Compilation - TD9

Glanage de cellules (GC)

1 Contexte et culture gé

Question 1-1 Racontez des histoires horribles de bugs dus à des `free` mal placés en C. Pourquoi ces histoires sont-elles si horribles ? Citez quelques langages modernes et civilisés, et constatez qu'ils sont tous munis d'un GC.

Question 1-2 Il y a deux grandes classes de GC : incrémental et par à-coups. Il y a trois grands algorithmes : par comptage de références, par *mark and sweep*, à deux bancs de mémoire. Expliquez avec les mains et classez.

Question 1-3 Expliquez le phénomène de *fragmentation* de la mémoire, et la nécessaire *compaction* qui en résulte.

Question 1-4 Le compilateur doit fournir plein d'information au GC : ce dernier aura besoin de connaître le type exact de la cellule au bout de n'importe quel pointeur pour pouvoir suivre les pointeurs qu'elle contient. Or il existe des choses affreuses comme des unions dont le type va changer dynamiquement, des languages permettant de *cast everything to* `void*` *to get rid of this annoying type-checking*, etc. La manière la plus simple de s'en sortir est d'utiliser des cellules auto-descriptives. Donnez au moins deux techniques.

2 Méthode des compteurs de référence

Question 2-1 Il y a une situation dans laquelle cette méthode ne récupérera jamais une cellule, laquelle ?

3 Méthode "marquage et balayage"

Dans toute la suite on appelle *graphe de vivacité* le graphe dont les noeuds sont des cellules (du tas ou des enregistrements d'activation), et les arêtes des pointeurs.

La méthode *mark and sweep* consiste à partir des *racines* du graphe de vivacité (les variables présentes dans la pile d'activation), marquer tous les enregistrements atteignables à partir des racines, et ensuite glaner tous les enregistrements non marqués.

Question 3-1 Quelle phase sera la plus longue, marquage ou glanage ?

Marquage L'algorithme suivant adopte une stratégie en profondeur d'abord pour explorer le graphe de vivacité :

DFS(x) `:=`
`si` x pointe vers un objet non marqué du tas
  marquer x
  `pour tout` champ f_i de x `faire` DFS(f_i)

Question 3-2 Quel est le problème si on implémente l'algorithme ci-dessus naïvement ?

Question 3-3 Comment l'algorithme suivant pallie-t-il à ce problème ?

```
DFS(x) :=
 si x pointe vers un objet non marqué du tas
 t ← nil
 marquer x ; done[x] ← 0
 tant que vrai
  i ← done[x]
  si i < nombre de champs dans x alors
     y ← x.f_i
     si y pointe vers un objet non marqué du tas alors
        x.f_i ← t ; t ← x ; x ← y
        marquer x ; done[x] ← 0
     sinon done[x] ← i + 1
  sinon
     y ← x ; x ← t
     si x = nil alors break
     i ← done[x]
     t ← x.f_i ; x.f_i ← y
     done[x] ← i + 1
```

Glanage On se propose d'utiliser l'algorithme suivant :

```
p ← première addresse du tas
tant que p < dernière addresse du tas
 si l'enregistrement pointé par p est marqué
  alors enlever la marque sur p
  sinon soit f1 le premier champ de p,
        p.f1 ← freelist
        freelist ← p
 p ← p + taille(p)
```

Question 3-4 Quel est le problème avec la liste chaînée freelist que l'on récupère ? Proposer une solution pour mieux organiser l'information que l'on glane.

4 Méthode générationnelle

L'expérience prouve que la durée de vie des objets alloués lors de l'exécution d'un programme est fort variable : certains meurent très rapidement, d'autres ne disparaissent qu'à la fin. L'idée est ici de diviser le tas en plusieurs "tranches d'âge" G0, G1,...,Gn. G0 contient les objets les plus jeunes, et Gn les objets les plus vieux.

Le plus souvent, on fait du GC dans G0 (zone où sont bien entendu placés les objets nouvellement alloués). Régulièrement, on bascule des objets de G0 dans G1, et on fait la même opération dans G1 (mais à fréquence moindre). Typiquement, la taille des Gi croît de manière exponentielle, et un objet vieillit d'un cran lorsqu'il survit à une ou deux passes de GC.

Question 4-1 Quel est le problème de cohérence qui se présente lorsque l'on s'attache à ne faire un GC que dans G0 ? Proposer une méthode pour se garder d'erreurs fâcheuses.

Question 4-2 *Le boa qui digère le mammouth (pig in snake problem) :* quel est le problème si un programme utilise un paquet d'objets alloués dans le tas pendant longtemps, et libère ceux-ci d'un coup ?

Aux dernières nouvelles, Caml utilise un GC en mark and sweep, *avec deux générations.*