

Computer Network : Rapport du projet

Charles Jacquet & Stéphane Kimmel

1 Introduction

Pour ce projet, il nous a été demandé de concevoir un programme en langage C permettant d'envoyer un fichier vers un hôte distant connecté en réseau local. La difficulté du programme résidait dans le fait qu'il devait permettre de détecter les erreurs lors du transfert ainsi que les informations non-reçues et renvoyer les parties défectueuses afin que le fichier reçu soit exactement le même que celui envoyé à la base. Tout ceci, alors qu'il nous est imposé de travailler sur un protocole de transmission qui ne le fait pas nativement (UDP).

2 Choix d'implémentation

Le protocole utilisé a dû être l'UDP, l'algorithme de transmission était aussi fixé au selective repeat et la structure d'un paquet est également donnée.

Par contre, le programme étant clairement divisé en 2 parties (envoyeur et receveur), nous avons décidé de nous le diviser en 2 parties distinctes tel quel. Ce qui fait que nous avons deux implémentations assez différentes.

2.1 Sender

Pour ce qui est du sender, premièrement, nous utilisons `getaddrinfo()` pour obtenir des structures `addrinfo`. Ensuite nous les testons grâce à un `connect()` pour trouver une structure valable. Ensuite, pour l'envoi des paquets ainsi que leur ack, nous avons mis toutes les opérations dans une boucle se terminant lors de la réception du dernier ack. Pour l'envoi, on utilise la fonction `send_window()` qui va regarder si il y a des éléments vides dans le buffer et si c'est le cas, créer et envoyer des paquets vers le receiver ainsi que remplir le buffer.

Ensuite grâce à la fonction `select` munie d'un timeout, nous pouvons gérer la réception des ack.

Lorsque le timeout expire, nous renvoyons le premier élément dans le buffer car il permet d'automatiquement libérer une place dans le buffer lorsque son ack est reçu.

Pour la réception des ack, la stratégie utilisée est de trouver la place de l'élément ack dans le buffer ensuite on peut mettre le ack des paquets se trouvant avant à

1(pour dire que les paquets ont été ack). Ensuite on décale la fenêtre tant que le premier paquet est ack. Enfin, si on reçoit 2 fois d'affilé un ack avec le même numéro de séquence, c'est qu'il n'a pas été reçu donc on réenvoie l'élément en question pour éviter d'attendre la fin du timer.

De plus, nous avons implémenté le delay, c'est à dire qu'avant d'envoyer un fichier on attends le delay de x ms. Pour ce faire, on a utilisé plusieurs structures (se trouvant dans struct.h).

De plus, au début nous avons des soucis car nous ne laissons pas la place pour le '\0' à la fin des tableaux de char. Nous avons donc décidé d'utiliser uniquement les 511 premier bytes de la payload et ne pas s'occuper du dernier.

- **msgUDP** Notre structure message UDP est en réalité la structure envoyée, soit le paquet comme spécifié dans les consignes.
- **paquet** Cette structure contient une structure msgUDP ainsi qu'un int pour savoir si le paquet est ack ou pas.
- **window** Cette structure permet de stocker des pointeurs vers chaque paquet dans le buffer. Et contient aussi deux int pour connaître le nombre d'éléments maximum ainsi que le nombre d'éléments vide dans le buffer.

Pour ce qui est du delay. Le delay est le temps à attendre avant de transmettre chaque paquets. Nous avons donc créé une fonction delay permettant d'attendre le temps nécessaire grâce à des structures timeval ainsi que la fonction gettimeofday() le delay nécessaire avant l'envoi du paquet.

2.2 Receiver

Le receiver étant un peu plus simple à réaliser à première vue. Pour ce qui est de l'implémentation de la structure d'un paquet, nous avons fait le choix de la définir dans un fichier externe pour ne pas devoir la redéfinir dans le sender et dans le receiver, de plus, lorsque la structure d'un paquet doit changer au cours du développement du programme. C'est le seul paquet externe utilisé par le receiver car même pour préparer le paquet d'un accusé, il n'est pas assez intéressant de créer un nouveau fichier externe car il est relativement simple à créer.

Tout d'abord, nous utilisons, tout comme le sender, la structure addrinfo pour la gestion des adresses de l'hôte et donc la méthode getaddrinfo. Cependant, pour une raison inconnue, lorsque l'argument hostname de cette méthode est différent de ":", il nous est impossible de faire fonctionner notre programme en local. Sinon celui-ci échoue chaque fois lors de l'appel à l'appel bind. Il faut donc chaque fois utiliser ":" comme argument hostname au programme (ou "::1" si l'on fait tourner notre programme en local).

Ensuite, nous rentrons dans une boucle qui ne s'arrête que lorsque la longueur d'un paquet reçu est plus petite que 512, auquel cas, cela marque la fin du

programme. Dans la boucle, nous recevons les paquets après un appel à la fonction select qui permet de gérer la réception des paquets lorsque le programme en traite un autre.

Un petit inconvénient de notre implémentation est qu'elle consomme une assez grande mémoire. En effet, il était obligatoire d'avoir un buffer tampon entre la réception d'un paquet et l'écriture dans le fichier car autrement, les paquets auraient pu être écrits dans le mauvais ordre dans le fichier (dû au protocole UDP qui ne certifie pas l'ordre d'arrivée des différents paquets). Nous avons décidé, par facilité, de créer un tableau de 256 lignes sur 512 colonnes comme buffer (131 072 bytes). Ceci est assez imposant comme taille et il aurait été possible de s'arranger en faisant un buffer tampon plus petit mais nous n'y avons pas porté trop d'importance tout d'abord car cela permettait d'y voir plus clair pour le débogage du programme et permettait aussi de mieux s'y retrouver dans la gestion de la fenêtre qui coulisse sur ce tableau. Ça reste aussi ridiculement petit pour les ordinateurs de nos jours mais c'est cela reste sans contester une des dernières parties restante à optimiser dans notre programme.

3 Conclusion

Pour conclure, les difficultés de ce projet ne sont arrivées que lors de la mise en oeuvre. Que ce soit pour l'envoi ou la réception des paquets, nous avons eu pas mal de bugs. Certain étant à première vue incompréhensible comme l'arrêt du programme entre 2 printf. Mais nous sommes parvenu à trouver des solutions et à avoir un programme (sender + receiver) fonctionnel.

4 Test interopérabilité

4.1 Effectué avec le groupe de Romain Henneton et Sundeeep Dhillon

Pour le test d'interopérabilité, nous avons été en salle intel où nous avons fait les différents tests. Nous avons juste eu quelques difficultés au début mais en comparant les codes on a pu vite voir les différences et prendre la solution qui paraissait la meilleure. Il nous a fallu une dizaine de minutes pour que les différents programmes soient fonctionnels entre eux.

4.2 Effectué avec le groupe de Sandrine Romainville et Jolan Werner

Ces tests-ci n'ont pas été très convainquant, et ce dû à pas mal de divergences dans nos 2 programmes. Mais le plus grand problème a été la structure de paquet qui est assez différente de la nôtre et du coup, incompatible. Les deux programmes arrivent à communiquer mais ne se comprennent pas sur les numéros de séquences, les CRC, ...