

Modèle d'un système de producteurs - consommateurs

TP de programmation concurrente

V. Pilloux

Objectif pédagogique

Développer un système de communication générique et personnalisé (queues) qui puisse servir à un système de producteurs – consommateurs concurrents.

Problème 1

On aimerait implémenter un système de gestion de queues qui facilite la communication entre producteurs et consommateurs, soit un ensemble de fonctions thread-safe qui gère le buffer central de la figure 1, puisque celui-ci sera sollicité par plusieurs tâches exécutées en parallèle. Le tampon est de taille limitée et peut contenir au maximum 10 éléments.

La tentative de lecture d'un buffer vide ou la tentative d'écriture dans un buffer plein entraîne une attente passive (jusqu'à ce que les conditions changent).

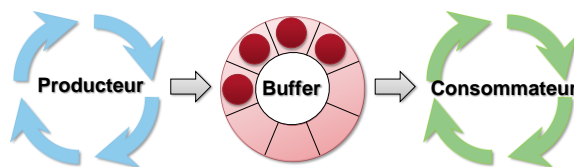


Figure 1

Pour tester ce système, un ensemble composé de 7 tâches A à G qui ont chacune un temps de travail variable (et concurrent) devra être créé. Ce système aura les dépendances présentées sur la figure 2.

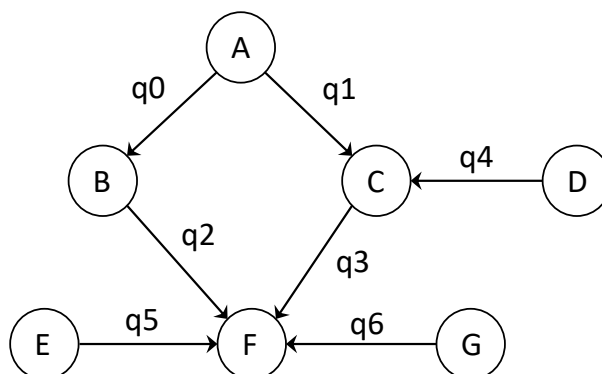


Figure 2

Chaque flèche représente la transmission d'une donnée ou d'un bloc de données selon la table 1.

Nom du label sur la figure 2	Données
q0	Un entier (int)
q1	Un entier (int)
q2	Un vecteur de 5 entiers (int)
q3	Un entier (int)
q4	Une chaîne de 35 caractères (maximum), \0 de fin de chaîne compris
q5	Un entier (int)
q6	Un entier (int)

Table 1

La teneur des données n'a pas d'importance (vous pouvez la choisir), mais il faut que celle-ci change à chaque envoi. Il est néanmoins conseillé d'incrémenter ces valeurs pour tester le système.

Toutes les tâches devront effectuer les actions suivantes en boucle tant que la fin du programme n'est pas déclarée:

- Attendre la réception de données (pour les tâches dont les flèches pointent vers elles uniquement). Afficher les données reçues.
- Attendre aléatoirement entre 1 et 11 μ s pour simuler un travail quelconque.
- Préparer puis émettre une donnée ou un bloc de données et afficher l'envoi de ces données.

La tâche F reçoit 1000 blocs de données respectivement des tâches B, C, E et G puis déclenche la fin du programme, détruit toutes les queues créées et sort. Un bloc de données correspond à la description de la Table 1.

Guide de développement

1. Décompressez l'archive TP4_prod_cons_etu.zip. Celle-ci contient un fichier pour la gestion des queues (**queue_mgt.c**) et un autre pour le `main()` et la déclaration des tâches (**prod_cons.c**).
2. Implémentez 4 fonctions de gestion de queue dans le fichier **queue_mgt.c**. Leur prototype et la fonctionnalité demandée est décrite dans **queue_mgt.h**. Notez que la fonction `destroy_queue()` doit libérer les fonction `send()` et `receive()` qui seraient en attente.
3. Testez la fonctionnalité de 2. en créant 2 thread : l'un qui émet et l'autre qui reçoit une information plusieurs fois de suite.
4. Implémentez toutes les tâches de la figure 1. Vérifiez que votre programme parvient toujours à la fin, avec ou sans délais.

Contraintes de développement

- **Les queues Posix ne doivent pas être utilisées (c'est-à-dire `mqd_t`)**
- Aucune tâche ne doit sortir avec `pthread_cancel()`, `pthread_exit()` ou `exit()`.
- **Toutes les attentes doivent être passives**
- Votre programme doit aussi pouvoir fonctionner en ôtant les attentes (du type `usleep()`).
- Rendez les fonctions `init_queue()`, `destroy_queue()`, `send()` et `receive()` robustes, c'est-à-dire testez les différents cas d'erreur et faites en sorte qu'on ne puisse exécuter le code des 3 dernière fonctions que si l'initialisation de la queue a été faite correctement.
- Propreté du code : voir les « consignes pour l'écriture de code C » données sur Cyberlearn. Cependant les variables globales **nécessaires** sont permises dans ce TP.

Problème 2

On aimerait maintenant créer un système de queues qui ressemble à celui de Posix en rajoutant une notion simple de priorité lors de l'envoi d'un message. Dans un sous-répertoire appelé **prod_cons_prio**, produisez un code contenant seulement 2 threads concurrents, l'un étant le producteur, l'autre le consommateur. Le producteur émet des nombres de 0 à 49 au consommateur par l'intermédiaire d'une queue (celle du problème 1) à laquelle on a ajouté un paramètre `bool sent_to_front`.

Lorsque la valeur vaut `true`, le nouvel élément est inséré en premier dans le tampon de la queue, au lieu de se placer en dernier. Sinon c'est le mode FIFO standard qui doit être utilisé.

Vérifiez le fonctionnement de ce nouveau programme en affichant les valeurs émises et reçues par les 2 threads.

Travail à rendre

Une seule archive nommée **G<numéro de votre groupe>prod_cons.zip** contenant vos fichiers sources et les makefile permettant de compiler vos deux programmes sans warning et sans erreur (le 2^e se trouvant dans le sous-répertoire **prod_cons_prio**).

Bon travail !