Exercice 1

On se propose de paralléliser le traitement d'une matrice de réels en confiant le traitement de chaque ligne – calculer la somme des éléments de cette ligne – à un thread.

Le thread initial saisit au clavier le contenu de la matrice, active les threads sous-traitants, puis calcule et affiche la somme des valeurs qu'ils ont calculé

Version 1) Syntaxe d'appel de la commande :

% traiterMatrice

Version 2) Syntaxe d'appel de la commande :

% traiterMatrice NB_LIGNES NB_COLONNES

L'évolution de la version 1 à la version 2 doit entraîner le minimum de modification

On suppose que les fonctions suivantes existent :

```
void saisirMatrice(float mat[NBLMAX][NBCMAX], int nbL, int nbC);
float sommeLigne(float mat[NBLMAX][NBCMAX], int nbL, int nbC, int numL);
```

Comment dire aux threads sous-traitants ce qu'ils ont à faire ?

- **Solution 1 : Les threads ne partagent aucun information**
  - **Le thread principal devra fournir à chaque thread sous-traitant**
    - La matrice
    - Sa taille : nombre de lignes et nombre de colonnes
    - Le numéro de la ligne à traiter
  - **Comment ?**
    - Grâce au paramètre (void *) de la fonction décrivant le traitement du thread
    - En regroupant les informations dans une structure ➔ paramètre vu comme l'adresse d'une telle structure
- **Solution 2 : Les threads partagent presque tout**
  - **La matrice et sa taille sont partagées en les déclarant en global**
  - **Le thread principal fournit à chaque thread sous-traitant**
    - Le numéro de ligne à traiter
  - **Comment ?**
    - Grâce au paramètre (void *) de la fonction décrivant le traitement du thread
    - Vu comme un int *
- **Choix ici : Solution 2, autant profiter des avantages des threads (vs processus) !**

Comment les threads sous-traitants font remonter leur résultat ?

- **Solution 1 : Les threads ne partagent pas cette information**
  - Chaque thread sous-traitant devra communiquer ce résultat au thread principal
  - Comment ?
    - À sa mort, grâce au compte-rendu d'exécution
    - En réservant une adresse à laquelle stocker ce résultat
- **Solution 2 : Les threads partagent cette information**
  - Chaque somme calculée par un sous-traitant est partagée en la déclarant en global
    - Ce qui revient à partager un tableau de nbLignes float
  - Comment ?
    - Un thread sous-traitant modifiera la case du tableau qui correspond au numéro de ligne qu'il a traité
    - Le thread principal consultera la case lorsqu'il sera sûr que le thread ne la modifiera plus ➔ à sa mort
    - En se terminant, un thread sous-traitant ne reverra rien en compte-rendu (NULL) et le thread principal n'aura pas besoin de récupérer quelque chose
- **Choix ici : Solution 1, plus simple vu le contexte (un thread ne fait rien d'autre après le calcul de la somme, il peut mourir en renvoyant la valeur)**

Le thread principal (1/2)

```
int main (void) {
/* Besoin de conserver tous les identifiants des threads sous-traitants */
pthread_t idTh[NB_LIGNES_MAX];
/* Besoin de donner une information différente par sous-traitant
                                        ➔ autant d'adresses que nécessaire */
int nb[NB_LIGNES_MAX];
/* Initialiser taille et contenu de la matrice */
. . . .
/* Créer les threads sous-traitants [! Gérer éventuels échecs en traitant retours fonctions] */
for (int i = 0; i < nbLignes; i++) {
   nb[i] = i;
   pthread_create(&idTh[i], NULL, calculerSommeLigne, (void *)&nb[i]);
 }
```

Le thread principal (2/2)

```
/* Attendre la fin des threads sous-traitants pour récupérer et utiliser les résultats */
float somTotale = 0.0;     /* Déclaré au début du main normalement */

for (int i = 0; i < nbLignes; i++) {
   float *somPartielle;
   pthread_join(idTh[i], (void **)&somPartielle);
   somTotale += *somPartielle;
   free(somPartielle);
 }

/* Afficher la somme totale */
...
}
```

Les threads sous-traitants

- Exécutent la fonction suivante

```
void *calculerSommeLigne (void *arg) {
    /* Récupérer le numéro de ligne dont l'adresse est passée en paramètre */
     int numLig = *(int *)arg;
    /* Se préparer à stocker le résultat de manière pérenne */
     float *som = malloc(sizeof(float))
    /* Calculer le résultat */
    *som = sommeLigne(laMat, nbLignes, nbColonnes, numLig);
    /* Le retourner en mourant */
     pthread_exit((void *)som);
}
```