



## 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 « **algJuin2009SFTP3.zip** ». Cette archive permet de visualiser un résultat respectant les spécifications de l'analyse. Elle contient notamment :

- une archive java « ServeurSFTPv2.jar »
- une archive java « ClientSFTPv2.jar »
- la classe « Protocole.java » à utiliser pour les règles de communication client-serveur
- un ensemble de fichiers de tests de transfert et un fichier « uids.dat ».

##### Ce qui vous est demandé :

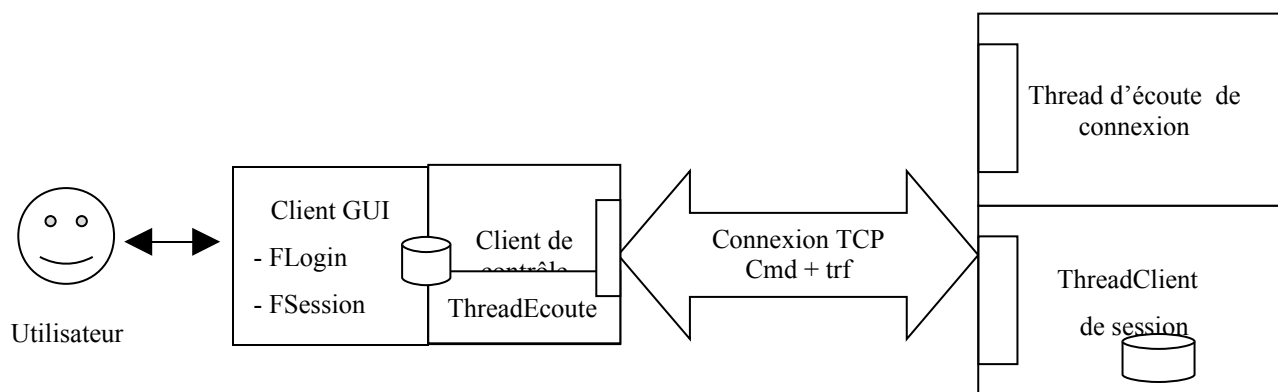
**Vous devez écrire l'application serveur SFTP sans interface graphique !**  
**Votre application doit offrir les fonctions suivantes à l'utilisateur : *connect*, *putFileChar* (mode caractère) et *disconnect*.**

**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 ». Une seule connexion TCP est nécessaire pour transmettre les commandes et pour réaliser les transferts de fichiers en mode « char ».

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 et d'ouvrir une session lui permettant de réaliser autant de uploads en mode caractère 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** » : démarré avec 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

La communication 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 applicatifs 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.

#### 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 selon le format ci-dessous :

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

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

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

##### 2. Phase session - Transfert de fichier PUTFILE\_CHAR (upload d'un fichier texte vers le serveur).

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

1- Le client envoie une requête au serveur spécifiant le nom du fichier.

<b>Requête n°1 « put file char »</b> == <PUTFILE_CHAR><nom_fichier><EOS><EOL>
-------------------------------------------------------------------------------

2- Le serveur envoie une réponse indiquant qu'il est prêt à recevoir le fichier.

<b>Réponse n°1</b> == <PUTFILE_CHAR><ACK><EOL>
------------------------------------------------

3- L'application transfère ensuite le fichier ligne à ligne du client vers le serveur. Le serveur acquitte chaque ligne de texte reçue après écriture dans le fichier de destination.

<b>... file transfert ligne à ligne client → serveur ; requete / réponse ...</b> <b>Requête n°2 à z-1</b> == <PUTFILE_CHAR><ACK><ligne de texte><EOL> <b>Réponse n°2 à z-1</b> == <PUTFILE_CHAR><ACK><EOL>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4- Le client rencontre la fin de fichier. Il transmet une requête contenant EOF. Le serveur acquitte et ferme le fichier. Le client affiche le résultat du transfert à son utilisateur.

<b>Requête n°z EOF</b> == <PUTFILE_CHAR><ACK><EOF><EOL> <b>Réponse n°z EOF</b> == <PUTFILE_CHAR><ACK><EOL>
---------------------------------------------------------------------------------------------------------------

Attention :

- La réponse n°z est obligatoire et est envoyée après fermeture du flux sur le fichier cible.

- Le fichier source pourrait être vide. Dans ce cas :  $z = 2$  ! Le fichier doit être créé vide sur le serveur.

### 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.

<b>Requête « déconnexion »</b> == <DISCONNECT>< EOL> <b>Réponse</b> == <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 en envoyant une réponse NACK contenant la cause. Le client interrompt son transfert. Il avertit son utilisateur. Vous devez traiter les cas suivants :

- Le serveur ne peut pas créer le fichier.
- Le serveur rencontre une IOException quelconque lors de l'écriture dans le fichier de destination.

<b>Réponse n°z NACK</b> == <PUTFILE_CHAR>< NACK><msg erreur><EOS>< EOL>
-------------------------------------------------------------------------

2- Côté client. Le client rencontre une erreur « fichier ». Il avorte le transfert en envoyant une requête NACK au serveur. Il avertit son utilisateur. Vous devez traiter les cas suivants :

- Le fichier source demandé est introuvable
- Une IOException quelconque se produit lors de la lecture du fichier

<b>Requête n°z NACK EOF</b> == <PUTFILE_CHAR><NACK>< EOL>
-----------------------------------------------------------