom officiel de la machine
  char **h_aliases ;    // liste d'alias
  int h_addrtype ;      // type d'adresse
  int h_length ;        // longueur de l'adresse
  char **h_addr_list ; // liste des adresses de l'hote
} ;
```

  - Les adresses sont des `char *` de 4 caractères
  - Seul `h_addr_list[0]` nous intéresse en pratique

Machines

- Nom de la machine
  - `int gethostname(char *name, size_t namelen);`
- Informations sur une machine à partir de son nom
  - ```
    struct hostent *gethostbyname(const char *name);
    struct hostent *gethostbyaddr(const char *addr,
                                  socklen_t len, int type);
    ```
    ```

Exemple (addr.c)

```
struct hostent *h;
union {
  in_addr_t entier;
  unsigned char car[4];
} u;
void print_addr (char *nom) {
  int i;
  h = gethostbyname(nom);
  if ( h == NULL) printf("%s: macine inconnue", nom);
  else {
    printf("%s: ", nom);
    u.entier = *(in_addr_t*)h->h_addr;
    for(i=0; i<4; i++) printf("%d ", u.car[i]);
    printf(" [%u]\n", ntohl(u.entier));
  }
}
```

Les ports

- Les ports représente un service
  - Ce sont des numéros entiers sur 16 bits
  - `/etc/services` associe un port à un service et un protocol
  - Fonctions d'accès : `getservbyname`, `getservbyport`
- Choix d'un numéro pour offrir un service
  - `0-1023` : Well-known (super-utilisateur)
  - `1024-49151` : Registred (Utilisateur, conflits possibles)
  - `49152-65535` : Private (Utilisateur, libres)

Structure des adresses de sockets

- Structure abstraite `struct sockaddr`
- Structure concrète `struct sockaddr_in`
-
```
struct sockaddr_in {
  sa_family_t sin_family ;
  struct in_addr sin_addr ;
  sin_port_t sin_port ;
} ;
```
  - `sin_family` vaudra toujours `AF_INET`
  - `sin_addr` est l'adresse IP
    - `int gethostname(char *name, size_t namelen);`
    - `INADDR_ANY` : toutes les adresses possibles de la machine
  - `sin_port` représente le numéro de port (utiliser `htons`)
    - 0 : un port quelconque

Offre de service (2.a)

- Lier la socket à une adresse
  - `int bind(int desc,`
    `const struct sockaddr *name, socklen_t namelen);`
  - Lie la socket `desc` à l'adesse `name`
  - Le descripteur `desc` sera utilisé pour écouter et récupérer les connections

Création de Socket AF_INET (creerSocket.c)

```
static struct sockaddr_in adresse;
int creerSocket(int type, int port, struct sockaddr_in *p_adresse) {
  int desc;
  socklen_t longueur = (socklen_t) sizeof(struct sockaddr_in);
  /* Creation de la socket */
  if ( (desc = socket(AF_INET, type, 0)) == -1) {
    perror("Creation de socket impossible"); return -1;
  }
  /* Préparation de l'adresse */
  adresse.sin_family = AF_INET;
  adresse.sin_addr.s_addr = htonl(INADDR_ANY);
  adresse.sin_port = htons(port);
  /* Attachement de la socket */
  if ( bind(desc, (struct sockaddr *)&adresse, longueur) == -1) {
    perror("Attachement de la socket impossible");
    close(desc); return -1;
  }
  /* Récupération de l'adresse effective d'attachement */
  if (p_adresse != NULL) getsockname(desc, (struct sockaddr *)p_adresse, &longueur);
  return desc;
}
```

Création de Socket AF_INET (creerSocket.h)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

extern int creerSocket(int type, int port, struct sockaddr_in *p_adresse);
```

Exemple (sockinet.c)

```
#include "creerSocket.h"
int main(){
  struct sockaddr_in adresse_udp, adresse_tcp;
  short port_udp, port_tcp;
  int socket_udp, socket_tcp;
  port_udp = 2467;
  socket_udp = creerSocket(SOCK_DGRAM, port_udp, &adresse_udp);
  if ( socket_udp != -1 )
    printf("Socket UDP sur port : %d\n", ntohs(adresse_udp.sin_port));
  port_tcp = 0;
  socket_tcp = creerSocket(SOCK_STREAM, port_tcp, &adresse_tcp);
  if ( socket_tcp != -1 )
    printf("Socket TCP sur port : %d\n", ntohs(adresse_tcp.sin_port));
}
```

Offre de service (2.a)

- Autoriser les connections
  - `int listen(int desc, int backlog);`
  - `desc` : socket
  - `n` : nombre de connections qui peuvent être en attente avant leur ouverture (typiquement quelques dizaines)

Offre de service (2.a)

- Accepter une connection
  - `int accept(int desc,`
    `struct sockaddr *addr, socklen_t *addrlen);`
  - `desc` : socket
  - `addr` : addresse du client (paramètre résultat)
  - Valeur de retour : descripteur correspondant à la connection

Exemple : (serveur0.c)

```
int main(int argc, char **argv) {
  struct sockaddr_in adr;
  socklen_t lg_adr = (socklen_t)sizeof(struct sockaddr_in);
  int port, ecoute, connexion;
  if (argc < 2) { fprintf(stderr, "usage: %s port\n",argv[0]); exit(2); }
  port = atoi(argv[1]);
  ecoute = creerSocket(SOCK_STREAM, port, &adr);
  if ( ecoute == -1 ) {
    fprintf(stderr, "Creation de socket impossible\n"); exit(2);
  }
  if (listen(ecoute, 10) == -1) { perror("listen"); exit(2); }
  while(1) {
    connexion = accept(ecoute, (struct sockaddr *)&adr, &lg_adr);
    if (connexion == -1) {
      if (errno == EINTR) continue;
      else { perror("accept"); exit(1); }
    }
    f(connexion);
    shutdown(connexion, SHUT_WR);
  }
}
```

Utiliser un service (2.b)

- On lie la socket à l'adresse du service
  - `int connect(int desc,`
    `const struct sockaddr *name, socklen_t namelen);`
  - `desc` : socket
  - `name` : adresse du serveur

Exemple (client0.c)

```
int main(int argc, char **argv) {
  struct sockaddr_in adr_ser, adr_cli;
  int port, sock;
  struct hostent *hp;
  if (argc < 3) {
    fprintf(stderr, "usage: %s machine port\n",argv[0]); exit(2);
  }
  if ( (hp = gethostbyname(argv[1])) == NULL ) {
    fprintf(stderr, "machine %s inconnue\n",argv[1]); exit(2);
  }
  port = 0;
  if ( (sock = creerSocket(SOCK_STREAM, port, &adr_cli)) == -1 ) {
    fprintf(stderr, "Creation de socket impossible\n"); exit(2);
  }
  adr_ser.sin_family = AF_INET;
  adr_ser.sin_port = htons(atoi(argv[2]));
  memcpy(&adr_ser.sin_addr.s_addr, hp->h_addr, hp->h_length);
  if (connect(sock, (struct sockaddr *)&adr_ser, sizeof(adr_ser)) == -1) { perror("conect"); exit(
  printf("Connexion acceptee\n");
  g(sock);
}
```

Communications

- Une fois connecté, le client et le serveur peuvent dialoguer
  - en écrivant les données dans le descripteur (`write`)
  - en lisant les données dans le descripteur (`read`)
- Les lectres écritures se comportent comme sur un tuyau :
  - `read` bloque s'il n'y a pas de données
  - `write` bloque s'il y a trop de données
  - Si la connection a été fermée `read` renvoie 0 et `write` déclence le signal `SIGPIPE`.

Exemple (serveur1.c)

```
#define BUFF_SIZE 256

void f(int desc) {
  char buff1[BUFF_SIZE];
  char buff2[BUFF_SIZE];
  read(desc, buff1, BUFF_SIZE);
  sprintf(buff2, "Bonjour %s", buff1);
  write(desc, buff2, BUFF_SIZE);
}
```

Exemple (client1.c)

```
#define BUFF_SIZE 256

void g(int desc) {
  char buff[BUFF_SIZE];
  write(desc, "client", 10);
  read(desc, buff, BUFF_SIZE);
  printf("%s", buff);

}
```

JoCaml