Systèmes distribués
Master Informatique 1ère année

UNIVERSITÉ PARIS 13

Travaux Pratiques, séance 1

Modèle de communication par passage de messages

Question 1. Écrivez un programme *C* qui affiche *Bonjour !* à l'écran (sans oublier le retour chariot). Compilez-le et lancez-le.

Vous allez maintenant programmer avec *MPI*. Il s'agit d'une librairie qui peut être incluse dans un source *C* avec la commande suivante, placée au tout début du fichier source, avec les autres `#include` :

```
#include "mpi.h"
```

L'utilisation de cette librairie demande de placer certaines commandes d'initialisation au début de la fonction `main` :

```
MPI_Init(NULL,NULL);
```

Les deux arguments `NULL` peuvent être respectivement remplacés par `&argc` et `&argv` pourvu que que votre fonction `main` prenne les deux arguments `int argc` et `char **argv` (mais ce n'est pas une obligation). La commande `MPI_Init` déclare un *communicateur* appelé `MPI_COMM_WORLD` : il s'agit d'un contenant dans lequel se trouvent les différents processus. Le nombre de ces processus peut être obtenu avec la commande suivante :

```
MPI_Comm_size(MPI_COMM_WORLD,&nb_processus);
```

Ici `nb_processus` est une variable de type `int` qui doit être préalablement déclarée dans la fonction `main`. L'identifiant du processus courant peut être obtenu avec la commande :

```
MPI_Comm_rank(MPI_COMM_WORLD,&identifiant);
```

Si n processus ont été lancés, la variable `identifiant` également de type `int`, sera un entier entre 0 et $n-1$. À la fin de la fonction `main`, juste avant le `return`, on ferme MPI par la commande :

```
MPI_Finalize();
```

Question 2. Écrivez un programme C, utilisant MPI tel que chaque processus affiche son identifiant (sans oublier le retour chariot). Appelez votre source `helloworld.c`. Pour compiler, utilisez la commande :

```
mpicc -Wall helloworld.c -o helloworld
```

Pour exécuter votre programme distribué avec n processus, utilisez la commande :

```
mpirun -np n ./helloworld
```

Compilez et lancez votre programme avec 4 processus.

Question 3. La commande `MPI_Wtime()` renvoie un nombre de type `double` contenant le temps en secondes qui s'est écoulé depuis un instant fixe (mais, sans autre information, arbitraire). Utilisez cette commande pour mesurer le nombre de secondes qui s'est écoulé entre la fin de `MPI_Init` et `MPI_Finalize` pour chacun des processus. Affichez cette durée juste avant l'appel de `MPI_Finalize`. Compilez et lancez votre programme avec 4 processus.

Afin d'envoyer ou de recevoir des messages d'un processus à l'autre, on utilisera les commandes `MPI_Send` et `MPI_Recv` :

```
MPI_Send(buffer,k,MPI_Type,destination_id,balise,MPI_COMM_WORLD);
```

Ici, `buffer` est un pointeur sur les données à envoyer, `k` est le nombre de donées envoyées, `MPI_Type` est le type des données envoyées (pour des nombres entiers, `MPI_INT`, pour des nombres de type double, `MPI_DOUBLE`, ...). En outre, l'entier `destination_id` est l'identificateur du processus auquel les données doivent être envoyées et l'entier `balise` permet de différentier différents types de messages.

```
MPI_Recv(buffer,k,MPI_Type,origine_id,balise,MPI_COMM_WORLD,&status);
```

Ici, les trois premiers arguments et le quatrième sont de même type que pour la commande précedente. L'entier `origine_id` est l'identificateur du processus duquel les données sont reçues. L'argument `status` de type `MPI_Status` permet au système de donner des détails supplémentaires sur le message reçu.

Question 4. Écrivez, compilez et exécutez un programme distribué, fait de deux processus (et qui vérifie bien qu'il n'y a que deux processus), tel que chacun des processus envoie son identifiant à l'autre processus, et affiche le message reçu.

Question 5. Écrivez et exécutez un programme distribué, fait de deux processus (et qui vérifie bien qu'il n'y a que deux processus), tel que l'un des processus envoie un certain nombre (par exemple 100 pour l'un des processus et 200 pour l'autre) à l'autre processus, puis reçoit de lui un certain nombre k. L'autre processus recevra un nombre k du premier et le lui renverra. Lors de chaque réception et de chaque envoi, chaque processus devra afficher une phrase qui précise son propre identifiant, et qui explique ensuite qu'il envoie (ou reçoit) une valeur qu'il précisera au (du) processus dont l'identifiant sera lui-aussi précisé.

Question 6. En vous inspirant du programme écrit à la question précédente, écrivez, compilez et exécutez un programme distribué, fait de deux processus (et qui vérifie bien qu'il n'y a que deux processus), tel que les deux processus jouent au *ping-pong* : l'un d'eux doit envoyer un nombre (par exemple 100 pour l'un des processus et 200 pour l'autre), l'autre doit le recevoir et le renvoyer au premier, qui le reçoit à son tour et le renvoie, etc... Pour chaque envoi et chaque réception, chaque processus affichera un phrase qui décrit cet envoi ou cette réception. Le nombre d'envois sera précisé par un entier N. Exécutez votre programme pour différentes valeurs de N (par exemple $N = 5, 10, 100$).

Question 7. Écrivez, compilez et exécutez un programme distribué, fait de n processus, tel que le premier processus envoie un nombre (par exemple 100) au deuxième processus, reçoit un nombre k du n-ième processus, et affiche ce nombre. Tous les autres processus doivent recevoir un nombre k du processus précédent (dans l'ordre circulaire des identifiants) et le renvoyer au processus suivant. Exécutez votre programme avec $n = 3$ et avec $n = 20$.

Question 8. En vous inspirant de la question 4 de la première séance de travaux dirigés, écrivez, compilez et exécutez un programme distribué, fait de n processus, tel que chaque processus génère un entier au hasard entre 0 et une valeur M. Ces processus doivent déterminer collectivement les valeurs maximale et minimale parmi ces nombres à l'aide de messages. Vous ferez 3 versions de ce programme :

a) Dans la première version, chaque processus n'enverra (et ne recevra) qu'un seul message fait de deux entiers placés de manière consécutive dans la mémoire.

b) Dans la deuxière version, chaque processus enverra (et recevra) deux mes-

sages faits d'un seul entier, discriminés en utilisant la balise.

c) Dans la troisière version, chaque processus enverra (et recevra) deux messages faits d'un seul entier, qui ne seront pas discriminés en utilisant la balise.

Exécutez chaque version de votre programme avec $M = 100$ et $n = 10$. Qu'observez-vous ?