UNIVERSITÉ DE ROUEN	ANNÉE 2015-2016
FACULTÉ DES SCIENCES	RÉSEAUX
LICENCE 3 INFORMATIQUE	FICHE DE TD N° 7-8 (PROJET)

Client/Serveur pour le protocole TFTP (1ère partie)

Le protocole TFTP (Trivial File Transfer Protocol) est un protocole simple, basé sur UDP, qui permet uniquement de lire/écrire des fichiers depuis/sur un serveur, sur le port 69, sans authentification. Ce protocole est décrit dans la RFC [1] 1350 `http://datatracker.ietf.org/doc/rfc1350/`.

Nous allons réaliser une bibliothèque d'outils `TFTP`, utilisant les bibliothèques `AdresseInternet` et `SocketUDP` définies lors des précédents TP, qui permettra de réaliser un client et un serveur TFTP.

Dans ce TD/TP, nous utiliserons une version simplifiée du protocole (utilisation d'un mode unique et uniquement lecture sur le serveur), sur le port 6969.

Description du protocole TFTP

Voici une vue d'ensemble de ce protocole.

- Il est basé sur `UDP`.
- Il permet l'échange de fichiers sans authentification.
- Il gère des connexions (au niveau applicatif) :
  - Une demande de lecture (`RRQ`) sert de demande de connexion.
  - La taille maximum des paquets transférés est fixe (512 octets de données).
  - Les paquets sont acquittés au fur et à mesure de leur arrivée.
  - Un paquet contenant moins de 512 octets signale la fin de la transmission.
  - La plupart des erreurs provoquent l'abandon de la transmission.

Les paquets TFTP

Un paquet TFTP comporte une entête (`OPcode`) sur 2 octets, suivi des données du paquet (maximum 510 octets). L'entête indique le type du paquet :

Type	Opération	Code
1	Demande de lecture	`RRQ`
2	Demande d'écriture	`WRQ`
3	Données	`DATA`
4	Acquittement	`ACK`
5	Erreur	`ERROR`

Les paquets de type `RRQ` comportent l'entête, suivi d'un nom de fichier (terminant par l'octet 0) :

```
 2 bytes     string   1 byte     string    1 byte
 ------------------------------------------------
|  RRQ   |  Filename  |   0  |    "octet"  |   0  |
 ------------------------------------------------
```

La chaîne de caractère `"octet"` correspond au mode de transfert binaire.

1. Request For Comments

Lorsque le serveur reçoit une requête RRQ, il répond en envoyant le contenu du fichier demandé par paquets de blocs de 512 octets maximum. Les numéros de bloc commencent à 1. Si la taille est inférieure à 512 alors cela indique la fin du transfert. Chaque bloc de données est du type DATA et a la forme suivante :

```
 2 bytes     2 bytes      n bytes
 ---------------------------------
|  DATA  |   Block #  |   Donnée   |
 ---------------------------------
```

Tous les paquets autres que les erreurs, les acquittements dupliqués et ceux utilisés pour la conclusion sont acquittés. Les paquets DATA sont acquittés par des paquets ACK ou ERROR alors que les paquets RRQ et ACK sont acquittés par des paquets DATA ou ERROR. Un paquet non acquitté doit être renvoyé. Les paquets d'acquittement ACK sont de la forme suivante :

```
 2 bytes     2 bytes
 --------------------
|  ACK   |   Block #  |
 --------------------
```

Les paquets d'erreurs ERROR sont de la forme suivante :

```
 2 bytes     2 bytes      string    1 byte
 ------------------------------------------
| ERROR  |  ErrorCode |   ErrMsg   |   0  |
 ------------------------------------------
```

Un message d'erreur peut servir d'acquittement pour n'importe quel autre type de paquets. Les codes d'erreurs sont les suivants :

Code	Type d'erreur
0	Non défini
1	Fichier non trouvé
4	Opération TFTP illégale
5	TID (adresse) inconnu

Initialisation

Une connexion est initiée avec l'émission d'une requête RRQ pour lire et la réception du premier bloc de données. A chaque paquet de données est associé un numéro de bloc. Ils sont consécutifs et commencent à 1. Un paquet d'acquittement (ACK) valide un bloc. Il contient le numéro de bloc qu'il valide. En cas de réponse d'un paquet erreur (ERROR), la demande est rejetée.

Les demandes de connexion se font sur un numéro de port particulier réservé (69). Le serveur défini ensuite un identifiant dont il va se servir afin d'établir la connexion (TID).

Exemple d'établissement de connexion

— Un client envoie un paquet de type RRQ sur le port 69 d'un serveur. Il a choisi aléatoirement un port local (port A).
— Le serveur reçoit la requête sur le port 69, crée un nouveau port (port B) et envoie un paquet DATA sur le port A du client avec comme port source le port B.

— La connexion est alors établie entre le client et son port A et le serveur et son port B.
— Pour la suite, client et serveur doivent s'assurer que les paquets qu'ils reçoivent proviennent bien du port établi pour la communication.

Fin de transfert

La fin de transfert a lieu lorsque le paquet de données est de taille inférieure à 512 octets. Dans ce cas, le récepteur envoie l'acquittement final et attend un certain temps avant de rompre la liaison (si le dernier paquet est réémis, c'est que l'acquittement final s'est perdu). La connexion est ensuite rompue.

Si une requête ne peut être accordée ou qu'une erreur se produit durant le transfert, un paquet erreur est émis. Les paquets erreurs ne sont jamais acquittés.

Exercice 1

1) Quelle est la taille maximale d'un fichier que l'on peut envoyer par ce protocole ?

2) Quel doit être le type des entiers qui interviennent dans les paquets (les codes d'entête ou d'erreur) ?

3) Décrire l'ensemble des paquets échangés dans le cas où un client demande un fichier `bonjour.txt` à un serveur, le fichier contenant le texte `bonjour !!!\n` suivi de 512 espaces. On suppose que tout se passe normalement.

4) Que se passe-t-il si le premier paquet DATA est perdu ? le premier ACK ? Si le premier ACK, qui s'était perdu, arrive après le deuxième DATA ?

5) Si le dernier ACK est perdu ? Si le premier RRQ est perdu ?

6) Si le premier RRQ est dupliqué par le réseau et arrive 2 fois au serveur ?

Exercice 2

1) Ce protocole peut être vu comme un ensemble de messages Question/Réponse. Pour chacun des types décrits ci-dessus, indiquer les messages Question et Réponse, du point de vue du client ou du serveur. Décrire aussi les messages qui n'attendent pas de réponse.

Implantation d'un client/serveur TFTP (1ère partie)

Exercice 3 ***Outils de manipulation de paquets TFTP***

1) Un paquet TFTP est un buffer de taille au plus 512 octets. Définissez les fonctions suivantes :

```
int tftp_make_ack(char *buffer, size_t *length, uint16_t block);

int tftp_make_rrq(char *buffer, size_t *length, const char *fichier);

int tftp_make_data(char *buffer, size_t *length, uint16_t block, const
char *data, size_t n);

int tftp_make_error(char *buffer, size_t *length, uint16_t errorcode,
const char *message);
```

qui créent un paquet tftp pour chaque type en mettant à jour la variable `length` avec la taille des données écrites.
Valeur de retour : 0 si tout s'est bien passé, -1 sinon.

Exercice 4 ***Fonctions d'envoi simples*** Écrire la fonction :

```
void tftp_send_error(SocketUDP *socket, const AdresseInternet *dst,
      uint16_t code, const char *msg);
```

qui envoie un message d'erreur à l'adresse `dst`.

Exercice 5 ***Fonctions de Question/Réponse***

1) Écrire les fonctions suivantes :

```
int tftp_send_RRQ_wait_DATA_with_timeout(SocketUDP *socket,
    const AdresseInternet *dst, const char *fichier,
    AdresseInternet *connexion, char *reponse, size_t *replength);
```

qui envoie une demande de connexion RRQ pour lire le fichier `fichier`, sur la socket `socket` à l'adresse `dst`. La fonction se place en attente de réponse au plus `TIMEOUT` secondes. Si elle reçoit une réponse valide (i.e. de type `DATA 1`), l'adresse de l'expéditeur est renseignée dans la variable `connexion`, le paquet reçu dans `reponse` et sa taille dans `replength`. Si elle reçoit une réponse non valide, un paquet d'erreur tftp est envoyé à l'expéditeur et les variables `connexion, reponse, replength` sont indéterminées.
Valeur de retour : 1 en cas de succès, 0 si le timeout est dépassé, -1 en cas d'erreur.

2) Écrire la fonction `tftp_send_RRQ_wait_DATA` qui fonctionne comme la fonction précédente, mais en cas de dépassement du timeout renvoie la demande de connexion, au plus `NB_MAX_ENVOI` fois.

3) Écrire les fonctions :

```
int tftp_send_DATA_wait_ACK(SocketUDP *socket,
   const AdresseInternet *dst, const char *paquet, size_t paquetlen);
```

qui envoie un `paquet` de taille `paquetlen` de type `DATA` à l'adresse `dst`, et attend de recevoir en réponse depuis `dst` un paquet de type `ACK` correspondant (même numéro de bloc). Les paquets `ACK` reçus de numéro de bloc inférieur sont ignorés, tous les autres paquets reçus génèrent l'envoi d'un paquet d'erreur. Si le paquet attendu n'est pas reçu au bout de `TIMEOUT`, le paquet `DATA` est renvoyé (au plus `NB_MAX_ENVOI` fois).
Valeur de retour : 1 en cas de succès, 0 si le timeout est dépassé `NB_MAX_ENVOI`, -1 en cas d'erreur.

4) Sur le même modèle, écrire la fonction `tftp_send_ACK_wait_DATA`.

5) Sur le même modèle, écrire la fonction `tftp_send_last_ACK`.

Exercice 6 ***Client/serveur TFTP***

1) Écrire le code d'un client TFTP qui peut réaliser un transfert de type RRQ.

2) Écrire le code d'un serveur TFTP multi-threads qui ne répond qu'à des requêtes de type RRQ.