TP 6 : Multiplexage des communications réseau
Implémentation avec *Infiniband*

Alexandre DENIS
denis@labri.fr

Systèmes et supports d'exécution — 2006–2007

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

Ce TP est **noté** et à rendre (code source + compte-rendu) par e-mail au plus tard le vendredi **12 janvier 2007**.

1 Implémentation de référence sur *Infiniband*

Nous commençons par écrire un protocole de communication sans multiplexage sur *Infiniband*, qui servira de référence dans la suite.

Téléchargez le nouveau squelette de programme sur la page du cours. Ce code correspond au 2.1.3 du TP 5.

1.1 Taille variable bornée

Modifiez le code de façon à ce que des messages de tailles différentes puissent être échangés. Vous pourrez supposer pour l'instant que les messages font moins de 8 ko.

1.2 Benchmark

Modifier le programme de façon à envoyer des messages de taille croissante (les puissances de 2 par exemple), et mesurer le temps de transfert pour chacune des tailles. Tracer ensuite les courbes de latence et de débit en fonction de la taille des messages à l'aide de `gnuplot`. 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 tailles de message qui vous semblent pertinentes.

1.3 Taille non-bornée

Ajoutez à votre code la prise en charge de messages de taille non-bornée, qui peuvent nécessiter une fragmentation en plusieurs envois sur le réseau. Pourquoi ne peut-on pas les envoyer directement sans fragmentation ? Expérimentez. Recommencez votre benchmark avec des messages jusque 4 Mo. Conclusion ?

Indication : il vous sera nécessaire d'implémenter un contrôle de flux, c'est-à-dire que le destinataire doit prévenir l'émetteur qu'il a reçu et « consommé » le paquet avant que l'émetteur puisse envoyer le paquet suivant. Dans le cas contraire, on risquerait d'écraser un paquet que le récepteur n'aurait pas encore consommé.

1.4 Double buffering

1. Le système de fragmentation précédent présente des performance assez faibles. Pourquoi ?
   Une optimisation classique consiste en un *double buffering*. Il s'agit de disposer de deux buffers d'émission : la carte *Infiniband* en envoie un pendant que le CPU prépare le suivant, en *pipeline*.
2. Implémentez un système de double buffering en émission et évaluez son impact sur les performances.
3. Implémentez le même système du côté récepteur, sans oubliez d'adapter le mécanisme de contrôle de flux en conséquence.

2 Multiplexage et interactions *threads*–réseau

On désire maintenant faire communiquer plusieurs *threads* (`pthread`) en même temps. Si l'on n'y prend pas garde, les *threads* peuvent recevoir un message destiné à un autre *thread*.

La solution est appelée *multiplexage* : on ajoute une étiquette (*tag*) ou encore « clef de multiplexage » à chaque message émis pour réaliser des canaux logiques.

1. Ajoutez un *tag* (entier) à chaque message émis.
2. Ajoutez la prise en charge des *threads* (verrouillage) et la possibilité de filtrer les tags en réception. Attention : un *thread* ne sait pas *a priori* si un message lui est destiné avant sa réception ! Quel *thread* effectue la scrutation de la boîte aux lettres ?
3. Transformer votre benchmark pour que deux *threads* effectuent un ping-pong en parallèle, chacun avec son propre *tag*.
4. Comparer les performances de votre code avec multiplexage et sans multiplexage dans le cas d'un seul *thread*. Quel est l'impact de votre implémentation du multiplexage sur les performances ? (copies éventuelles, en-têtes, ...)
5. Mesurer et commentez les performances de votre code quand deux *threads* exécutent le benchmark. Donnez (sans l'implémenter) les grandes lignes d'un système qui garantirait une meilleure performance et une meilleure équité.

3 Optimisations *(bonus)*

Nous cherchons maintenant à améliorer la latence de notre protocole.

1. Quel est le nombre de paquets émis sur le réseau pour chaque message ?
2. Revoyez la disposition en mémoire dans la MR pour pouvoir effectuer du *piggy backing* et réduire le nombre de paquet émis à un seul par message quand la fragmentation n'est pas nécessaire.
3. Au final, quel est le surcoût du multiplexage ?