Projet : Protocole connecté au dessus d'UDP

L'objectif de ce projet et de mettre en œuvre un protocole « à la » TCP en utilisant le protocole non-connecté UDP. On attend de ce protocole qu'il soit de type connecté, qu'il assure un contrôle de flux et de congestion, qu'il soit de type FIFO (c'est-à-dire que les messages sont envoyés et délivrés dans l'ordre), afin qu'il puisse finalement être considéré comme fiable.

Il vous est demandé d'implémenter le code correspondant à chacune des questions. De plus, vous enverrez une archive du code (pensez à faire un `make clean` avant de créer l'archive), ainsi qu'un rapport de cinq pages maximum dans lequel vous justifierez vos choix (attention il ne faut pas mettre de code dans le rapport). La date de remise du rapport et de l'archive est le Vendredi 7 Décembre.

À noter : Il est important de signaler que dans les parties suivantes, chacune des deux entités prenant part à la communication (i.e. le serveur et le client) peuvent à la fois jouer le rôle d'émetteur et de récepteur. De plus, pour simuler la perte de paquets, l'option `-p` pourra être passée en paramètre de l'application.

1 Protocole *envoyer et attendre*

Dans cette partie, il vous est demandé de mettre en place un mécanisme d'acquittement, c'est-à-dire qu'à chaque fois qu'une trame est envoyée, l'émetteur attend l'acquittement de cette dernière pendant un certain temps (« timeout »). Si au bout du temps d'attente, l'acquittement n'est toujours pas reçu, l'émetteur renvoie la trame. Une attention toute particulière devra être apportée à la numérotation des trames et des acquittements pour éviter les problèmes vus en TD (TD4 Nachos). De plus, tout comme dans le TD4, il vous est demandé de mettre en place un tampon de réception au niveau de la couche de transport de manière à pouvoir recevoir un certain nombre de paquets avant que l'application ne les consomme.

Remarque : On veut éviter d'appeler le gestionnaire de signaux quand on est bloqué dans un sémaphore ou un verrou ! Pour cela, dès que l'on exécute une primitive utilisant ces mécanismes, on intercepte le signal `SIGALARM` utilisé pour notre timer. Ce dispositif est implémenté dans les modules `synch`, `timeout` et `my_signal`. Vous pouvez vous y intéresser mais en aucun cas les modifier ! Tout bug provenant de la modification de ces fichiers sera laissé à votre charge !

2 Protocole avec fenêtre coulissante

Il vous est demandé de mettre en place un protocole à fenêtre coulissante au niveau de l'émetteur. C'est-à-dire que l'émetteur doit être capable d'envoyer un nombre n (n étant la taille du tampon d'émission) de trames sans recevoir d'accusé de réception. Si au bout des n messages, aucun accusé de réception n'est reçu, l'émetteur se bloque en émission en attendant les accusés de réception (attention, le mécanisme de « timeout » est toujours présent pour chacune des trames).

3 Gestion de la congestion

Dans le cas où l'application ne consommerait pas suffisamment rapidement les paquets, le tampon de réception peut se remplir complètement. De ce fait, la couche de transport ne peut plus recevoir de paquets (elle ne peut plus les copier dans le tampon). Pour éviter un tel scénario, il est nécessaire de mettre en place un mécanisme de gestion du trafic qui s'inspire de ce qui a été fait dans en TD (TD5 Nachos). L'idée est que chaque acquittement envoyé par le destinataire à la source doit contenir le nombre de positions disponibles dans le tampon de réception. Ainsi, si le tampon est complètement plein, l'émetteur s'arrête d'émettre jusqu'à nouvel ordre. Pour éviter de tomber dans une situation d'interblocage, dès que l'émetteur reçoit un acquittement dans lequel le récepteur dit qu'il n'a plus de place dans son tampon, il déclenche un temporisateur. Dès que la durée correspondante est écoulée, il envoie un message vide (`NOP`) au client, pour recevoir un accusé de réception qui contiendra l'état du tampon de réception au niveau du récepteur.

4 Bonus

4.1 Tampon de réordonnancement

UDP n'étant pas un protocole connecté, il ne garantit pas que les messages émis arrivent dans l'ordre d'émission. Ceci représente un gros désavantage par rapport aux applications qui vont utiliser notre couche de transport. Pour pallier ce problème, il vous est demandé de ne faire remonter les paquets à l'application que dans l'ordre de leur émission (un tri sur le numéro du paquet permet de garantir un ordre conforme à l'ordre d'émission). Par exemple, si le récepteur s'attend à recevoir le paquet n° 3 et qu'il reçoit le paquet n° 4, il acquitte la réception du paquet n° 4 mais fait en sorte de ne pas le faire remonter à l'application. Dès que le paquet n° 3 est reçu, la couche de transport le fait remonter à l'application. Attention, pour éviter des problème d'interblocage, il est nécessaire au niveau de l'émetteur, de ne pas autoriser l'envoi d'un paquet si le numéro de séquence de ce dernier vérifie la relation suivante :

$$\textit{numéro de séquence du paquet} > \textit{plus petit numéro de séquence non acquitté} + \textit{taille de la fenêtre} - 1$$

4.2 Support multi-clients

La dernière partie de ce projet concerne la possibilité de faire communiquer plusieurs machines avec un même serveur (sur des ports différents). Il vous est demandé de mettre en place les outils nécessaires à la gestion de plusieurs clients. Cette partie se basera principalement sur l'utilisation de l'appel système `select`.