Un peu d'UDP

Il s'agit d'un TP noté à rendre sur Arche au plus tard le **mercredi 17 novembre 2021 à 23h59**. Écrivez votre code en **C ou en Python** (le sujet est plus simple à faire en Python). S'il y a plusieurs fichiers pour votre code, mettez-les tous dans un zip. Pour toute question, écrivez à guilhem@gamard@loria.fr.

Le but de ce TP est d'écrire un **serveur UDP** qui écoute à la fois sur les ports `2000`, `2002`, `2004`, etc. jusqu'à `2020`. Chaque fois qu'un paquet UDP est reçu sur un port N (par exemple `2004`), il est renvoyé à l'expéditeur sur le port $N + 1$ (par exemple `2005`).

Plus précisément, si le serveur reçoit un paquet UDP :

IP source	`10.0.42.54`
IP cible	`10.0.37.13`
Port source	`27015`
Port cible	`2004`
Contenu	`Hello`

alors il doit répondre avec un paquet UDP :

IP source	`10.0.37.13`
IP cible	`10.0.42.54`
Port source	`2004`
Port cible	`2005`
Contenu	`Hello`

en admettant que `10.0.37.13` soit l'adresse IP du serveur, `10.0.42.54` celle d'un client, et que le port source `27015` ait été tiré au hasard par l'OS.

Les questions qui suivent sont là pour vous **guider**. Chaque question (sauf la première) vous demande de reprendre le code de la question précédente et de l'améliorer.

Consigne : pour le rendu, envoyez le code de la **dernière question que vous avez réussi à faire marcher.** Précisez de quelle question il s'agit en commentaire.

Conseils : faites une copie de sauvegarde séparée de votre code chaque fois que vous réussissez une question, avant de passer à la suivante.
Testez votre programme (par exemple avec `netcat`) après chaque question.

Critères de notation :

- Quelle question a été atteinte ?
- Le code fonctionne-t-il dans des cas simples (s'il reçoit un paquet après l'autre) ?
- Le code fonctionne-t-il dans des cas limites (s'il reçoit 10 paquets à la fois) ?
- Le code est-il lisible (noms de variables/fonctions explicites, commentaires, etc.) ?
- Si besoin, le code utilise-t-il correctement des tableaux ou des listes de sockets ?
- Si le code est long, a-t-il un découpage en fonctions pertinent ?

Rappel : en UDP, on utilise `recvfrom` et `sendto` qui garantissent qu'**un et un seul** paquet est reçu (ou envoyé) à chaque appel. C'est bien plus simple qu'en TCP.

Hypothèse : on suppose toujours que le contenu d'un paquet fait **100 octets** maximum.

Question 1. Écrivez un programme qui écoute en UDP sur le port `2000` et qui renvoie chaque paquet reçu à son expéditeur, sur son port source (donc comme d'habitude).

Question 2. Modifiez votre programme pour qu'il utilise deux sockets afin écouter en UDP sur les ports `2000` et `2002` (toujours pour renvoyer les paquets).
Utilisez au choix `select`, les *threads,* ou les processus.
(Les processus vont rendre certaines questions plus difficiles, mais pas impossibles.)

Question 3. Modifiez votre programme pour que, au lieu de renvoyer chaque paquet sur le port source correspondant, il renvoie chaque paquet sur le port cible + 1.
Ainsi, la socket qui écoute sur le port `2000` renvoie tous les paquets sur le port `2001` (mais à la bonne adresse d'expéditeur), et la socket qui écoute sur le port `2002` renvoie tous les paquets sur le port `2003`.

Question 4. Comment testez-vous votre programme de la question précédente ? Donnez votre procédure de test, soit en l'expiquant dans un fichier texte à part, soit sous forme de script shell.

(**Piste possible :** `nc -l`.)

Question 5. Modifiez votre programme pour qu'il écoute sur tous les ports pairs entre `2000` et `2020` (inclus). Utilisez un tableau (ou une liste) de sockets et faites les appels système voulus (`bind`, etc.) dans des boucles.

- Si vous utilisez select, a priori vous n'avez pas besoin d'indications supplémentaires.
- Si vous utilisez des *threads,* faites un tableau de threads et créez-les avec une boucle.
- Si vous utilisez des processus, gardez un processus parent pour le contrôle et créez un processus enfant par socket. Dans le parent, maintenez un tableau des processus enfants.

Question 6. Modifiez votre programme pour que, si la création d'une seule socket échoue, alors tout le programme termine (y compris les processus enfants s'il y en a). Si une réception ou un envoi échoue, alors le programme affiche un message d'erreur dans la console et continue.