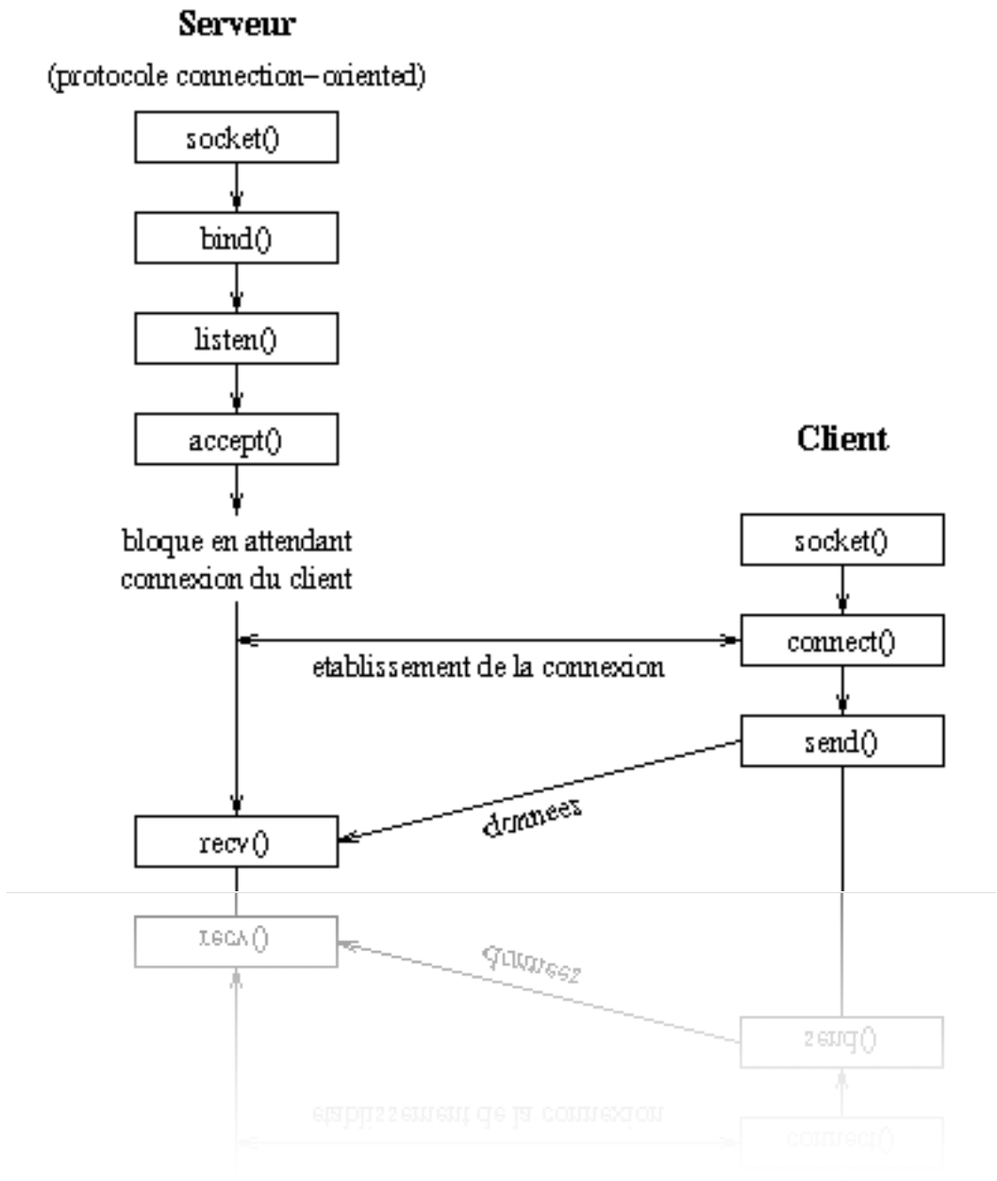


Communication distante

Communication distante

- Notion de communication distante
  - Communication entre deux processus présents sur deux machines différentes
  - Un processus est identifié par la combinaison de :
    - une adresse de machine, ex : adresse IP
    - un numéro de port : entier entre 0 et 65535

Communication distante

- Définition de rôles
  - pré-entente entre les processus
  - Le serveur :
    - fournit un ***service***, raison motivant la connexion du client
    - rôle : répondre aux demandes des clients
  - Le client :
    - envoie des demandes au serveur
    - initialise le dialogue client-serveur

Communication distante

- Réaliser une communication dans un réseau : 5 éléments
  - un protocole : définit les règles de communication
  - une adresse de machine locale (ex. adresse IP)
  - un port local : numéro de port associé au processus
  - une adresse de machine distante (ex. adresse IP)
  - un port distante : numéro de port associé au processus

Communication distante

- Support de la communication : la **socket**
  - interface logicielle
  - assure l'exploitation de manière aisée et uniforme des services d'un protocole réseau

- Intérêt de l'utilisation de socket
  - Abstraction des spécificités d'une connexion réseau
  - Facilité de programmation
  - Protocoles réseau intégrés
  - Porte de communication au travers de laquelle les processus extérieurs vont dialoguer

Communication distante

- Connexion en mode connecté
  - Mécanismes d'envoi / réception synchronisés
  - Données acquittés
  - Nécessite une phase d'établissement de connexion
  - Exemple : TCP
- Connexion en mode non connecté
  - Mécanismes asynchrones. Pas d'acquittement
  - Approche plus simple, mais communication non garantie
  - Exemple : UDP

Quel que soit le mode employé, la socket reste le support de communication

Gestion des sockets

- Identification de socket dans un processus
  - mécanisme identique à l'identification de fichier
  - une socket active est identifiée par un descripteur de socket
  - un descripteur de socket :
    - est stocké dans la table des descripteurs du processus
    - ne peut avoir la même valeur qu'un descripteur de fichier
      est également dupliqué en cas de **fork()**

- Création d'une socket
  - appel système **socket()**
  - ne requiert que le 1er des cinq éléments d'une connexion
    - nécessite un protocole
    - inutile de connaître les adresses locales et distantes

Création d'une socket

- Dans un programme : appel système ***socket()***
  - prototype : ***int socket (int PF, int type, int protocol)***
    - PF : famille de protocoles à utiliser
    - type : type de communication (connecté ou non)
    - protocol : nom du protocole à utiliser
  - code retour :
    - >0 : valeur du descripteur associé
    - -1 : erreur (protocole invalide, ...)

Création d'une socket

- Quelle famille de protocoles utiliser ?

Identifiant	Signification
PF_INET	Socket IPv4
PF_INET6	Socket IPv6
PF_CCITT	Interface X25
PF_LOCAL	Boucle locale
PF_KEY	Accès à une table de clé (IPsec)
PF_APPLETALK	Réseaux Apple
PX_IPX	Protocole Internet de Novell
...	...

Création d'une socket

- Quel type de protocole utiliser ?

Identifiant	Signification
SOCK_STREAM	Mode connecté (couche transport)
SOCK_DGRAM	Mode non connecté (couche transport)
SOCK_RAW	Dialogue direct avec la couche IP

Création d'une socket

- Quel protocole utiliser ?

Identifiant	Signification
IPPROTO_TCP	Protocole TCP
IPPROTO_UDP	Protocole UDP
IPPROTO_RAW	Dialogue direct avec IP (SOCK_RAW)
IPPROTO_ICMP	Protocole ICMP (SOCK_RAW)

  - En général : protocole mis à 0
  - L'association de la famille de protocole et du type de communication définit explicitement le protocole :
    - PF_INET + SOCK_STREAM => TCP = IPPROTO_TCP
    - PF_INET + SOCK_DGRAM => UDP = IPPROTO_UDP

Attribution d'une adresse

- Adresse d'un processus
  - sur internet : composée d'une IP et d'un numéro de port
  - **Au niveau du serveur :**
    - Nécessité de spécifier le numéro de port utilisé pour que les clients sachent où se connecter
    - Si la machine du serveur possède plusieurs adresses IP, spécifier sur quelle(s) adresse(s) IP le service doit s'exécuter
  - **Au niveau du client :**
    - pas besoin de spécifier un numéro de port
    - attribué automatiquement et aléatoirement
    - mais besoin de spécifier l'adresse de la socket du serveur

Attribution d'une adresse

- Comment décrire une adresse de processus?
  - L'adresse change selon la famille de protocoles
  - Passage de l'adresse au travers d'une structure générale
    - **struct sockaddr**
  - Chaque famille de protocole possède sa propre structure :
    - PF_INET : struct sockaddr_in
    - PF_INET6 : struct sockaddr_in6
    - PF_LOCAL : struct sockaddr_un
    - ...
  - Dans les déclarations de fonctions : type général sockaddr
  - Dans leur appel, :
    - structure correspondant à la famille de protocole utilisée
    - on «fait passer» (transtypage) la structure utilisée pour une structure de type sockaddr.

Attribution d'une adresse

- La structure sockaddr_in
  - Famille de protocole PF_INET : IPv4

```
struct sockaddr_in {
 short           sin_family;   // PF_INET
 unsigned short  sin_port;     // port
 struct in_addr  sin_addr;     // 32 bits
 char            sin_zero[8]; // inutilisé
};

struct in_addr {
 unsigned long   s_addr;  // adresse 32 bits
};
```

Attribution d'une adresse

- Remplir les éléments de la structure sockaddr_in
  - Le port doit être converti au format réseau :
    - le codage de la valeur du port doit être adapté
    - fonction **htons( )**
  - L'adresse doit être convertie au format réseau :
    - Adresse en notation à points "xxx.xxx.xxx.xxx"
    - Doit être convertie au format réseau
    - fonction **inet_addr()**
  - Dans le cas d'un serveur :
    - Si plusieurs adresses IP, intéressant et pertinent d'associer le serveur avec toutes les adresses IP
    - Entier *INADDR_ANY* à convertir au format réseau
    - fonction **htonl()**

Attribution d'une adresse

- Coté Client :
  - Structure contenant l'adresse du serveur auquel se connecter
  - Exemple : 82.34.54.63 sur le port 80

```
struct sockaddr_in servAddr;
...
servAddr.sin_family = PF_INET;
servAddr.sin_port = htons(80);
servAddr.sin_addr.saddr = inet_addr("82.34.54.63");
...
```

Attribution d'une adresse

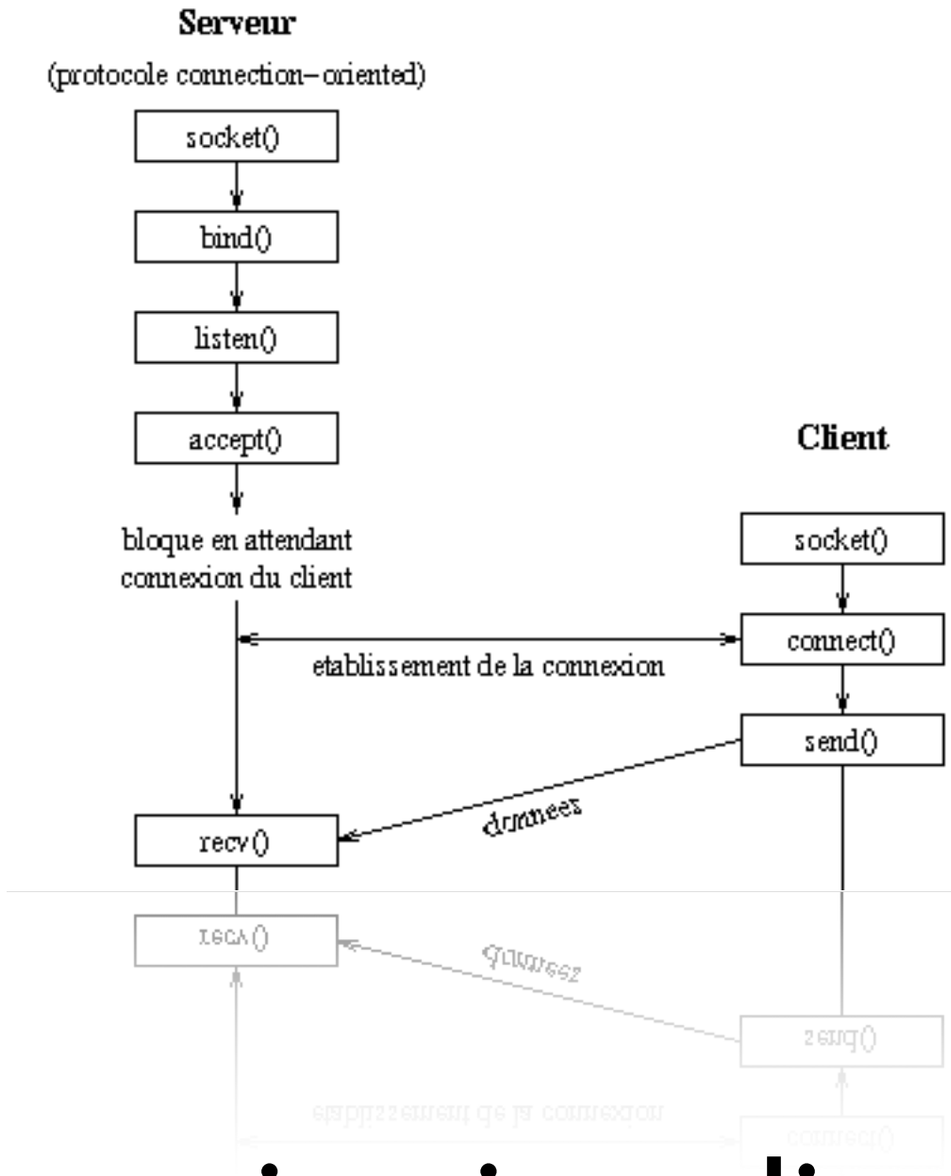
- Coté Serveur :
  - Structure contenant l'adresse du service à mettre en place
  - Exemple : toutes les adresses IP du serveur sur le port 80

```
struct sockaddr_in myAddr;
...
myAddr.sin_family = PF_INET;
myAddr.sin_port = htons(80);
myAddr.sin_addr.saddr = htonl(INADDR_ANY);
...
```

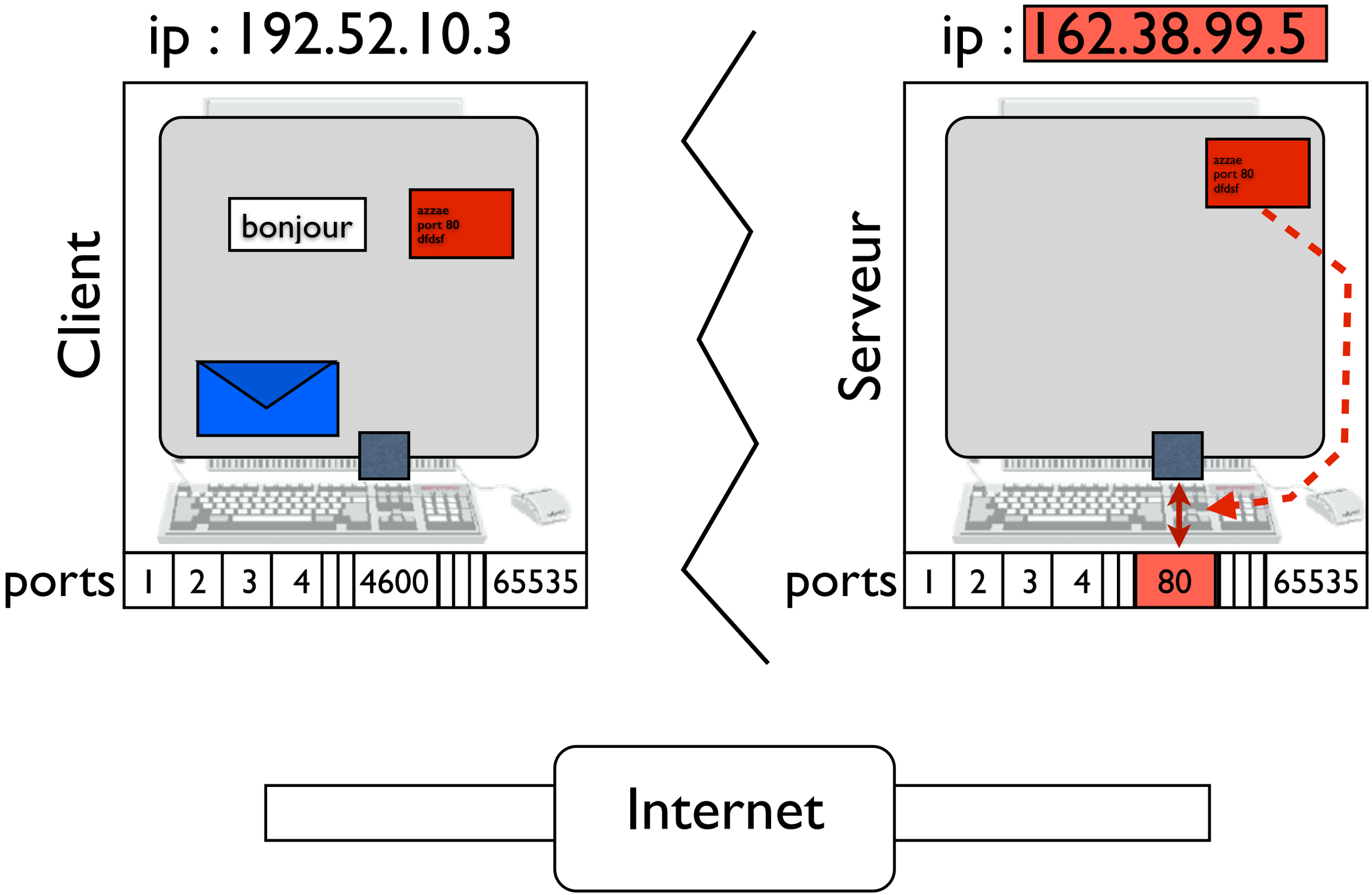
Attribution d'une adresse

- Coté Serveur :
  Associer l'adresse du serveur à la socket locale
  - appel système **bind()**

- Dans un programme : appel système ***bind()***
  - prototype : ***int bind (int socket, struct sockaddr * address, socklen_t address_len)***
  - socket : descripteur de la socket
  - address : structure contenant l'adresse à associer
  - address_len : taille de la structure contenant l'adresse
  - code retour :
  - 0 : attribution d'adresse OK
  - -1 : erreur (protocole invalide, ...)



Communication distante : Le mode non connecté


ip : 192.52.10.3
Client
bonjour
azzae
port 80
dfdsf
ports
1
2
3
4
4600
65535
ip : 162.38.99.5
Serveur
azzae
port 80
dfdsf
ports
1
2
3
4
80
65535
Internet

ip : 192.52.10.3

Client

azzae
port 80
dfdsf

azzae
port 80
dfdsf

ports | 1 | 2 | 3 | 4 | 4600 | 65535

ip : 162.38.99.5

Serveur

azzae
port 80
dfdsf

bonjour

ports | 1 | 2 | 3 | 4 | 80 | 65535

Internet

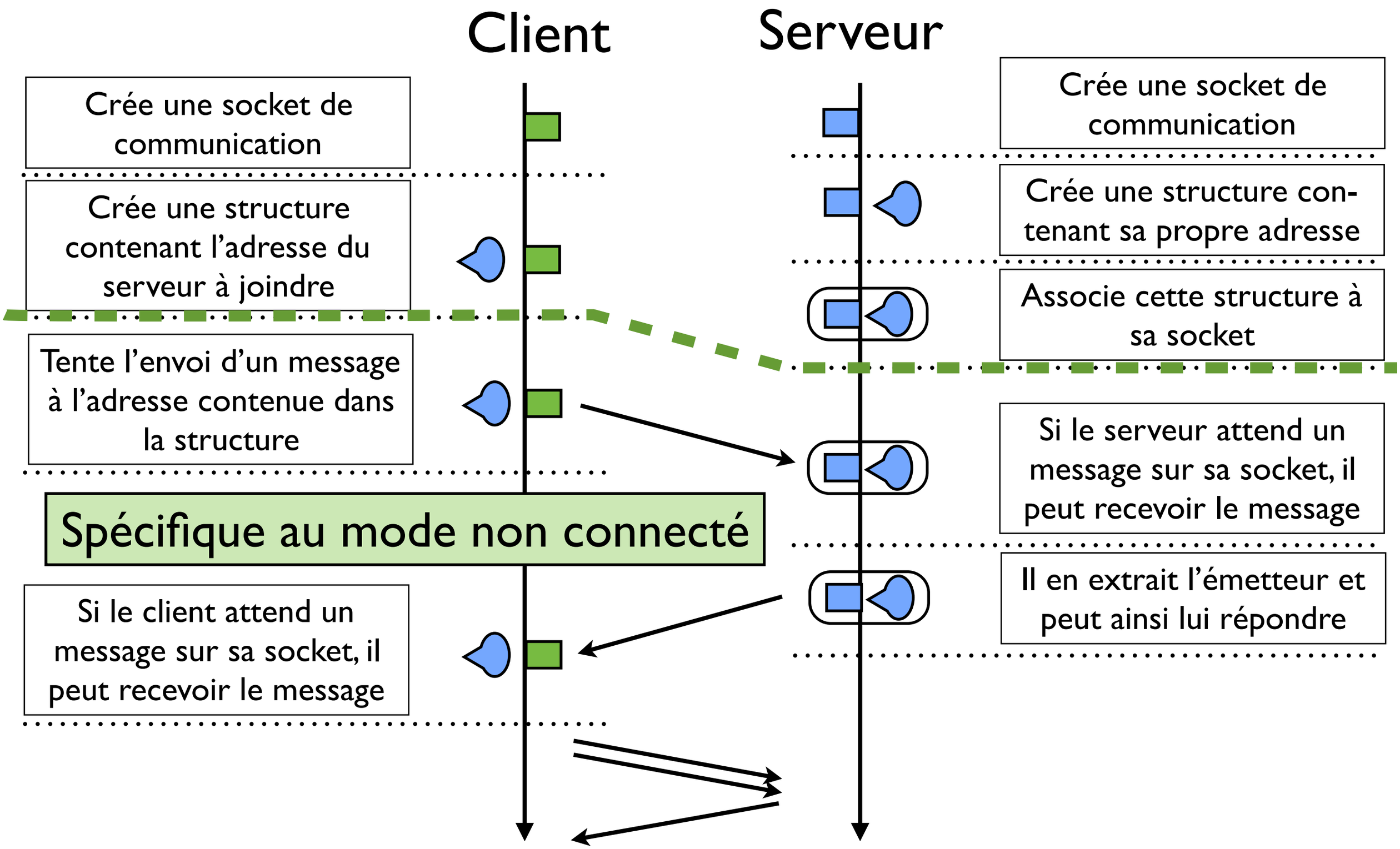
Le mode non connecté

- Principe du mode non connecté
  - Mécanismes asynchrones.
  - Pas d'acquittement
  - Communication non garantie
  - Perte de paquets possibles
  - On parlera de datagrammes transmis

Le mode non connecté en bref

- Un client
  - crée une socket de communication
  - crée une structure contenant l'adresse du serveur
  - envoie un message à l'adresse contenue dans la structure
- Un serveur
  - crée une socket de communication et lui attribue un nom
  - écoute sur cette socket en attente de messages
  - lorsqu'un message arrive :
    - lecture du message
    - récupère une structure contenant l'adresse de l'émetteur
    - peut donc répondre à l'émetteur (si ce dernier sait qu'il doit attendre une réponse du serveur !)

Le mode non connecté en bref



Envoi et réception de messages

- Recevoir un message en mode non connecté
  - Appel système **recvfrom()**
  - Reçoit un message
  - Remplit une structure avec les coordonnées de l'émetteur

- Envoyer un message en mode non connecté
  - Appel système **sendto()**
  - Envoi un message vers l'adresse indiquée dans une structure
  - L'émetteur doit être en position de recevoir

Réception en mode non connecté

- Dans un programme : appel système ***recvfrom()***
  - prototype : ***int recvfrom (int socket, void *buffer, size_t length, int flags, struct sockaddr *address, socklen_t address_len)***
    - socket : descripteur de la socket de communication
    - buffer : zone dans laquelle stocker le message reçu
    - length : taille max du buffer
    - flags : options facultatives
    - address : structure qui va contenir l'adresse de l'émetteur
    - address_len : taille de la structure stockant l'adresse
  - code retour :
    - > 0 : nombre d'octets reçus
    - -1 : erreur (mauvaise socket, timeout de connexion...)

Envoi en mode non connecté

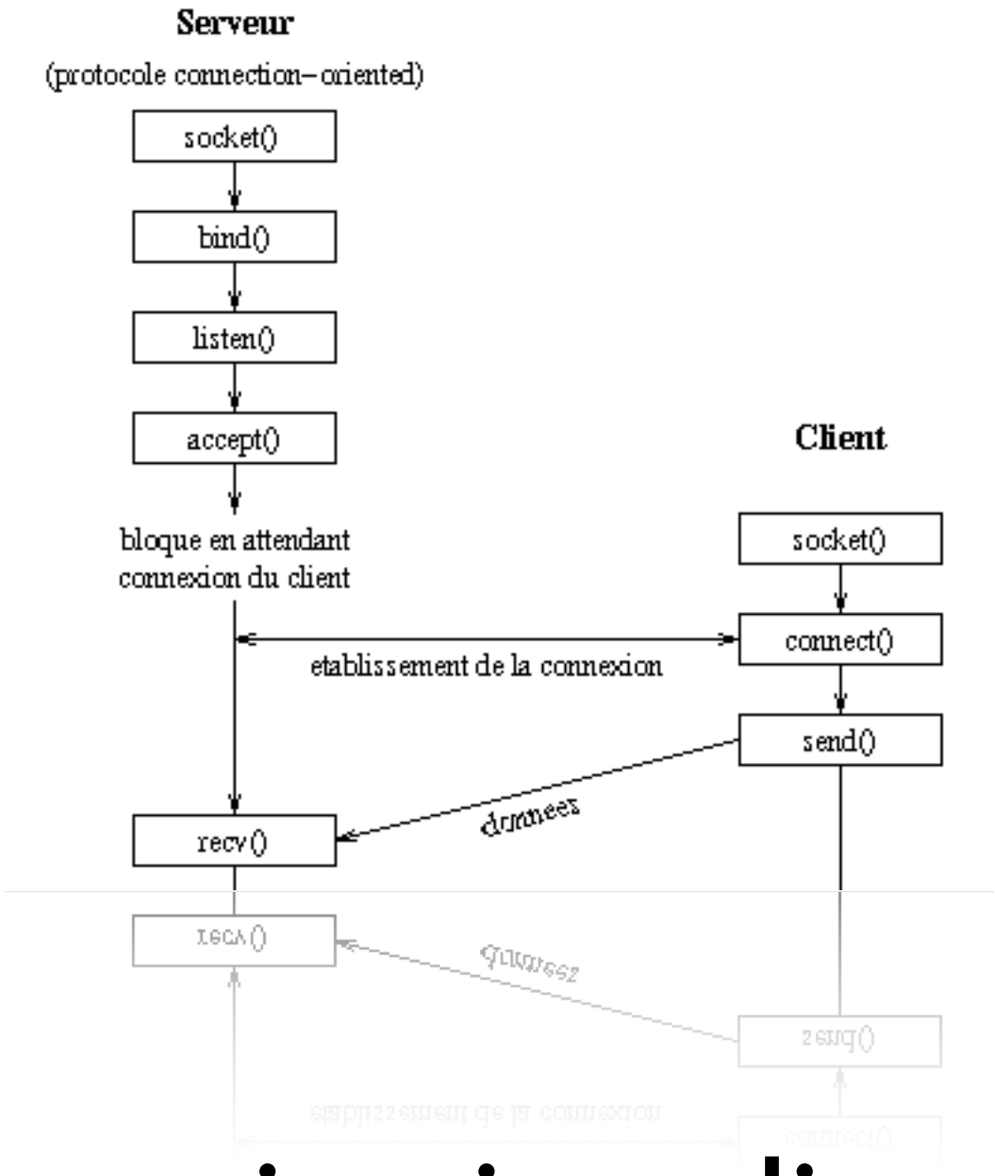
- Dans un programme : appel système ***sendto()***
  - prototype : ***int sendto (int socket, const void *buffer, size_t length, int flags, const struct sockaddr *address, socklen_t address_len)***
    - socket : descripteur de la socket de communication
    - buffer : zone dans laquelle lire le message à envoyer
    - length : taille du message à envoyer
    - flags : options facultatives
    - address : structure contenant l'adresse du destinataire
    - address_len : taille de la structure stockant l'adresse
  - code retour :
    - > 0 : nombre d'octets envoyés
    - -1 : erreur (mauvaise socket, timeout de connexion...)

Serveur perroquet non connecté

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
  int sock = socket(PF_INET,SOCK_DGRAM,0);      // socket en mode non connecté UDP
  struct sockaddr_in servAddr, cliAddr;
  int addrLength=sizeof(struct sockaddr_in);
  int recus;                                    // stockage du nombre d'octets recus
  char buffer[1024];                            // buffer pour lecture du message

  servAddr.sin_addr.s_addr = htonl(INADDR_ANY); // association sur toutes les adresses
  servAddr.sin_family= AF_INET;                 // famille d'adresse IPv4
  servAddr.sin_port   = htons(4444);            // port 4444

  bind (sock, (struct sockaddr *) & servAddr, addrLength); // attribution du nom

  while (1) {                                   // boucle infinie
    recus = recvfrom(sock, buffer, 1024, 0,(struct sockaddr *)& cliAddr,&addrLength);
    sendto(sock, buffer, recus, 0, (struct sockaddr *)& cliAddr,&addrLength);
  }
}
```

Client en mode non connecté

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
  int sock = socket(PF_INET, SOCK_DGRAM,0);      // socket en mode non connecté UDP
  struct sockaddr_in servAddr, rcptAddr;
  int addrLength=sizeof(struct sockaddr_in);
  int recus;                                     // stockage du nombre d'octets recus
  char buffer[1024];

  servAddr.sin_addr.s_addr = htonl("1.2.3.4");  // adresse IP du serveur
  servAddr.sin_family= AF_INET;                 // famille d'adresse IPv4
  servAddr.sin_port  = htons(4444);               // port 4444

  // envoi vers serveur d'un message hello
  sendto(sock, "Hello World ", 11, 0, (struct sockaddr *)& servAddr, addrLength);

  // en attente de la réception de la réponse du serveur
  recus = recvfrom(sock, buffer, 1024, 0,(struct sockaddr *)& rcptAddr,&addrLength);

  close(sock);                                    // fermeture de la socket
  }
}
```



Communication distante : Le mode connecté

ip : 192.52.10.3

ip : 162.38.99.5

Client

bonjour

azzae
port 80
dfdsf

connection?

Serveur

azzae
port 80
dfdsf

ports 1 2 3 4 4600 65535

ports 1 2 3 4 80 65535

Internet

ip : 192.52.10.3

Client

bonjour

azzae
port 80
dfdsf

azz
por
dfd

azzae
port 80
dfdsf

ports 1 2 3 4 4600 65535

ip : 162.38.99.5

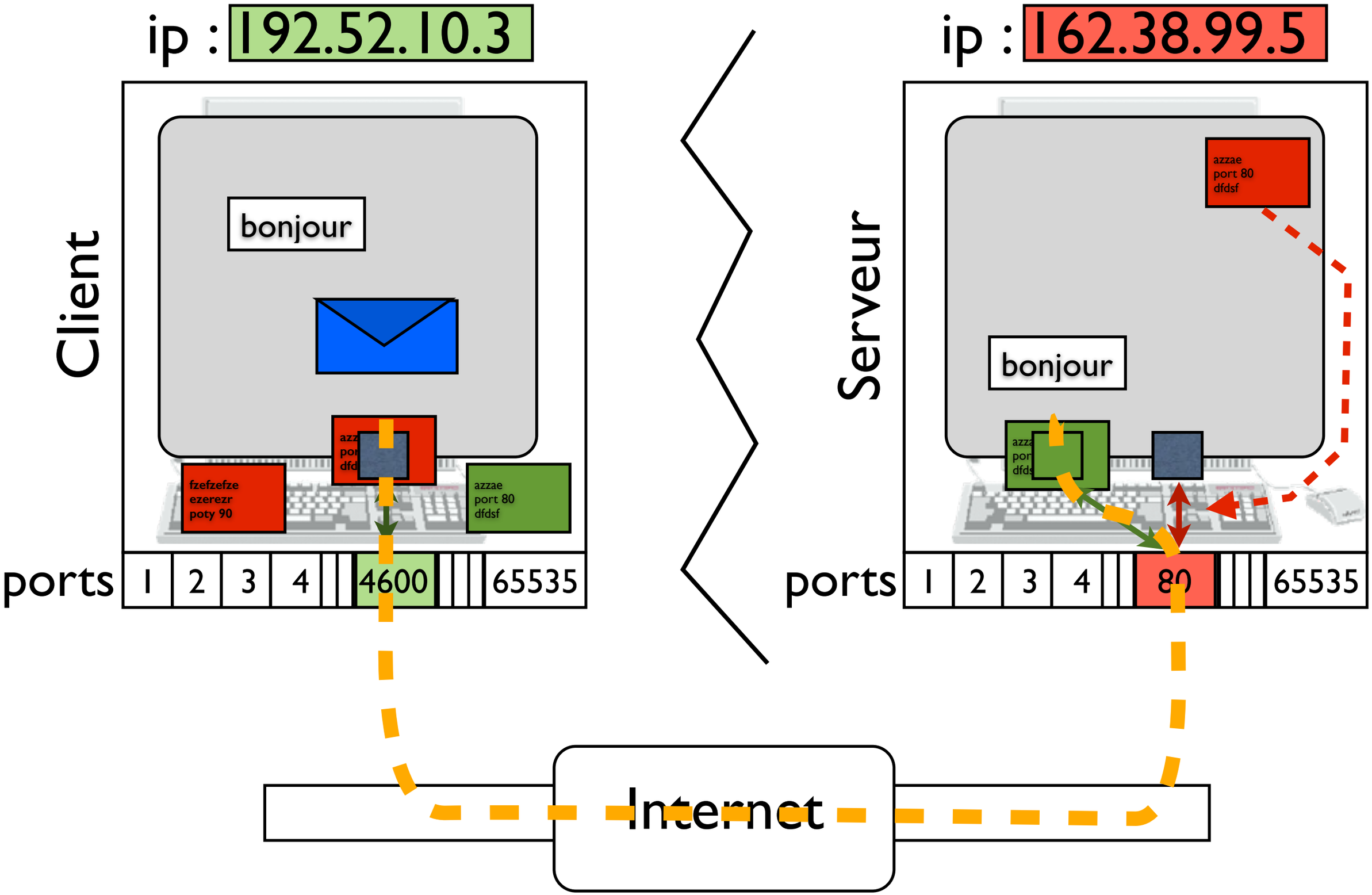
Serveur

azzae
port 80
dfdsf

connection?

ports 1 2 3 4 80 65535

Internet


ip : 192.52.10.3
Client
bonjour
azz
por
dfd
fzefzefze
ezerezr
poty 90
azzae
port 80
dfdsf
ports
1
2
3
4
4600
65535
ip : 162.38.99.5
Serveur
azzae
port 80
dfdsf
bonjour
azza
por
dfds
ports
1
2
3
4
80
65535
Internet

Le mode connecté

- Principe du mode connecté
  - Transfert des données synchronisé
  - Données transférées en flots
  - Vérification de la bonne réception des données
  - Etablissement d'un liaison virtuelle entre les deux processus

Le mode connecté en bref

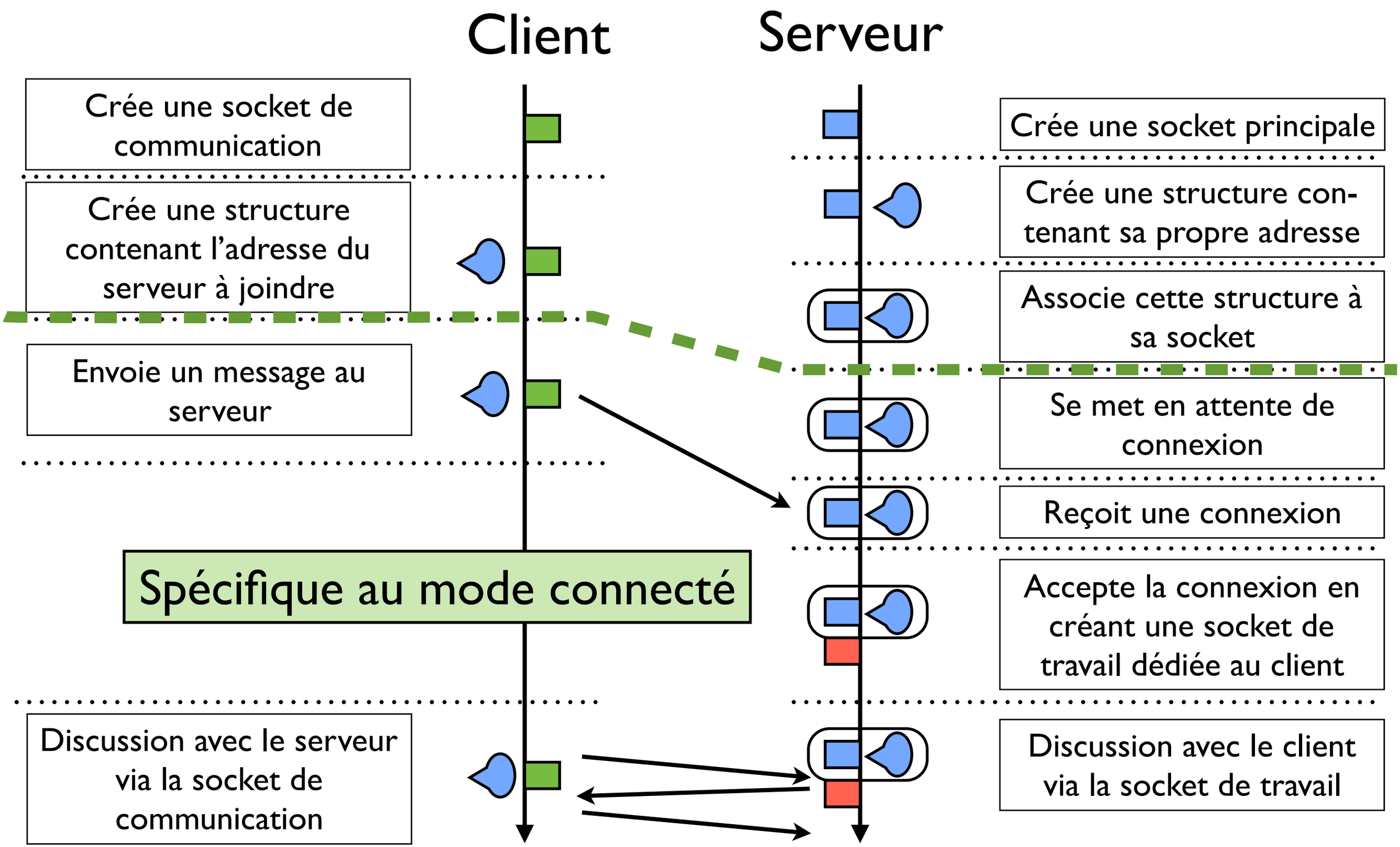
- Un client
  - crée une socket de communication
  - crée une structure contenant l'adresse du serveur
  - se connecte vers un serveur au travers de cette socket

- Un serveur
  - Doit pouvoir gérer plusieurs clients simultanément
  - crée une socket principale et lui attribue un nom
  - écoute sur cette socket : en attente de clients
  - lorsqu'un client se connecte :
    - création d'une socket dédiée, dite socket de travail
    - dialogue avec le client sur la socket de travail
  - autant de sockets de travail que de clients connectés

Le mode connecté en bref

- Une fois la connexion établie ...
  - Client et serveur peuvent communiquer simplement en lisant ou en écrivant sur les socket mises en place
  - Le client :
    - écrit sur sa socket les messages à destination du serveur
    - lit sur cette dernière les messages provenant du serveur
  - Le serveur :
    - écrit sur la socket de travail dédiée au client les messages à destination de ce dernier
    - lit sur cette socket les messages provenant de ce client

Le mode connecté en bref



Demande de connexion

- Coté Client :
  Initier une connexion vers le serveur
  - lancer une demande de connexion vers une adresse connue
  - initier cette connexion sur la socket locale
  - appel système **connect()**
  - appel bloquant :
  - tente de se connecter jusqu'à un succès ou un timeout

Demande de connexion

- Dans un programme : appel système ***connect()***
  - prototype : ***int connect (int socket, const struct sockaddr *address, socklen_t address_len)***
    - socket : descripteur de la socket
    - address : structure contenant l'adresse de destination
    - address_len : taille de la structure contenant l'adresse
  - code retour :
    - 0 : attribution d'adresse OK
    - -1 : erreur (timeout, socket déja connectée ...)

Acceptation d'une connexion

- Acceptation d'une connexion par le serveur
  - Appel système **accept()**
  - Retourne un descripteur de socket dédié à un client

- Dans un programme : appel système ***accept()***
  - prototype : ***int accept (int socket, struct sockaddr *address, socklen_t address_len)***
    - socket : descripteur de la socket principale
    - address : structure qui va contenir l'adresse du client
    - address_len : taille de la structure stockant l'adresse
  - code retour :
    - > 0 : descripteur de socket retourné
    - -1 : erreur (mauvaise socket principale ...)

Discuter en mode connecté

- Discussion possible une fois la connexion établie
  - Les concepteurs ont tenté de mettre en place des API aussi proches que possible que l'écriture / lecture sur fichier
  - On peut utiliser les appels système **write()** et **read()**
  - **read()** : lire un message envoyé par le processus distant
  - **write()** : écrire un message au processus distant
  - utilisation limitée à UNIX (ne marche pas sous windows)
  - mécanismes simplifiés
- Mécanismes dédiés au mode connecté
  - routines spécifiques à la communication en mode connecté
  - appels systèmes **send()** et **recv()**
  - ajout d'options, gestion différente de la déconnexion.
  - préconisé pour le dialogue en mode connecté

Fermeture d'une socket

- Pourquoi fermer une socket ?
  - supprimer le descripteur de la table des descripteurs
  - informer le processus distant de la fin de connexion

- Dans un programme : appel système ***close()***
  - prototype : ***int close (int fd)***
    - fd : descripteur de socket à fermer
  - code retour :
    - 0 : descripteur correctement fermé
    - -1 : erreur (mauvais paramètre, erreur d'entrée sortie ...)

Serveur «hello» connecté

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
  int sockPrim = socket(AF_INET,SOCK_STREAM,0); // socket en mode connecté TCP
  struct sockaddr_in servAddr, cliAddr;
  int addrLength=sizeof(struct sockaddr_in);
  int sockWork;                                 // descripteur de la socket de travail

  servAddr.sin_addr.s_addr = htonl(INADDR_ANY); // association sur toutes les adresses
  servAddr.sin_family= AF_INET;                 // famille d'adresse IPv4
  servAddr.sin_port   = htons(4444);            // port 4444

  bind (sockPrim, (struct sockaddr *) & servAddr, addrLength); // attribution du nom

  listen(sockPrim,10);                          // socket principale en mode écoute

  while (1) {                                   // boucle infinie
    sockWork = accept(sockPrim, (struct sockaddr *)& cliAddr,&addrLength);
    send(sockWork, "hello world", 11, 0);       // envoi du message
    close(sockWork);                            // fermeture du descripteur
  }
}
```

Client «hello» connecté

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
  int sockComm = socket(AF_INET,SOCK_STREAM,0); // socket en mode connecté TCP
  struct sockaddr_in servAddr;
  int addrLength=sizeof(struct sockaddr_in);
  int buffer[1024];                             // buffer accueillant le message

  servAddr.sin_addr.s_addr = htonl("1.2.3.4");  // IP du serveur
  servAddr.sin_family= AF_INET;                 // famille d'adresse IPv4
  servAddr.sin_port  = htons(4444);               // port 4444

  connect(sockComm, (struct sockaddr *)& servAddr, addrLength);
  recv(sockComm, buffer, 1023, 0);              // reception du message

  close(sockWork);                              // fermeture du descripteur

  exit(0);
}
```