Présentation de Valgrind

1. Présentation générale

Valgrind est un outils Open-Source permettant de mettre à jour les problèmes de gestion de mémoire pour des programme Linux-x86. Il détecte les fuites de mémoires pendant l'exécution du programme. Il a été développé par Julian Seward.

L'allocation dynamique de conteneurs de stockages pour les données jouent un rôle important en C, mais c'est aussi une source de nombreuse erreurs difficiles à trouvées. Libérer un bloc de donnée deux fois, surcharger le buffer du malloc ou même encore en perdre une adresse sont des cas d'erreurs fréquents qui frustrent le programmeur. Trouver ces erreurs et les corriger est très difficile car elles se manifestent toutes de la même façon à des endroits souvent lointain de leur cause.

Pour chacune de ces erreurs Valgrind est utile. Valgrind travail directement avec les exécutables, sans besoin de recompiler, relinker ou de modifier le programme à vérifier. Valgrind décide si le programme a besoin d'être modifié pour éviter une fuite de mémoire, et pointe aussi l'endroit de la "fuite".

Valgrind simule toutes les instructions exécutées par le programme. Pour cette raison Valgrind trouve les erreurs non seulement dans l'application mais aussi dans toutes les librairies linkées dynamiquement, ceci incluant les librairies GNU, les librairies des clients X, et même Qt si l'on travail avec KDE, etc… Ceci incluse également les librairie, celle GNU C par exemple, qui peuvent contenir des accès de violation mémoire.

2. Procédure d'installation

Pour installer Valgrind sous un système d'exploitation de type Unix, il faut tout d'abord télécharger la distribution binaire, disponible sur le site http://developer.kde.org/~sewardj/ (une alternative est http://freshmeat.net/projects/valgrind/). Ensuite, il est nécessaire de compiler le logiciel en saisissant les commandes suivantes :

```
$ tar zxvf valgrind-1.0.0.tar.gz
$ cd valgrind-1.0.0
$ ./configure --prefix=”/usr/local/valgrind”
$ make
$ make install
```

Valgrind est alors installé dans /usr/local/valgrind. Son exécutable est situé soit dans /usr/local/bin ou dans /usr/local/valgrind/bin.

3. Utilisation de Valgrind

Le début de la vérification est lancée en plaçant simplement le mot valgrind devant la command de l'exécutable. Par exemple :

```
$ valgrind ls -laF
```

Valgrind propose une multitude d'options dont nous ne parlerons pas ici pour ne pas surcharger cette presentation. La sortie présente la sortie habituelle du "ls -laF" avec aussi un rapport détaillé de la part de Valgrind. Toutes les erreurs liées à la mémoire sont données dans le rapport.

Comprendre le rapport de Valgrind :
Considérons la sortie fournie par Valgrind suivante :

```
   ==1353== Invalid read of size 4
   ==1353==    at 0x80484F6: print (valg_eg.c:7)
   ==1353==    by 0x8048561: main (valg_eg.c:16)
   ==1353==    by 0x4026D177: __libc_start_main
(../sysdeps/generic/libc-start.c   :129)
   ==1353==    by 0x80483F1: free@@GLIBC_2.0 (in /home/valg/a.out)
   ==1353==    Address 0x40C9104C is 0 bytes after a block of size 40 alloc'd
   ==1353==    at 0x40046824: malloc (vg_clientfuncs.c:100)
   ==1353==    by 0x8048524: main (valg_eg.c:12)
   ==1353==    by 0x4026D177: __libc_start_main
(../sysdeps/generic/libc-start.c   :129)
   ==1353==    by 0x80483F1: free@@GLIBC_2.0 (in /home/valg/a.out)
```

Ici, 1353, est le numéro du processus qui est suivit. Cette partie du rapport d'erreurs dit qu'une erreur de lecture à lieu à la ligne 7, dans la fonction print. La fonction print est appellée par la foncytion main, et toutes deux sont dans le fichier "valg_eg.c". La fonction main est appelée par la fonction __libc_start_main à la ligne 129 dans le fichier ../sysdep/generic/libc-start.c, elle-même appelée par free@@GLIBC_2.0 dans le fichier /home/valg/a.out. Des détails similaires sont données pour l'appel de la fonction malloc.

Un type d'erreur avec un Exemple :
Valgrind ne peut réellement détecter deux types d'erreurs : l'utilisation illégale d'adresses et l'utilisation de valeures non définies. Néanmoins c'est suffissant pour trouver de nombreuses sources d'erreurs dans un programme.

```
#include <stdlib.h>
int main()
{
        int p, t;
        if (p == 5)             /*Source de l"erreur*/
                t = p+1;
        return 0;
}
```

Ici la valeur de p n'est pas initialisée, donc p peut contenir des valeurs aléatoires provenant de mémoire non effacée. Donc une erreur peut se produire lorsque la condition est vérifiée. Une variable non initialisée peut devenir source de problème dans deux cas :

- lorsqu'elle est utilisée pour déterminer quelle route conditionnelle emprunter.
- Lorsqu'elle est utilisée pour générer une adresse (comme l'appel tableau[p] avec un p non initialisé).

Exemple d'une fuite de mémoire :
Soit le programme suivant :

```
#include <stdlib.h>
int main()
{
	int *p, i;
	p = malloc(5*sizeof(int));
	for(i = 0;i < 5;i++)
		p[i] = i;
	return 0;
}
```

Voici la sortie fournie par Valgrind :

```
==1048== LEAK SUMMARY:
==1048==    definitely lost: 20 bytes in 1 blocks.
==1048==    possibly lost:   0 bytes in 0 blocks.
==1048==    still reachable: 0 bytes in 0 blocks.
```

Dans le programme ci-dessus p contient l'adresse d'un block de 20 byte. Mais ce bloc n'est pas libéré par le programme. Donc le pointeur pour ce bloc de 20 byte est perdu à jamais. C'est une fuite de mémoire. On peut alors obtenir le résumé sur les fuites en utilisant l'option de Valgrind : --leak-check=yes.

Comment supprimer les erreurs :

Valgrind détecte de nombreuses erreurs dans de multiples programme qui sont pré-installés dans un système GNU/Linux. Il n'est pas facile de résoudre de tels problèmes mais il faut le faire. Alors Valgrind peut en faire la liste dans un fichier de supression .supp. Le format de ces fichier est décrit ci-dessous.

```
{
	Nom de l'erreure
	Type
	fun : nom de la fonction qui contient l'erreur
   fun : nom de la fonction qui appelle la fonction contenant l'erreur
}
```

On obtient ces fichier avec la commande :
valgrind --suppressions=chemin_du_fichier.supp le_nom_du_prog

4. Limitations de Valgrind

Aucun logiciel n'est libre de limitations. C'est la même chose dans le cas de Valgrind, même si il fonctionne bien. Voici une liste des limitations de Valgrind :

- Les programmes tournent 25 à 50 fois moins vite
- Augmentation de la consomation de mémoire.
- Les codes optimisés de manière aggressive (-O1, -O2 par exemple) peuvent certaines fois abuser Valgrind.
- Valgrind repose sur des librairie linkées dynamiquement.
- Valgrind ne fonctionne qu'avec un environement Linux x86 (kernel 2.2.X ou 2.4.X) ainsi qu'avec la librairie Glibc 2.1.X ou 2.2.X.