TP 7 : Multiplexage des communications réseau
Implémentation avec MX

Alexandre DENIS
denis@labri.fr

Systèmes et supports d'exécution — 2005–2006

L'objectif de ce TP est de d'implémenter un mécanisme de *multiplexage* des communications réseau permettant à plusieurs *threads* de communicaquer simultanément sur un seul canal partagé.

Ce TP est **noté** et à rendre (code source + compte-rendu) par e-mail au plus tard le **11 janvier 2005**.

Le réseau Myrinet, le pilote MX

Ce TP a lieu en salle 07, sur les machines équipées de cartes réseau *Myrinet*. Pour la configuration de MX, reportez-vous au TP 6.

Note : il vous sera confié un CD *Knoppix* et une clef USB, à rendre en même temps que votre travail. Ce CD peut théoriquement démarrer sur n'importe quel PC, y compris votre machine personnelle, et permettre de compiler votre code. Cependant, il va de soi que vous ne pourrez pas démarrer un code utilisant MX sur une machine qui n'est pas équipée de carte *Myrinet* !

Un exemple de code d'initialisation de MX vous est fourni sur la page web du cours SSECPD.

1 Benchmark de référence

Nous commençons par écrire un programme de benchmark qui servira de référence dans la suite.

Le benchmark doit avoir les propriétés suivantes :
- être en *ping-pong* (alternativement un message dans un sens, un message dans l'autre sens). Pourquoi est-ce meilleur qu'une succession d'envois dans le même sens ? Que risquerait-on de mesurer sinon ?
- mesurer les performances pour des messages de taille variable — faire les puissances de 2 en ordre croissant, de 4 octets à 4 Mo.
- réaliser une *moyenne* de plusieurs allers-retours pour chaque taille, pour une meilleure précision. Choisissez et justifiez le nombre d'allers-retours qui vous semble pertinent.

Tracer ensuite la courbe de débit et la courbe de temps de transfert en fonction de la taille des messages à l'aide de `gnuplot` (ou de tout autre outil de tracé de courbes). Pour la courbe de débit, choisissez une échelle logarithmique en x, pour la courbe de temps une

échelle logarithmique en x et en y. N'hésitez pas à zoomer uniquement sur les petites tailles de message pour la courbe de temps.

2 Multiplexage

2.1 Communications avec plusieurs *threads*

On désire maintenant faire communiquer plusieurs *threads* (`pthread`) en même temps.
- Réalisez un petit programme qui fait un ping-pong simple avec un jeton de taille déterminée.
- Ajoutez un deuxième *thread* qui fait circuler son propre jeton. Que se passe-t-il si les deux jetons ne font pas la même taille ? Essayez, pour voir ! Comment expliquez-vous le résultat obtenu ?
- Réfléchissez à un mécanisme qui permettrait d'assurer que chaque *thread* reçoit son jeton et pas le jeton destiné à l'autre *thread*.

L'idée naturelle serait d'utiliser les fonctionnalités de matching *de* MX*. Cependant, le but de ce TP étant d'implémenter vous-mêmes un mécanisme de multiplexage, il vous est demandé de ne* pas *utiliser le matching* MX *pour multiplexer. Il faut noter que* MX *est le seul pilote pour réseau haute performance à inclure en standard le mécanisme de matching.*

2.2 Multiplexage sur MX

Nous allons écrire une surcouche basée sur MX que l'application utilisera à la place de MX. Cette surcouche offrira un certain nombre de canaux logiques. Nous proposons d'écrire les fonctions suivantes :

`mux_init(...)`
Fonction d'initialisation appelée au début du programme (devra appeler `mx_init` et réaliser diverses initialisations).

`mux_end()`
Fonction de terminaison qui doit être appelée par le programme avant sa terminaison.

`mux_send(int size, const void*data, ...)`
Envoie un message sur le canal spécifié.

`mux_recv(int size, void*data, ...)`
Reçoit un message sur le canal spécifié.

1. Réfléchissez aux prototypes complets des fonctions proposées ci-dessus. Vous pouvez en ajouter d'autres si nécessaire.
2. Proposez une implémentation de ces fonctions sans vous soucier des performances pour l'instant.
3. Qui doit faire le `mx_irecv` ? Quels sont les choix possibles ? Quel choix avez-vous fait ? Pourquoi ?
4. Proposez une nouvelle version de votre programme du 2.1 qui utilise ces nouvelles fonctions plutôt que MX. Vérifiez que, cette fois, les jetons ne sont plus mélangés !

2.3 Performances et optimisation du multiplexage

Nous nous soucions maintenant du surcoût introduit par le multiplexage.

1. Quelle est l'impact de votre version de multiplexage sur les performances ? (copies éventuelles, en-têtes, ...)
2. Quelle optimisation proposez-vous ? Est-ce possible en gardant la même interface de programmation ou faut-il la modifier ? Justifiez.
3. Modifiez votre implémentation pour prendre en compte vos optimisations, ré-essayez le benchmark, et comparez aux résultats obtenus sans multiplexage. Quel est le surcoût, au final ?
4. Le multiplexage introduit-il forcément un surcoût ou est-il possible de le réaliser « quasi-gratuitement » ? Justifiez.