Gabriel C-Parent et Julie Rivière
Rapport de TP3: garbage collector

Introduction

Le rapport qui suit décrit notre expérience de la construction d'un garbage collector écrit en C. Nous tenons à souligner que certains choix peu communs ont été pris lors de la construction du programme, des choix que nous pensons être valables étant donné les faiblesses des constructions proposées. Le premier effet est que le fichier d'interface gc.h n'est plus adéquat pour le système que nous avons créé, étant donné l'utilisation de double pointeurs pour l'allocation d'objets dans la mémoire.
Les fonctions de test sont à la fin du fichier gc.c. Elles permettent de tester différents aspects du ramasse-miettes (le système de mémoire "virtuelle" si on peut l'appeler de la sorte, l'assignation, la défragmentation et le système de racines).
Nous allons développer les difficultés rencontrées lors de la création du système et justifier les choix faits.

Architecture du système

Le système de ramasse-miettes construit dans gc.c alloue de la mémoire sous la forme de doubles pointeurs. Avec le niveau d'indirection supplémentaire et un système de "pagination" qui garde en mémoire la position, la taille des objets alloués et un pointeur vers ceux-ci, le système ne souffre pas de fragmentation (ceci a évidemment un prix).

Allocation de mémoire

L'allocation de mémoire se fait à deux niveaux. Lorsque le programme réquisitionne de l'espace (via struct GCobject** gc_malloc(struct GCclass* class)), une nouvelle page est crée et représente l'objet dans le tableau de bytes. Cette structure de page comporte un pointeur vers la position de l'objet dans le tableau et est chargée de l'actualiser lors de la défragmentation. Un pointeur vers ce pointeur est retourné à travers gc_malloc() et c'est depuis celui-ci que le programme externe travaille. Chaque objet est ajouté à la fin de la liste chaînée de pages. Ceci se fait en temps constant, un pointeur vers la dernière page étant toujours conservé en mémoire.

Ramassage

La collection se fait à travers la fonction gc_collect(). Cette fonction est normalement appelée par lors de l'allocation, lorsqu'il n'y a plus de place pour

allouer à la fin du tableau. Le ramassage se fait en deux étapes (c'est mark & sweep évidemment), le marquage, durant lequel les racines sont traversées et les objets y étant liés sont marqués sur un byte artificiellement ajouté dans la structure (l'idée d'opérer sur un bit seulement n'est pas si avantageuse et modifie l'objet, ce qui peut amener des problèmes intéressants). La liste de pages est ensuite traversée, les derniers bytes vérifiés et la page est libérée si l'objet n'est pas marqué (on assigne une valeur char 'U') ou sinon la page est conservée et "glissée" le plus à gauche possible dans la mémoire, ce qui enlève la fragmentation.

Critique du modèle
Le problème évident est qu'il y a un coût en performance à cause de la perte d'un alignement optimal (en plus, pragma pack(1) est utilisé afin de minimiser l'espace utilisé). Ceci est toutefois contrebalancé par le fait que ce système permet d'allouer facilement des objets de tailles différentes et que l'espace est utilisé efficacement. De plus, il n'y a aucune garantie lors de l'utilisation du modèle de blocs que l'alignement sera adéquat dans l'architecture particulière de l'ordinateur et donc la performance n'est probablement pas un argument convaincant.

Conclusion
En somme, nous croyons que le problème de fragmentation rencontré dans les systèmes recommandés était suffisant pour justifier la création d'un système à deux niveaux d'indirection qui permet d'éliminer la fragmentation. Nous sommes sincèrement désolés des problèmes de correction que ceci engendre, mais nous croyons que ce modèle est plus pertinent et plus intéressant.