

1 Introduction

Pour ce projet, il nous a été demandé de concevoir un programme en langage C permettant d'envoyer un fichier vers hôte distant connecté en réseau locale. 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). Il a donc fallu surmonter manuellement obstacles lors de la transmission.

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 packet étant donnée, les grands points de l'implémentation étaient fixés dès le début.

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 reception 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 un attends le delay de x ms. Pour ce faire, on a utilisé plusieurs structures (se trouvant dans `struct.h`).

- **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.

2.2 Receiver

Le receiver étant un peu plus simple à réaliser à première vue, nous avons juste préféré que le receiver utilise la structure packet qui est stockée dans un fichier externe ainsi, lorsque la structure d'un packet doit changer au cours du développement du programme, comme il a été le cas, nous n'avons jamais eu de problèmes à cause de ça. C'est le seul packet externe utilisé par le receiver car même pour préparer le packet d'un accusé, il n'est pas intéressant de créer un nouveau fichier externe.

Nous aurions pû, après réception du premier packet, créer une connection avec l'hôte local. Cela aurait permis de passer en TCP mais nous avons décidé dès le début de travailler constamment en UDP, ce qui fait que nous n'avons aucune connection avec l'autre hôte. Cela permet entre outre, de n'avoir qu'une boucle dans notre programme et que le programme soit plus malléable, nous ne dépendons effectivement jamais du protocole de plus haut niveau.

Le troisième choix d'implémentation a été plus délicat à prendre. En effet, il est obligatoire d'avoir un buffer tampon entre la réception d'un packet et l'écriture dans le fichier car autrement, les packets auraient été écrits dans le mauvais ordre dans le fichier. Nous avons décidé, par facilité, de créer un tableau de 256 lignes sur 512 colonnes comme buffer. 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 car cela permettrait d'y voir plus clair pour le débogage du programme et cela reste ridiculement petit pour les ordinateurs de nos jours mais c'est cela reste sans contester une des dernières parties possibles 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 avec le groupe de Romain Henneton et Sundeep 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 fonctionnel entre eux.