M3101: Mini-projet

Lundi 17 janvier

Projet par groupes de 2 ou 3. À rendre dans une archive zip à dimitri.lajou@u-bordeaux.fr.

L'objectif de ce projet est de créer un programme capable de calculer la suite de Syracuse en parallèle.

On rappelle que la suite de Syracuse (s_n) est définit comme suit:

$$s_{n+1} = \begin{cases} s_n/2 & \text{si } u_n \text{ est pair,} \\ 3 * s_n + 1 & \text{sinon.} \end{cases}$$

Il est conjecturé que pour n'importe quel nombre positif s_0, cette suite finit par attendre 1. Par exemple, si $s_0 = 5$, on a $s_1 = 16$, $s_2 = 8$, $s_3 = 4$, $s_4 = 2$, et $s_5 = 1$. Votre objectif est d'afficher pour chaque valeur de s_0, le premier incice qui attent 1. Pour 16, on affichera par exemple le message suivant : "Syracuse(16) = 5".

Bien entendu, nous sommes dans un cours de programmation système, donc on va ajouter de la difficulté. Votre programme se résumera ainsi:

- des threads producteurs (au nombre de NB_THREADS_PROD) ajoutent les nombres de 1 à MAX_NB dans la file d'attente struct waitingList.
- des threads consommateurs (au nombre de NB_THREADS_CONSO) récupèrent ces nombres, calculent les valeurs de la suite de Syracuse qui correspondent et affiche le message.

Comme nous travaillons avec des threads, faites attention aux sections critiques. On ne cherche pas forcément à traiter tous les nombres dans l'ordre.

Vous avez à votre disposition une structure de données WaitingList disponible dans les fichiers waitingList.h et waitingList.c.

```c
#define MAX_SIZE 20

struct WaitingList{
    int array[MAX_SIZE];
    int size;
};

void wl_init(struct WaitingList* wl);
int wl_pop(struct WaitingList* wl);
void wl_push(struct WaitingList* wl, int x);
```

Question 1. Codez les fonctions wl_pop et wl_push qui enlèvent et ajoutent un élément à la liste respectivement. On ferra attention à l'entier size qui représente le nombre d'élements dans la liste. On n'enlèvera rien si la liste est vide et on ajoutera rien si elle est pleine.

Question 2. Ecrire une fonction syracuse qui étant donné un entier n retourne le premier indice de la suite de Syracuse à attendre 1 quand $s_0 = n$.

Question 3. Completer la fonction producteur. Cette fonction ajoutera le nombre currentNb à la struct WaitingList waitingList si c'est possible. Pour chaque nombre ajouté, la fonction incrémentera currentNb. Une fois tous les nombres de 1 à MAX_NB ajoutés la fonction terminera.

Question 4. Completer la fonction consomateur. Cette fonction devra tenter de récupérer un nombre dans la struct WaitingList waitingList si celle-ci n'est pas vide et ce tant que tous les nombres n'ont pas été traités. Pour chaque nombre récupéré, la fonction affichera la valeur de la suite de Syracuse pour ce nombre. Ex: "Syracuse(16) = 5".

Question 5. Dans la fonction main, lancer des threads qui exécuteront les fonctions producteur et consomateur. Comme les données utilisés par nos threads sont dans des variables globales, ces fonctions n'ont pas besoin de paramètres.

Question 6. Testez votre programme avec différents nombres de threads. **Si vous avez des bugs, c'est surement que vous avez oublié de gerer les sections critiques.**

Question 7. Mesurer le temps d'exécution de votre programme avec 2 producteurs et 4 consommateurs pour MAX_NB = 100000 (*c.f.* ci-dessous). Comparez vos résultats avec vos camarades.

```
1  struct timeval tvB, tvE;
2  gettimeofday(&tvB, NULL);
3  // Ce que vous voulez mesurer
4  gettimeofday(&tvE, NULL);
5  long int tpsD =
6      (tvE.tv_sec - tvB.tv_sec) * 1000000 + tvE.tv_usec - tvB.
     tv_usec;
7  printf("Temps = %ld micro seconds\n", tpsD);
```