Examen de Calcul Distribué : correction

27 mai 2016

Question 1 `MPI_COMM_WORLD` :
- créé par `MPI_Init`,
- détruit par `MPI_Finalize`,
- permet d'assigner un rang à chaque processus,
- permet les communications entre tous les processus.

Question 2 L'enveloppe d'un message est constituée de :
- un communicateur,
- un identifiant (tag) de message,
- le rang du processus émetteur,
- le rang du processus récepteur.

Question 3
- `MPI_Ssend` : mode synchronisé. Le processeur émetteur ne sort pas de la fonction tant que le message n'a pas été complètement réçu par le processus récepteur.
- `MPI_Bsend` : mode bufferisé. Les données émises sont copiés dans un buffer temporaire. Ce buffer est utilisé pour la transmission, et le processus émetteur peut sortir de la fonction avant que les données n'aient été envoyées.
- `MPI_Rsend` : mode ready. Le processus récepteur doit être en attente du message pour que la communication s'effectue.
- `MPI_Send` : mode standard. À chaque appel, le programme décide entre deux comportements possibles : mode synchronisé ou mode bufferisé. Il n'est pas possible de prévoir le comportement.

Question 4

1. mpicc -o source source.c
2. mpirun -n 4 source
3. 28
   28
   28
   28

Question 5 On ne peut pas prévoir l'ordre d'écriture de la sortie. Il y a plusieurs comportements possibles.
- `MPI_Send` se fait en mode synchronisé. Dans ce cas, le processus de rang 0 doit attendre que le processus de rang finisse l'appel à `MPI_Recv`, et la sortie est donc nécessairement :
  Red leader standing by.
  Gold leader standing by.
- `MPI_Send` se fait en mode bufferisé. Dans ce cas, le processus de rang 0 peut sortir immédiatement de la fonction `MPI_Send` (indépendamment du processus 1). Les deux processus tentent d'écrire en même temps, et on ne peut pas prévoir lequel accèdera en premier à la resource de sortie.

Question 6 Un appel à `MPI_Ssend` ne peut se terminer que si un appel à `MPI_Recv` se fait. Chaque processus restera bloqué dans `MPI_Ssend`, attendant que l'autre processus n'appelle `MPI_Recv`.

Question 7 Le programme provoque un *segmentation fault*. Le tableau `array` passé en argument de `MPI_Bcast` doit être alloué pour contenir les données émises (pour le processus 0) **et** les données reçues (pour les autres processus), mais l'allocation n'a été faite que pour le processus 0.

Question 8 Le programme provoque un deadlock. Tous les processus doivent appeler la fonction `MPI_Scatter` : elle agira comme une émission ou comme une réception suivant le rang du processus qui l'appelle (émission pour le processus dont le rang est égal au 7ième argument de la fonction, réception pour les autres). La réception d'un `MPI_Scatter` **ne peut pas** s'effectuer par un appel à `MPI_Recv`.

Le processus 0 peut ou non sortir du `MPI_Scatter` (le comportement est similaire à celui d'un `MPI_Send`, des fois synchronisé, des fois bufferisé). Les autres processus resteront bloqués dans le `MPI_Recv` puisqu'il n'y a pas d'envoi correspondant.

Question 9 Le programme s'exécute correctement et la sortie est :
0 0 0 0
1 1 1 1
2 2 2 2
3 3 3 3