

Rapport Projet de SE (S3) : Service de messagerie instantanée

Bertrand DAVID

18 novembre 2007

Table des matières

I	Introduction sur le projet	2
1	Avant propos	2
1.1	Etat du projet	2
1.2	Rapport	2
2	Objectifs	2
3	Buts	3
II	Discussion entre fenêtres	3
4	pmi.c	3
4.1	Rôle	3
4.2	Interprète de commande	3
4.3	Parsing de la saisie avec likeArgv()	3
4.4	Mécanisme de duplication de processus : fork()	4
4.5	Mécanisme de spécialisation d'un processus : exec()	4
4.6	Mécanisme de communication entre processus : pipe()	4
5	talk.c	4
5.1	Mécanisme de parallélisme : les threads	4
III	Gestion d'une liste d'utilisateur	5
6	Erratum	5
7	Segment de mémoire partagé : structure usersRep	5

8 Allocation dynamique d'un tableau à 2 dimensions	5
9 Libération de la mémoire d'un tableau 2d alloué dynamiquement	6
IV Méthodologie	6
10 Outils utilisés	6
10.1 Système d'exploitation	6
10.2 Environnement de développement : Emacs21	6
10.3 Debogueur : gdb	6
10.4 Inspecteur de mémoire : valgrind	7

Première partie

Introduction sur le projet

1 Avant propos

1.1 Etat du projet

Je suis très fier de pouvoir dire que mon projet fonctionne entièrement. J'avoue avoir eu des difficultés dans la partie II du projet consistant à gérer une liste des utilisateurs. Cela s'explique par le fait que notre enseignant de TD (Mr Lesner) s'est grandement attardé sur la partie I à la demande générale du groupe. Ce n'est donc pas sans difficulté que j'ai pu achevé la partie II de ce projet.

1.2 Rapport

Je suis désolé de ne pas avoir écrit un rapport assez concis. Toutefois, la mise en page, les marges, la titraille ne doivent pas vous induire en erreur : Ce rapport est moins long que l'on peut le penser en observant le nombre de pages.

2 Objectifs

Les objectifs étaient les suivants :

- Permettre le dialogue entre deux clients connectés à un service.
- Permettre l'inscription et la desinscription d'une liste.

3 Buts

Les buts étaient bien sûr inhérents au cours de système d'exploitation. A l'aide d'un code incomplet, nous devons implémenter certains mécanismes vus en cours de SE :

1. Duplication de processus (fork, thread)
2. Spécialisation d'un processus (exec)
3. Communication inter-processus (IPC, mémoire partagée, pipe)
4. Exclusion mutuelle(IPC)

Deuxième partie

Discussion entre fenêtres

4 pmi.c

4.1 Rôle

Pmi est le service de messagerie instantanée. C'est ce fichier qui va se charger de créer divers processus et de lancer certaines fonctionnalités demandées par l'utilisateur :

- Parler à un autre utilisateur
- S'enregistrer ou se supprimer de la liste des utilisateurs
- Quitter le service

4.2 Interprète de commande

J'ai choisi d'utiliser la bibliothèque GNUReadline pour la saisie des commandes utilisateurs. L'ayant déjà utilisé dans un projet personnel, il me semblait plus qu'approprié de l'utiliser dans l'invite de commande du pmi. Ajoutons également qu'il est très simple et peu contraignant d'utiliser readline() dès qu'il s'agit d'invite de commande. La bibliothèque GNUreadline permet :

1. L'édition de ligne (ncurses)
2. La constitution d'un historique¹.

4.3 Parsing de la saisie avec likeArgv()

Au 1er TD, j'avais codé une fonction qui analysait la saisie et la découpée. Un tableau à 2 dimensions était renvoyé avec les arguments. J'ai programmé cette fonction avant d'apprendre l'existence de la fonction strtok(). Par la suite, j'ai intégré strtok() dans ma fonction. Toutefois, il est possible de reprogrammer

¹L'historique de GNUReadline fonctionne comme l'historique des commandes dans bash (appuie sur la touche du haut).

likeArgv() en 4 fois moins de lignes et sans faire une allocation dynamique d'un tableau en 2d dans la fonction.

4.4 Mécanisme de duplication de processus : fork()

La fonction système fork() permet de dupliquer un processus. Or nous cherchons à créer plusieurs processus pour créer les deux clients.

L'utilisation de thread permettrait aussi d'effectuer les tâches voulues en parallèle. Le code incomplet propose, d'ailleurs, l'utilisation d'un thread pour la lecture de caractère dans talk.c

4.5 Mécanisme de spécialisation d'un processus : exec()

L'appel système exec() permet d'écraser le code présent par le code passé en paramètre de la fonction. Pmi gère toutes les interactions possibles de l'utilisateur sur le service. Seulement, pour ouvrir une fenêtre xterm, il faut que pmi puisse lancer xterm avec comme paramètre le programme de discussion (talk).

Dans le projet, il nous est imposé de faire un fork() pour dupliquer le service pmi puis un exec() pour remplacer le code de pmi par le code de xterm². Nous aurions pu utiliser la fonction system() fourni par la bibliothèque stdlib.h de la norme du langage C ANSI/ISO qui effectue un fork() suivi d'un exec() de manière implicite.

4.6 Mécanisme de communication entre processus : pipe()

La fonction pipe() permet d'ouvrir deux descripteurs :

- Un descripteur en lecture.
- Un descripteur en écriture.

Une des propriétés de fork() et de pouvoir partager des descripteurs en communs. Nous allons donc créer 2 tubes par processus pour la simple et bonne raison qu'un pipe est unidirectionnel.

5 talk.c

5.1 Mécanisme de parallélisme : les threads

Une discussion par messagerie instantanée implique chaque client puissent faire ces deux choses à la fois et de manière synchrone :

- Ecrire un message
- Lire un message

Dans cette optique, le code à compléter proposait d'utiliser un thread qui s'exécute en parallèle de la fonction talk chargé de l'affichage du prompt et de la

²Xterm est lancé avec comme paramètre le programme talk. Par transitivité, on peut dire que cela revient à lancer talk.

saisie. Ce thread est chargé de la lecture : dès qu'il recoit un caractère, il l'envoie sur la sortie `STDOUT`³.

Troisième partie

Gestion d'une liste d'utilisateur

6 Erratum

Il y a une erreur dans le code fourni pour l'étape #2 :

```
id = shmget(..., ..., IPC_CREAT|O_EXCL|0666);
```

doit être remplacé par :

```
id = shmget(..., ..., IPC_CREAT|IPC_EXCL|0666);
```

C'est une erreur qui m'a un peu perturbé dans la mesure où elle faisait partie du code incomplet.

7 Segment de mémoire partagé : structure `user-Rep`

J'ai décidé, comme dans l'exemple fourni, de définir le segment de mémoire partagé comme une structure à 2 types :

- `int nb` : contient le nombre d'utilisateurs enregistrés
- `char users[MAX_USER][TAILLE_NOM_MAX]` : tableau à 2 dimensions contenant les noms des utilisateurs enregistrés.

Le champ `nb` est très pratique car il me permet de savoir où est ce que mon code doit s'arrêter de remplir le tableau à 2 dimensions renvoyer par la fonction `get_liste_utilisateurs()` : j'évite l'erreur de segmentation. D'autre part, le fait d'utiliser le champ `nb` comme "limite" m'évite d'utiliser l'astuce proposée dans le code incomplet qui consiste à remplir une chaîne de caractère par "NULL" afin de délimiter ce qui est rempli dans le tableau de ce qui ne l'est pas⁴.

8 Allocation dynamique d'un tableau à 2 dimensions

La fonction `get_liste_utilisateurs()` retourne un pointeur de pointeur que l'on peut voir comme un tableau de pointeurs de char. On alloue d'abord un tableau qui va contenir les adresses vers les noms d'utilisateurs. La taille du tableau dans chaque dimension est défini par des symboles `MAX_USERS` et

³Stdout : affichage à l'écran.

⁴J'ai quand même implémenté l'astuce bien qu'elle ne me serve pas.

TAILLE_NOM_MAX dans le fichier header “chat.h” fourni avec le code incomplet⁵.

9 Libération de la mémoire d’un tableau 2d alloué dynamiquement

La démarche est la même que pour allouer dynamiquement sauf que l’on utilise la fonction `free()` au lieu de `malloc()`. La libération de la mémoire se fait à la suite de l’appel à la fonction `get_liste_utilisateurs()` dans `pmi.c`. À l’aide d’une boucle `for`, on libère les pointeurs dans le tableau de pointeurs puis on termine en libérant l’adresse du tableau de pointeurs.

Quatrième partie

Méthodologie

10 Outils utilisés

10.1 Système d’exploitation

Nous avons utilisé des appels systemes définies par la norme POSIX. Cela explique le besoin d’utiliser un système d’exploitation implémentant tous ces appels. Nous avons donc utilisé GNU/Linux.

10.2 Environnement de developpement : Emacs21

Emacs21 est un puissant outil de developpement contenant de nombreuses fonctionnalités :

- Possibilité de “couper” l’écran en deux (pour visualiser le code de `pmi.c` pendant que j’écris `pmi2.c` par exemple).
- Lancer la compilation avec “make” sans quitter l’IDE.
- Coloration syntaxique et indentation automatique et normée.

10.3 Debogueur : gdb

Pour ma part, je n’aurais jamais pu livrer un projet pleinement fonctionnel sans cet outil : le débogueur. Dans la partie 2, il m’a même été indispensable pour inspecter mes variables de `pmi2.c`. Ce projet aura eu également le mérite de permettre de maîtriser GDB de manière plus approfondie en allant inspecter des fichiers objets⁶. Il faut de même signaler que le Makefile fourni gracieusement compile avec l’option de compilation symbolique “-g”, facilitant ainsi le debugage.

⁵cf mon code pour voir l’allocation dynamique d’un tableau 2d avec une boucle `for`.

⁶`pmi2.o`

10.4 Inspecteur de mémoire : valgrind

Cet outil me permet de vérifier l'état de la mémoire :

- Vérifier que je fais autant de `free()` que de `malloc()`
- Voir quelle ligne du code provoque des “Segmentation Fault” ou des “SegInt”⁷.

⁷Cela est possible grâce à l'option de compilation symbolique “-g” passé en option lors de la compilation.