université PARIS DIDEROT PARIS 7

Projet C6 2016

tCHATche

un système de communication interactif multi-utilisateurs

1 Introduction

Un utilisateur se connecte au système en lançant une application cliente de nom `tchatche`. Celle-ci propose alors à l'utilisateur de s'identifier (par exemple en choisissant `yunes`) et une fois ceci effectué, chaque ligne tapée par l'utilisateur (par exemple `bonjour ca va ?`) est distribuée à tous les autres utilisateurs déjà identifiés (le message est alors préfixé par l'identification de l'utilisateur afin que la lecture du flot soit facilité, ex. : `[yunes] bonjour ca va ?`, comme dans la session hypothétique suivante :

```
Pseudo> barbatruc
Connected.
> Bonjour ça va ?
[barbatruc] Bonjour ça va ?
[bidibulle] Salut barba!
[trucmuche] Yo barbatruc!
> Je crois que mon projet marche!
[barbatruc] Je crois que mon projet marche!
...
```

Durant la conversation, l'utilisateur a la possibilité de demander la réalisation de certaines fonctionnalités, comme envoyer un fichier à un autre utilisateur, établir une conversation privée avec un utilisateur particulier, se déconnecter, etc.

```
...
> private to bidibulle C'est vraiment un blaireau ce trucmuche, non ?
[barbatruc --> bidibulle] C'est vraiment un blaireau ce trucmuche, non ?
> ...
[bidibulle --> barbatruc] Ouais, trop!
...
> send file trucmucheenmauvaiseposture.jpg to bidibulle
> quit
Bye barbatruc...
```

Bien entendu toute autre présentation et langage de commande sont acceptés.

2 Architecture

Le système repose sur deux exécutables, `tchatche` qui réalise l'interface avec l'utilisateur et `tchatche_serveur` qui réalise la distribution des messages. `tchatche_serveur` doit être préalablement lancé si l'on souhaite que le système de messagerie fonctionne. Les deux applications communiquent via un protocole décrit plus loin. Les messages de ce protocole circulent dans des tubes nommés. `tchatche_serveur` possède son propre tube nommé dont le chemin est une constante prédéfinie que nous nommerons S. Chaque processus d'interface avec l'utilisateur (`tchatche`) possède un tube nommé qui lui est propre que nous nommerons c_i pour tout client i.
`tchatche` envoie ses messages du protocole dans S et lit dans c_i. `tchatche_serveur` écrit au client i en écrivant des messages du protocole dans c_i, et il lit des commandes à réaliser dans S.

3 Protocole

Le protocole définit à la fois le format des messages qui circulent et leur ordonnancement. On notera que le serveur n'envoie jamais de message de sa propre initiative (sauf extension à inventer ?), mais toujours en réponse à une sollicitation provenant d'un utilisateur.
Dans tout message :

- les **nombre**s représentés sous la forme d'entiers codés en décimal et représentés sous la forme de 4 caractères (les **nombres longs** seront codés sur 8 caractères),
- les **type**s sont des suites de 4 caractères ASCII (ce ne sont pas des chaînes),
- les **chaîne**s de caractères sous la forme d'une longueur (codée comme un nombre) suivie par la suite des caractères de la chaîne sauf le caractère nul,
- et des **donnée**s sous la forme d'une longueur (codée comme un nombre) suivie par la suite des octets.

Tout message est constitué *a minima* d'une longueur et d'un type, suivi par un possible corps de message, la longueur mesurant la totalité du message (c'est-à-dire y compris l'espace occupé par la longueur elle-même) :

éléments :	*l*	*type*	*corps*
type :	nombre	type	

Par exemple, si l'on souhaite faire circuler la chaîne de caractère `Bonjour` dans un message de type `XYZT`, il faut envoyer les 15 octets suivants :

'0'	'0'	'1'	'5'	'X'	'Y'	'Z'	'T'	'0'	'0'	'0'	'3'	'B'	'o'	'n'

3.1 Connexion au système

Un utilisateur obtient sa connexion en envoyant son pseudo et en recevant en cas de succès un identifiant. Les messages échangés correspondants sont :

- envoyé dans S par l'utilisateur :

éléments :	*l*	`HELO`	pseudo	tube
	nombre	type	chaîne	chaîne

  où pseudo représentera l'utilisateur dans le système (c'est une valeur qu'il choisit) et tube le nom du tube nommé qui sera utilisé pour lui écrire (c'est aussi une valeur qu'il choisit).
- en cas de succès le serveur renverra, dans le tube de nom indiqué, un message de la forme :

éléments :	*l*	`OKOK`	*id*
	nombre	type	nombre

où *id* représentera l'identifiant interne de l'utilisateur dans le système. À chaque connexion un identifiant est attribué par le serveur (cet identifiant peut-être différent d'une connexion à l'autre, c'est un identifiant dynamique, une clé de session). On notera que le système doit garder une trace de la liaison entre le pseudo, l'identifiant et le tube.

— en cas d'échec le serveur renverra :

éléments :	*l*	`BADD`
	nombre	type

À partir du moment où la connexion a été acceptée, l'utilisateur peut recevoir toute sorte de messages et à n'importe quel moment.

3.2 Déconnexion

Pour obtenir sa déconnexion, l'utilisateur doit envoyer dans S le message :

éléments :	*l*	`BYEE`	*id*
	nombre	type	nombre

et en réponse le serveur doit lui envoyer dans son tube client :

éléments :	*l*	`BYEE`	*id*
	nombre	type	nombre

Suite à quoi le serveur fait disparaître toute donnée interne correspond à l'utilisateur et l'utilisateur nettoie correctement son environnement (tubes, etc.) avant de quitter.

3.3 Envoi d'un message public

Pour obtenir la distribution d'un message à tous les utilisateurs identifiés, un utilisateur doit envoyer au serveur un message de la forme :

éléments :	*l*	`BCST`	*id*	message
	nombre	type	nombre	chaîne

ce qui en retour provoque l'envoi par le serveur et à **tous** les utilisateurs d'un message de la forme :

éléments :	*l*	`BCST`	pseudo	message
	nombre	type	chaîne	chaîne

3.4 Envoi d'un message privé

Pour obtenir la distribution d'un message privé, un utilisateur doit envoyer au serveur un message de la forme :

éléments :	*l*	`PRVT`	*id*	pseudo	message
	nombre	type	nombre	chaîne	chaîne

ce qui en retour provoque l'envoi par le serveur et uniquement à l'utilisateur de pseudo indiqué d'un message de la forme (où pseudo est cette fois celui de l'envoyeur) :

éléments :	*l*	`PRVT`	pseudo	message
	nombre	type	chaîne	chaîne

et l'envoi en cas de succès à l'utilisateur d'origine d'une réponse `OKOK`, et en cas d'échec un message `BADD`.

3.5 Liste des utilisateurs

Pour obtenir la liste des utilisateurs, il faut envoyer au serveur le message :

éléments :	*l*	`LIST`	id
	nombre	type	nombre

qui provoque en retour l'envoi de n messages (où n est le nombre d'utilisateurs du système) de la forme :

éléments :	l	LIST	n	pseudo
	nombre	type	nombre	chaîne

pseudo correspond au nom d'un utilisateur actuellement identifié.

3.6 Message de contrôle

Il existe deux messages particuliers dits de contrôle.
Le premier permet l'extinction totale du système, c'est-à-dire la déconnexion forcée de tous les utilisateurs et la terminaison propre du serveur. Pour cela il faut envoyer au serveur un message de la forme :

éléments :	l	SHUT	id
	nombre	type	nombre

qui provoque l'envoi à tous les utilisateurs connectés l'envoi immédiat du message :

éléments :	l	SHUT	pseudo
	nombre	type	nombre

Le second message permet d'obtenir des informations internes du serveur (affichage des tables, etc), son effet réel n'est pas spécifié mais il peut-être utilisé pour aider au déboguage, il a la forme :

éléments :	l	DEBG
	nombre	type

Ce message n'a pas pour effet de produire un message en réponse.

3.7 Transfert de fichier (commande avancée)

L'obtention d'un transfert de fichier nécessite l'envoi de plusieurs messages, car ils sont transférés par paquet de 256 octets sauf le dernier.
Le premier message a la forme :

éléments :	l	FILE	série	id	pseudo	longueur du fichier	nom de fichier
	nombre	type	nombre	nombre	chaîne	nombre long	chaîne

dans lequel série doit correspondre au nombre 0. Ce message doit provoquer en cas de succès l'envoi des deux message suivants, le premier au réceptionnaire du transfert :

éléments :	l	FILE	série	idtransfert	longueur fichier	nom fichier
	nombre	type	nombre	nombre	nombre long	chaîne

et l'autre à la source du transfert :

éléments :	l	OKOK	idtransfert
	nombre	type	nombre

dans ces messages idtransfert est un identifiant interne de transfert de fichier qui est attribué par le serveur.
En cas d'échec l'envoyeur reçoit :

éléments :	l	BADD
	nombre	type

Si le transfert est possible alors l'envoyeur doit envoyer le contenu à l'aide de messages :

éléments :	l	FILE	série	idtransfert	données
	nombre	type	nombre	nombre	données

qui doivent correspondre à l'envoi de paquets de 256 octets de données du fichier sauf éventuellement pour le dernier message qui ne doit contenir que les octets restant à envoyer. Le numéro de série correspond au paquet, ainsi pour un fichier de 513 octets il doit y avoir 3 messages de la forme décrite : le premier avec le numéro de série 1 et les 256 premiers octets du fichier, le second avec le numéro 2 et les 256 octets qui suivent puis le troisième avec le numéro 3 et le dernier octet du fichier.
Chaque message doit provoquer l'envoi auprès du destinataire des messages :

éléments :	l	FILE	série	idtransfert	données
	nombre	type	nombre	nombre	données

4 Réalisation

Le projet est à réaliser en C et doit pouvoir fonctionner sur les machines de l'UFR (impératif).
Il ne peut être réalisé que par équipes de 2 ou 3 étudiants.
On veillera à produire un code modulaire, clair et documenté.
Une bonne partie du projet consiste à établir une interface entre le clavier/écran de l'utilisateur et le protocole sous-jacent au système. Attention il ne s'agit pas d'utiliser une interface graphique (c'est même formellement interdit), par contre si vous voulez utiliser une bibliothèque comme `ncurses` se sera considéré comme un plus (si vous ne vous sentez pas à l'aise avec le C et le système, n'essayez pas !) [1]. Il est encore plus important de songer à des problèmes comme la terminaison correcte sans laisser de fichier temporaire ou autres scories d'exécution. Vous veillerez aussi à faire en sorte que les erreurs pouvant se produire n'empêche pas la bonne continuation du service ; il faut peut-être pour cela identifier les problèmes qui pourraient survenir lors des communications, etc.
Vous pouvez commencer par la partie serveur, c'est même conseillé... Vous pouvez même aller assez loin puisque le protocole permet d'envoyer des messages «à la main», donc un client peut être réduit (dans un premier temps) à de simples manipulations de commandes shell...
La partie «transfert de fichier» n'est pas à considérer en premier lieu, c'est plutôt une partie avancée du projet, il faut vous focaliser sur le reste. Il est préférable d'avoir une version fonctionnelle sans transfert de fichiers, plutôt que des bouts de fonctionnalités mal terminées.
Lors de la soutenance il sera important de préparer des cas d'utilisation, des exemples parlants et démonstratifs du bon fonctionnement général. Vous ne devez pas négliger pour autant la maîtrise des limites ; il n'est pas très grave que certaines choses posent problème, il est plus grave de ne pas en être conscient.
Vous pouvez ajouter des commandes au protocole, **uniquement** si ce qui a été demandé fonctionne correctement. Mais vous devez impérativement respecter le protocole, pour cela vous devriez pouvoir essayer d'utiliser votre `tchatche` avec le serveur d'un autre groupe, par exemple. Ce genre d'échange sera apprécié à sa juste valeur.

1. Si vous utilisez `ncurses` il faudra au moins séparer, dans la fenêtre principale, la zone de saisie de l'utilisateur et le flux des communications.