TP11 - Valgrind

Langage C (LC4)

Semaine du 12 avril 2010

1 Trouver des fuites de mémoire avec Valgrind

En C, où il n'y a pas de garbage collector (contrairement à d'autres langages comme Java, Caml...), il est important de s'assurer qu'un programme ne présente pas de fuites de mémoire. Valgrind permet de vérifier cela facilement.

Question 1. Compilez le code suivant (`gcc -Wall -Wextra -g example.c -o example`) :

```
#include <stdlib.h>

int main(void)
{
  char* x = malloc(100 * sizeof(char));

  return EXIT_SUCCESS;
}
```

Maintenant, lancer votre programme avec Valgrind :
`valgrind --tool=memcheck ./example` (ou simplement `valgrind ./example`).
Que remarquez-vous ? Comment faire pour trouver l'erreur ?

Question 2. Recompilez maintenant le même programme sans l'option `-g` de `gcc` :
`gcc -Wall -Wextra example.c -o example`
et relancer Valgrind. Quelle différence observez-vous ?

Remarque : il se peut que pour afficher absolument toutes les erreurs d'allocation, il faille ajouter l'option `--show-reachable=yes`.

Question 3. Vérifiez que le code suivant ne s'exécute pas comme il faut :

```
#include <stdlib.h>

int main(void)
{
  char* x = malloc(100 * sizeof(char));
  free(x);
  free(x);

  return EXIT_SUCCESS;
}
```

Corrigez-le grâce à Valgrind.

Question 4. Comparez maintenant le temps d'exécution du programme avec et sans valgrind à l'aide de `time`. Que remarquez-vous ?

2 Détecter des utilisations invalides de pointeur

Valgrind permet aussi de détecter des utilisations invalides de pointeur, par exemple lors d'une tentative d'accès en dehors d'un tableau qui a été alloué.

Question 5. Compilez maintenant le code suivant :

```
#include <stdlib.h>

int main(void)
{
  char* x = malloc(10 * sizeof(char));
  x[10] = 'a';
  free(x);

  return EXIT_SUCCESS;
}
```

Trouver l'erreur grâce à Valgrind.

3 Trouver des variables non initialisées

Valgrind trouve aussi les variables non initialisées.

Question 6. Corrigez l'erreur dans le code suivant grâce à Valgrind :

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  int i;

  if (i == 42) {
    printf("Hello!\n");
  }

  return EXIT_SUCCESS;
}
```

Question 7. Faites de même avec le code suivant :

```
#include <stdlib.h>
#include <stdio.h>

int foo(int a)
{
  if (a == 42) {
```

```
    return 42;
  }
  return 0;
}

int main(void)
{
  int i;
  foo(i);

  return EXIT_SUCCESS;
}
```

4 Limitation de Valgrind

Question 8. Le code suivant est-il valide ? Si ce n'est pas le cas, Valgrind vous aide-t-il à le débugguer ?

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  char x[10];
  x[11] = 'a';
  printf("%c\n", x[42]);

  return EXIT_SUCCESS;
}
```

5 Profilage de code avec Valgrind

Valgrind permet aussi de faire du profilage de code, c'est-à-dire d'analyser une application afin de connaître la liste des fonctions appelées et le temps passé dans chacune d'elles, ce afin d'en identifier les parties de code qu'il faut optimiser.
Pour cela, il faut exécuter :
`valgrind --tool=callgrind ./example`.
Cela va générer un fichier `callgrind.out.25137` qui est lisible par un logiciel tel Kcachegrind.