Sockets

ADSCCD : Administration et Développement Système pour les Centres de Calcul et de Donnée

M2-CNS, parcours SA et SR
Université Paris-Saclay, site d'Evry, UEVE

Patrice LUCAS
CEA-DAM, patrice.lucas@cea.fr
PAST, département informatique de l'UEVE

Définition

- Point de connexion sur lequel, on peut écrire et lire des messages.
- Asynchrone
- Standard POSIX

Cycle de vie d'une communication par socket

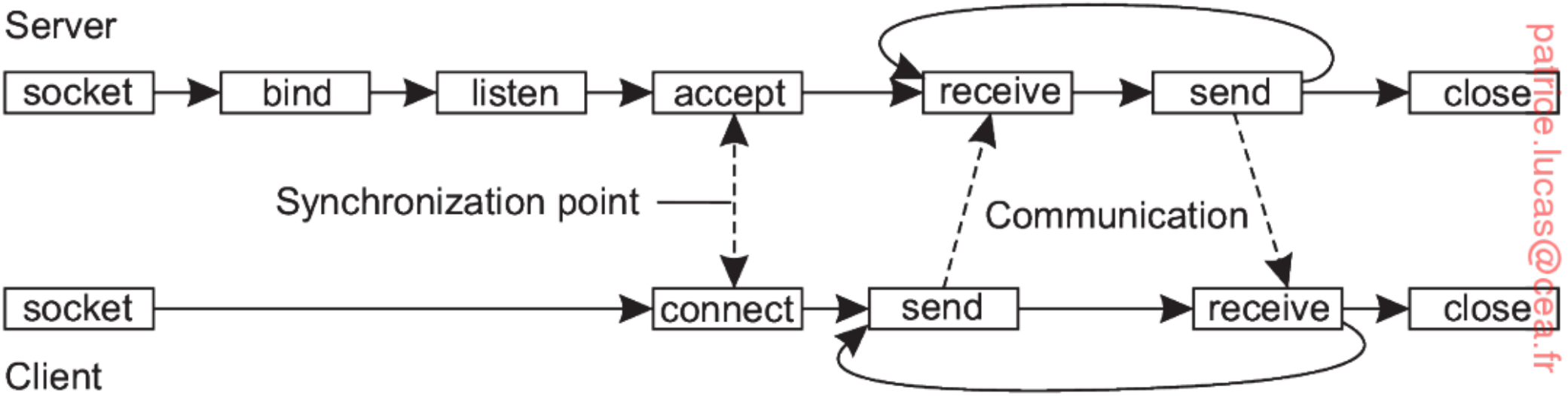

Figure 4.19: Connection-oriented communication pattern using sockets.

Distributed Systems, Maarten van Steen, Andrew S. Tanenbaum, Third Edition

Primitives d’une socket

Operation	Description
`socket`	Create a new communication end point
`bind`	Attach a local address to a socket
`listen`	Tell operating system what the maximum number of pending connection requests should be
`accept`	Block caller until a connection request arrives
`connect`	Actively attempt to establish a connection
`send`	Send some data over the connection
`receive`	Receive some data over the connection
`close`	Release the connection

Figure 4.18: The socket operations for TCP/IP.

Distributed Systems, Maarten van Steen, Andrew S. Tanenbaum, Third Edition

C Socket API : socket

- En C, la socket une fois créée apparait comme un “file descriptor”.

Création d’une socket

```
SOCKET(2)                                                  Linux Programmer's Manual
                                                                          SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);
```

C Socket API : socket

int socket(**int domain**, int type, int protocol);

- Domain :
  - AF_UNIX/AF_LOCAL (communication locale),
  - AF_INET (IPv4), AF_INET6 (IPv6),
  - …

C Socket API : socket

int socket(int domain, **int type**, int protocol);

- Type :
  - SOCK_STREAM : TCP, connection based protocol (packets are delivered in order), *“Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.”*
  - SOCK_DGRAM : UDP, datagram-based protocol (packet are limited in size), *“Supports datagrams (connectionless, unreliable messages of a fixed maximum length).”*
  - ...

  - Deux “options” peuvent être combinées par un “ou” logique avec le type :
    - SOCK_NONBLOCK : la socket devient “non-bloquante, un “accept” sans connexion en attente renvoie immédiatement EAGAIN ou EWOULDBLOCK au lieu d’attendre.
    - SOCK_CLOEXEC : la socket sera fermée sur un “execve”.

C Socket API : socket

int socket(int domain, int type, **int protocol**);

- Protocol :
  - 0 : désigne lorsque qu'il est unique le protocole disponible pour le couple "domaine/type" désigné,
  - Si plusieurs protocoles existent, on peut trouver la bonne valeur en utilisant "getprotobyname" (cf le contenu du fichier /etc/protocols) .

C Socket API : socket

int socket(int domain, int type, int protocol);

- Exemples AF_UNIX/AF_LOCAL:
  - unix_socket = socket(AF_UNIX, type, 0);
- Exemples IPv4 (cf : “man 7 ip”):
  - tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
  - udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
  - raw_socket = socket(AF_INET, SOCK_RAW, protocol);

C Socket API : bind

Permet de lier une adresse locale à la socket.

```
int bind(int sockfd, const struct sockaddr *addr,
socklen_t addrlen);
```

- “struct sockaddr” est un type générique permettant de recevoir les adresses propres à chaque domaine : il contient a-minima le champ sa_family du type sa_family_t (utilisé par exemple avec la valeur AF_UNSPEC dans le connect en mode déconnecté pour supprimer une association).
- Adresse unix (domain: AF_UNIX/AF_LOCAL)

```
#define UNIX_PATH_MAX    108

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

- Adresse IPv4 (domain: AF_INET)

C Socket API : bind

Permet de lier une adresse locale à la socket.

```
int bind(int sockfd, const struct sockaddr *addr,
socklen_t addrlen);
```

- “struct sockaddr” est un type générique permettant de recevoir les adresses propres à chaque domaine.
- Adresse unix (AF_UNIX/AF_LOCAL)
- Adresse IPv4

```
struct sockaddr_in {
    sa_family_t sin_family;/* address family: AF_INET */
    in_port_t sin_port;      /* port in network byte
order */
    struct in_addr sin_addr; /* internet address */
};

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

C Socket API : construction d'adresses

- inet_aton : "192.168.2.37" → struct in_addr

```
int inet_aton(const char *cp, struct in_addr *inp);
```

- getaddrinfo : "myserver.com", "http"/"6000", "AF_INET" → liste de "struct sockaddr"

```
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
struct addrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char *ai_canonname;
    struct addrinfo *ai_next;
};
```

C Socket API : listen / accept / connect

- Déclarer la socket en écoute et fixer le nombre de connexion en attente pour le serveur (réservé aux types : SOCK_STREAM, SOCK_SEQPACKET):

```
int listen(int sockfd, int backlog);
```

- Accepter une connexion pour un serveur, récupérer la socket obtenue et l'adresse du client qui vient de se connecter (réservé aux types : SOCK_STREAM et SOCK_SEQ_PACKET):

```
int accept(int sockfd,
           struct sockaddr *addr,
           socklen_t *addrlen);
```

- Se connecter sur une adresse précise :
  - Mode connecté (SOCK_STREAM, SOCK_SEQ_PACKET) : réalise une connection (généralement utilisé une unique fois)
  - Mode non-connecté (SOCK_DGRAM) : adresse par défaut d'emission et adresse unique de réception (peut s'utiliser plusieurs fois, ou jamais au profit de sendto et recvfrom)

```
int connect(int sockfd,
            const struct sockaddr *addr,
            socklen_t addrlen);
```

C Socket API : send/recv

- Envoyer un message en mode connecté : write / send, sans “flag” send est équivalent à un write

```
ssize_t send(int sockfd, const void *buf,
size_t len, int flags);
```

- Recevoir un message en mode connecté : read / recv

```
ssize_t recv(int sockfd, void *buf,
size_t len, int flags);
```

  - Flags :
    - MSG_PEEK : laisse les données en place,
    - MSG_DONTWAIT : comportement non-bloquant.

C Socket API : sendto/recvfrom

- Envoyer un message en mode non-connecté : sendto,

```
ssize_t sendto(int sockfd, const void
*buf, size_t len, int flags,
                  const struct sockaddr
*dest_addr, socklen_t addrlen);
```

- Recevoir un message en mode non-connecté : recvfrom

```
ssize_t recvfrom(int sockfd, void *buf,
size_t len, int flags,
                    struct sockaddr
*src_addr, socklen_t *addrlen);
```

C Socket API : network endianness

- Attention, sur une socket on transmet uniquement un ensemble d'octets ...

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

uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);.
```

C Socket API : shutdown, close

- Clore la connection : SHUT_RD, SHUT_WR, SHUT_RDWR
  - Ferme la connection dans un sens ou les deux.
    - Mode connecté : correspond au “FIN”/”FIN-ACK”
    - Mode déconnecté : bloque simplement l’utilisation de la socket en envoi ou en réception
  - En réception, un recv renvoie une taille nulle sur une socket dont l’émetteur est en shutdown

```
int shutdown(int sockfd, int how);
```

- Fermer les sockets : close
  - Mode connecté : correspond à un “RST” reset.

```
int close(int fd);
```

C Socket API : exemples

- socket_c_tcp_draw
- socket_c_udp_draw
- socket_c_udp_max
- socket_c_udp_nonblocking

C Socket API : exercice

- socket_c_tcp_mystery

- Ecrire le code d’un client qui
  - contacte un premier serveur (socket_c_tcp_giver : dont le code compilé x86_64 est fourni) en lui transmettant un ID et reçoit de la part du serveur la clé correspondant à cet ID,
  - transmet ensuite cet ID puis la clé à un deuxième serveur (socket_c_tcp_checker : dont le code compilé x86_64 est fourni)

Socket en python

- Au sein de la librairie standard
  - module “socket” : https://docs.python.org/3.8/library/socket.html
  - https://docs.python.org/3.8/howto/sockets.html#socket-howto
- Aussi proche que possible de l’API BSD d’origine mais avec des aspects simplifiés naturels en python
  - Allocation automatique des buffers à la réception
  - Taille implicite des buffers à l’émission
- Adresse propre à la famille
  - AF_UNIX : une string décrivant une entrée du système de fichier
  - AF_INET : (host, port)
    - host : une string avec soit le hostname avec un nom de domaine, soit l’IP ‘A.B.C.D’ (avec un nom de domaine, la première entrée de la résolution est utilisée)
    - port : un entier

Socket en python : serveur

- Création, renvoie une socket :

```
socket.socket(family=AF_INET, type=SOCK_STREAM, proto=0, fileno=None)
```

- Attachement sur la socket :

```
socket.bind(address)
```

- Création + attachement AF_INET/AF_INET6, renvoie une socket déjà attachée (python 3.8):

```
socket.create_server(address, *, family=AF_INET, backlog=None, reuse_port=False, dualstack_ipv6=False)
```

- Limitation sur la socket :

```
socket.listen([backlog])
```

- Accept sur la socket, renvoie (conn, address) (avec “conn”, la socket et “address”, l’adresse distante)

```
socket.accept()
```

Socket en python : client

- Création, renvoie une socket :

```
socket.socket(family=AF_INET,
type=SOCK_STREAM, proto=0,
fileno=None)
```

- Connection de la socket :

```
socket.connect(address)
```

- Création + connection AF_INET/AF_INET6, renvoie une socket déjà connectée :

```
socket.create_connection(address[, timeout[,
source_address]])
```

Socket en python : envoi / réception

- Obtention d'un fichier sur la socket :

```
socket.makefile(mode='r', buffering=None, *,
encoding=None, errors=None, newline=None)
```

  - mode : 'r', 'w', 'b' .

- Envoi sur la socket :

```
socket.send(bytes[, flags])
```

  - Renvoie le nombre d'octets effectivement envoyés

```
socket.sendall(bytes[, flags])
```

  - Renvoie "`None`" ou bien lève une exception si une erreur se produit

- Réception sur la socket :

```
socket.recv(bufsize[, flags])
```

  - bufsize : taille maximale à recevoir

Socket en python : fermeture

- Shutdown de la socket sur la socket :

```
socket.shutdown(how)
```

  - how : socket.SHUT_RD, socket.SHUT_WR, `socket.SHUT_RDWR`

- Fermeture de la socket sur la socket :

```
Socket.close()
```

  - Les sockets sont automatiquement fermées avec le “garbage-collecting” mais il est conseillé de les fermer explicitement ou de les utiliser au sein d’une directive “`with`”.

Socket en python : sérialisation

- Le module python “struct” permet de faire ses premiers pas en sérialisation.

```
buffer = struct.pack(format, v1, v2, …)

(v1, v2, …) = struct.unpack(format, buffer)

nb_bytes = struct.calcsize(format)
```

- Format :
  - Byte order, size, alignment : @ (native), = (native, standard, pas d’alignement), < (little endian), > (big endian), ! (network)
  - Characters: i (int), I (unsigned int), d (double) ...

Socket en python : exemple

- socket_python_tcp_draw

Socket en python : exercices

- socket_tcp_mystery (en python)
- socket_python_cp
  - Écrire un client lisant le contenu d’un fichier et l’envoyant à un serveur qui recopie ce contenu dans un nouveau fichier.