HMIN105M TD-TP
IPC : files de messages
Durée moyenne : une séance de 3h

1 Gestion des messages

On veut mettre en place une application composée de deux programmes : le premier, appelé *client*, génère et envoie des requêtes au second, appelé *calculatrice*. La calculatrice résout les requêtes et renvoie les réponses au client. Les requêtes sont des opérations mathématiques simples à effectuer (par exemple : $7 * 5 =?$). Les deux programmes (client et calculatrice) communiquent à l'aide d'une **file de message**.

1. Décrire comment mettre en place la file de message et le protocole de communication entre le client et la calculatrice.
2. Ecrire le schéma algorithmique du client et de la calculatrice. Dans ce schéma, vous devez définir la structure des requêtes et celle des réponses. Les requêtes possibles sont les opérations $+, -, *$ et / avec deux opérandes.
3. Modifier le schéma algorithmique précédent (toujours en utilisant une seule file de messages) pour que la calculatrice puisse répondre aux requêtes de différents clients.
4. Modifier le schéma algorithmique proposé pour qu'il y ait autant de calculatrices que de types de requêtes, c'est-à-dire, une calculatrice pour répondre aux requêtes +, une autre pour répondre aux requêtes −, etc. Remarque : utilisez toujours une seule file de messages.
5. Dans l'ordre, implémenter les 3 schémas algorithmiques précédents. Lors de l'implémentation, la fonction **fork()** n'est pas utile.
6. A l'exécution, mettre en œuvre les situations suivantes :
   - Exécuter votre application de manière à constater l'existence de la file hors de la vie des processus engendrés (avec la commande `ipcs`).
   - Déposer des messages dans la file jusqu'à constater le blocage lorsque la file est pleine.
   - Exécuter l'application de manière à constater un blocage lorsque la file est vide.
   - Dans une situation où la file existe, modifier ou supprimer et recréer le fichier de calcul de la clé. Relancer des processus clients. Que se passe-t-il ?

2 Décomposition parallèle

Il s'agit de la décomposition de nombres en facteurs premiers, en utilisant une file de messages et un nombre quelconque de processus partageant la file et participant à cette décomposition.

Le déroulement demandé est le suivant :

1. Un processus *parent* dépose un entier n quelconque dans une file de messages. Des processus *enfants*, engendrés par *parent* et appelés *décomposeurs* vont prendre en charge la décomposition et la restitution des résultats. Le nombre d'*enfants* devra être paramétré.
2. Chaque décomposeur dépose dans la file soit les deux facteurs qu'il a trouvés, soit un seul si la décomposition s'arrête là (un facteur premier a été trouvé). Les deux facteurs déposés devront être à nouveau décomposés, etc. ce qui doit aboutir à plusieurs décompositions parallèles (dans le cas de plusieurs enfants).
3. Le *parent* récupère les résultats (les facteurs premiers seulement) au fur et à mesure de leurs arrivées. Il multiplie ces facteurs entre eux pour savoir si la décomposition est terminée (le produit est égal au nombre de départ).
4. Le *parent* peut réinjecter des nouveaux nombres à décomposer ou signaler par un message spécial la fin aux *enfants*. Un message de fin par *enfant* est conseillé, pourquoi ?

3 (Exercice optionnel : File de messages pour la synchronisation (sur papier)

L'objectif de cet exercice est d'utiliser une file de messages pour mettre en place l'exclusion mutuelle, sans introduire un autre objet IPC.

Dans une application, un programme P_{file} doit gérer une ressource commune à plusieurs programmes P_i ($1 \leq i \leq N$), par exemple un accès à un espace mémoire commun. Ainsi, lorsqu'un programme P_i souhaite accéder à cette ressource, il doit passer par une demande adressée à P_{file}. A chaque instant, un et un seul programme P_i peut disposer de la ressource : il peut modifier le contenu de l'espace mémoire et le passage par P_{file} doit garantir l'exclusivité d'accès.

Le rôle de P_{file} est :
- de recevoir des requêtes demandant l'accès,
- d'annoncer au premier demandeur que la ressource est disponible,
- de prendre connaissance de la libération de la ressource pour pouvoir l'annoncer à nouveau comme disponible.

On suppose dans un premier temps un fonctionnement sans pannes, c'est-à-dire que tout programme P_i accédant à la ressource annonce bien sa fin d'exécution et relâche correctement la ressource.

Pour mettre en place l'exclusion d'accès à la ressource commune, l'objectif est d'utiliser une file de messages. Pour la ressource, on suppose qu'on dispose d'un accès à une mémoire partagée.

1. Décrire le fonctionnement général du système.
2. Comment est matérialisée la file d'attente pour l'accès à la ressource commune ?
3. Que fait P_{file} pour la prise en compte des requêtes ? A-t-il besoin d'entrées-sorties non bloquantes ? Comment est-ce que P_{file} prend connaissance de la fin d'utilisation de la ressource par un programme P_i ?

Allons un peu plus loin :

1. On admet maintenant qu'il peut y avoir des pannes. Que se passe-t-il si un P_i disparait ? En particulier, peut-on revenir à une situation correcte ? Pour ce faire, étudier les situations possibles d'un P_i et dire face à chaque situation ce qui peut être fait.
2. Si on ne dispose pas du concept de mémoire partagée, quels choix ferez-vous pour mettre en place un tel partage et satisfaire les requêtes ?