TP10

Page web du cours :

`http://www.lsv.ens-cachan.fr/~schwoon/enseignement/systemes/ws1516/`.

1 Allocation mémoire dynamique

L'objective de cette exercice est de travailler avec une structure de données qui réalise une pile qui stocke des entiers. La structure est définie comme suivant :

```
typedef struct stack {
  int content;
  struct stack * prev;
} stack_t;
```

`prev` pointe vers l'élément précédent (plus bas) sur la pile. Une pile est représentée par un pointeur vers l'élément le plus haut; un pointeur NULL représente une pile vide.

Un implémentation devrait fournir les trois fonctions suivantes :

```
// renvoyer l'élément le plus haut dans la pile
int peek(stack_t * s)

// rajouter un élément sur la pile
stack_t * push(stack_t * s, int content)

// enlever l'élément le plus haut
stack_t * pop(stack_t * s)
```

(a) Réalisez ces fonctions en temps constant ($O(1)$).

Un problème récurrent avec l'allocation dynamique de mémoire sont les fuites, c'est à dire des morceaux de mémoire qui non libérées. Un outil qui peut tester l'existence des fuites s'appele `valgrind`.

(b) Utilisez `valgrind` avec votre programme pour identifier des fuites éventuelles et corrigez-les.

```
valgrind --tool=memcheck --leak-check=full --track-origins=yes ./mem
```

(c) Utilisez votre implémentation des piles pour tester si une chaîne de caractères est un palindrome. (Remarque : on se limite aux cas ou un caractère est un simple octet.)

```
// renvoie 1 si txt est un palindrome et 0 sinon
int palindrome(char * txt)
```

2 Élection d'un leader

Un problème important dans la programmation concurrente est de choisir un *leader*, p.ex. pour l'organisation et coordination d'un réseau. On part avec un nombre n des ordinateurs ou processus (on parlera des *nœuds*) qui sont à priori tous pareil, sauf un identifiant unique. On suivant un même protocol, tous les nœuds vont se mettre d'accord sur un seul nœud qu'ils accepteront comme leader.

Il existe une grande variété de protocoles pour cet objectif. On va en étudier (et réaliser) un protocole par Dolev, Klawe, and Rodeh qui se distingue par le faible nombre de messages nécéssaire pour l'élection qui sera de $\mathcal{O}(n \log n)$. Par comparaison, les approches simples ont tendance a nééssiter prendre $\mathcal{O}(n^2)$ messages. Le protocole va être presenté pendant le TP, les diapos sont aussi disponible sur la page web.

Le protocole part du principe que les nœuds sont organisé en anneau, chaque nœud envoie des messages vers son voisin à droite. La page web contient un code squelette qui va créer n processus (les nœuds) pour un n donné ; ces processus seront déjà équipées des pipes pour communiquer. Votre tâche est de compléter le programme en réalisant le protocole :

- Au départ, tous les nœuds sont *actifs* et possèdent un identifiant unique.
- Un nœud actif attend une petite période aléatoire (fonction `delay()`), puis envoie son identifiant à son voisin à droit. Ensuite, il attendra les identifiants des *deux* plus proches voisins actifs à gauche. Il décidera ensuite s'il reste actif, devient passif, ou se déclare leader. (Les conditions seront précisées dans la présentation.) S'il reste actif, il répète le comportement ci-dessus.
- Un nœud *passif* transmet simplement tous les messages reçus de gauche vers son voisin à droit. En plus, si le message déclare un leader, il affiche un message correspondant à l'écran.
- Il y a trois types de messages échangés par les nœuds, tous dans le format (*type*,*identifiant*).
  - "voisin" (v): pour envoyer son identifiant vers le voisin à droit.
  - "prochain" (p): un nœud qui a reçu l'identifiant de son voisin à gauche l'envoie vers son voisin à droit.
  - "gagnant" (g): un nœud se déclare leader.

Essayez le protocole pour des différente valeurs de n. Observez si votre programme termine toujours et s'il y a toujours un seul leader.

Précisions : Vous devez modifier la fonction `node` et la compléter avec les réactions aux trois types de messages. La fonction `send_message` vous permet d'envoyer des messages.