Réseaux : modèles, protocoles, programmation

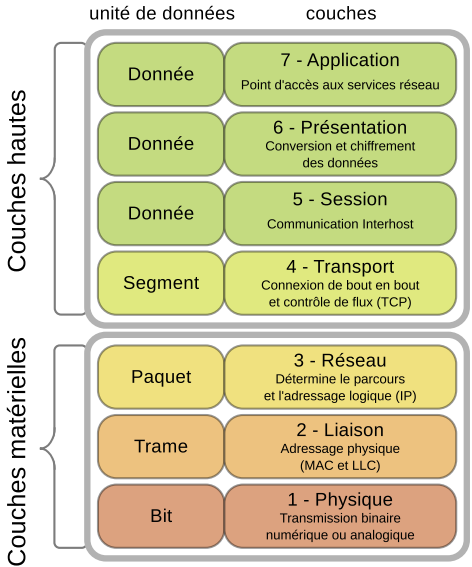
Pablo Rauzy
pablo.rauzy@univ-paris8.fr
pablo.rauzy.name/teaching/rmpp

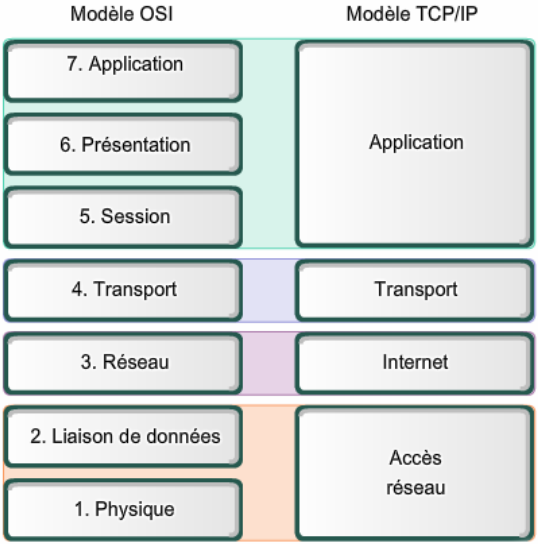
UFR MITSIC / L2 informatique

Séance 6

Couches session, présentation, et application
Programmation TCP

Couches session, présentation, et application


unité de données
couches
Couches hautes
Couches matérielles
Donnée
7 - Application
Point d'accès aux services réseau
Donnée
6 - Présentation
Conversion et chiffrement des données
Donnée
5 - Session
Communication Interhost
Segment
4 - Transport
Connexion de bout en bout et contrôle de flux (TCP)
Paquet
3 - Réseau
Détermine le parcours et l'adressage logique (IP)
Trame
2 - Liaison
Adressage physique (MAC et LLC)
Bit
1 - Physique
Transmission binaire numérique ou analogique


Modèle OSI
Modèle TCP/IP
7. Application
6. Présentation
5. Session
Application
4. Transport
Transport
3. Réseau
Internet
2. Liaison de données
1. Physique
Accès
réseau

- La couche session est la cinquième couche du modèle OSI.
- Elle est en charge de la synchronisation des communications et de la gestion des transactions :
  - permettre à n'importe quel participant de parler à tout moment,
  - rétablir la connexion en cas de coupure,
  - synchroniser deux flux (e.g., audio et vidéo),
  - gérer les communications multipoints.
- En pratique, donc dans le modèle TCP/IP, la couche session n'a pas vraiment d'existence propre.

- La couche présentation est la sixième couche du modèle OSI.
- Les cinq couches du dessous ne transportent que des données brutes, vu simplement comme une suite d'octets, la couche présentation est donc en charge :
  - de l'encodage (ASCII, Latin-1, Unicode, ...),
  - de la compression,
  - du chiffrement.
- En pratique, donc dans le modèle TCP/IP, la couche présentation n'a pas vraiment d'existence propre, et chaque protocole implémente cela à sa façon.

- Dans le monde TCP/IP, on convertit typiquement tout en texte.
- MIME signifie *Multipurpose Internet Mail Extensions*.
- Il s'agit à l'origine d'un standard pour les courriels, permettant de spécifier l'encodage des caractères en les convertissant vers de l'ASCII, seul format supporté nativement par les protocoles de courriel.
- Aujourd'hui les types MIME prennent aussi en compte les différents formats de médias (document, image, audio, vidéo, etc.).
- Son utilisation s'est répandue au delà du courriel : HTTP utilise aussi le standard MIME.

Entêtes MIME

- `Content-type:` donne le type MIME du contenu (par défaut, `text/plain`).
  - C'est ici qu'on précise un *charset* dans le cas de type texte.
  - Exemples :
    - `text/html; charset=utf-8`
    - `image/png`
- `Content-transfer-encoding:` donne l'encodage de transfert (la méthode de conversion vers ASCII).
  - Ne pas confondre avec le charset.
  - Exemples :
    - `quoted-printable`
    - `base64`

Entêtes MIME

- `Content-type:` donne le type MIME du contenu (par défaut, `text/plain`).
  - C'est ici qu'on précise un *charset* dans le cas de type texte.
  - Exemples :
    - `text/html; charset=utf-8`
    - `image/png`
- `Content-transfer-encoding:` donne l'encodage de transfert (la méthode de conversion vers ASCII).
  - Ne pas confondre avec le charset.
  - Exemples :
    - `quoted-printable`
    - `base64`
- Le `Content-type` peut aussi spécifier
  - qu'il y a plusieurs parties indépendantes dans les données (e.g., pour les pièces jointes)
    - `multipart/mixed; boundary="=---=sep=---="`
  - l'existence d'alternatives (e.g., pour envoyer un courriel en texte et en HTML)
    - `multipart/alternative`

- La couche application est la septième couche du modèle OSI.
- Elle représente le point d'accès au service réseau pour l'utilisateur.
- La diversité des protocoles à ce niveau là est très large.
  - BGP, DNS, HTTP, FTP, SMTP, IMAP, POP, SSH, IRC, XMPP, ...
  - Et bientôt le vôtre !
- Nous étudierons certains de ces protocoles dans les séances à venir.
- Pour la suite du cours d'aujourd'hui, nous allons revenir sur la programmation réseau.

Programmation TCP

- Rappel des appels systèmes réseau qu'on a vu jusqu'à présent.
- On reste encore sur de l'IPv4, on verra une prochaine fois comment être compatible avec IPv6 aussi.

- Créer une socket.
- `int socket (int domain, int type, int protocol);`
  - `domain` : le domaine de communication (unix, internet, ...).
  - `type` : le type de communciation (raw, datagramme, stream).
  - `protocol` : le protocole qu'on utilisera avec la socket.
  - Retourne le file descriptor de la socket créée, ou -1 en cas d'erreur.

- Créer une socket.
- `int socket (int domain, int type, int protocol);`
  - `domain` : le domaine de communication (unix, internet, ...).
  - `type` : le type de communciation (raw, datagramme, stream).
  - `protocol` : le protocole qu'on utilisera avec la socket.
  - Retourne le file descriptor de la socket créée, ou -1 en cas d'erreur.

```
1 #include <sys/socket.h>
2   int sfd;
3   sfd = socket(AF_INET, SOCK_STREAM, 0);
4   if (sfd < 0) {
5     perror("socket");
6     exit(1);
7   }
```

- Assigner une adresse à une socket.
- `int bind (int sockfd, const struct sockaddr *addr, int addrlen);`
  - `sockfd` : la socket à laquelle on veut assigner l'adresse.
  - `addr` : l'adresse que l'on souhaite assigner.
  - `addrlen` : donne la taille (en octet) de l'adresse.
  - Retourne 0, ou -1 en cas d'erreur.

bind

- Assigner une adresse à une socket.
- `int bind (int sockfd, const struct sockaddr *addr, int addrlen);`
  - `sockfd` : la socket à laquelle on veut assigner l'adresse.
  - `addr` : l'adresse que l'on souhaite assigner.
  - `addrlen` : donne la taille (en octet) de l'adresse.
  - Retourne 0, ou -1 en cas d'erreur.

```
1 #include <sys/socket.h>
2   struct sockaddr_in addr;
3   addr.sin_family = AF_INET;
4   addr.sin_addr.s_addr = INADDR_ANY;
5   addr.sin_port = htons(port);
6   if (bind(sfd, (struct sockaddr *) &addr, sizeof(addr)) < 0)  {
7     perror("bind");
8     exit(1);
9   }
```

- Marquer une socket comme passive, pour accepter des connexions entrantes.
- `int listen (int sockfd, int backlog);`
  - `sockfd` : la socket à rendre passive.
  - `backlog` : taille maximum de la queue de connexion en attente.
  - Retourne 0, ou -1 en cas d'erreur.

listen

- Marquer une socket comme passive, pour accepter des connexions entrantes.
- `int listen (int sockfd, int backlog);`
  - `sockfd` : la socket à rendre passive.
  - `backlog` : taille maximum de la queue de connexion en attente.
  - Retourne 0, ou -1 en cas d'erreur.

```
1 #include <sys/socket.h>
2   if (listen(sfd, 8) < 0)  {
3     perror("listen");
4     exit(1);
5   }
```

- Accepter une connexion entrante.
- `int accept (int sockfd, struct sockaddr *addr, int *addrlen);`
  - `sockfd` : la socket sur laquelle on écoute.
  - `addr` : sera rempli avec l'adresse de la socket distante.
  - `addrlen` : sera rempli avec la taille (en octet) de l'adresse.
  - Retourne le file descriptor de la socket distante, ou -1 en cas d'erreur.

accept

- Accepter une connexion entrante.
- `int accept (int sockfd, struct sockaddr *addr, int *addrlen);`
  - `sockfd` : la socket sur laquelle on écoute.
  - `addr` : sera rempli avec l'adresse de la socket distante.
  - `addrlen` : sera rempli avec la taille (en octet) de l'adresse.
  - Retourne le file descriptor de la socket distante, ou -1 en cas d'erreur.

```
1 #include <sys/socket.h>
2   int client;
3   if ((client = accept(sfd, NULL, 0)) < 0) {
4     perror("accept");
5     exit(1);
6   }
```

- Connecter une socket à une adresse distante.
- `int connect (int sockfd, const struct sockaddr *addr, int addrlen);`
  - `sockfd` : la socket à connecter.
  - `addr` : l'adresse de la socket distante.
  - `addrlen` : la taille (en octet) de l'adresse.
  - Retourne 0, ou -1 en cas d'erreur.

connect

- Connecter une socket à une adresse distante.
- `int connect (int sockfd, const struct sockaddr *addr, int addrlen);`
  - `sockfd` : la socket à connecter.
  - `addr` : l'adresse de la socket distante.
  - `addrlen` : la taille (en octet) de l'adresse.
  - Retourne 0, ou -1 en cas d'erreur.

```
1  #include <sys/socket.h>
2    host = gethostbyname(server);
3    if (host == NULL) {
4      printf("ERROR: gethostbyname");
5      exit(1);
6    }
7    addr.sin_family = AF_INET;
8    addr.sin_addr.s_addr = htonl(*((uint32_t *) host->h_addr));
9    addr.sin_port = htons(port);
10   if (connect(sfd, &addr, sizeof(addr)) < 0) {
11     perror("connect");
12     exit(1);
13   }
```

send / write

- Envoyer des données à travers une socket connectée.
- `int send (int sockfd, const void *buf, size_t len, int flags);`
- `int write (int fd, const void *buf, size_t count);`
  - `sockfd` : la socket à travers laquelle on envoie les données.
  - `buf` : un pointeur vers la zone mémoire où sont les données.
  - `len` : la longueur (en octets) des données.
  - `flags` : des options (cf. le manuel), à 0 pour `write`.
  - Retourne le nombre d'octets envoyés, ou -1 en cas d'erreur.

send / write

- Envoyer des données à travers une socket connectée.
- `int send (int sockfd, const void *buf, size_t len, int flags);`
- `int write (int fd, const void *buf, size_t count);`
  - `sockfd` : la socket à travers laquelle on envoie les données.
  - `buf` : un pointeur vers la zone mémoire où sont les données.
  - `len` : la longueur (en octets) des données.
  - `flags` : des options (cf. le manuel), à 0 pour `write`.
  - Retourne le nombre d'octets envoyés, ou -1 en cas d'erreur.

```
1 #include <unistd.h>
2   if (write(client, buf, len) < 0)  {
3     perror("write");
4   }
```

recv / read

- Recevoir des données à travers une socket connectée.
- `int recv (int sockfd, const void *buf, size_t len, int flags);`
- `int read (int fd, const void *buf, size_t count);`
  - `sockfd` : la socket à travers laquelle on reçoit les données.
  - `buf` : un pointeur vers la zone mémoire sont écrite les données.
  - `len` : la taille (en octets) de cette zone mémoire.
  - `flags` : des options (cf. le manuel), à 0 pour `read`.
  - Retourne le nombre d'octets reçus, ou -1 en cas d'erreur.

recv / read

- Recevoir des données à travers une socket connectée.
- `int recv (int sockfd, const void *buf, size_t len, int flags);`
- `int read (int fd, const void *buf, size_t count);`
  - `sockfd` : la socket à travers laquelle on reçoit les données.
  - `buf` : un pointeur vers la zone mémoire sont écrite les données.
  - `len` : la taille (en octets) de cette zone mémoire.
  - `flags` : des options (cf. le manuel), à 0 pour `read`.
  - Retourne le nombre d'octets reçus, ou -1 en cas d'erreur.

```
1 #include <unistd.h>
2   if ((len = read(client, buf, sizeof(buf))) < 0) {
3     perror("read");
4   }
```

- Fermer la connexion.
- `int shutdown (int sockfd, int how);`
  - `sockfd` : la socket dont on ferme la connexion.
  - `how` : comment fermer la connexion :
    - `SHUT_RD` : refuser la réception,
    - `SHUT_WR` : refuser l'envoie,
    - `SHUT_RDWR` : refuser les deux.
  - Retourne 0, ou -1 en cas d'erreur.

- Fermer définitivement la socket.
- `int close (int fd);`
  - `fd` : le file descriptor de la socket à fermer.
  - Retourne 0, ou -1 en cas d'erreur.

close

- Fermer définitivement la socket.
- `int close (int fd);`
  - `fd` : le file descriptor de la socket à fermer.
  - Retourne 0, ou -1 en cas d'erreur.

```
1 #include <unistd.h>
2   close(sfd);
```