TP 4 : Communication par tubes

Encadrant : R. Aparicio-Pardo
Author TP : Etienne Lozes

11 décembre 2017

1 Introduction

1.1 Présentation

Le but de ce TP est de faire un petit logiciel de déplacement de curseur sur un terminal. Le logiciel fait intervenir trois modules :

- un gestionnaire de clavier qui traduit les pressions clavier retenues en instruction de mouvement,
- un gestionnaire d'affichage qui exécute des instructions d'affichage,
- un "court-circuit" qui convertit les instructions de mouvement en instruction d'affichage

Dans un premier temps, ces trois modules sont implémentés par des procédures qui s'exécutent séquentiellement au sein d'un seul processus. Ensuite, on implémente chaque module par des processus séparés qui communiquent par des tubes nommés. Enfin, on crée un lanceur unique qui crée les tubes (anonymes) et lance les trois processus.

1.2 Recommandations d'usage

Ce sujet fait intervenir plusieurs fichiers C, fait appel à la bibliothèque `ncurses` qu'il faut lier au moment de l'édition de lien en ajoutant l'option `-lncurses` à `gcc`. Par ailleurs, on recommande d'utiliser l'option `-Wall` pour afficher tous les avertissements de `gcc` (idéalement, on ne doit avoir aucun warning !).

Important : Pensez à garder une copie d'une version aboutie à chaque étape, la prochaine séance on reprendra du code que vous aurez écrit aujourd'hui dans les premières étapes.

2 Échauffement : le jeu simple

On s'intéresse tout d'abord à programmer le jeu simple où les trois modules sont implémentés au sein d'un même programme par trois procédures différentes. Il s'agit maintenant de comprendre comment gérer le clavier et l'écran, ce qui fait appel à la bibliothèque `curses`, et sous quel format doivent communiquer les trois modules.

2.1 La bibliothèque curses

Pour comprendre comment utiliser la bibliothèque curses, récupérer le fichier `http://www.i3s.unice.fr/~raparicio/teaching/sys/?dir=TP4/src`. Le programme comporte diverses instructions d'initialisation, puis une boucle qui lit la touche pressée, met à jour les variables de coordonnées, et actualise l'affichage.

2.2 La modularisation

Il s'agit maintenant d'adapter ce programme pour le rendre plus modulaire. Le programme comportera désormais 3 fonctions :

- une fonction *gestionnaire de clavier* `char gc(int key)` qui traduit la touche pressée en une instruction de mouvement correspondant, codée sur une lettre : 'u', 'd', 'l', 'r', ou 'q' ;
- une fonction *gestionnaire d'affichage* `void ga(char instr[4])` qui reçoit une instruction d'affichage de la forme | w | X | Y | C | ou l'instruction | q | u | i | t |, et exécute cette instruction ;
- une fonction *court-circuit* `void cc(char c,char instr[4])` qui calcule les instructions d'affichage en fonction de la touche pressée.

Remarque : ici, X et Y sont les caractères dont le code ASCII correspond à l'entier (qui peut prendre des valeurs > 9). Il faut donc utiliser une coercion à un moment donné. Pour faire des tests "lisibles", surtout par la suite, vous pouvez décaler ce code ASCII vers une partie lisible du code ASCII, par exemple `(char) x + '0'` pour avoir 0 sur '0' etc. Mais il faut faire le décalage des deux côtés (gc et cc).

Etape 1 : *Réécrivez le jeu précédent en utilisant le découpage suggéré. Le programme marche toujours bien ? On continue...*

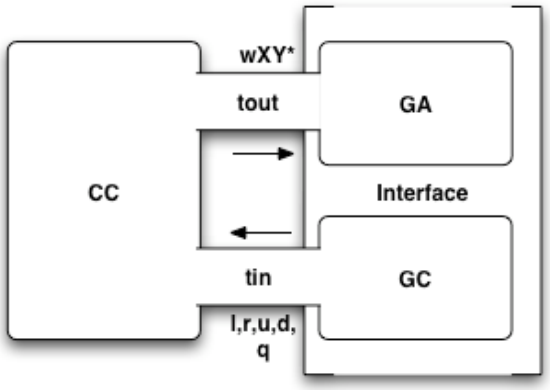
3 Communication par tubes nommés

On veut maintenant découper le jeu en plusieurs processus, chaque processus implémentant un des modules précédents. Les processus communiquent par deux tubes nommés, `tin`, partagé par `gc` et `cc`, et `tout` partagé par `ga` et `cc`. Pour le moment, `tin` et `tout` sont créés par l'utilisateur au niveau système (`mkfifo tin`), et chaque programme est lancé manuellement dans un terminal différent. En conséquence, `gc` et `ga` doivent chacun initialiser la bibliothèque `ncurses`. De plus, en vue de la suite, on conviendra que `gc` et `ga` connaissent les tubes `tin` et `tout` et les ouvrent directement, tandis que `cc` lit et écrit sur ses i/o standards (comme `cat`).

Etape 2 *Faire trois programmes à partir de votre code. Pour tester vos programmes, lancer dans trois terminaux différents* `./gc`, `./ga`, *et* `./cc <tin >tout` *(en ayant créé les tubes nommés depuis le shell).*

4 Encapsulation de l'interface

La méthode précédente présente le défaut d'utiliser trois terminaux. On veut maintenant améliorer cela en encapsulant les deux programmes `gc` et `ga` dans un même programme `interface` qui initialise la bibliothèque `ncurse` (ce n'est donc plus la tâche de `gc` et `ga`), et appelle les deux sous-programme `gc` et `ga`. Ceci aura aussi l'intérêt de faciliter les redirections [1].



On a déjà vu comment un processus peut en lancer un autre : la commande fork() crée deux copies du même programme, que l'on peut distinguer comme suit.

```
if (fork()){ /* suite du premier processus}
else { /*suite du second *}
/* code exécutable par les deux */
/* a proscrire : on ne doit pas sortir des blocs du if */
}
```

1. En effet, si vous essayez de lire et écrire sur les descripteurs standards dans `gc` et `ga`, et de faire des redirections en les appellant depuis le shell (ex : gc | cc | ga), vous observerez que cela ne marche plus : la raison pourrait être que `ncurses` a besoin de l'entrée standard reliée au terminal pour fonctionner correctement.

Etape 3 *Remettez ensemble le code de* `gc` *et* `ga` *dans un seul programme* `interface`. *L'initialisation de curses se fait au début, puis l'appel à fork lance les routines* `maingc` *et* `mainga` *correspondant (la partie initscr() en moins) aux mains des programmes précédents. Testez votre programme depuis le shell :* `./cc <tin >tout &` *puis* `./interface`.

5 Communication par tubes anonymes

Comme vous avez pu le constater, la solution du programme interface est un peu meilleure mais permet à un joueur extérieur de venir perturber votre partie (comment ?). Par ailleurs, il faut encore lancer le `cc` "à la main". On veut maintenant pouvoir jouer en privé et enlever le lancement du `cc`.

Pour cela, on rajoute au tout début d' interface la création de deux tubes anonymes (par appel à `pipe`), on lance cc avec les redirections d'entrée et sortie standard (voir exemple du cours). Il faut aussi changer les descripteurs utilisés par `gc` et `ga`, qui doivent maintenant travailler avec les tubes anonymes.

Etape 4 *Mettez dans un même programme "jeu" les maingc, mainga et maincc correspondant aux anciens main. Le main de jeu doit d'abord initialiser le terminal, créer les tubes, puis lancer les trois modules par deux appels à fork, en faisant les redirections.*