Séance de mise à niveau. Outils de débogage: gdb, valgrind, gprof

Objectifs du TP : Le débogage de programme C est un art en soi. Le travail le plus important se fait lors de la conception par la structuration en modules et fonctions ainsi que lors du choix de la structure de donnée. La programmation et le test de chaque module indépendamment permet aussi de gagner beaucoup de temps sur l'ensemble du projet. Malgré cela, il reste souvent des bugs difficiles à trouver rapidement. Nous passons en revue divers outils d'aide à la programmation : valgrind permettant de détecter les fuites mémoires (très utile car si votre programme fonctionne sur un exemple, il peut très bien être buggé quand même), grpof permettant de faire du profiling rapidement et de détecter ou votre programme perd son temps et enfin gdb, debuggeur indispensable pour comprendre d'où viennent les erreurs. Avant de commencer, récupérez l'archive du TD disponible sur `http://graal.ens-lyon.fr/~eagullo/asr2/` ou localement sur Europe (`europe:/home/eagullo/teach/asr2/`).

Le document original provient de la page web `http://perso.citi.insa-lyon.fr/trisset/cours/AGP/` et a été écrit par T. Risset.

1 Compilation

Compilez votre programme. Modifiez votre *Makefile* pour positionner le drapeau *all* dans la variable *CFLAGS*. Recompilez et corrigez les erreurs signalés par les avertissements de compilation. Notez que d'autres outils permettent le débogage à la compilation tel que *lint* :
`http://www.infoworld.com/article/06/01/26/74270_05FEcodelint_1.html?s=feature`

2 Gdb

Regarder puis compiler le programme *bug.c* (Makefile fourni), exécutez-le avec un entier comme argument sur la ligne de commande. Essayez de trouver l'erreur à vue (record sans débuggeur : 2'30). Même si vous avez trouvé, lancer *gdb*. Sous *gdb*, tapez *help* pour avoir la liste de commande, taper *help cmd* pour l'aide sur la commande *cmd*.

1. Chargez le programme *bug* (commande file).
2. Lancez l'exécution avec 3 comme argument (*run 3*), repérer la ligne ou a lieu l'erreur. Afficher les lignes autour (commande *list*).
3. Afficher la pile d'appel de fonction menant à l'erreur (commande *backtrace*).
4. Afficher la valeur de i (commande *print*).
5. Tapez *help breakpoints* pour avoir la liste des commandes permettant d'utiliser les point d'arrêt. Mettez un point d'arrêt au début de la fonction *main*, un autre au début de la fonction *traitement*. Enlevez celui de la fonction main.
6. Relancez le programme bug.
7. Lorsque vous êtes arrêté dans la fonction traitement, mettez une surveillance de la variable i (commande *watch*). Entrez la commande *watch i* pour afficher i à chaque arrêt.
8. Tapez *cont* pour continuer, surveillez l'exécution jusqu'à ce que vous trouviez le problème. Notez que *ddd* permet(trait) de faire de meme avec une interface graphique.

3 Valgrind

L'outil *valgrind* permet de vérifier le code à l'exécution. Le code est exécuté dans une machine virtuelle qui vérifie les appels, adresses mémoires, etc. valgrind est essentiellement utilisé pour découvrir tous les problèmes mémoire : "fuites mémoire" (memory leak : mémoire

allouée mais non libérée) et accès en dehors des bornes des tableaux. Les fuites mémoires ont peu d'importance pour les programmes qui terminent car l'espace alloué est libéré par le système d'exploitation, mais lorsque le programme est utilisé comme librairie dynamique, les fuites répétées peuvent rapidement saturer la mémoire. L'outil *valgrind* est extrêmement simple d'utilisation. Si votre programme se lance avec la commande *monProg arg1 arg2*, il suffit d'exécuter *valgrind [option] monProg arg1 arg2* où *[option]* indique ce que l'on désire mesurer avec valgrind. L'option par défaut (pas d'option) exécute *memcheck* qui vérifie l'essentiel des problèmes mémoire comme l'accès en dehors des bornes des tableaux, l'utilisation de mémoire après libération ou de mémoire non initialisée et les fuites mémoire. Le rapport donnée par valgrind n'est pas très convivial. Dans les salles libres services, vous trouverez un exécutable *valgrind* sur `/soft/enseignants/Emmanuel.Agullo/valgrind/bin/valgrind`.

1. Visualisez le programme *valgrindEx.c*, repérez les erreurs de la fonction *f*. Compiler (fichier *Makefile* fourni) et exécutez ce programme. Exécutez ce programme avec *valgrind*. Repérez dans le rapport de *valgrind* les erreurs que vous avez notées. Exécutez ce programme avec *valgrind* en mettant l'option permettant de visualiser où la fuite mémoire a lieu.
2. Valgrind permet aussi d'analyser le comportement des caches. Le cache est une portion de la mémoire rapidement accessible qui sert à stocker temporairement une petite partie de la mémoire. Lors de plusieurs accès successifs à une même donnée, si la donnée reste dans le cache, le programme est beaucoup plus rapide. Compiler et exécutez les programmes gentil.c et mechant.c. On remarquera que les deux programmes font la même chose, mais dans un ordre différent. À l'aide de la commande shell *time* mesurez la différence de temps d'exécution des deux programmes. D'où vient cette différence ? En déduire comment sont stockés les tableaux bi-dimensionnales (*i.e.* les matrices) en *C*. Notez que *valgrind* permet de vérifier cette hypothèse avec l'option *–tool=cachegrind* ("moins moins"). Interprétez les résultats.

Dans beaucoup de cas, valgrind permet de trouver les erreurs plus vite que gdb/ddd, le choix d'un outil particulier de debugage pour un bug particulier doit se faire en prenant en compte l'estimation de la difficulté du bug et le temps de mettre en place l'outil de debug. Souvent, une simple lecture très attentive du code suffit (ou une lecture par un tiers).

4 gprof

L'outil *gprof* permet d'identifier les fonctions pouvant nécessiter une optimisation. Il comptabilise le nombre d'accès total à une fonction et le nombre d'accès à une fonction par fonction appelante. Il stocke également les temps d'exécution de chaque fonction. Lors de la compilation, il est nécessaire d'ajouter au moins l'option *-p*. L'exécution du programme ainsi compilé va créer le fichier *gmon.out*. Ensuite, pour exploiter le fichier généré, il faut exécuter *gprof* suivi du nom de votre programme (*gmon.out* doit être dans le même répertoire que l'exécutable). Regardez rapidement le programme *testGprof.c* et en particulier la fonction *main* ; compilez le programme (*Makefile* fourni) et exécutez *gprof* dessus, analyser les résultat (la sortie de *gprof* est commentée).

5 Pour finir...

1. Modifier le programme tp0.c pour qu'il n'y ait plus de fuite mémoire.
2. Pour ceux qui ne sont toujours pas convaincu qu'il est important d'écrire lisiblement des programmes C, jetez un oeil au programme *phillipps.c* (obtenu sur `http://www.ioccc.org/`), que fait-il (exécutez-le pour vérifier votre hypothèse).

6 Autres points ?

C'est le moment de poser des questions sur des questions relatives à *C* / *Makefile* / `PATH` / *.bashrc* / ...

A Rendons à César

Le document original provient de la page web `http://perso.citi.insa-lyon.fr/trisset/cours/AGP/` et a été écrit par T. Risset.