Rapport : Projet 1 de programmation système, Chat - édition processus et pipe

Min-Tchun Li, Maxence Perera Gonzalez, Ayman Kahouache

Contents

1 Introduction

Pour le premier projet du cours INFO-F-201, nous avons créé un *chat* en C permettant deux utilisateurs de discuter entre eux à l'aide de pipes nommés, et un *chat-bot* qui pourra remplacer un utilisateur et effectuer certaines commandes.

Les buts de ce projet sont de gérer les signaux **SIGINT** et **SIGPIPE**, ainsi que les communications entre les processus à l'aide de pipes nommés.

2 Programme en C : chat

2.1 Paramètres et affichage

Avant tout, les pseudonymes des utilisateurs seront toujours donnés en premier et deuxième. Si ce n'est pas le cas, nous considérerons tout de même ces paramètres en tant que pseudonymes. De plus, ces derniers ne peuvent pas dépasser 30 caractères, sinon un message d'erreur est retourné. Par la suite, le programme possède trois modes : le mode par défaut, le mode *bot* et le mode *manuel*. Le premier affiche les messages normalement, c'est-à-dire que le pseudonyme des utilisateurs sont soulignés et sont suivis par les messages qu'ils ont envoyé. Le deuxième affichera les messages mais ne soulignera pas le texte, et ces derniers ne seront pas coloré. Et le dernier mode n'affichera pas les messages à la réception. Ces derniers ne s'afficheront seulement dans ces trois cas suivants :

- Si le signal **SIGINT** est reçu (c'est-à-dire que l'utilisateur a pressé les touches CTRL-C);
- Les messages en attente seront affichés après que l'utilisateur ait envoyé son message;
- Lorsque plus de 4096 octets seront en attente d'être affichés.

Par ailleurs, nous considérons que l'utilisateur ne peut pas utiliser de ponctuation dans le choix du pseudonyme.

Pour pouvoir faire la différence entre les pseudonymes des paramètres donnés au programme, nous avons implémenté les fonctions *checkParseArgv* et *parseUsernames*. Le premier consiste à parcourir les paramètres donnés au programme et retourne ce message d'erreur si aucun n'est donné : *chat pseudo_utilisateur pseudo_destinataire [–bot] [–manuel]*. Si des paramètres sont donnés au programme, nous les parcourons à l'aide d'une boucle for et utilisons la deuxième fonction. Cette dernière consiste à comparer les deux premiers paramètres et les chaines de caractère *bot* et *manuel*.

Listing 1: parseUsernames

footnotesize

```
void parseUsernames(char* argv[], int argv_index,
                        const char** user1, const char** user2,
                                const char* special_name) {
        if (special_name != NULL) {
                if (argv_index == 1) {
                        *user1 = special_name;
                }
                else {
                        *user2 = special_name;
                }
        }
        else {
                if (argv_index == 1) {
                        *user1 = argv[argv_index];
                }
```

```
                    else {
                    *user2 = argv[argv_index];
                    }
                }
    }
```

Listing 2: checkParseArgv

footnotesize

```
    int checkParseArgv(int argc, char* argv[], bool* bot_mode,
                       bool* manual_mode, const char** user1, const char** user2) {
if (1 == argc) {
   fprintf(stderr, "chat pseudo_utilisateur pseudo_destinataire [--bot] [--manuel]\n");
   return 1;
}
else {
   for (int i = 1; i < 3; i++) {
      // On ne regarde que les deux premiers params. argv[1] et argv[2] correspondant a
      if (strlen(argv[i]) > MAX_LENGHT_USERNAME) {
         fprintf(stderr, "Error : the maximum length of usernames is 30.\n");
         return 2;
      }
      // CHECK IF --PARAM IN ARGV[1] OR ARGV[2] SHOULD RISE AN ERROR OR BE CONSIDERED A
      else if (strcmp(argv[i], "--bot") == 0) {
         parseUsernames(argv, i, user1, user2, "Bot");
      }
      else if (strcmp(argv[i], "--manuel") == 0) {
         parseUsernames(argv, i, user1, user2, "Manuel");
      }
      else {
         for (size_t j = 0; j < strlen(argv[i]); j++) {
         // On parcourt chaque caractere j de argv[i] avec argv[i][j]
         // On appelle ispunct() pour verifier s'il y a un caractere de ponctuation dan
         // size_t pour la variable j == type de retour de strlen() (long unsigned int)
            if (ispunct((unsigned char)argv[i][j])) {
               fprintf(stderr, "Error : punctuation characters are forbidden for usernan
               return 3;
            }
         }
         parseUsernames(argv, i, user1, user2, NULL);
      }
   }
   if (argc == 4) {
      if (strcmp(argv[3], BOT_MODE) == 0) {
         *bot_mode = true;
      }
      else if (strcmp(argv[3], MANUAL_MODE) == 0) {
         *manual_mode = true;
      }
   }
   else if (argc == 5) {
      if (strcmp(argv[3], BOT_MODE) == 0 || strcmp(argv[4], BOT_MODE) == 0) {
```

```
            *bot_mode = true;
        }
        if (strcmp(argv[3], MANUAL_MODE) == 0 || strcmp(argv[4], MANUAL_MODE) == 0) {
            *manual_mode = true;
        }
    }
  }
  return 0;
}
```

2.2 Processus et pipes nommés

Le programme utilisera deux processus pour que les utilisateurs puissent communiquer de manière simultanée. Le processus père sera responsable de la lecture des messages sur l'entrée standard et de l'écrire sur le pipe. Quant au processus fils, il lira sur le pipes et s'occupera d'afficher les messages envoyé par l'utilisateur.

Ces pipes nommés seront initialisés grâce à la fonction *mkfifo()* et le chemin d'accès doit être */tmp/WRITER-READER.chat* où

- *WRITER* correspond à la personne qui a envoyé le message;
- *READER* correspond à la personne qui reçoit le message.

Comme nous ne connaissons pas les pseudonymes des utilisateurs à l'avance et qu'on ne peut pas concaténer des chaines de caractère en C, nous avons utiliser la fonction *snprintf*. Nous pourrions utiliser la fonction *sprintf* mais l'avantage de la première fonction est que nous pouvons limiter une taille de buffer. De ce fait, nous n'aurons jamais l'erreur *buffer overflows*. S'il y a tout de même un excès, la fonction tronquera la partie en trop.

2.3 Mémoire partagée

en attente

2.4 Gestion des signaux et fin de programme

en attente

3 Programme en bash : chat-bot

3.1 Paramètres

Pour cette partie du projet, les paramètres donnés au programme sont le pseudonyme de la personne qui discute avec le robot et le pseudonyme utilisé par le robot (si deuxième paramètre n'est pas fourni, le nom par défaut du robot sera *bot*)

Si aucun paramètre n'est donné au programme, alors le programme retournera un message d'erreur.

Listing 3: Gestion de paramètre

```
ARGC=$#
destinataire=$1
bot=${2-'bot'}
if [ $ARGC != 2 ]; then
    echo "chat-bot destinataire [pseudo]" >&2
```

```
exit 1
```

Nous pouvons noter que $ARGC$ contiendra le nombre de paramètre donné. De plus, la dernière ligne consiste à rediriger vers la sortie standard *stderr*(car le file descriptor correspondant est 2)

3.2 Interface

en attente

3.3 Commandes

en attente