M2 FEADéP – Systèmes et réseaux
TP3 Communication entre processus – mmap

Nicolas Chappe
École Normale Supérieure de Lyon

1 Introduction

`mmap` permet d'associer un fichier du disque à une zone mémoire, et de partager cette zone mémoire entre plusieurs processus.

Exercice 1

On s'intéresse ici au fonctionnement d'un serveur d'affichage (par exemple, un compositeur Wayland). Il s'agit d'un processus qui gère de manière centralisée un certain nombre de choses en lien avec l'affichage d'un environnement graphique de bureau. Cela implique diverses communications avec les différents processus des fenêtres ouvertes par l'utilisateur.

Pour chacune des situations suivantes, indiquer en justifiant quelle méthode de communication inter-processus vous paraît la plus appropriée entre les fenêtres clientes et le serveur d'affichage, soit de la mémoire partagée avec mmap, soit un canal de communication avec des tubes ou sockets.

1. La récupération par les fenêtres de l'information qu'un clic de souris a eu lieu.
2. La récupération par les fenêtres de l'image du curseur de souris.
3. L'envoi au serveur des pixels d'une fenêtre prête à être affichée.

2 Discussion instantanée avec `mmap`

Le fichier `chat.c` fourni contient un début d'outil de discussion local fondé sur `mmap`.

La structure `Message` contient la date d'envoi d'un message (`time_t` est un entier, cf. `man 2 time`), le nom de l'expéditeur et le contenu du message.

La structure `Historique` contient le nombre total de messages envoyés (`compte`) et les 16 derniers messages envoyés, stockés dans un *buffer circulaire*. En pratique, le n-ième message est stocké à l'indice `n % 16` du tableau, écrasant ainsi les messages plus anciens. La structure `Historique` est destinée à être stockée en mémoire partagée, afin que chacun des participants à la conversation puisse écrire des messages et les rendre visible à tous immédiatement à

```
Entrez votre nom : Nicolas
(e) écrire (r) rafraîchir (q) quitter r
Pas de nouveau message
(e) écrire (r) rafraîchir (q) quitter e
Entrez votre message : bonjour
[Wed Jan 26 10:15:00 2022] @Nicolas: bonjour
(e) écrire (r) rafraîchir (q) quitter r
[Wed Jan 26 10:18:00 2022] @Eddy: hello
(e) écrire (r) rafraîchir (q) quitter q
```

FIGURE 1 – Exemple de sortie possible

faible coût. `Historique` contient donc un mutex, qui est verrouillé lors de l'écriture d'un message par un processus.

Exercice 2

L'objectif de cet exercice est de compléter le programme pour qu'il propose deux options à l'utilisateur :

- Rafraîchir son état, en vérifiant dans la mémoire partagée s'il y a eu des nouveaux messages, et si c'est le cas les afficher.
- Écrire un message, et l'insérer dans le buffer circulaire de la mémoire partagée.

Ces deux fonctionnalités constitueront la boucle principale du programme, et pourront être implantées dans des fonctions auxiliaires.

Au-delà de la boucle principale, des indications sur les différentes étapes d'initialisation du programme sont données dans le squelette de code.

Dans un premier temps, vous pourrez ignorer l'initialisation et le verrouillage de la mutex.

Manipulation de dates. Voir `man 2 time` pour la récupération d'un `time_t` et `man ctime` pour son affichage.

Note. Cet exercice peut s'effectuer entièrement sans malloc ni création de thread.

Exercice 3

Nous allons maintenant compléter notre code pour y intégrer le rafraîchissement en temps réel des messages, sans avoir besoin d'action de l'utilisateur.

Par souci de simplicité, nous le ferons dans un premier temps avec attente active. L'idée est d'avoir un premier thread qui se charge de surveiller régulièrement les nouveaux mes-

sages, par exemple toutes les 5 secondes, et un autre thread qui se charge de gérer l'écriture de messages par l'utilisateur.

Exercice 4

Les primitives de synchronisation que nous avons vues jusqu'à maintenant ne permettent pas simplement de nous séparer de l'attente active introduite dans l'exercice précédent. Nous allons utiliser une *variable de condition* pour surveiller l'apparition de nouveaux messages en mémoire partagée. Une variable de condition `pthread_cond_t` propose trois opérations :

- `int pthread_cond_signal(pthread_cond_t *cond)` réveille un des threads qui attend la variable de condition.
- `int pthread_cond_broadcast(pthread_cond_t *cond)` réveille tous les threads qui attendent la variable de condition.
- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)` déverrouille la mutex en argument (supposée verrouillée), met en pause le thread, et au réveil reverrouille la mutex.

Ajoutez une variable de condition à `Historique`, et utilisez broadcast et wait pour notifier les threads de l'existence d'un nouveau message.

Comme pour la mutex, un appel à `pthread_condattr_setpshared` sera nécessaire durant l'initialisation de la variable de condition.