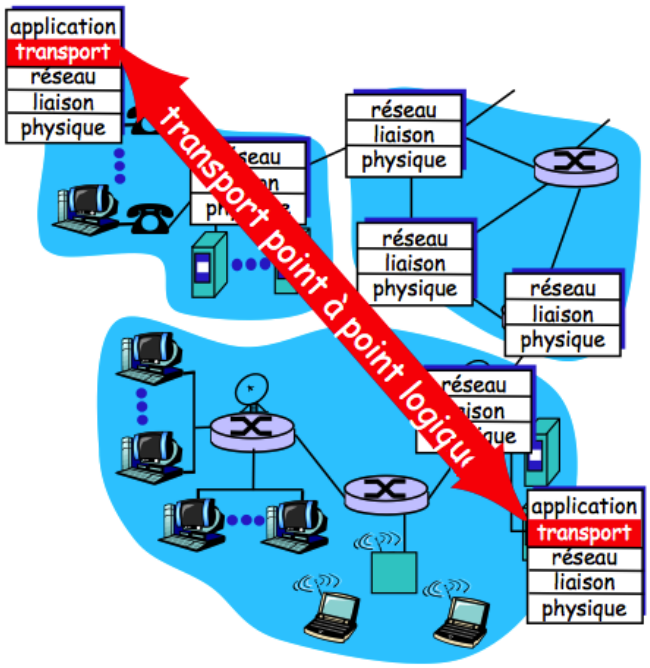
Chapitre 7

Couche Transport

Le service fourni par les protocoles de la couche réseaux (IP par exemple) n'étant pas fiable, il faut implanter par dessus un protocole supplémentaire, en fonction de la qualité de service dont les applications ont besoin. Ce protocole est appelé protocole de transport et appartient à la couche OSI numéro 4, la couche transport.

La couche transport assure une communication entre deux machines dans un réseau quelque soit leur localisation en masquant les détails vus dans la couche réseaux tel que le routage. Elle fournit donc une communication logique de bout en bout entre processus tournant sur des machines différentes. Les protocoles de transport tournent sur les hôtes et non pas sur les routeurs.



Il est important de noter qu'il existe une grande ressemblance entres les fonctionnalités de la couche transport 4 et la couche liaison 2. Cependant, la couche liaison adresse las machines tandis que la couche transport adresse les applications. Les tâche essentielles de la couche transport sont les suivantes :

- Segmentation des messages,
- Séquencement des messages,
- Multiplexage et démultiplexage des données des applications,
- Protection contre les éventuelles erreurs,
- Assurer les qualité de service souhaitée par les applications.

7.1 Segmentation des messages

La couche transport émettrice divise (fragmente) le message reçu de la couche supérieure (session pour OSI et application pour TCP/IP) en segments qui les passe à la couche réseau. L'unité de données manipulée par le ouche transport est donc le segment. La couche transport réceptrice, reforme le message à partir des segments obtenus de la couche réseau.

7.2 Séquencement des messages

Si le service offert par la couche réseau n'est pas fiable, la couche transport doit le fiabiliser en garantissant la reconstruction correcte du message pour les application ayant besoin. Pour cela les segments doivent être numérotés et acquittés exactement comme dans la couche liaison, mais ici entre deux applications distantes.

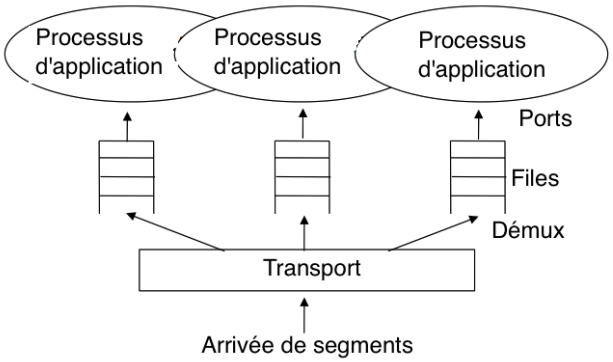
7.3 Multiplexage et démultiplexage

Puisque plusieurs application peuvent tourner sur une machine et peuvent toutes envoyer et recevoir des messages, la couche transport doit pouvoir :

- lors de l'émission, collecter ces données et et les encaplsuler dans le même médium : service multiplexage,
- lors de la réception, livrer les segments reçus aux bonnes applications : service de démultiplexage.

Pour pouvoir distinguer les applications (processus) des unes des autres, elles doivent être identifiées par des adresses. Chaque processus utilisant le réseau doit obtenir une adresse. Par exemple, dans le protocole TCP d'Internet on utilise les numéro de port pour

adresser les applications. Si les trames portent des adresses MAC et les paquets des adresses IP, les segments portent des numéros de port.



Dans le protocole TCP, le numéro de port est un entier représenté sur deux octets. Par exemple, les applications http utilisent le port numéro 80.

7.4 Protection contre les erreurs

En plus de la protection contre les erreurs effectuée à chaque liaison, la couche transport en effectue une autre de bout en bout. Généralement, c'est une somme (checksum) effectuée sur la totalité du segment envoyé. Le champ Checksum est codé sur 16 bits et représente la validité du paquet de la couche 4 TCP.

Pour TCP, le Checksum est constitué en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets de l'en-tête et des données pris deux par deux (mots de 16 bits).

Par example, soit le segment à envoyer est "0x450000304422400080060000 8c7c19acae241e2b". On commence par diviser le segment en blocs de 16 bits chacun puis calculer leurs somme :

$$4500 + 0030 + 4422 + 4000 + 8006 + 0000 + 8c7c + 19ac + ae24 + 1e2b = 2BBCF$$

Puis mettre le résultat en 16 bits par l'ajout du reste au résultat :

$$2 + BBCF = BBD1 = 1011101111010001$$

Finalement, on calcule le complément à 1 du résultat

$$\text{checksum} = \text{complement à un } (1011101111010001) = 0100010000101110 = 442E$$

La vérification chez le récepteur est faite en utilisant le même algorithme avec l'initialisation du checksum à la valeur 442E.

2BBCF + 442E = 2FFFD, alors 2 + FFFD = FFFF

On prend le complément à un de FFFF = 0.

Le segment est correct si le checksum est à 0.

7.5 Protocoles de transport

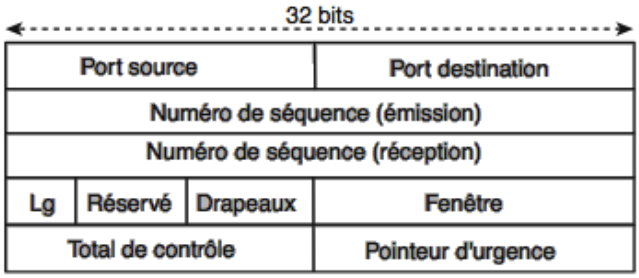
Les applications utilisant les réseaux peuvent avoir besoin de deux type de service : connecté et non connecté. Le service connecté peut être utilisé lorsqu'on connait l'adresse du destinataire au préalable tel que les serveurs http, ou ftp. Tandis que le service non connecté peut être utilisé par exemple pour la diffusion. Le monde d'Internet (TCP/IP) utilise les deux protocoles TCP et UDP correspondant chacun à un service.

7.5.1 TCP

TCP (Transport Control Protocol) assure un service de transmission de données de bout en bout, connecté, fiable avec détection et correction d'erreurs .

7.5.1.1 Format d'un segment TCP

Un segment TCP est constitué comme suit :



- Port Source (16 bits) : Numéro du port utilisé par l'application en cours sur la machine source.
- Port Destination (16 bits) : Numéro du port relatif à l'application en cours sur la machine de destination.
- Numéro d'ordre (32 bits) : numéro du premier octet du flux de données qui sera transmis (Initial Sequence Number)
- Numéro d'accusé de réception (32 bits) : Numéro d'ordre du dernier octet reçu par le récepteur (par rapport à tous les octets du flot de données reçues).

- Longueur en-tête (4 bits) : Il permet de repérer le début des données dans le segment. Ce décalage est essentiel, car il est possible que l'en-tête contienne un champ d'options de taille variable. Un en-tête sans option contient 20 octets, donc le champ longueur contient la valeur 5, l'unité étant le mot de 32 bits (soit 4 octets).
- Réservé (6 bits) : Champ inutilisé.
- Drapeaux ou flags (6 bits) Ces bits sont à considérer individuellement :
  - URG (Urgent) : Si ce drapeau est à 1, le segment transporte des données urgentes dont la place est indiquée par le champ Pointeur d'urgence (voir ci-après).
  - ACK (Acknowledgement) : Si ce drapeau est à 1, le segment transporte un accusé de réception.
  - PSH (Push). Si ce drapeau est à 1, le module TCP récepteur ne doit pas attendre que son tampon de réception soit plein pour délivrer les données à l'application. Au contraire, il doit délivrer le segment immédiatement, quel que soit l'état de son tampon (méthode Push).
  - RST (Reset) : Si ce drapeau est à 1, la connexion est interrompue.
  - SYN (Synchronize) : Si ce drapeau est à 1, les numéros d'ordre sont synchronisés (il s'agit de l'ouverture de connexion).
  - FIN (Final) : Si ce drapeau est à 1, la connexion se termine normalement.
- Fenêtre (16 bits) : Champ permettant de connaître le nombre d'octets que le récepteur est capable de recevoir sans accusé de réception.
- Total de contrôle ou checksum (16 bits). Le total de contrôle est réalisé en faisant la somme des champs de données et de l'en-tête. Il est calculé par le module TCP émetteur et permet au module TCP récepteur de vérifier l'intégrité du segment reçu.
- Pointeur d'urgence (16 bits). Indique le rang à partir duquel l'information est une donnée urgente.
- Options (taille variable).

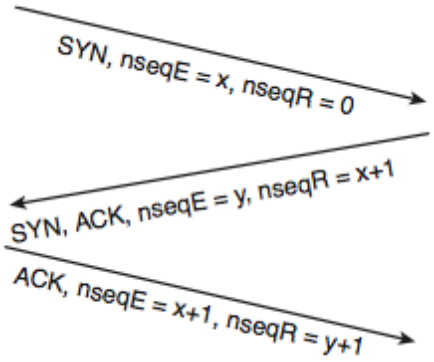
7.5.1.2 Connexion

TCP est un protocole qui fonctionne en mode client/serveur : l'un des utilisateurs est le serveur offrant des services (serveur http : site web), l'autre est le client (client http : navigateur web) qui utilise les services proposés par le serveur.

Le serveur doit être initialisé le premier ; on dit qu'il exécute une ouverture passive. Dès qu'il est opérationnel, il attend les demandes des clients, qui peuvent alors faire une ouverture active. Pour ouvrir une connexion, les clients doivent connaître le numéro de

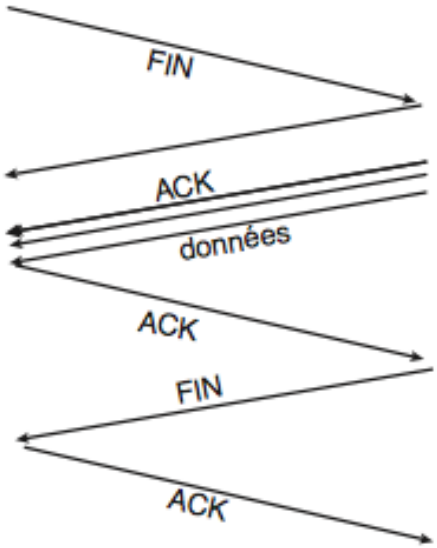
port de l'application distante. En général, les serveurs utilisent des numéros de ports bien connus, mais il est possible d'implanter une application sur un numéro de port quelconque. Il faut alors prévenir les clients pour qu'ils sachent le numéro de port à utiliser.

Le client ouvre la connexion en envoyant un premier segment, parfois appelé séquence de synchronisation. La connexion est établie en trois phases (*three-way-handshake*) en spécifiant les numéro de séquences des octets à échanger.



Une fois la connexion établie, elle peut être utilisée pour échanger les segments dans les deux sens.

La déconnexion est indépendante pour les deux sens ; un échange dans un sens peut être interrompu en gardant celui de l'autre sens. Pour fermer la connexion dans les deux sens, on utilise quatre phases.



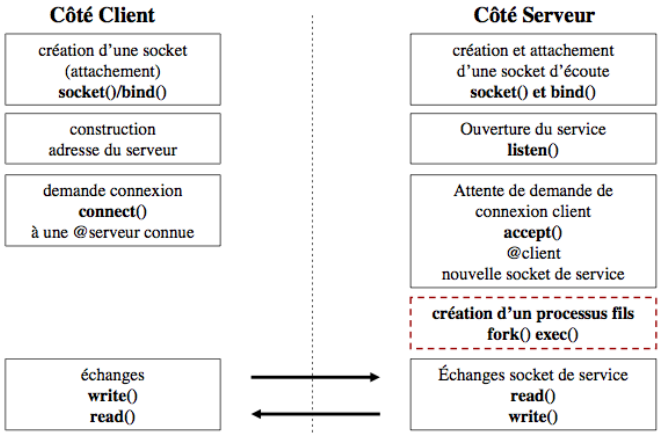
Les applications utilisent un concept appelé socket (initialement développé dans UNIX), permettant d'identifier les différentes communication sur la même machine. Le socket est constitué de la paire :

< adresse IP locale, numéro de port local >

Pour identifier de manière unique l'échange de données avec le processus applicatif distant, le protocole de transport utilise un ensemble de cinq paramètres formé par le nom du protocole utilisé, le socket local et le socket distant :

< protocole, adresse IP locale, numéro de port local ; adresse IP distante, numéro de port distant >

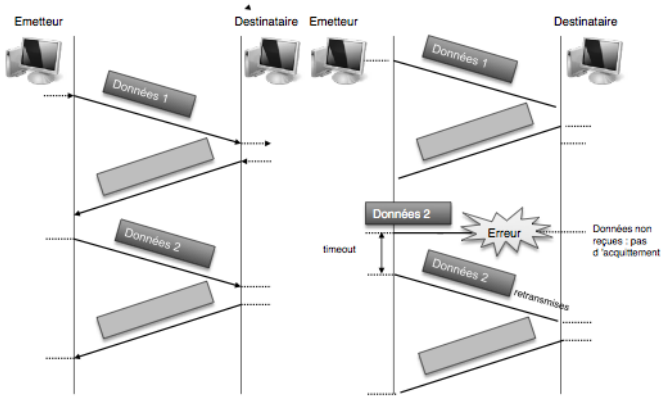
Les serveur peut accepter plusieurs connexions simultanément en créant pour chacune son propore soket avec un processus différent pour sa gestion.



Dans le modèle OSI, les sockets sont gérés par la couche session.

7.5.1.3 Fiabilité

L'utilisation d'un mécanisme appelé PAR (Positive Acknowlegment with Retransmission, Accusé de réception positif avec la retransmission) permet à TCP de garantir des transmissions fiables.



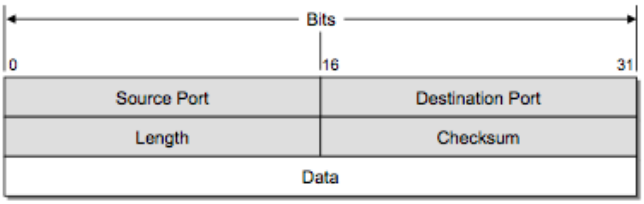
Chaque segment envoyé contient un checksum que le destinataire utilise pour vérifier que les données n'ont pas été endommagées pendant leur transmission. Si le segment est correctement reçu, le récepteur renvoie un accusé de réception positif à l'émetteur. Dans la négative, le récepteur élimine ce segment de données. Après un délai d'attente déterminé, le module TCP d'envoi retransmet les segments pour lesquels aucun accusé de réception positif n'a été reçu.

7.5.2 UDP

Le protocole UDP permet aux applications d'accéder directement à un service de transmission de datagrammes, tel que le service de transmission qu'offre IP. UDP est caractérisé par :

- UDP possède un mécanisme permettant d'identifier les processus d'application à l'aide de numéros de port UDP.
- UDP est orienté datagrammes (sans connexion), ce qui évite les problèmes liés à l'ouverture, au maintien et à la fermeture des connexions.
- UDP est efficace pour les applications en diffusion/multidiffusion. Les applications satisfaisant à un modèle du type "interrogation-réponse" peuvent également utiliser UDP. La réponse peut être utilisée comme étant un accusé de réception positif à l'interrogation. Si une réponse n'est pas reçue dans un certain intervalle de temps, l'application envoie simplement une autre interrogation.
- UDP ne séquence pas les données. La remise conforme des données n'est pas garantie.
- UDP peut éventuellement vérifier l'intégrité des données (et des données seulement) avec un total de contrôle.
- UDP est plus rapide, plus simple et plus efficace que TCP mais il est moins robuste.

Le datagramme UDP est organisé comme suit :



- Port source (16 bits) : Il s'agit du numéro de port correspondant à l'application émettrice du paquet. Ce champ représente une adresse de réponse pour le destinataire.
- Port destination (16 bits) : Contient le port correspondant à l'application de la

machine à laquelle on s'adresse.

– Longueur (16 bits) : Précise la longueur totale du datagramme UDP, exprimée en octets.
– Total de contrôle ou checksum (16 bits). Bloc de contrôle d'erreur destiné à contrôler l'intégrité de l'entête du datagramme UDP.

7.6 Les sockets

Au début des années 80, l'ARPA (Advanced Research Projects Agency du département de la défense américaine) a assigné à l'Université de Berkeley en Californie la responsabilité de construire un système d'exploitation qui pourrait être utilisé comme plate-forme standard pour l'ARPANet, le prédécesseur de l'actuel Internet.

Berkeley, déjà très connu pour son travail sur Unix, a ajouté une nouvelle interface au système d'exploitation pour implémenter les communications réseaux. Cette interface est généralement connue sous le nom de Berkeley-Socket. Sockets Interface (BSD : Berkeley Software Distribution) est à l'origine de presque tout ce qui existe comme interface pour TCP/IP, et à l'origine même des Windows Sockets (WinSock).

7.6.1 Définition d'une socket

Une Socket désigne l'extrémité d'un canal de communication bidirectionnel, elle ressemble beaucoup à un téléphone : En connectant deux sockets ensemble, on peut faire passer des données entre processus, même entre processus s'exécutent sur des machines différentes, exactement de la même façon qu'on parle à travers le téléphone une fois qu'on s'est connecté chez quelqu'un d'autre en l'appelant. Une socket est donc une interface entre les applications des utilisateurs et la couche transport.

Modèle des sockets	**Modèle OSI**
Application utilisant les sockets	Application Présentation Session
UDP/TCP	Transport
IP/ARP	Réseau
Ethernet, X25, ...	Liaison Physique

Les sockets peuvent être utilisée en deux modes de communication :

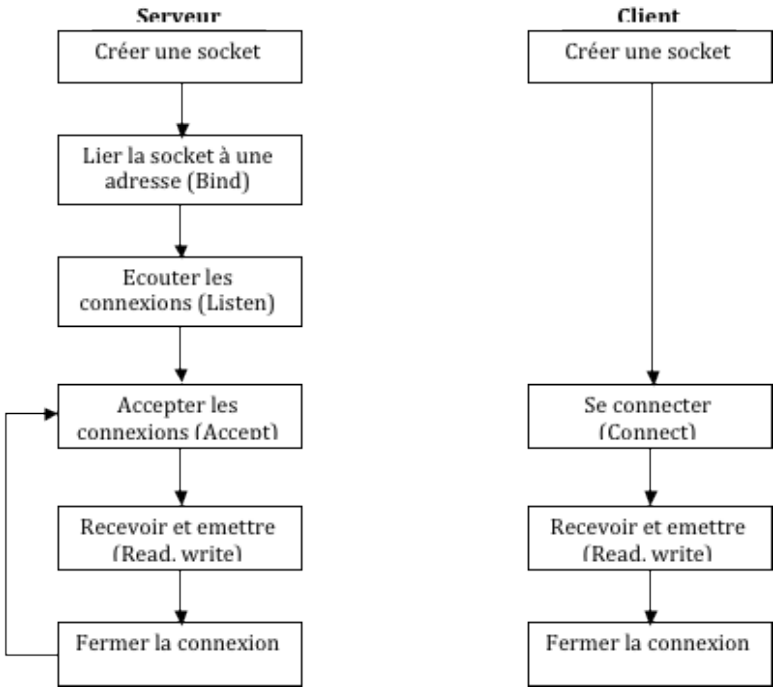
- Le mode connecté (comparable à une communication téléphonique), utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données.
- Le mode non connecté (analogue à une communication par courrier), utilisant le protocole UDP. Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

Pour faciliter leur utilisation les sockets sont conçues en conservant la sémantique des primitives d'E/S système exactement comme les fichier (création, ouverture, lecture, écriture, fermeture).

Pour communiquer des données entre deux sockets, dans un mode connecté, on doit faire une distinction entre celle utilisée par le programme demandeur de connexion (Client) et celle utilisée par le programme qui accepte les connexions (Serveur).

- Un serveur est un programme qui attend les connexions à travers une Socket et qui gère ensuite toutes les connexions qui arrivent.
- Un client est un programme qui se connecte à un serveur à travers une socket.

Le schéma suivant montre les différentes étapes exécutées par un client et un serveur pour communiquer :



7.6.2 Utilisation des sockets

Les sockets sont généralement implémentés en langage C, et utilisent des fonctions et des structures disponibles dans la librairie <sys/socket.h>.

Pour recevoir des appels téléphoniques, il faut d'abord installer votre téléphone. Pour ce faire, il faut créer une socket pour écouter les connections, un procédé qui se passe en plusieurs étapes :

7.6.2.1 Création d'une socket → Socket()

Ce qui ressemble à se faire installer une ligne de téléphone par la compagnie des télécoms. La fonction socket() necéssite de Spécifier le type de la socket. Les deux types les plus connus sont SOCK_STREAM et SOCK_DGRAM. SOCK_STREAM indique que les données seront transportées par le protocole TCP, alors que SOCK_DGRAM indique que les données seront transportées par le protocole UDP. Nous ne nous intéresserons ici qu'aux sockets SOCK_STREAM, qui sont les plus courantes.

7.6.2.2 Lier la Socket à une adresse → Bind()

Après avoir créé une socket, il faut lui donner une adresse à écouter, de la même façon qu'on prend un numéro de téléphone pour pouvoir recevoir des appels, mais avant ça, on doit spécifier la famille d'adresse à utiliser. Dans le cas de TCP/IP, la famille d'adresse est "AF_INET" où chaque socket a une adresse unique composée de deux éléments : une adresse IP et un numéro de port.

- La première partie est l'adresse IP (comme 192.9.200.10), avec lequel on veut communiquer. Malheureusement, les nombres sont difficiles à retenir, surtout quand on doit travailler avec beaucoup de nombres différents. L'interface la plus utilisée pour retrouver une adresse est la fonction gethostbyname(), qui prend le nom d'un ordinateur et vous renvoie son adresse IP. De même, il est possible de retrouver le nom d'un ordinateur quand on a son adresse IP en utilisant la fonction gethostbyaddr().
- La seconde partie est le numéro de port, qui autorise plusieurs conversations simultanées sur chaque ordinateur. Une application peut soit prendre un numéro de port réservé pour son type d'application, soit en demander un au hasard du système lorsqu'il lie une adresse à sa socket ; une application peut en utiliser plusieurs (un serveur par exemple exploite une socket par client connecté). Les numéros de port sont uniques sur un système donné. Certains numéros sont réservés pour des appli-

cations (ou protocoles) précises, par exemple le port 80 est réservé aux serveurs web (HTTP). Ces ports réservés ont un numéro inférieur à 1024.

Application	Port	Application	Port
echo	7	telnet	23
www	80	pop3	110
ftp	21	smtp	25
kerberos	88	nntp	119
time	37	imap2	143
nameserver	42	irc	194
domain	53	imap3	220

Une connexion est identifiée, donc, de façon unique par la donnée de deux couples, une adresse IP et un numéro de port, un pour le client et un autre pour le serveur. Il est important de noter qu'un dialogue client/serveur n'a pas forcément lieu via un réseau. Il est en effet possible de faire communiquer un client et un serveur sur une même machine, via ce qu'on appelle l'interface de loopback, représentée par convention par l'adresse IP 127.0.0.1.

Pour lier une socket à une adresse on utilise la fonction bind() (to bind veut dire "lier"). Une adresse de socket Internet est spécifié en utilisant la structure sockaddr, qui contient les champs qui spécifient la famille d'adresse, l'adresse et sa longueur.

7.6.2.3 Ecouter les appels → Listen()

Les sockets de type SOCK_STREAM ont la capacité de mettre les requêtes de connexion en files d'attentes, ce qui ressemble au téléphone qui sonne en attendant que l'on réponde. Si c'est occupé, la connexion va attendre qu'on libère la ligne. La fonction listen() est utilisée pour donner le nombre maximum de requêtes en attente avant de refuser les connexions.

7.6.2.4 Accepter les appels → Accept()

Après avoir créé une socket pour recevoir des appels, il faut accepter les appels vers cette socket. La fonction accept() est utilisée pour se faire. Appeler la fonction accept() est équivalent à prendre le combiné lorsque le téléphone sonne. Accept() renvoie une nouvelle socket qui est connecté à celui qui appelle.

7.6.2.5 Appeler une socket → Connect()

Après avoir créé une socket, et après lui avoir donné une adresse, il faut utiliser la fonction connect() pour essayer de se connecter à une socket qui attend les appels ce qui est équivalent à composer numéro d'un téléphone.

7.6.2.6 Emettre et recevoir des données → Send() & Recv()

Maintenant qu'on a une connexion entre deux sockets, on peut envoyer des données entre elles. On utilise les fonctions send() and recv().

7.6.2.7 Fermer une connexion → closesocket()

De la même façon qu'on raccroche après avoir eu quelqu'un au téléphone, il faut fermer la connexion entre les deux sockets. La fonction closesocket() est utilisée pour fermer chaque extrémité de la connexion. Si une des extrémité est fermée et qu'on essaye d'utiliser la fonction send() à l'autre, la fonction send() renverra une erreur. Un recv() qui attend quand la connection à l'autre extrémité est fermé ne retournera aucun octet.