Programmation Réseaux
TP 1 – 2 séances de 2 heures

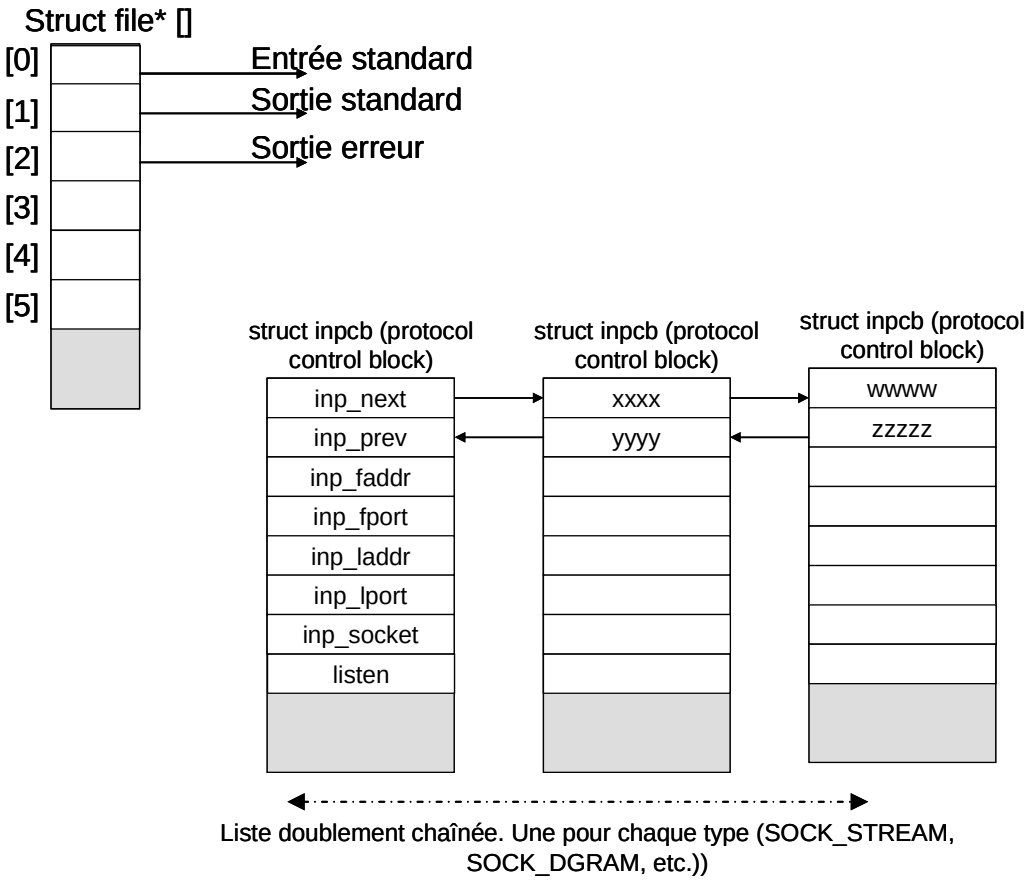
Exercice 1 : Serveur TCP.

L'idée de ce premier exercice est de voir en détail les structures mises en œuvre par le noyau et les valeurs de celles-ci lors d'une communication TCP - IP. Nous considérons ci-dessous une version allégée des structures gérées par le noyau. L'exercice consiste à compléter les structures avec les valeurs appropriées vis à vis du code qui est donné ici.

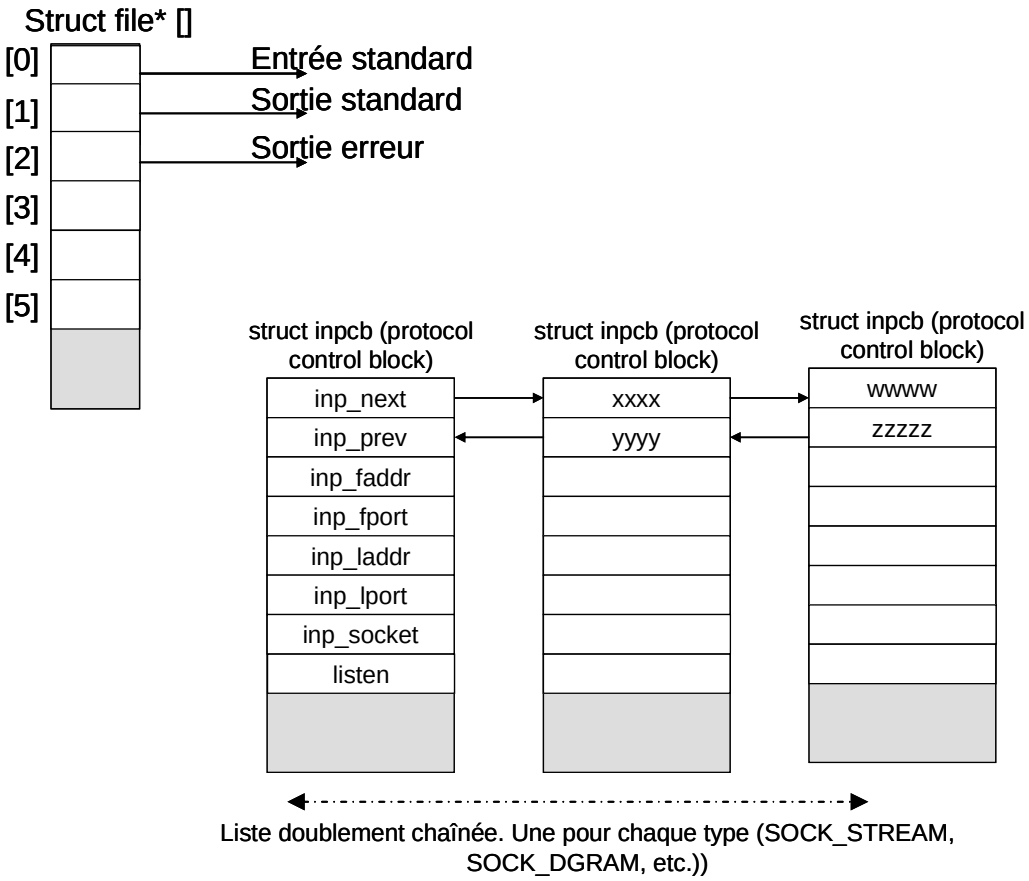
```
char buffer[100], reponse[30]=« Bien recu »;

memset(&hints,0,sizeof(hints));
hints.ai_family = AF_INET6;          //IPv4 ou IPv6
hints.ai_socktype = SOCK_STREAM;     //TCP
hints.ai_flags = AI_PASSIVE;         //Permet d'associer la socket a toutes les adresses locales

if(getaddrinfo(NULL,"80",&hints,&res)!=0) erreur("Erreur getaddrinfo()",1);   0


if((sockfd=socket(..., ..., ...))<0) perror("Erreur socket():" );                                  1

if(bind(..., (struct sockaddr *) ..., ...)<0) perror("Erreur bind()");                                  2

if(listen(..., ...)<0) erreur("Erreur listen()",2);                                3

if ((confd=accept(..., ..., ...))>0)                                4
{
            if(recv(..., buffer, ... , ...)<0) perror (« Erreur recv »);                                5
            if(send(..., reponse, ..., ...)!=strlen(reponse)) perror (« Erreur send »);                                6
} else perror (« Erreur accept »);
```

Au point 0, les structures gérées par le noyau n'ont pas été mis à jour ou créées, le tableau de descripteurs de fichiers associé au processus ne contient donc pas de lien/pointeur vers des connexions (structure inpcb).

1. Une structure inpcb est initiée à l'appel avec l'appel à `socket()` (point 1 dans le code) et le lien vers le tableau de descripteur est mis à jour. Mettez à jour les champs des structures en conséquences, juste après l'appel à socket (au niveau du point 1 dans le code donc). Quelle est alors la valeur de l'entier `sockfd` ?



2. Faites de même pour les points 2, 3 et 4. On suppose qu'un client avec l'adresse IP 129.1.1.2 avec le numéro de port local (client) 1078 se connecte sur ce serveur. L'adresse IP destination (serveur) utilisée par le client est 167.1.3.4. Quelle est alors la valeur de `confd` ? Note : le drapeau (booléen) listen se trouve en fait dans une autre structure (la structure `socket` qui n'a pas été vu en cours). Voir le cours polycopié pour plus de détails (le transparent décrivant l'appel à `listen()` et une des annexes).



3. Que se passe-t-il à l'arrivé d'un paquet (avant et après l'appel à `recv()`)?
4. Que se passe-t-il à l'émission d'un paquet provoqué par `send()` ?

Exercice 2: Gestion des erreurs

Cet exercice a pour but de vous familiariser avec l'utilisation de la fonction `perror()`. Cette fonction va lire la valeur de la variable `errno`. C'est une variable gérée par le système d'exploitation (local à chaque processus) qui indique la valeur de l'erreur du dernier appel système. Elle vaut *0* en cas de succès de l'appel et une valeur positive sinon. Lorsque c'est positif, la valeur indique le type d'erreur. Vous devez coder un petit programme en C qui appellera un appel système (`socket()`). Il vous est demandé d'appeler l'appel `socket()` avec un argument erroné et afficher la valeur de `errno`. La lecture de la section *Erreur du man* vous permettra d'interpréter l'erreur correspondant à la valeur de `errno`. Vous pourrez ensuite utiliser la fonction `perror()` qui fait simplement l'affichage de l'erreur correspondant à la valeur de `errno` (à utiliser juste après l'appel système).

Exercice 3: Gestion des adresses

Cet exercice vise à vous familiariser avec la fonction vous permettant de manipuler les adresses et les numéros de ports.

Utiliser la fonction `getaddrinfo()` pour mettre à jour une structure `addrinfo`. Cette structure devra vous permettre d'initialiser un serveur écoutant sur toutes ces adresses et sur le port

2000. Pensez à utiliser la fonction `gai_strerror()` pour afficher le type d'erreur renvoyé par `getaddrinfo` (sil il y en a une bien sûr).

Exercice 4 : Un serveur simple

Reprenez le code précédent (exercice 3). Vous devez compléter ce code afin de programmer un serveur qui écoutera sur le port 2000 et affichera au format ASCII toutes les données reçues des clients.

Pour tester votre serveur vous pouvez utiliser une des deux solutions suivantes. La première consiste à utiliser votre navigateur avec la syntaxe suivante :

`http://127.0.0.1:2000/toto.html`

et

`http://[::1]:2000/toto.html`

L'autre solution consiste à utiliser le client telnet :

`telnet 127.0.0.1 2000`
Vous taper le texte ici …
`Ctrl-D` pour sortir

et

`telnet6 ::1 2000`
Vous taper le texte ici …
`Ctrl-D` pour sortir

Vous devrez tester les deux versions du protocole IP.

Note : Utiliser les appels `socket()`, `bind()` et `recv()`.
Note 2 : Si vous avez des erreurs du type « `Adress already in use` » lorsque vous relancez votre serveur, vous pouvez rajouter l'instruction suivante avant l'appel à `bind()` :

```
int tr=1 ;
if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&tr,sizeof(int)) == -1)
  perror("Erreur setsockopt() SO_REUSEADDR");
```