



ALG2°IR Examen de Juin 2009

Serveur de fichier simplifié

1. Directives de l'examen

- Il importe avant tout de bien lire l'énoncé et de respecter rigoureusement ce qui est demandé.
- L'examen est individuel ! Toute tentative de fraude (communication d'information locale ou à distance, orale ou électronique, utilisation de supports non explicitement autorisés) sera sanctionnée d'une **cote nulle** !
- Vous pouvez disposer des notes du cours (TD ALG2°IR) et accéder à votre répertoire Z: sur le réseau ainsi qu'aux eDistri. Vous avez ½ heure pour préparer votre environnement de travail après quoi vous ne pouvez plus utiliser aucun support électronique (clé USB, CD, ...) ni de connexion Internet !
- Vous êtes seul responsable de la sauvegarde régulière de votre travail : créez de temps à autre une copie de sécurité.
- Déposez dans le eCasier de votre professeur une archive de votre projet NetBeans complet. Donnez à cette archive votre nom et suffixez-le d'un numéro de version (ex. Dupont_V1, Dupont_V2,...).

Bon travail !

2. Fonctionnalités en bref

L'application réseau « SFTP (Simple FTP) » permet de gérer un répertoire de fichiers centralisés sur un serveur et de le partager parmi un ensemble d'utilisateurs réseau pré-enregistrés. L'utilisateur doit d'abord ouvrir une session distante avec le serveur SFTP avant de pouvoir utiliser les autres commandes de son interface graphique.

Ce qui vous est fourni :

Vous trouverez dans le eDistri de votre professeur une archive compressée du nom de « **algJuin2009SFTP - PMA.zip** ». Cette archive permet de visualiser un résultat respectant les spécifications de l'analyse. Elle contient notamment :

- une archive java « ServeurSFTPv1.jar »
- une archive java « ClientSFTPv1.jar »
- la classe « Protocole.java » à utiliser pour les règles de communication client-serveur
- un ensemble de fichiers de tests de transfert.

Ce qui vous est demandé :

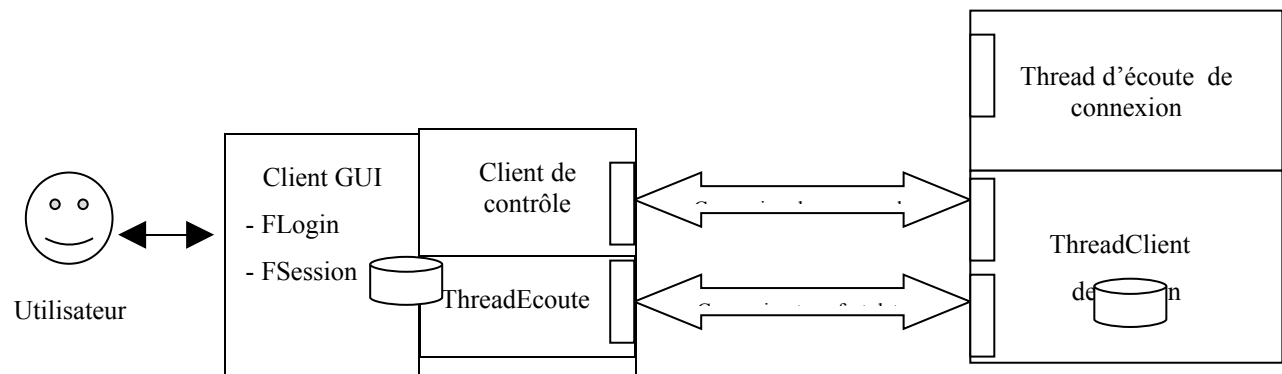
Vous devez écrire l'application serveur SFTP sans interface graphique !
Votre application doit offrir les fonctions suivantes à l'utilisateur : *connect*, *getFileBin* (mode byte) et *disconnect*.

Vous devez absolument utiliser la classe « pma.pgr.examen2009.sftp.Protocole » qui vous est fournie. Vous devez respecter les spécifications du protocole et produire une architecture client-serveur multithread.

3. Architecture et spécifications de l'application

3.1. Architecture client-serveur multithread

L'architecture applicative complète imposée est décrite par le schéma ci-dessous.



La partie cliente offre une interface GUI en deux fenêtres :

- FLogin : permet de se connecter au serveur et de réaliser le login de l'utilisateur
- FSession : permet une session de transfert de fichier

Le client crée un répertoire de travail « c:\clientSFTP » et un sous-répertoire pour contenir les fichiers de transferts : « c:\clientSFTP\rootSFTP ». Il est vital de constater qu'une deuxième connexion (« transfert data ») est nécessaire pour le transfert de fichier en mode « byte ».

Le Client vous étant donné nous n'entrerons pas plus loin dans ses détails d'implémentation. Il vous est fourni afin que vous puissiez tester les 3 fonctions demandées à votre serveur.

3.2. Le serveur

Votre application doit permettre à l'utilisateur de se connecter à votre serveur, d'accéder à ses fichiers distants et de réaliser autant de download en mode binaire qu'il le désire avant de quitter en se déconnectant proprement.

La partie serveur n'offre aucune interface utilisateur. Elle s'exécute en deux types de threads :

- Le « **Thread d'écoute** » : lancé par la méthode main, il boucle à l'infini à l'attente de connexions de clients. Il ne s'arrête jamais de lui-même. Il est arrêté par un outil système externe (ex. le task manager de Windows ou via netbeans). A la connexion d'un client il démarre un « ThreadClient ».
- Le « **ThreadClient** » : il gère la session SFTP en étant à l'écoute des commandes du client.

Pour faire simple, le répertoire de travail du serveur peut être fixé de façon statique et valoir « c:\serveurSFTP ». Il contiendra le fichier des utilisateurs enregistrés. Pour le stockage des fichiers votre serveur créera un sous-répertoire du précédent « c:\serveurSFTP\rootSFTP ».

Au démarrage, le serveur met en cache la liste des utilisateurs lue dans le fichier « uids.dat ».

Lorsqu'un client se connecte, il transmet le `userId` / `password` de son utilisateur. Le serveur vérifie l'existence de l'utilisateur dans son cache. On ne vous demande aucun traitement particulier d'exception pour cette commande. Il en est de même lorsque votre serveur traite la commande de déconnexion. Cette commande ne ferme pas les sockets ce qui permet de faire à nouveau un login. Par contre, le client peut « quitter » ce qui fermera les sockets utilisés.

3.3. Le protocole de communication de l'application complète

A) Règles générales

1. Connexion TCP de « commande »

La communication de requêtes / réponses entre les processus client et serveur de l'application se fait via une connexion de transport fiable de niveau 4 (TCP). La classe Protocole définit la constante **PORT** qui est le port TCP standard d'écoute du serveur pour les demandes de connexion des clients.

Tous les échanges de messages de commande se font sous la forme de chaînes de caractères terminées par le délimiteur « Protocole.EOL » (End Of Line). Tous les champs de longueur variable sont délimités par le délimiteur « Protocole.EOS ». La classe Protocole définit le codage des valeurs des principaux paramètres des messages.

2. Connexion TCP de « transfert data »

Le transfert en mode binaire nécessite d'ouvrir des flux de type byte de bout en bout. Cela implique donc l'ouverture d'une connexion TCP parallèle à la précédente pour implémenter le transfert de fichier. Le port utilisé côté serveur est égal à `PORT+1` !

Le serveur ouvre un flux `FileInputStream` sur le fichier source. Il lit le fichier source et écrit sur la connexion de transfert data par buffer de `BUF_SIZE` (= 256) bytes. Il convient d'ouvrir un flux `BufferedOutputStream` coté serveur et `BufferedInputStream` coté client sur les sockets de transfert data. Enfin, le client ouvre un flux `FileOutputStream` sur le fichier de destination. Il lit sur la connexion de transfert data et écrit par buffer de `BUF_SIZE` (= 256) bytes sur le fichier de destination.

La fin du transfert est reconnue par le fait que le dernier buffer transféré est de taille inférieure à `BUF_SIZE`. Tous les autres transferts sont de taille exactement égale à `BUF_SIZE`.

B) Flux principaux de l'application

1. Phase de login et ouverture d'une session SFTP

L'utilisateur démarre le programme client de contrôle. Sur l'interface graphique `FLogin`, il fournit son `userId`, son `password` ainsi que les infos de connexion au serveur (nom d'hôte et port si différent de la constante `PORT`). La session s'établit et ouvre une interface graphique `FSession` permettant la gestion des fichiers centralisés.

1- Le client envoie une requête au serveur sur la connexion de commande selon le format ci-dessous :

Requête « connexion » == <code><CONNECT><userId><EOS><password><EOS><EOL></code>

2- Le serveur examine les informations transmises et il envoie une réponse positive sur la même connexion. L'application passe à la phase « session ».

Réponse « connexion ok » == <code><CONNECT><ACK><EOL></code>

2. Phase session - Transfert de fichier `GETFILE_BIN` (download d'un fichier binaire depuis le serveur).

L'utilisateur encode le nom du fichier à transférer et clique sur le bouton « ← get ». L'opération est réalisée par un flux de messages selon le protocole ci-dessous.

1- Le client envoie une requête initiale au serveur sur la connexion de commande spécifiant le nom du fichier.

Requête n°1 « get file bin » == < GETFILE_BIN ><nom_fichier><EOS>< EOL>

2- Le serveur envoie une réponse sur la connexion de commande indiquant qu'il est prêt à envoyer le fichier et spécifiant sa taille en nombre de bytes.

Réponse n°1 == < GETFILE_BIN ><ACK><nb_bytes><EOS>< EOL>

3- L'application transfère ensuite le fichier par paquets de BUF_SIZE bytes du serveur vers le client sur la connexion de transfert data. Le client doit lire tout le flux envoyé par le serveur jusqu'à détection de EOF. On remarquera l'absence de format dans les envois.

... file transfert par paquets de BUF_SIZE bytes sur la connexion de transfert data

....

Réponse n°2 à z-1 == <byte [] buffer [BUF_SIZE]>

... EOF rencontré : le dernier envoi contient de 0 à BUF_SIZE-1 bytes ...

Réponse n°z == <byte [] buffer [0 .. BUF_SIZE-1]>

4- Remarques sur les conditions de fin du transfert.

a) Le serveur ayant rencontré la fin normale du fichier source et ayant envoyé une réponse z au client doit fermer le fichier et considérer le transfert comme terminé. Il n'a pas besoin de signaler la bonne fin du transfert au client.

Le client par contre, doit comparer le nombre de bytes reçus avec celui annoncé et afficher le résultat final du transfert à son utilisateur.

b) Le fichier source pourrait être vide. Dans ce cas : z = 2 ! Le fichier doit être créé vide sur le client.

3. Phase de déconnexion

L'utilisateur peut se déconnecter ou quitter le programme à tout moment. Le client doit avertir le serveur, si une connexion est établie. Le serveur doit confirmer la demande de déconnexion car il peut y avoir un transfert en cours.

Requête « déconnexion » == <DISCONNECT>< EOL>

Réponse == <DISCONNECT>< EOL>

C) Flux secondaires et traitement des exceptions

Des erreurs peuvent survenir durant le transfert du fichier.

1- Côté serveur. Le serveur peut rencontrer une erreur « fichier » et avorter le transfert. Vous devez traiter les cas suivants :

- Le serveur ne trouve pas le fichier ou n'arrive pas ouvrir un flux de lecture. Le transfert n'a pas lieu. Une réponse négative est envoyée sur la connexion de commande .

Réponse n°1 NACK == <GETFILE_BIN>< NACK><msg_erreur><EOS>< EOL>

- Le serveur rencontre une IOException quelconque lors de la lecture du fichier source. Il envoie un buffer vide sur la connexion de transfert data. Le client comprend que le transfert est avorté par le serveur.

... IOException rencontrée : le dernier envoi contient 0 bytes ...

Réponse n°z == <byte [] buffer [0]>

2- Côté client. Le client peut rencontrer une erreur « fichier » et avorter le transfert en fermant le fichier (s'il a été ouvert). Il doit absolument purger la connexion de transfert data en jetant tous les bytes qu'elle contient. La purge est terminée lorsqu'il reçoit un buffer de lecture de

taille inférieure à `BUF_SIZE`. Il avertit toujours son utilisateur du résultat final. Vous devez traiter les cas suivants :

- Le fichier destination ne peut pas être créé localement. En principe, le transfert n'a même pas lieu mais il faut notifier l'utilisateur !
- Une `IOException` quelconque se produit lors de l'écriture sur le fichier destination. Il avorte le transfert mais il lit jusqu'au bout.