Méthodologie de conception et de programmation_S2_P

Exercices sur mémoire, pointeurs, tableaux et allocation dynamique

Exo 1 : Pile et tas

Les variables déclarées dans une fonction sont allouées sur la pile.
La mémoire allouée dynamiquement (par malloc) est allouée sur le tas.

Sur un système Linux,
La plage d'adresse de la pile appartient "aux adresses hautes" (valeurs élevées) et la pile croît vers les adresses basses.
La plage d'adresse du tas appartient "aux adresses basses" (valeurs faibles) et le tas croît vers les adresses hautes.

Écrire un programme qui vous permet d'illustrer cette caractéristique.

Exo 1bis : Toucher les limites de la pile

Une façon de toucher les limites de la pile est d'empiler sans fin les appels de fonctions. Cette "erreur" est facile à obtenir grâce à une fonction récursive dans laquelle on a oublié de coder le *cas de base.*

Écrivez une fonction récursive qui déclare une variable puis affiche son adresse. Elle s'appelle donc elle-même et nous allons ici omettre sciemment le cas de base.

Appelez cette fonction dans le main, avec le paramètre que vous voulez puisque de toute façon elle va boucler indéfiniment jusqu'à ce que la pile soit pleine et provoque une erreur.

Quelle est la valeur de la dernière adresse affichée ?

Exo 2 : Visualisation de la mémoire d'un processus

Pour cet exercice, rédigez vos réponses dans un fichier **`exo2_TD4.txt`**. Vous en aurez besoin un jour.

Sur les systèmes de type Unix, il est possible d'avoir des informations sur les processus grâce au pseudo-système de fichier **`/proc`**.

Aller voir la page de man de proc : taper dans un terminal **`man proc`**

Ce qui va nous intéresser ici est le pseudo-fichier **`/proc/<pid>/maps`**

Chercher dans le man de proc la partie qui traite de ce pseudo-fichier et ce qu'il contient.

Modifier le programme de la question précédente pour qu'il ne se termine que quand l'utilisateur a entré la lettre **`q`** au clavier.

Dans un terminal, lancer le programme et ne taper PAS sur q 🙂

Dans un autre terminal, nous allons chercher quel est son numéro **`pid`** (processus id)

Une façon d'avoir une vue (jolie) des processus actuellement présents sur votre système est d'utiliser la commande **`pstree`**

Taper la commande **`pstree`**. Observez l'affichage, commentez puis comparez aux commentaires du prof (que vous pouvez noter aussi).

Comme nous souhaitons en outre obtenir les numéros pid :

Taper la commande man **`pstree`** et trouver l'option qui permet d'afficher ce numéro ; taper cette commande.

Taper la commande **`pstree <option_que_vous_avez_trouvée> | grep <nom_de_l_exécutable>`**

Nous obtenons le numéro pid de l'exécutable.

Nous pouvons maintenant observer le pseudo-fichier : taper la commande **`cat /proc/<pid_trouvé>/maps`** dans un terminal (maximiser la taille du terminal pour avoir un affichage joli). Commentez puis comparez aux commentaires du prof (que vous pouvez noter aussi).

Exo 3 : Pointeurs et tableaux

Les tableaux en **`C`** se comportent (de loin) comme un pointeur constant. Dans la déclaration :

```
int tab[10];
```

`tab` peut (dans une première approximation) être assimilé à une variable de type **`(int * const)`** dont la valeur est **`&tab[0]`** c'est-à-dire l'adresse de la première case de **`tab`**.

Déclarer un tableau de 10 entiers.
Déclarer un pointeur sur un entier et l'utiliser pour afficher le contenu du tableau. (Utiliser l'exercice sur l'arithmétique des pointeurs).

Dans quelle zone de mémoire se situent les éléments du tableau ?
Quelles différences faites-vous entre ce tableau et une allocation dynamique pour 10 entiers ?

Exo 4 : Pointeurs de pointeurs de taille variable, calcul de coefficients binomiaux avec le triangle de Pascal

On va calculer des coefficients binomiaux en utilisant la formule de Pascal :
C^p_n = C^{p}_{n-1} + C^{p-1}_{n-1}.

Une première propriété des coefficients binomiaux est que dès que p > n, C^p_n = 0.
Ainsi il n'est pas utile de stocker toutes ces valeurs de C^p_n. Pour utiliser facilement la formule de Pascal, on stockera C^{n+1}_{n}.

Une seconde propriété utile pour le calcul est que C^0_n = 1 pour tout n.

Écrire une fonction **`binom_coeff_alloc`** qui prend en paramètre N et retourne l'adresse de la zone mémoire allouée dynamiquement pour le calcul du triangle de Pascal de taille N.

Écrire une fonction **`binom_coeff_comp`** prend en paramètre N et l'adresse d'une zone mémoire pour le calcul du triangle de Pascal de taille N et qui calcule toutes les valeurs de coefficients binomiaux dans le triangle.

Écrire une fonction **`binom_coeff_free`** qui prend en paramètres une zone mémoire allouée dynamiquement qui contient un triangle de Pascal et son nombre de lignes N et libère l'espace mémoire alloué dynamiquement.

Écrire une fonction **`binom_coeff_print`** qui prend en paramètres une zone mémoire allouée dynamiquement qui contient un triangle de Pascal, n et p et affiche le coefficient binomial C^p_n (traitez les différents cas des valeurs de n et p possibles).

Écrire une fonction **`binom_coeff_print_triangle`** qui prend en paramètres N et une zone mémoire allouée dynamiquement qui contient un triangle de Pascal et affiche le triangle de Pascal pour le paramètre N.

Tester toutes ces fonctions à l'aide de plusieurs exemples.

Modifié le: mercredi 8 février 2017, 09:07