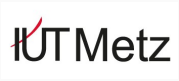
R3.05 - Programmation Système

Sockets

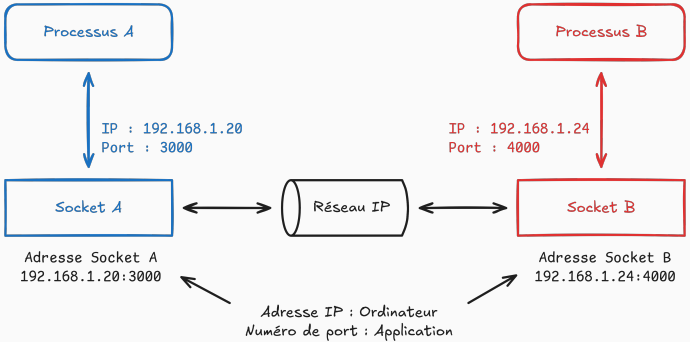
Cyril Grelier

Université de Lorraine - IUT de Metz



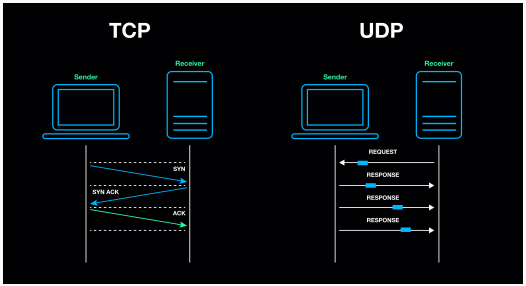
Introduction

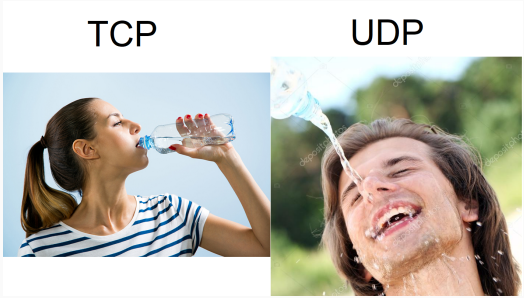
- Un **socket** = point de communication bidirectionnel entre deux processus.
- Communication possible :
  - **Locale** (`AF_UNIX`)
  - **Réseau** (`AF_INET` IPv4, `AF_INET6` IPv6)
- Manipulés comme des **descripteurs de fichiers** : `read`/`write`, `send`/`recv`.



Modes de transport : TCP vs UDP

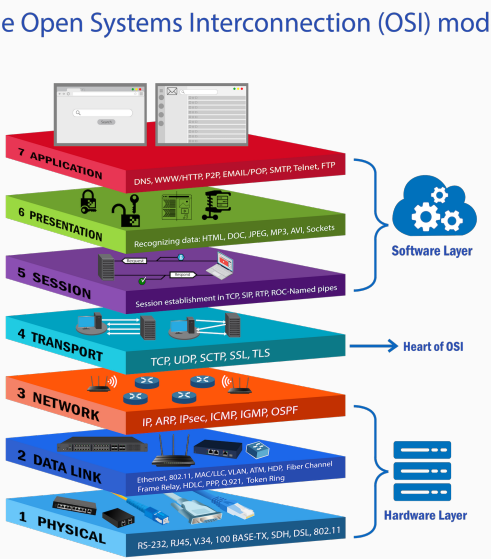
- **TCP** : `SOCK_STREAM`, connecté, fiable, ordonné. Idéal pour HTTP, SSH, SMTP, bases de données.
- **UDP** : `SOCK_DGRAM`, non connecté, non fiable (best-effort). Idéal si la **latence** prime : streaming, VoIP, jeux.





Modèle OSI et pile TCP/IP

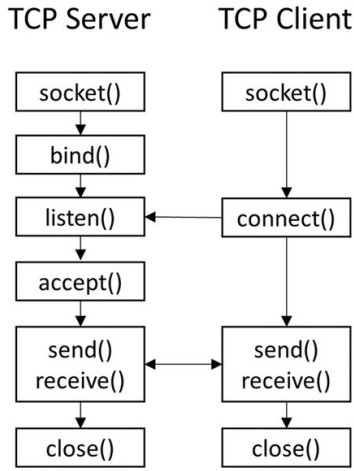
- Les sockets se situent au-dessus des protocoles de transport (TCP/UDP).
- Exposent une API de programmation réseau.



Principe d'une communication TCP

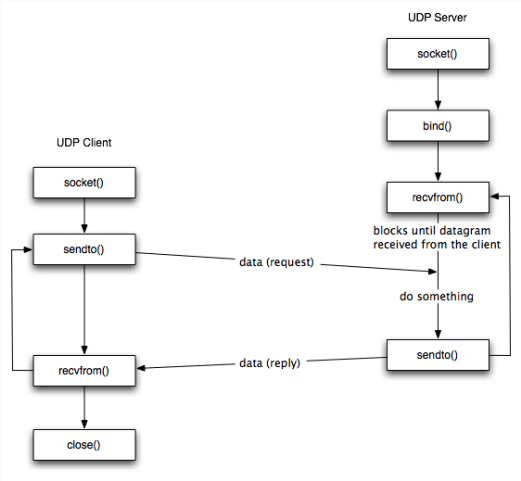
Deux applications : serveur et client.

1. Serveur : `socket`, `bind`, `listen`, puis `accept` pour créer une socket connectée.
2. Client : `socket`, puis `connect` avec IP/port du serveur.
3. Les deux échangent avec `send`/`recv`.
4. Fermeture avec `shutdown` + `close`.



Principe d'une communication UDP

1. Serveur : `socket(SOCK_DGRAM)`, `bind`, puis `recvfrom`.
2. Client : `socket`, envoi avec `sendto`.
3. Pas de notion de session : chaque message est indépendant.
4. Fermeture avec `close`.



Création d'un socket

Prototype C :

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

- domain : AF_INET, AF_INET6, AF_UNIX
- type : SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
- protocol : souvent 0 (choix par défaut)

```c
// TCP
int fd_tcp = socket(AF_INET, SOCK_STREAM, 0);
// UDP
int fd_udp = socket(AF_INET, SOCK_DGRAM, 0);
```

Définition de l'adresse : bind

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

- Associe une socket à une adresse (IP + port ou chemin AF_UNIX).
- Utilise des structures spécialisées (sockaddr_in, sockaddr_in6, sockaddr_un).
- Conversion en `struct sockaddr *`.

Exemple IPv4 :

```c
struct sockaddr_in addr = {0};
addr.sin_family      = AF_INET;
addr.sin_port        = htons(8080);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(s, (struct sockaddr*)&addr, sizeof addr);
```

Écoute et acceptation TCP

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

- `listen` : met la socket en attente.
- accept : crée une nouvelle socket connectée pour un client.

```c
int s = socket(AF_INET, SOCK_STREAM, 0);
bind(s, ...);
listen(s, 16);
int c = accept(s, (struct sockaddr*)&cli, &len);
```

Connexion côté client

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

```c
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv = {0};
serv.sin_family = AF_INET;
serv.sin_port   = htons(8080);
inet_pton(AF_INET, "127.0.0.1", &serv.sin_addr);
connect(sock, (struct sockaddr*)&serv, sizeof serv);
```

Envoyer et recevoir des données

TCP :

```
1  send(sock, buf, len, 0);
2  recv(sock, buf, len, 0);
```

UDP :

```
1  sendto(sock, buf, len, 0, (struct sockaddr*)&dst, len);
2  recvfrom(sock, buf, len, 0, (struct sockaddr*)&src, &len);
```

Fermeture des sockets

TCP :

```
1  shutdown(fd, SHUT_WR); // fin des envois
2  close(fd);             // libère la ressource
```

UDP :

```
1  close(fd_udp);
```

Sockets locales (AF_UNIX)

```c
int fd = socket(AF_UNIX, SOCK_STREAM, 0);
struct sockaddr_un addr = {0};
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "/tmp/mysock");
bind(fd, (struct sockaddr*)&addr, sizeof addr);
```

- Communication locale rapide (sans pile IP).
- Nécessite de `close` et `unlink` le fichier socket côté serveur.