

# BE processus communicants (minichat)

## 1 Objectifs

Ce BE vise à vous permettre de mettre en pratique les notions de base vues en TD et TP, autour de l'interaction par tubes et par segments de mémoire partagée, par la réalisation d'une application pratique : une messagerie interactive.

Le BE consiste à compléter deux versions successives (serveur et tableau blanc) de l'application.

## 2 Application de messagerie interactive

L'objectif est de permettre à un ensemble de participants situés sur des machines différentes d'échanger et de partager des messages de manière interactive. Chaque message échangé sera visible instantanément par chacun des participants connectés. Chacun des messages échangés consistera en une ligne de texte, limitée à 128 caractères. La succession des messages échangés s'appellera une *discussion*.

L'application fournie à chaque participant utilisera un terminal, décomposé en 2 zones :

- la partie supérieure sera utilisée pour afficher les derniers messages échangés depuis que l'utilisateur a rejoint la discussion ;
- la partie inférieure (dernière ligne) sera réservée à la saisie d'une ligne de texte, correspondant au prochain message à envoyer.

Un participant rejoint une discussion en lançant l'application, en fournissant un paramètre correspondant à son *pseudonyme* pour l'application. Ce pseudonyme préfixera les différents messages envoyés par l'utilisateur, afin de permettre d'identifier les auteurs des différents messages. Une fois la discussion rejointe, les messages s'affichent (dans la zone supérieure) au fur et à mesure de leur émission par les participants. Le défilement des messages est géré de sorte que le message le plus récent se trouve en bas, juste au dessus de la zone de saisie. Le participant peut saisir un message à tout moment, l'envoi du message étant provoqué par le retour chariot. Pour quitter (proprement) la discussion, un participant peut saisir un message prédéfini par l'application (par exemple : "au revoir").<sup>1</sup>

Afin de fournir une idée plus précise de ce qui est attendu, l'exemple ci dessous regroupe les écrans d'une discussion, entamée par **sosthène**, rejointe par **euphrasie**, puis **theodore** (qui quitte presque aussitôt).<sup>1</sup>

```

Terminal — -tcsh — 364
- tcsh

[service] theodore rejoint la conversation
[euphrasie] salut theodore
=====
au revoir
fin client
[aacpro-intel-wf:minichat/code/serveur] philippe%

Terminal — serveur — 365
serveur
[aacpro-intel-wf:minichat/code/serveur] philippe% ls
console      console.c    serveur      serveur.c
[aacpro-intel-wf:minichat/code/serveur] philippe% ./serveur
participants actifs : 0
participants actifs : 1
participants actifs : 2
participants actifs : 2
participants actifs : 2
participants actifs : 3
participants actifs : 3
participants actifs : 3
participants actifs : 2
participants actifs : 2

Terminal — console euphrasie — 363
console euphrasie

[service] theodore quitte la conversation
=====

[service] euphrasie rejoint la conversation
[sosthène] y a quelqu'un ?
[euphrasie] oui
[sosthène] socrate est un homme
[service] theodore rejoint la conversation
[euphrasie] salut theodore
[theodore] au revoir
[service] theodore quitte la conversation
[sosthène] déjà parti !
=====

Terminal — console sosthène — 361
console sosthène

[service] sosthène rejoint la conversation
[service] euphrasie rejoint la conversation
[sosthène] y a quelqu'un ?
[euphrasie] oui
[sosthène] socrate est un homme
[service] theodore rejoint la conversation
[euphrasie] salut theodore
[theodore] au revoir
[service] theodore quitte la conversation
[sosthène] déjà parti !
=====

```

Une courte vidéo, disponible sur Moodle fournit un exemple animé de session.

1. Le dernier écran est une trace de l'application serveur, puisqu'il s'agit ici de la version serveur (cf section suivante)

**Remarques**

- afin d'éviter les complications liées à l'utilisation de services et protocoles réseau, les applications des différents utilisateurs s'exécuteront sur une même machine, ce qui ne réduit pas pour autant les possibilités d'utilisation en réseau, puisque ces applications peuvent être lancées à distance, via **ssh**.
- l'interface, les fonctionnalités et l'ergonomie doivent être très simples, sans pour autant réduire drastiquement l'utilisabilité. Ainsi, limiter l'affichage aux 20 derniers messages, limiter la taille des messages, ne pas gérer le fait qu'une saisie peut être coupée par un affichage sont des contraintes raisonnables. A contrario, permettre d'introduire des émoticônes dans les messages semble peu pertinent, au regard de la complication introduite, ou ne pas accompagner le message par le pseudo de son auteur semble une simplification trop réductrice.
- pour chacune des versions demandées, un squelette est fourni pour les différents composants de l'application. Ce squelette fournit une implantation minimale de l'interface utilisateur que vous pouvez considérer comme suffisante pour les besoins du BE : l'objectif du BE est de développer des schémas d'interaction entre processus, et non des interfaces graphiques. Un fichier **LisezMoi.md** accompagne le squelette, et en précise les particularités d'utilisation.

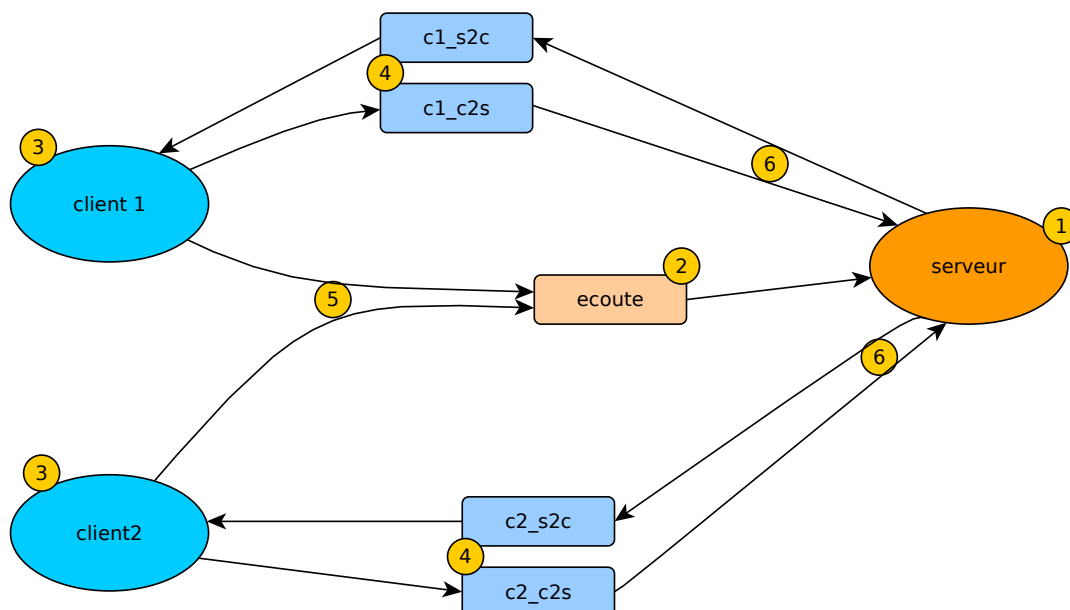
### 3 Version serveur [effort de développement estimé < 100 lignes]

Pour cette version, les messages échangés entre processus participants transitent par des tubes, selon le principe suivant :

- un processus particulier, le **serveur**, centralise les messages émis par les participants, et les retransmet à chacun des participants. Le serveur est donc relié à chacun des participants par deux tubes :
  - un tube du participant vers le serveur, par lequel le participant envoie les messages saisis
  - un tube du serveur vers le participant, par lequel le participant reçoit les messages émis par le serveur
- les processus participants (**clients**) attendent des messages soit de l'entrée standard (auquel cas ils les retransmettent au serveur), soit du serveur (auquel cas ils actualisent la liste des derniers messages de la discussion, et en rafraîchissent l'affichage)
- un tube particulier (**tube d'écoute**), lu par le serveur, écrit par les participants, permet à un nouveau participant de s'enregistrer auprès du serveur, en fournissant son identité et celle de ses tubes de communication.

La figure suivante illustre cette architecture, avec les conventions de nommage proposées dans le squelette de code fourni avec la version serveur :

- ① le processus **serveur** est lancé ;
- ② le serveur crée le tube d'écoute (**écoute**) ;
- ③ les participants sont lancés. Leurs pseudonymes respectifs sont **c1** et **c2** ;
- ④ le client **c1** crée 2 tubes de service : un tube nommé **c1\_c2s** pour l'envoi des messages du client vers le serveur, et un tube nommé **c1\_s2c** pour la réception des messages diffusés par le serveur. De même, client **c2** crée 2 tubes nommés **c2\_c2s** et **c2\_s2c**
- ⑤ les clients s'enregistrent auprès du serveur en écrivant leur pseudonyme sur le tube d'écoute.
- ⑥ à la réception d'un pseudonyme **xxx** sur le tube d'écoute, le serveur intègre le nouveau participant en ouvrant les tubes de service nommés **xxx\_c2s** et **xxx\_s2c**, d'après le pseudonyme du participant.
- ⑦ le serveur attend (en lecture) des messages sur l'un des tubes **xxx\_c2s**. Lorsqu'un message est lu sur l'un de ces tubes, il est écrit sur l'ensemble des tubes **xxx\_s2c**.



Une courte vidéo, disponible sur Moodle commente cette illustration de manière un peu plus développée.

### 3.1 Remarques

- Les processus participants et le serveur étant lancés séparément, les tubes utilisés seront des **tubes nommés** (ou **fifo**), pour permettre de les désigner en dehors du contexte d'exécution des processus.
- Soyez particulièrement attentifs à la bonne gestion de la connexion/déconnexion des participants : il ne faut pas que l'attente d'un participant en cours de connexion ou ayant quitté la discussion bloque les autres participants ou le serveur (ou pire : en provoque la panne).
- L'exécution du serveur ne doit pas dépendre de l'existence (ou non) de participants : il est normal (et conforme à ce qui est attendu) que le serveur tourne, et soit prêt à accepter de nouveaux participants, même si aucun participant n'est connecté à un moment donné.

### 3.2 Points techniques

- l'ouverture des tubes nommés fait l'objet d'une synchronisation (*rendez-vous* entre lecteur et écrivain) : un lecteur (resp. écrivain) qui ouvre un tube doit attendre qu'au moins un écrivain (resp. lecteur) l'ait ouvert. Il est donc important que, dans le cas où plusieurs tubes sont utilisés, ces tubes soient ouverts **dans le même ordre** : sinon, il y aurait un risque d'interblocage entre deux correspondants s'attendant mutuellement... mais sur des tubes différents. Une autre possibilité (mais qui retire de l'intérêt à l'utilisation d'un **select**) est de rendre les E/S non bloquantes à l'ouverture (option **O\_NONBLOCK**) et/ou via **fcntl**.
- **select** signale qu'un descripteur est prêt en lecture si le **read** est non bloquant, ce qui englobe la disponibilité de données en lecture sur ce descripteur **mais aussi** la fin de fichier ou une erreur de lecture.

## 4 Version tableau blanc [effort de développement estimé < 20 lignes]

Dans cette version, les messages sont directement écrits par les participants dans un segment de mémoire partagée, sans transiter par un serveur intermédiaire. Tous les processus sont donc similaires.

### 4.1 Point technique

**mmap** ne spécifie pas quel est le résultat d'une écriture **après** la fin d'un fichier couplé (**SIGBUS** est une possibilité, fréquente). Il faut donc fixer la taille du fichier destination à la taille du fichier source **avant** le couplage. Pour cela, on peut se positionner à la taille souhaitée pour le fichier, par **lseek()**, et écrire un octet à cette position.

## 5 Modalités pratiques

Un squelette de code est fourni pour les deux versions, qu'il vous faut compléter, essentiellement en mettant en place les schémas de communication appropriés. Chacun de ces squelettes comporte la déclaration de structures de données, procédures et variables pouvant vous être utiles dans le développement de votre solution, ainsi que quelques fonctions utilitaires. Ces squelettes ne sont pas forcément complets en termes de définitions, et votre solution peut nécessiter des procédures, structures ou variables différentes.

Les rendus sont **optionnels**. Chaque version pourra être rendue, respectivement à l'issue du TP sur les tubes nommés et **select/fcntl** (semaine du 11 mai) et à l'issue du TP sur la mémoire partagée (semaine du 18 mai). Chaque rendu sera évalué sur 5 points. La ou les note(s) obtenue(s) **s'ajouteront** à la note du projet minishell. **La date de clôture des rendus est fixée au 1 juin, 23h59.**