Programmation Parallèle et Distribuée
TD 3

Exercice I : Deadlock

Programmez la section de code suivante (uniquement pour deux processus MPI) :

```
char *buf = calloc(n, sizeof(char)) ;
char *buf2 = calloc(n, sizeof(char)) ;

if (rang == 0)
        vois = 1 ;
else
        vois = 0 ;

MPI_Send(buf, n, MPI_BYTE, vois, 0, MPI_COMM_WORLD) ;
MPI_Recv(buf2, n, MPI_BYTE, vois, 0, MPI_COMM_WORLD, &sta) ;
```

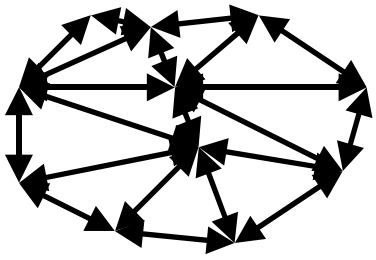
1. Expliquez en quoi cette section de code n'est pas sure.
2. Déterminez la valeur seuil de n pour laquelle le programme bloque.
3. Remplacez l'envoi standard par :
    a. un envoi synchrone ;
    b. un envoi bufferisé ;

   Que remarquez vous ?
4. Réécrivez cette section de code en utilisant des communications non bloquantes.

Exercice II : Graphe de communication

On représente par un graphe (non orienté) les communications entre *P* processus.

Chaque nœud du graphe représente un processus MPI.
Un arc entre deux nœuds définit l'existence d'envois/réceptions entre les deux processus correspondants.



Un graphe de communication est représenté par la structure suivante :

```
struct graphe_t
{
        int nb_noeuds ;

        /* tableau dimensionné à nb_noeuds
             nb_voisins[p] : retourne le nombre de nœuds directement connectés au nœud p
        */
        int *nb_voisins ;

        /* tableau à 2 dimensions
                voisins[p] : tableau dimensionné à nb_voisins[p]
                contient les numéros des nœuds directement connectés au nœud p
        */
        int **voisins ;
} ;
```

L'ensemble des voisins d'un processus p est $\{q = \text{voisins}[p][\text{iv}]$ où $0 \leq \text{iv} < \text{nb\_voisins}[p]\}$. Tout processus p ($0 \leq p < P$) doit envoyer nb_voisins[p] messages et recevoir nb_voisins[p] messages.

Pour un processus p donné, les buffers des messages à envoyer se trouvent dans le tableau char **msg_snd ; les tailles des buffers sont dans le tableau int *taille_msg_snd .
Autrement dit, le processus p doit envoyer le message msg_snd[iv] de taille taille_msg_snd[iv] au voisin q = voisins[p][iv] pour tout $0 \leq$ iv < nb_voisins[p].

Pour un processus p donné, les buffers des messages à recevoir se trouvent dans le tableau char **msg_rcv ; les tailles des buffers sont dans le tableau int *taille_msg_rcv .
Autrement dit, le processus p doit envoyer le message msg_rcv[iv] de taille taille_msg_rcv[iv] au voisin q = voisins[p][iv] pour tout $0 \leq$ iv < nb_voisins[p].

Soit la fonction

```
void echange( struct graphe_t *graphe,
                char **msg_snd, int *taille_msg_snd,
                char **msg_rcv, int *taille_msg_rcv ) ;
```

appelée par chaque processus p, et qui effectue les envois/réceptions définis par le graphe de communication graphe.

1. Écrire la fonction echange :
    a. en utilisant des communications non bloquantes ;
    b. en utilisant des communications bloquantes.

2. Que faudrait il faire pour écrire la fonction echange en utilisant des communications bloquantes en mode synchrone ?