ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Architectures matérielles et logicielles des réseaux

S. Genaud

ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

Partie 3 : Socket

Socket

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

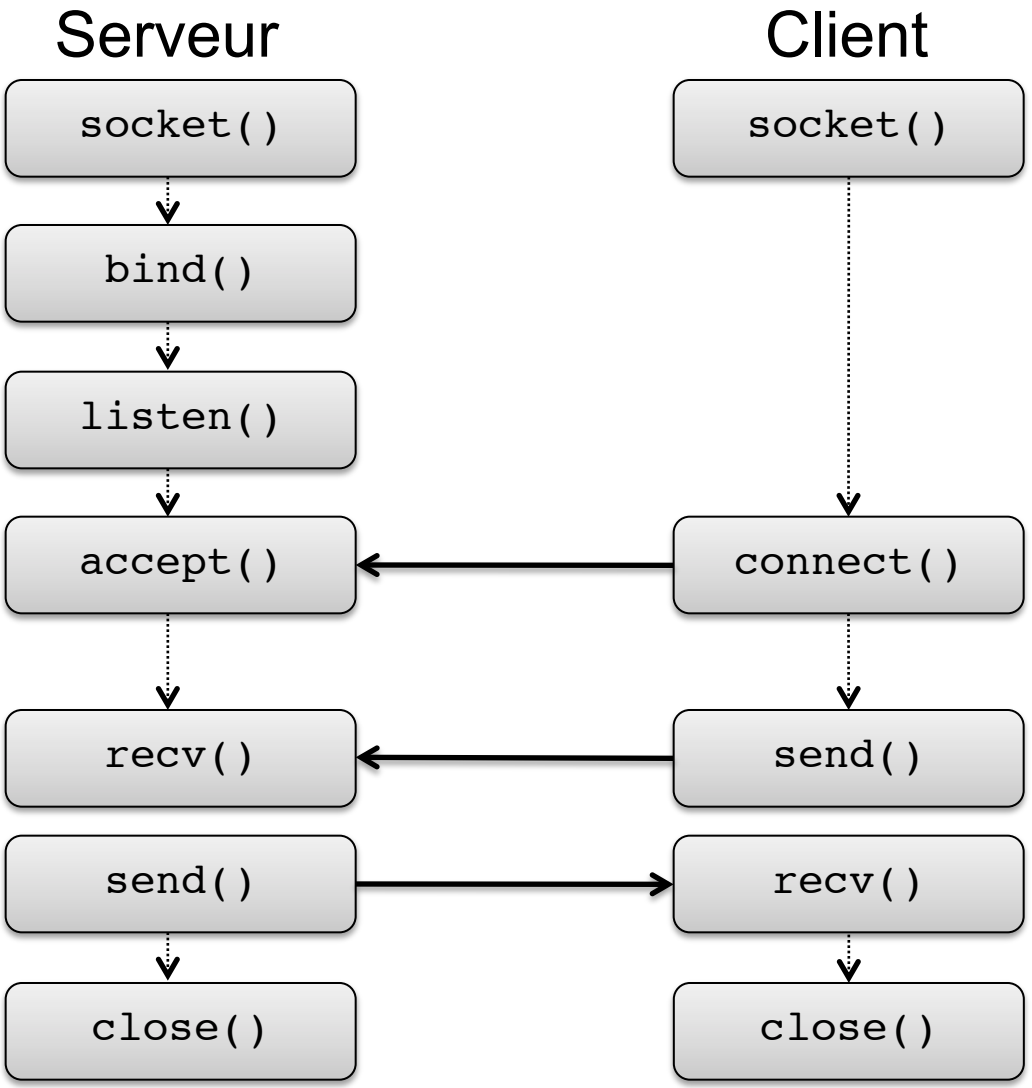
- Un socket permet une communication entre processus (Inter Process Communication-IPC), localisés soit :
  - à l'intérieur d'un même hôte
  - sur des hôtes différents (connectés par TCP/IP)
- Prennent place au-dessus de la couche transport (couche 4 du modèle OSI)
- Un socket peut utiliser le mode
  - connecté : TCP
  - non-connecté : UDP

Socket côté serveur: cycle de vie

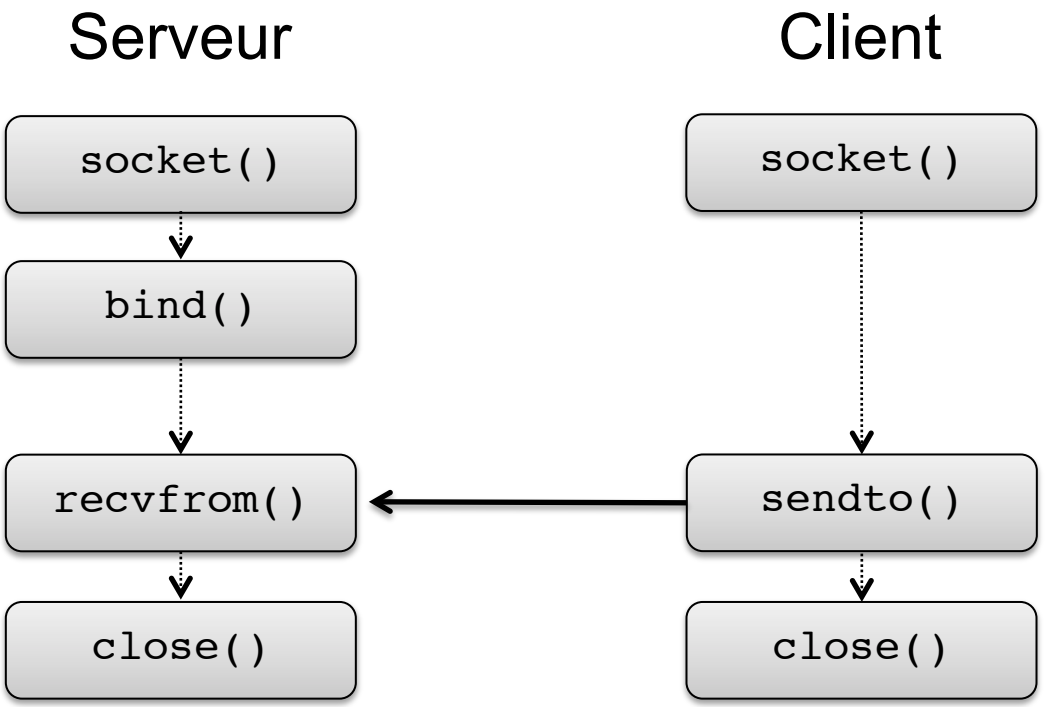
ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

1. **création** socket : retourne un descripteur (un entier), identifiant unique.
2. **spécification** du mode de communication (bind): port, et mode (TCP ou UDP)
3. **écoute** des messages:
   - en mode connecté : attente passive (listen) d'ouverture de connexion (accept) puis réception globale (recv)
   - en mode non-connecté : réception par morceaux sans ordre (recvfrom)
4. **fermeture** (close)

Socket mode connecté (TCP)



Socket mode non-connecté (UDP)



Serveur et client : création d'un socket

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

- **domain** : *AF_INET* pour TCP/IP communication avec, *AF_UNIX* pour communication à l'intérieur d'une machine même machine Unix
- **type:** connecté (TCP) : SOCK_STREAM ou non-connecté (UDP) : SOCK_DGRAM
- **protocol:** permet de spécifier un protocole permettant de fournir le service désiré. Dans le cas de la suite TCP/IP il n'est pas utile, on le mettra ainsi toujours à 0
- `socket()` retourne un descripteur du socket nouvellement ou -1 en cas d'erreur.

Serveur : bind() : associer un port

```
bind (int descripteur, sockaddr localaddr, int addrlen);
```

- **descripteur** : celui du socket
- **localaddr** : structure de communication. Structure générique à 3 champs:
  - **sa_len** : longueur utile de l'adresse
  - **sa_family** famille de protocole (AF_INET pour TCP/IP)
  - **sa_data** est une chaîne de 14 caractères (au maximum) contenant l'adresse
- **addrlen** indique la taille du champ `localaddr` (utiliser `sizeof(localaddr)`)

=> renvoie 0 si ok, -1 sinon.

Serveur : attribution d'une adresse

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- La structure `sockaddr` est générique. On utilise ensuite des structures spécifiques au domaine de communication.

- Pour internet:

```
struct sockaddr_in {
       uint8_t   sin_len; /* longueur totale */
       sa_family sin_family; /* famille d'adresse */
       in_port_t sin_port; /* numero de port */
       struct in_addr sin_addr; /* valeur sur 32 bits de
l'adresse */
       char   sin_zero[8]; /* champ rempli de zeros */
};
```

Exemple bind()

```
struct sockaddr_in addr;

 addr.sin_family = AF_INET;
 addr.sin_port = htons(1100);
 addr.sin_addr.s_addr = INADDR_ANY;

 if(-1 == bind(s,(struct sockaddr *)&addr,
sizeof(addr)))
 {
     perror("error bind failed");
     close(s);
     exit(EXIT_FAILURE);
 }
```

- On choisit ici le port 1100.
- Les ports inférieurs à 1024 sont réservés au système.
- Si on avait mis 0, le système aurait choisi pour nous.

Serveur : écoute

```
listen (int descripteur, int maxconn);
```

=> permet l'écoute sur le port indiqué dans bind et traite chaque connexion par un appel à `accept()`

- **descripteur** : celui du socket
- **maxconn:** nombre maximum de demandes de connexions en attente (c-a-d nombre de connexions simultanées possibles).

Serveur : accepte connexion

```
int accept (int descripteur, struct sockaddr * addr, int *
addrlen)
```

- **descripteur** : celui du socket
- **sockaddr** : mis à jour avec les coordonnées de l'appelant
- addrlen : longueur de l'adresse

⇒ `accept()` :
- ⇒ extrait la première connexion en file d'attente
- ⇒ créé un **nouveau socket** avec les mêmes caractéristiques que le socket original
- ⇒ retourne l'**identificateur** de ce socket (ou INVALID_SOCKET en cas d'erreur)
- ⇒ si pas de connexion en file d'attente, se met en **attente bloquante**

  (sauf si socket marqué non-bloquant i.e. `fcntl(descripteur, F_SETFL, O_NONBLOCK);` auquel cas l'absence de connexions en attente provoque une erreur)

Client: établit connexion

```
int connect (int descripteur, struct sockaddr * addr, int *
addrlen)
```

- **descripteur** : celui du socket
- **sockaddr** : adresse à appeler
- addrlen : longueur de l'adresse

=> connecte() retourne 0 si la connexion s'est bien passée (-1 sinon)

Exemple connect()

```
struct sockaddr_in addr;

 addr.sin_family = AF_INET;
 addr.sin_port = htons(1100);
 inet_pton(AF_INET, "192.168.1.3", &addr.sin_addr);

 if(-1 == connect(s, (struct sockaddr *)&addr,
sizeof(addr)))
 {
     perror("connect failed");
     close(s);
     exit(EXIT_FAILURE);
 }
```

Serveur et Client: envoi

```
int send (int descripteur, const void *msg, int len, int
flags);
```

- **descripteur** : celui du socket
- **msg** : pointeur vers données à envoyer
- len : longueur du message en octets
- flags: habituellement 0

⇒ send() renvoie le nombre d'octets effectivement envoyés. Si tout n'a pas été envoyé, il faut renvoyer le reste.

```
char *msg ="Bonjour Monde";
bytes_sent = send(sockfd, msg, strlen(msg), 0);
```

Serveur et Client : réception

```
int recv (int descripteur, const void *msg, int len, int
flags);
```

- **descripteur** : celui du socket
- **msg** : pointeur vers données à envoyer
- len : longueur du message en octets
- flags: habituellement 0

⇒ recv() renvoie le nombre d'octets effectivement reçus

Serveur TCP 1/3 : socket

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define MYPORT 3490    /*port où les utilisateurs se connecteront */
#define BACKLOG 10     /* max connexions acceptées en file */

  main() {
    int sockfd, new_fd;  /*sock_fd, nouvelle connection sur new_fd */
    struct sockaddr_in my_addr;    /* Adresse */
    struct sockaddr_in their_addr; /* Adresse du connecté  */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
         perror("socket");
         exit(1);
    }
```

TCP (stream=connecté)

Serveur TCP 2/3

```
.../...

       my_addr.sin_family = AF_INET;
       my_addr.sin_port = htons(MYPORT)
       my_addr.sin_addr.s_addr = INADDR_ANY;   /*le syst va mettre mon addresse*/
       bzero(&(my_addr.sin_zero), 8);

       if (bind(sockfd, (struct sockaddr *)&my_addr,
                        sizeof(struct sockaddr))== -1) {
           perror("bind");
           exit(1);
       }

       if (listen(sockfd, BACKLOG) == -1) {
            perror("listen");
            exit(1);
        }


.../...
```

Serveur TCP 3/3: traitement données

Bloquant !

```
.../...
     while(1) {  /* main accept() loop */
          sin_size = sizeof(struct sockaddr_in);
          if ((new_fd = accept(sockfd,
                               (struct sockaddr *)&their_addr,
                                &sin_size)) == -1) {
              perror("accept");
              continue;
           }
           printf("serveur: Reçu connexion de %s\n",
                inet_ntoa(their_addr.sin_addr));
           if (!fork()) { /* processus fils */
              if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
                  perror("send");
              close(new_fd);
              exit(0);
           }
           close(new_fd);  /* Le parent n'a pas besoin de cela */
           /* Nettoyage des processus fils */
           while(waitpid(-1,NULL,WNOHANG) > 0);
       }
   }
```

Client TCP 1/2

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

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

#define THEPORT 3490    /*port sur le serveur */
#define MAXDATASIZE 100

  main(int argc, char **argv) {
    char buf[MAXDATASIZE];
    int sockfd;
    struct sockaddr_in srv_addr;    /* Adresse */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
         perror("socket");
         exit(1);
    }
```

Client TCP 2/2

```
 .../...
        if ((he=gethostbyname(argv[1])) == NULL) {
           herror("gethostbyname");
           exit(1);
        }
        srv_addr.sin_family = AF_INET;
        srv_addr.sin_port = htons(MYPORT)
        srv_addr.sin_addr = *((struct in_addr *)he->h_addr);
        bzero(&(srv_addr.sin_zero), 8);

        if (connect(sockfd, (struct sockaddr *)&srv_addr, sizeof(struct
sockaddr))== -1) {
            perror( "connect");
            exit(1);
        }

        if ((numbytes=recv(sockfd, buf, MAXDATASIZE, 0)) == -1) {
            perror("recv"); exit(1);
        }
        buf[numbytes] = '\0';
        printf("Reçu: %s",buf);
        close(sockfd);
        return 0;
    }
```

Server UDP

```
/***
 * IDEM entête server TCP
 ***/
  int addr_len, numbytes;

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
               perror("socket"); exit(1);
  }
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(MYPORT)
  my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
  bzero(&(my_addr.sin_zero), 8);

  if (bind(sockfd,
       (struct sockaddr *)&my_addr, sizeof(struct sockaddr))== -1) {
               perror("bind");
               exit(1);
        }
  }
  if ((numbytes=recvfrom(sockfd, buf, MAXBUFLEN, 0,
                (struct sockaddr *)&their_addr, &addr_len)) == -1) {
               perror("recvfrom"); exit(1);
  }
  printf("* Lu %d octets avec recvfrom(). Contenu=[%s]\n",numbytes,buf);
```

UDP

Bloquant !

Surveillance socket

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- accept() ou recvfrom() sont bloquants => comment gérer l'arrivée d'événements sur plusieurs descripteurs ?
  - ex1: on veut surveiller une entrée de caractères sur stdin, et un message reçu sur un socket TCP.
  - ex2: on veut surveiller plusieurs sockets en même temps
- Soit
  - créer plusieurs « fils d'exécution » (processus, threads)
  - utiliser `select()`

```
int select(int numfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);
```

Surveillance socket

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- `select()` peut surveiller des ensembles de sockets, de type lecture ou écriture.

- On ajoute les sockets à surveiller à des ensembles (fd_set)
- Macros usuelles :
  - FD_ZERO(fd_set *set) - initialise l'ensemble
  - FD_SET(int fd, fd_set *set) - Ajoute fd à l'ensemble
  - FD_CLR(int fd, fd_set *set) - Elimine fd de l'ensemble
  - FD_ISSET(int fd, fd_set *set) - test pour savoir si fd fait partie de l'ensemble
- Après, le timeout, select()
  - retourne le nombre total de descripteurs qui sont prêts
  - réduit les ensembles de descripteurs aux sous-ensembles de descripteurs prêts.

Exemple select()

```
 ...
if ((sockfd= socket(AF_INET, SOCK_STREAM, 0)) == -1) {
       perror("socket");exit(1);
}
if ((listen( sockfd, 10)==-1) {
       perror("listen");exit(1);
}

FD_ZERO(&readfds);
FD_SET(sockfd, &readfds);

while (select(sockfd+1, &readfds, NULL, NULL, 0)!=0) {
   if (FD_ISSET(sockfd, &readfds))
                printf("le descripteur %d contient une
donnée à lire !\n », sockfd);
}
```

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Partie 4 : Applications

Exemples de protocoles applicatifs : POP, HTTP

Protocoles Applicatifs

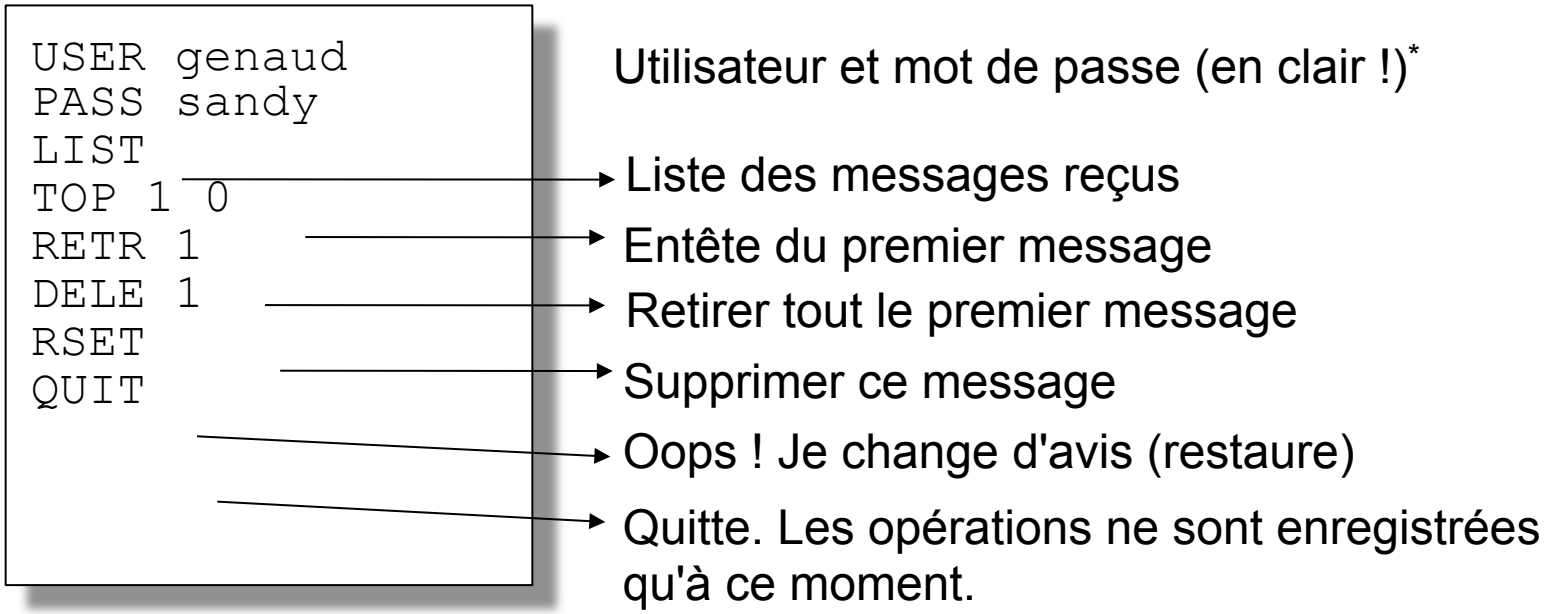
ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- Les protocoles applicatifs les plus répandus sont historiquement:
  - simples : un petit nombre de commandes suffit
  - textuels : évite les problèmes de codage, permet une compréhension/interaction humaine

- Exemples:
  - SMTP,
  - POP,
  - IMAP,
  - HTTP,
  - FTP,
  - ...

POP (Gestion mail)

Exemple de session POP 3



* sauf si `APOP` supporté par serveur

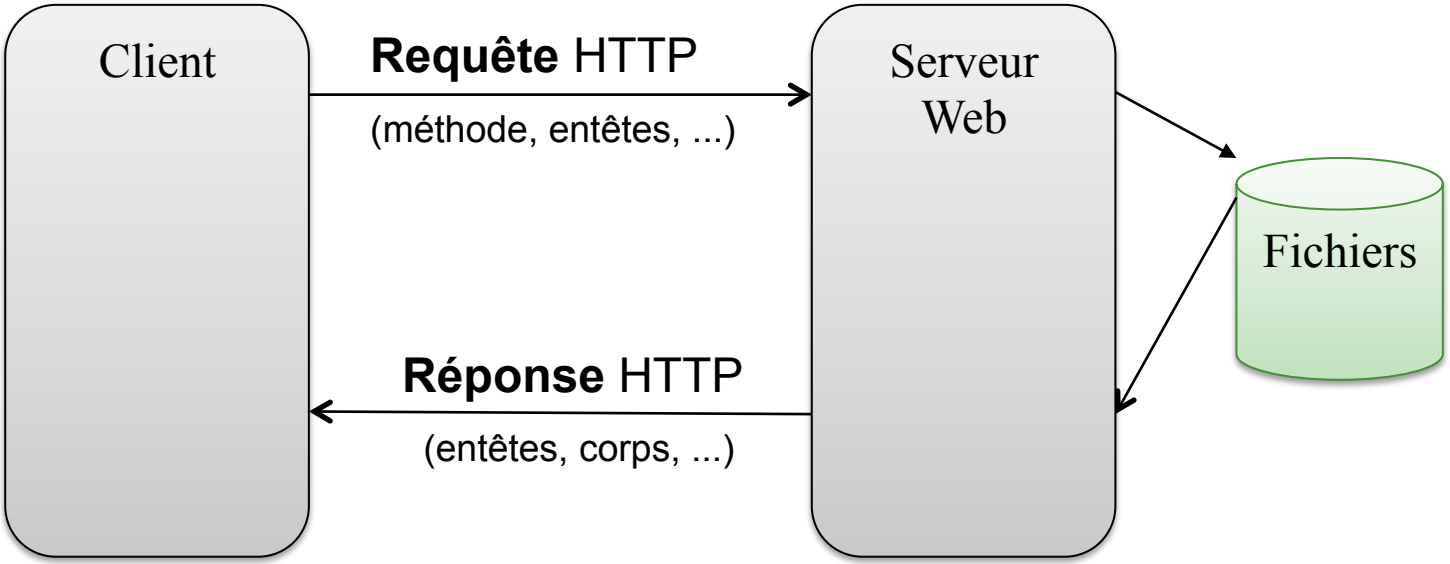
HTTP

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- HyperText Transfer Protocol (HTTP) : protocole de communication client-serveur de la couche application
- Peut utiliser une couche transport offrant connexion fiable (e.g TCP).
- Développé pour le World Wide Web. Les clients les plus nombreux sont les navigateurs web.

- Versions:
  - HTTP/1.0 (1996): standard de l'IETF, et est décrit dans la RFC 1945. Supporte les serveurs HTTP virtuels, la gestion de cache et l'identification.
  - HTTP/1.1 (1997), RFC 2068. Ajoute le support du transfert en *pipeline* et la négociation de type de contenu (format de données, langue).

HTTP


Client
Requête HTTP
(méthode, entêtes, ...)
Serveur Web
Fichiers
Réponse HTTP
(entêtes, corps, ...)

HTTP - Requête

<commande> <URI> <protocole>
<entête> (optionnel (sauf `Host` en HTTP/1.1))
...
<entête> (optionnel)
<ligne vide>
<corps> (optionnel)

```
POST /page.html HTTP/1.0
Host: mamachine.com
Referer: http://mamachine.com/redirect
User-Agent:Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:
1.9.1.7) Gecko/20091221 Firefox/3.5.7 (.NET CLR 3.5.30729)


name=toto&class=B&tel=049101234
```

HTTP - Commandes

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Commande	Effet
GET	Requête de la ressource située à l'URL spécifiée
HEAD	Requête de l'en-tête de la ressource
POST	Envoi de données à la ressource située à l'URL spécifiée
PUT	Envoi de données à l'URL spécifiée
DELETE	Suppression de la ressource située à l'URL spécifiée

HTTP – Entêtes Requêtes

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Exemples d'entêtes (non exhaustif)

Commande	Effet
Accept	format (mime type) attendu par le client
Accept-Charset	jeu de caractère attendu par le client
Accept-Encoding	Indique l'encoding supporté (e.g `Accept-Encoding: compress, gzip`)
Content-Length	longueur du corps de la requête (octets)
Content-Type	format (mime-type) du corps de la requête
Date	date de début de connexion
Host	précise site web cible. Nécessaire si plusieurs site webs virtuels sont hébergés à la même IP. (**obligatoire** en 1.1)
Referer	URL du lien à partir duquel la requête a été effectuée
User-Agent	Chaîne donnant des informations sur le client, comme le nom et la version du navigateur, du système d'exploitation

HTTP - Réponse

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

<protocole <code de statut> <signification status>
<entête> : <valeur>(optionnel)

...
<entête>: <valeur> (optionnel)
<ligne vide>
<corps> (optionnel)

```
HTTP/1.0 200 OK
Date : Sat, 15 Jan 2000 14:37:12 GMT
Server : Microsoft-IIS/2.0
Content-Type : text/HTML
Content-Length : 1245
Last-Modified : Fri, 14 Jan 2000 08:25:13 GMT
```

HTTP – Entêtes Réponse

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Exemples d'entêtes (non exhaustif)

Commande	Effet
Content-Encoding	type d'encode utilisé (e.g Content-Encoding : gzip)
Content-Language	langage de réponse
Content-Length	longueur du corps de la requête (octets)
Content-Type	format (mime-type) du corps de la requête
Date	date de début de connexion
Expires	date de péremption des données
Location	URL où se rediriger pour trouver les données
Server	chaîne donnant des informations sur serveur
Set-Cookie	place un cookie (e.g Set-Cookie: UserID=foobar; Max-Age=3600; Version=1)
Transfer-Encoding	manière dont sont transférés les données: chunked, compress, deflate, gzip, identity.

HTTP 1.1 vs 1.0

ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

- HTTP/1.1 ajoute:
  - méthode OPTIONS pour s'enquérir de capacités du serveur
  - Améliore le CACHING
  - Améliore le transfert réseau
    - Client peut indique quelle compression il supporte (Accept-Encoding)
    - Client peut demander une partie seulement du document (Range)
    - Serveur peut envoyer par morceau (Transfer-Encoding: chunked, chunked <taille> donne la taille du morceau)
    - Connexion persistante (Keep-alive) par défaut en HTTP/1.1