Systèmes et supports d'exécution pour le calcul parallèle et distribué

Université de Bordeaux 1 — Master 2 SDRP, 2005 – 2006

Partiel

Alexandre DENIS

23 novembre 2005

Durée : 2 h. **Documents interdits.**
Les exercices sont indépendants les uns des autres. Le barème est donné à titre indicatif. Les parties de code demandées doivent être rédigées en langage C.

1 Questions de cours : multi-threading et SMP (6 points)

1. Rappelez brièvement la définition et les propriétés essentielles de :
   (a) fibre/filament
   (b) ordonnanceur de *threads* coopératif
   (c) ordonnanceur de *threads* préemptif
   (d) ordonnanceurs de niveau noyau, utilisateur, hybride
2. Qu'est-ce qu'un processeur *dual core* (ou *multi core*) ? Quel type d'ordonnanceur de *threads* sont adaptés à leur programmation ?
3. Quel est le problème avec le placement des données sur une machine NUMA ? Est-il possible que le même programme soit plus lent sur une NUMA que sur une machine mono-processeur ?

2 Architecture des intergiciels (6 points)

1. Qu'appelle-t-on une construction d'intergiciel par abstraction ?
2. Qu'appelle-t-on une construction d'intergiciel par composants logiciels ?
3. Quels sont les avantages et les inconvénients de chacune de ces deux approches ?
4. *Globus XIO* est la bibliothèque de communication de l'intergiciel pour grilles *Globus*. *Globus XIO* est bâti autour d'une interface de programmation simple composée de quatre fonctions (`xio_open`, `xio_close`, `xio_read`, et `xio_write`) et repose sur deux types d'entités :
   – des *pilotes*, qui implémentent l'API *Globus XIO* sur diverses ressources physiques ;
   – des *filtres*, qui offrent l'API *Globus XIO* et utilisent cette même API *Globus XIO*.

Les entrées/sorties (fichiers, réseaux) réalisées avec *Globus XIO* se font donc au travers d'une pile XIO composée d'un nombre variable de *filtres* (éventuellement zéro) superposés au-dessus d'un pilote (toujours exactement un et un seul pilote, toujours en bas de pile). Les applications assemblent elles-mêmes la pile qu'elles désirent utiliser.

(a) Dans quelle catégorie d'intergiciel classeriez-vous *Globus XIO* ? Justifiez.

(b) Quels sont les avantages de cette approche ?

(c) Quelles sont les limitations de cette approche ? Donnez au moins un exemple de scénario difficile à faire rentrer dans l'architecture prévue par *Globus XIO*.

(d) Comment *Globus XIO* peut-il réussir à programmer des réseaux avec pour seul primitive d'ouverture `xio_open` ? Essayez de deviner le prototype de cette fonction et les implications sur l'implémentation.

3 Problème : synchronisation — boîte aux lettres (8 points)

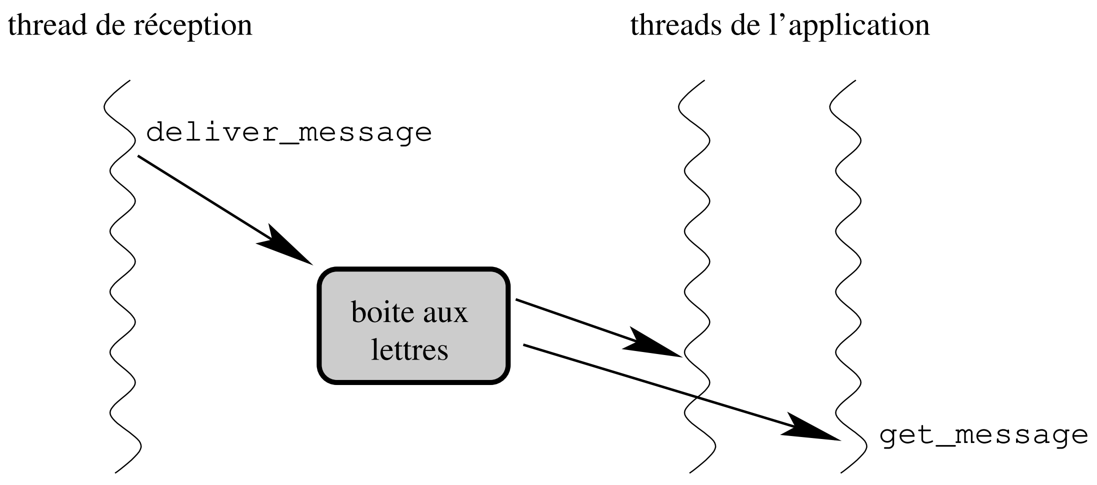
Nous nous plaçons dans le cadre d'un intergiciel qui, pour certaines raisons techniques, réalise toutes ses réceptions réseau dans le même *thread* interne à l'intergiciel ; l'application qui utilise cet intergiciel dispose de ses propres *threads*. L'objectif de cette partie est d'étudier le mécanisme de *boîte aux lettres* utilisé pour faire passer les données du *thread* de réception aux *threads* de l'application.

En pratique, le *thread* de réception est une boucle qui effectue la réception réseau puis appelle la fonction :

```
deliver_message(int size, void*buffer);
```

en passant en paramètre la taille du message reçu et un pointeur vers les données reçues. Les *threads* applicatifs récupèrent les données dans la boîte aux lettres en appelant la fonction :

```
get_message(int*size, void**buffer);
```



3.1 Boîte aux lettres simple

Dans un premier temps, nous supposons que la boîte aux lettres ne peut contenir qu'un seul message et qu'elle est toujours vide quand un message arrive depuis le réseau (*i.e.* quand `deliver_message` est appelée).

1. Écrire la structure de données représentant une boîte aux lettres.
2. Écrire les fonctions `deliver_message` et `get_message`.
3. Que se passe-t-il si plusieurs *threads* appellent `get_message` en même temps ?

3.2 Boîte aux lettres simple avec attente

Nous supposons maintenant qu'un message peut arriver sur le réseau alors que la boîte aux lettres n'est pas vide (*i.e.* `deliver_message` est appelé alors qu'aucun `get_message` n'a encore vidé la boîte aux lettres).

1. Comment gère-t-on ce cas ? Quel est l'impact éventuel sur les performances ?
2. Proposer une nouvelle version de votre code qui gère ce cas.

3.3 Boîte aux lettres complète

Nous nous souhaitons maintenant pouvoir placer plusieurs messages en attente dans la boîte aux lettres.

1. Quel est l'intérêt d'avoir une boîte aux lettre pouvant contenir plusieurs messages ?
2. Donner une idée des modifications à apporter à la version précédente (sans tout réécrire).

3.4 Zéro copie (bonus)

Les schémas précédents introduisent une copie en mémoire des données reçues. Dans la réalité, ces copies sont très couteuses. Il est possible de les éviter dans les cas où le `get_message` a lieu avant le `deliver_message`.

1. Quel sont les principaux problèmes pour pouvoir recevoir les données en zéro copie ?
2. Quelles modifications faut-il apporter au code de gestion des boîtes aux lettres et au *thread* de réception ?