TP à rendre 3

Vous devez écrire le programme `reseau` qui simule un réseau Ethernet de plusieurs stations connectées via un commutateur. La syntaxe est la suivante :

```
./reseau nb_sta
```

Ce programme crée *nb_sta* processus fils qui sont les stations du réseau et le processus père est le commutateur. Chaque station est connectée au commutateur via deux tubes anonymes comme illustré sur la figure 1 :

— le premier tube est partagé par toutes les stations pour envoyer des trames au commutateur ;
— le second tube est spécifique à chaque station pour recevoir des trames depuis le commutateur.

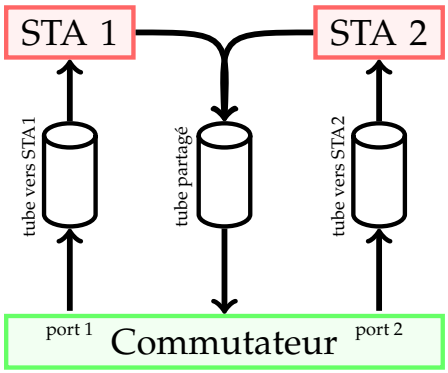

FIGURE 1 – Topologie du réseau avec 2 stations

Pour simplifier l'exercice, on limitera le nombre de stations à `MAXSTA` fixé à 10. Aussi, les adresses utilisées par les stations sont des entiers : `1` pour la première station, `2` pour la seconde, etc.

Les paramètres des trames que chaque station doit transmettre sont présents dans des fichiers `STA_`*x* où *x* précise l'adresse de la station source. Ces fichiers comportent des couples (`destination`,`payload`) où `destination` est un entier et `payload` est le message à transmettre sous la forme d'un bloc de `PAYLOAD_SIZE` (fixé ici à 4) octets. Par exemple, si on souhaite que la station 1 envoie une trame dont le *payload* est « toto » à la station 2, le fichier `STA_1` contient :

```
> hexdump -C STA_1
00000000  02 00 00 00 74 6f 74 6f           |....toto|
00000008
```

Le programme `trame` fourni permet de créer les fichiers `STA_`*x*. Sa syntaxe est :

```
> ./trame source destination payload
```

Chaque nouvel appel à `trame` pour une même source complète le fichier existant. Pour écraser le contenu d'un fichier, il faut d'abord le supprimer.

Une station doit lire son fichier `STA_`*x* et transmettre les trames correspondantes au commutateur via le tube partagé, puis dans un second temps attendre les trames reçues du commutateur. De son côté, après la réception d'une trame, le commutateur doit la *commuter*, c'est-à-dire déterminer le port (donc le tube) qui permet de joindre la destination de la trame et l'envoyer uniquement sur ce port. La table de commutation présente sur le commutateur fait la correspondance entre les destinations et les ports par lesquelles elles sont joignables. Pour simplifier l'exercice, on suppose cette table complète au démarrage du réseau. Il n'est donc pas nécessaire de l'alimenter en temps réel. Si une trame comporte une adresse de destination qui ne correspond à aucune station présente, le commutateur doit la diffuser à toutes les stations, sauf bien sûr la station source.

Chaque station qui reçoit une trame doit afficher sur la sortie standard son adresse, l'adresse de l'expéditeur de la trame, l'adresse de la destination de la trame et le contenu du *payload* sous la forme (un exemple concret est fourni en annexe) :

```
MOI - SRC - DST - PAYLOAD
```

Le programme doit prendre fin lorsque tous les processus fils sont terminés. Le code de retour doit être nul si tous les processus fils ont terminé via un appel à `exit(0)`, ou égal à 1 dans le cas contraire.

Vous écrirez le programme `reseau` en langage C en utilisant uniquement les primitives système. Les fonctions de bibliothèque sont autorisées uniquement pour l'affichage et la manipulation de la mémoire ou de chaînes de caractères. Pour des raisons d'efficacité, vous ne ferez pas d'appels redondants aux primitives systèmes. Votre programme doit vérifier les valeurs de retour de toutes les primitives système utilisées.

Vous apporterez un soin particulier à la mise en forme de façon à rendre un code lisible et commenté à bon escient. Référez-vous au document « Conseils de programmation pour réussir vos projets et TP » (mis à votre disposition sur Moodle) et, si besoin, utilisez l'utilitaire `clang-format` avec les options données dans ce document.

Votre programme doit compiler avec la commande suivante (disponible dans le `Makefile` mis à disposition sur Moodle) : `cc -Wall -Wextra -Werror`
Les programmes qui ne compilent pas avec cette commande **ne seront pas examinés**. Un script de test est mis à votre disposition sur Moodle. Celui-ci exécute votre programme sur des jeux de tests qui serviront de base à l'évaluation de votre rendu. La commande suivante permet de lancer les tests :

```
> ./test.sh
```

Vous pouvez également lancer un test en particulier en précisant son numéro (de 1 à 4) :

```
> ./test.sh 1
> ./test.sh 2
...
```

N'hésitez pas à contacter votre enseignant si vous constatez un comportement anormal ou si vous souhaitez ajouter un test.

Le mécanisme décrit dans ce sujet rend possible un blocage du réseau. Vous écrirez dans un commentaire au début de votre programme les conditions dans lesquelles un tel blocage peut survenir. On ne vous demande pas de modifier votre programme pour corriger ce problème de blocage.

Vous devrez rendre sur Moodle un *unique* fichier nommé `reseau.c` (Moodle sait qui vous êtes, il est inutile d'appeler votre programme `Jean-Claude_Dusse_reseau.c`, et il est interdit de rendre un fichier d'un autre nom ou une archive au format du jour). De plus, assurez-vous de rendre des fichiers utilisant l'encodage UTF-8 — il y aura de sévères pénalités sinon.

Ce TP à rendre est **individuel**. On rappelle que la copie ou le plagiat sont sévèrement sanctionnés.

Annexe

L'exemple suivant crée un réseau avec trois stations dans lequel la station `1` envoie une trame à la station `2`, la station `2` envoie une trame à la station `3`, et la station 3 envoie une trame vers la station `4` (donc inconnue du commutateur) avec respectivement `aaaa`, `bbbb` et `cccc` comme *payload*. La sortie observée sur la sortie standard doit être (pas nécessairement dans cet ordre) :

```
> ./reseau 3
2 - 1 - 2 - aaaa
3 - 2 - 3 - bbbb
1 - 3 - 4 - cccc
2 - 3 - 4 - cccc
```