TD-TME 4

Simulation à la main d'algorithmes de GC

Soit le programme suivant :

pseudo-C	pseudo-ML

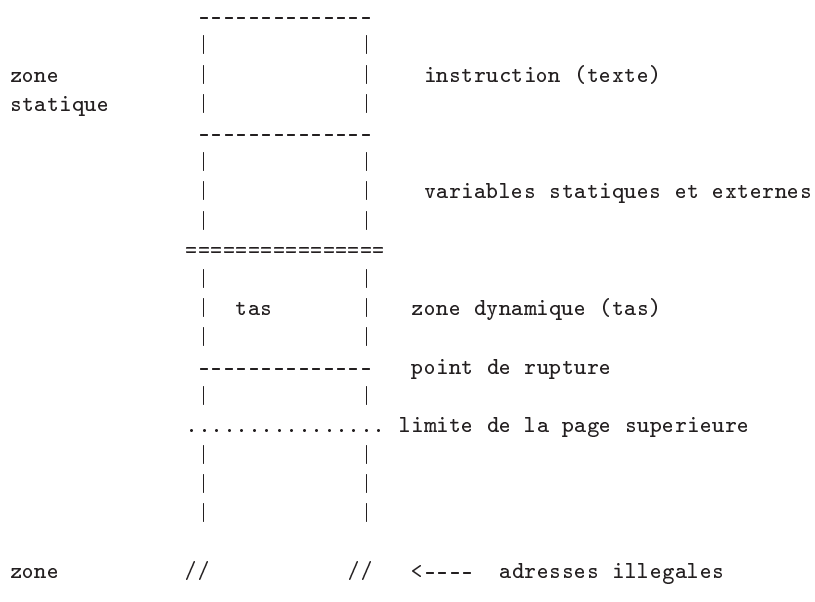
x = make2(NULL,NULL);
y = make2(x, NULL);
z = make2(x, NULL);
x = make2(y,z);
y[2] = x;
y = NULL;
x = z;
z = make2(x,NULL);
	type ('a, 'b) cell =
	  {mutable tete : 'a; mutable queue : 'b};;
	let x = {tete = None; queue = None};;
	let y = {tete = Some x; queue = None};;
	let z = {tete = Some x; queue = None};;
	let x = {tete = Some y; queue = Some z};;
	y.queue <- Some x;;
	let y = None;;
	let x = z;;
	let z = {tete = Some x; queue = None};;

On suppose que l'ensemble des racines est composé de toutes les variables globales du programme.

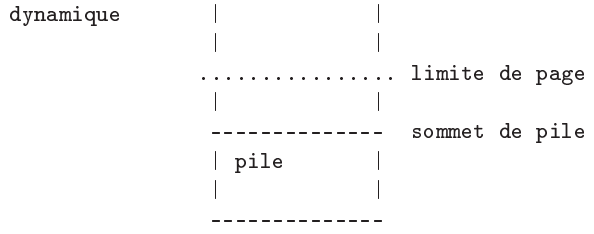
1. indiquer l'état mémoire à la fin de son exécution
2. simuler un algorihtme de compteurs par références sur cette mémoire;
3. simuler les trois algorithmes explorateurs suivants si le GC se déclenche à la fin de l'exécution de ce programme.
   (a) Mark&Sweep
   (b) Mark&Compact
   (c) Stop&Copy

Organisation mémoire d'un programme

Le schéma suivant [1] montre l'organisation de l'espace d'adressage d'un processus.



[1] est tiré de l'ouvrage de Jean-Marie Rifflet "La communication sous Unix"



En C la pile est gérée automatiquement par le système. Elle contient les déclarations locales, les paramètres des fonctions. Comme la portée de ces variables est connue, le compilateur C ajoute dans le code engendré les instructions de gestion de pile.

Le programmeur peut agir sur la zone des données (statique ou dynamique). L'accès à une adresse illégale engendre le signal `SIGSEGV`. Sous Unix, le point de rupture est l'adresse virtuelle la plus petite en dehors de cet espace de données. Comme l'allocation de mémoire s'effectue par page, il est possible d'accéder au delà du point de rupture si on reste dans la page allouée.
Il peut être utile de connaître et de modifier le point de rupture. Pour cela on utilise les fonctions `brk` et `sbrk` :

`int brk(char *adr)`	positionne le point de rupture en `adr`	retourne 0 ou -1
`caddr_t sbrk(inc)`	ajoute `inc` au point de rupture	retourne l'ancienne valeur ou `(caddr_t)-1`

Optimisation de l'allocation

On cherche à évaluer les performances de `malloc` et `free`. Pour cela on écrit tout d'abord un programme gourmand en mémoire, ensuite on écrit une gestion mémoire plus rapide que `malloc` en allouant en une seule fois une grande zone mémoire pour les listes d'entiers.

Crible d'Eratosthène

On travaillera sur le type `liste_entier` suivant :

```
struct cons {int car; struct cons *cdr;};
typedef struct cons *liste_entier;
```

1. Ecrire une fonction `cons` qui ajoute un entier à une liste d'entier et retourne la nouvelle liste en utilisant `malloc`.
2. Ecrire une fonction `intervalle` qui prend deux entiers a et b et construit la liste de tous les entiers entre a et b.
3. Ecrire une fonction qui calcule la liste des nombres premiers inférieurs à un n donné. On utilisera l'algorithme du crible d'Eratosthène. Ecrire une version sans récupération mémoire et une autre version avec récupération mémoire.
4. Que pouvez-vous dire sur l'occupation mémoire après l'appel de `eratosthene(1000)` dans ces deux cas?

Allocation optimisée

1. Ecrire une fonction `init_alloc` qui alloue en une fois une zone mémoire pouvant contenir n *cons*. Que faut-il faire sur cette zone pour pouvoir l'utiliser ensuite par un `malloc` utilisateur?
2. Ecrire une fonction `mon_malloc` qui utilise la zone déclarée par `init_alloc`.
3. Ecrire une fonction `mon_free` qui libère une cellule de cette zone.

Récupération automatique de liste_entier

1. Ecrire une fonction `init_GC`, version modifiée de `init_alloc`, qui conserve les informations de début et de fin du tas, ainsi que les informations sur la zone statique et sur la pile. On utilisera pour déterminer la zone statique les symboles globaux `end` et `edata`. On calculera l'adresse de début de pile et son sens en faisant un calcul sur les adresses des variables locales.
2. Ecrire une fonction qui détermine si une valeur (un pointeur) est un pointeur sur une liste d'entier.

3. Ecrire une fonction `mark` qui prend un pointeur valide en entrée et le marque dans le tas (`| 1`) sur le champ `cdr`) et marque les descendants s'il y en a de ce pointeur.
4. Ecrire une fonction `sweep` qui explore le tas et reconstruit la liste des cellules libres en enlevant les marques des cellules à conserver.
5. Ecrire une fonction `gc` qui récupère automatiquement l'espace disponible dans le tas.
6. Utiliser les fonctions précédentes pour modifier la fonction `mon_malloc` qui déclenche le GC s'il n'y a plus de place disponible et qui retourne l'espace libéré.
7. Que se passe-t-il s'il n'y a toujours pas assez de place après un GC?
8. Que se passe-t-il si un entier (correspondant à la valeur d'un pointeur valide) se trouve dans la pile?
9. Que se passe-t-il si une liste d'entiers apparaît comme champ d'une valeur `struct`?

Utilisation des programmes décrits

On trouvera sur les machines `ari-31-312-XY` les programmes suivants :

- `stat.c` : programme de calcul de la zone de données statiques;
- `pile.c` : programme de calcul du sens de la pile;
- `crible.c` : programme classique du crible (utilise `crible_gen.c`);
- `crible_gen.c` : le programme générique pour le crible;
- `alloc_man.c` : optimisation de l'allocation;
- `crible_mm.c` : utilise `alloc_man.c` et `crible_gen.c`;
- `gc_alloc.c` : GC `Mark&Sweep` pour les listes d'entiers;
- `crible_gc.c` : utilise `gc_alloc.c` et `crible_gen.c`.

dans le catalogue `/users/Enseignants/chaillou/install/td4ex`.
N'hésitez pas à les compiler, les tester, les lire et les modifier!!!

GC de Boehm

1. A l'URL suivante : `http://www.hpl.hp.com/personal/Hans_Boehm/gc/` récupérez une version stable du GC de Boehm (`http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/gc.tar.gz`).
2. Compilez-le et testez-le.
3. Modifier le programme du crible d'Eratosthène pour le compiler et le linker en utiulisant le GC de Boehm.

Récupération automatique de liste_entier (variante)

`Stop & Copy`

1. Modifier les fonctions d'allocation et de récupération pour implanter ce GC pour les listes d'entiers. Pour la copie, on inclura dans le champ `cdr` la nouvelle adresse de la cellule déplacée.
2. Que se passe t-il si un entier correspond à une adresse valide? Comment peut-on différencier un pointeur d'un entier?