Cours n° 2

Sockets en C

Safa YAHI
safa.yahi@univ-amu.fr

Serveur conccurent avec fork()

- Après le fork(), la socket d'écoute et la socket de connexion sont dupliquées dans le fils.
- La socket de connexion sera utilisée uniquement par le fils pour dialoguer avec le client => elle doit être fermée par le père.
- La socket d'écoute est utilisée uniquement par le père pour attendre d'autres clients => elle doit être fermée par le fils.
- Il faut gérer les zombies : supprimer les pid des processus fils et leur état de sortie de la table des processus quand ils se terminent en utilisant une fonction de type wait().

Serveur conccurent avec fork()

```
int main() {
int sock_serveur = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in sockaddr_serveur;
sockaddr_serveur.sin_family = AF_INET;
sockaddr_serveur.sin_port = htons(NUM_PORT);
sockaddr_serveur.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sock_serveur, (struct sockaddr *) &sockaddr_serveur, sizeof(sockaddr_in)) == -1) exitErreur("bind");
if (listen(sock_serveur, BACKLOG) == -1) exitErreur("listen");
int sock_client; char * msg; time_t date;
// Mise en place d'un handler pour supprimer les pid des fils qui se terminent
struct sigaction sa;
sa.sa_handler = supprimer_zombies;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART ;
sigaction(SIGCHLD, &sa, NULL);
```

Serveur conccurent avec fork()

```
for (int i = 1; i <= NB_CLIENTS; i++) {
        sock_client = accept(sock_serveur, NULL, NULL);
        if (sock_client == -1) exitErreur("accept");
        pid_t f = fork() ;
        if (f == 0) { // C'est le processus fils
                close(sock_serveur);
                date = time(NULL); msg = ctime(&date);
                if (write(sock_client, msg, strlen(msg)) == -1) exitErreur("write");
                close(sock_client);
                exit(EXIT_SUCCESS) ; }
        // ici c'est le père qui ferme la socket du fils et retourne à accept
        close(sock_client);  }
close(sock_serveur);
return 0;}
```

Serveur conccurent avec fork()

```
void exitErreur(const char * msg) {

perror(msg);

exit( EXIT_FAILURE);

}


void supprimer_zombies(int s) {

while (  waitpid(-1, NULL, WNOHANG ) > 0) ;

// WNOHANG : le père ne se bloque pas s'il n'ya pas plus de fils terminé alors qu'il y'a des fils en cours d'execution

// si un fils génère un signal alors que le signal (du m^me type)  d'un autre fils n'a pas été traité, il sera perdu d'où
la boucle pour supprimer des pid dont les signaux ont été supprimés.

}
```

recv()

```
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- Les trois premiers paramètres et la valeur de retour sont les mêmes que pour read().
- Le paramètre flags modifie le comportement par défaut. Il peut prendre une valeur ou un 'or' entre plusieurs valeurs. Exemples de valeurs :
  - MSG_DONTWAIT pour faire un recv() non bloquant
  - MSG_PEEK pour prendre juste une copie du message reçu sans le supprimer du buffer.
- recv(sockfd, buf, len, 0) <=> read(sockfd, buf, len)

send()

```
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

- Les trois premiers paramètres et la valeur de retour sont les mêmes que pour write().
- Quelques valeurs pour flags :
  - MSG_OOB pour envoyer des données urgentes
  - MSG_DONTWAIT pour faire un send non bloquant.
- send(sockfd, buf, len, 0) <=> write(sockfd, buf, len)

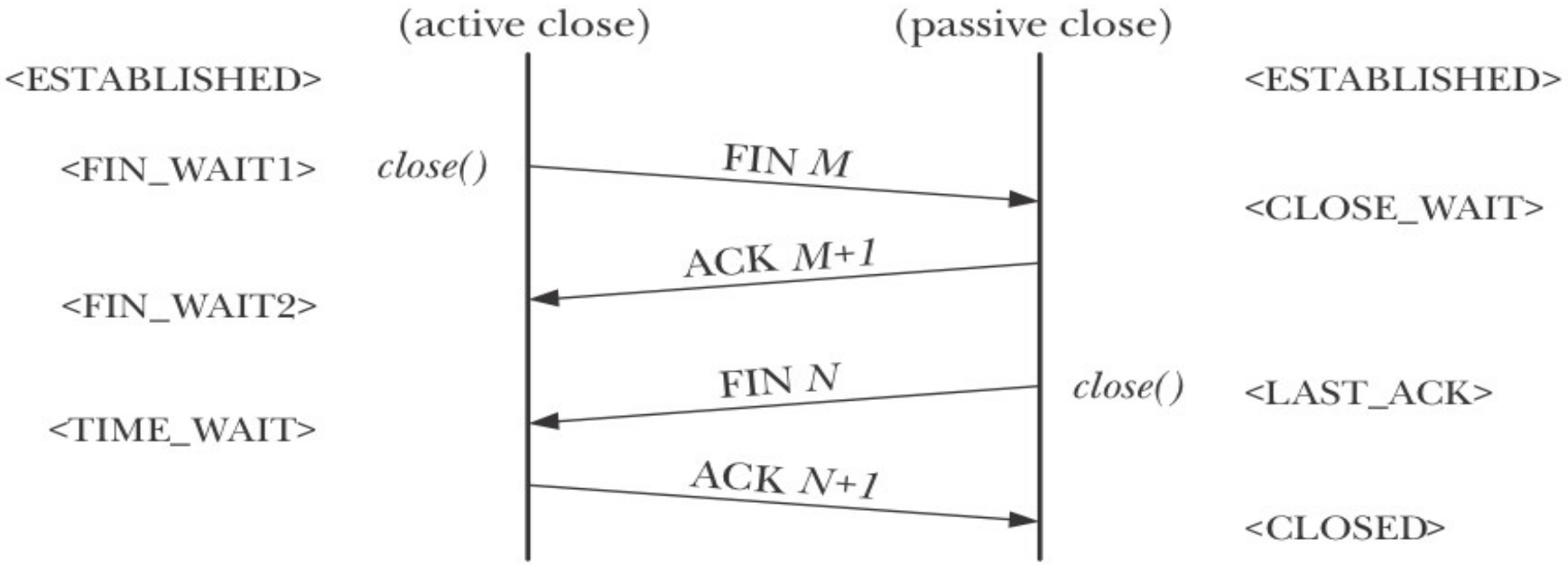
Relancer un serveur

Relancer un serveur ? Eh oui ce n'est pas aussi simple qu'on le croit. Voyons la démo suivante.

- Lancer un serveur daytime, lui envoyer une requête et l'arrêter.
- Le relancer juste après. On a un message d'erreur lié à bind() : *address already in use*.
- On vérifie avec **netstat -lt** : tout semble correct, il n'y pas une autre instance lancée de ce serveur...
- Et si on affichait toutes les sockets TCP avec **netstat -at** (à faire dans les 60s après la fin de la requête intiale) ?

Relancer un serveur

- On constate la socket qui a été créée par le serveur (à la connexion du client) dans l'était TIME_WAIT.
- Cet état permet, par exemple, de restransmettre le dernier ACK s'il est perdu.



Relancer un serveur

- La socket TIME_WAIT, comme toutes les sockets créées par le serveur pour servir un client, utilisent le même port local et la même adresse IP locale que sa socket d'écoute. Comment les distinguer ?
- Une socket TCP connectée est identifiée par le tuple :

  (port local, @ IP locale, port distant, @ IP distante)
- Le protocole TCP exige l'unicité de ce tuple.
- La plupart des implémentations rajoutent une contrainte en plus : un port ne peut être utilisé avec bind() s'il existe une connexion TCP avec ce même port d'où l'erreur lors du redémarrage du serveur.

L'option SO_REUSEADDR

La solution (qui reste conforme aux spécifications de TCP) est d'utiliser l'option de socket SO_REUSEADDR : il faut lui donner une valeur non nulle avant l'appel bind() via la fonction setsockopt().

setsockopt()

```
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
```

- sockfd : le descripteur de fichier de la socket dont on veut modifier une option.
- level : niveau de l'option, par exemple, SOL_SOCKET pour le niveau API Sockets.
- optname spécifie l'option à modifier
- optval : pointeur vers un entier ou une structure qui contient la valeur de l'option.
- optlen : la taille du buffer pointé par optval.

setsockopt() pour modifier l'option SO_REUSEADDR

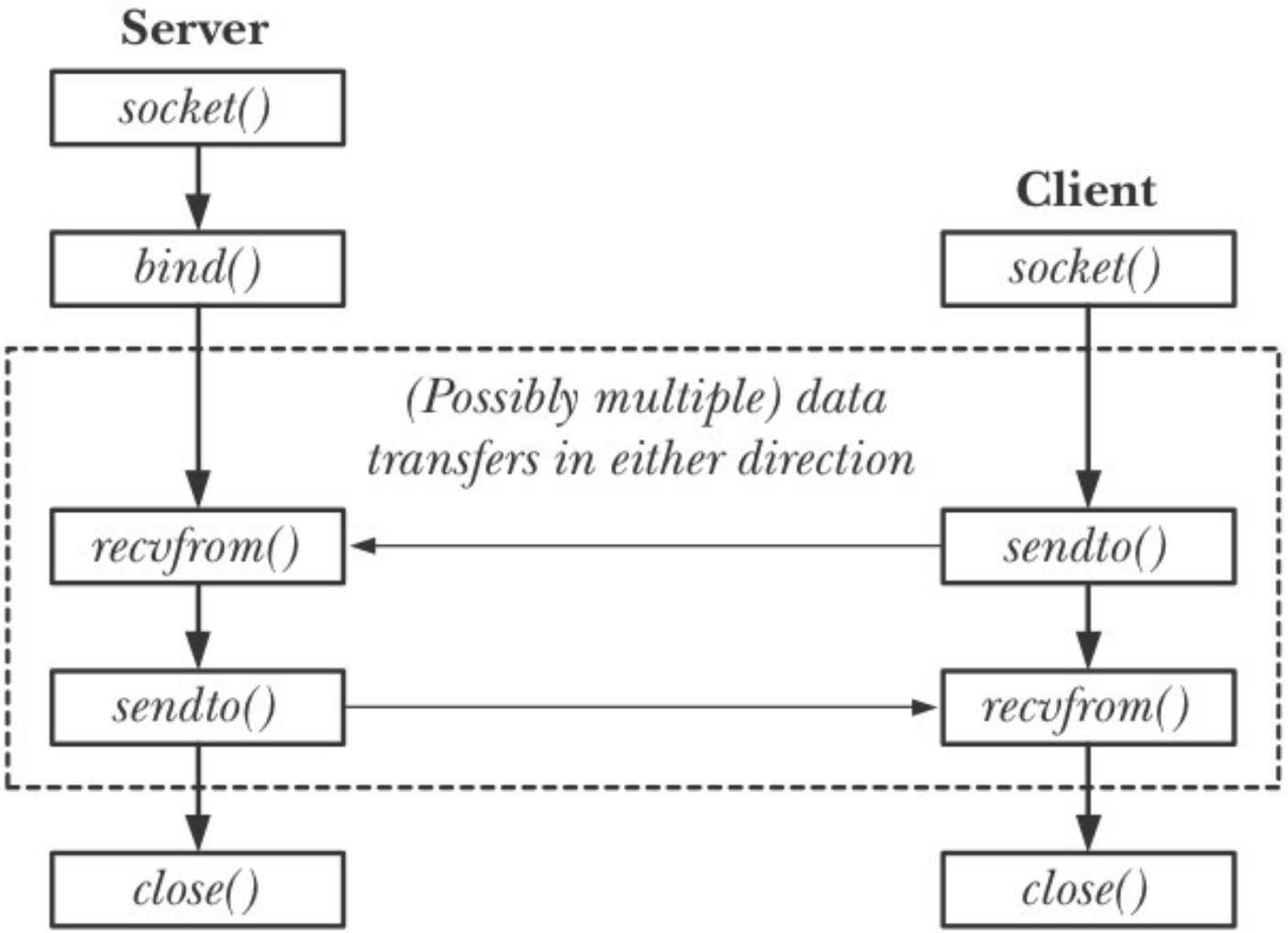
```
int optval = 1;

setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
&optval, sizeof(optval));
```

Un autre exemple d'utilisaiton de setsockopt()

```
int optval = 1;
 setsockopt(sock_client, IPPROTO_TCP, TCP_NODELAY,
&optval, sizeof(optval));
```

Ceci permet de désactiver l'algorithme de Nagle.

Client / Serveur UDP



sendto()

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);
```

- Les quatre premiers paramètres et la valeur de retour sont ceux de send().
- dest_addr est l'adresse de la socket du destinataire.
- addrlen est la taille de l'adresse du destinatair

recvfrom()

```
#include <sys/socket.h>

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen);
```

- Les quatre premiers paramètres et la valeur de retour sont les mêmes que pour recv().
- **src_addr** l'adresse de l'émetteur
- **addrlen** la taille de l'adresse de l'émetteur. Elle doit être initialisée par l'appelant (sizeof(sockaddr_in) pour IPv4). Elle est modifiée en retour pour indiquer la taille réelle de l'adresse enregistrée.

Exercice

Réécrivez l'application (serveur et client) daytime en mode UDP.

recvfrom() et sendto () en TCP

- Les fonctions recvfrom() et sendto() peuvent être aussi utilisées en TCP : comme on connait l'interlocuteur dans ce cas, on ne spécifie pas l'adresse du destinataire dans le sendto() et on n'essaye pas d'avoir celle de l'émetteur dans le recvfrom().
- send(sockfd, buf, len, flags) <=>

  sendto(sockfd, buf, len, flags, NULL, 0);
- recv(sockfd, buf, len, flags) <=>

  recvfrom(sockfd, buf, len, flags, NULL, NULL);