GNU Debugger (gdb)

TP n°4 - Hoerdt Mickaël/Quirin Arnaud - Avril 2004

1 Manipulation de gdb

GDB est un debugger symbolique sous Unix permettant le contrôle de programmes C, C++, Pascal, Fortran.

gdb permet de :
- mettre des points d'arrêt dans le programme (breakpoints)
- visualiser l'état de la pile, des variables
- calculer des expressions
- modifier le contenu de variables
- ...

xxgdb : interface graphique qui facilite l'utilisation de gdb sous X-Windows. Malheureusement, elle n'est pas installé sous Ada.

1.1 Préliminaires

- Lancement du debugger :
  gdb <*nom_programme_executable*>
  (gdb) *commande*
- Lancement de l'exécution (sous gdb)
  (gdb) run <*arguments_du_programme*>

Le programme est exécuté comme s'il avait été lancé normalement. L'exécution se poursuit jusqu'à la fin du programme sauf si :
- des points d'arrêt ont été demandés (avant run)
- erreur du programme avec arrêt inopiné
- interruption de l'utilisateur (touche CTRL+C)

➜ arrêt de l'exécution sans quitter gdb, des commandes peuvent être saisies.

1.2 Points d'arrêt (breakpoint)

Permet d'interrompre l'exécution et revenir au mode commande à un endroit donné du programme.
- Arrêt début fonction : break <*nom_fonction*>
- Arrêt corps de la fonction : break <*numéro_ligne*> (exécution interrompue juste **avant** la ligne)
- Numéro de ligne obtenue à l'aide de la commande list (abrégée en l)
  (gdb) list
  (gdl) l

➜ affichage des 10 lignes qui suivent ou chevauchent le listing précédent.

(gdb) l <*nom_fonction*>
(gdb) l <*fichier : fonction*>

1.3 Contrôle de l'exécution

step : exécute une ligne de code (en entrant éventuellement dans les sous-fonctions)
next : exécute une ligne en sautant les procédures ou fonction
cont : continue jusqu'au prochain breakpoint

On n'utilise pas run.

1.4 Compléments sur les points d'arrêt

Afficher la liste des points d'arrêt : info breakpoints
Exemple de résultat :

Num	Type	Disp	In ?	Address	What
1	breakpoint	keep	y	0x0804885c	in mult at mult.c :13
2	breakpoint	keep	y	0x0804897b	in pro at pro.c :15
		breakpoint already hit 1 time			

(gdb) info breakpoint <*numéro*>

Exemple : info breakpoint 2

1.5 Effacer un point d'arrêt

clear <*numéro_ligne*> ex : clear pro.c :15
clear <*nom_fonction*> ex : clear pro
clear *<*adresse*> ex : clear *0x0804897b

1.6 Visualisation des variables

print <*nom_variable*>
print <*expression*>
print *<*nom_variable*> valeur pointée par un pointeur
display <*expression*> affichage de manière permanente (à chaque arrêt) du contenu de la va
ou de l'expression
undisplay <*expression*> annuler la demande d'affichage

Exemple :

display i	récupération du numéro de display 1
display j	récupération du numéro de display 2
undisplay 1	pour ne plus afficher le contenu de i

1.7 Modification d'une variable

set <*variable*>=<*expression*>

1.8 Pile d'appels

where : liste des procédures ou fonctions empilées jusqu'à celle où on est momentanément arrêté
up : pour remonter dans la liste
down : pour descendre d'un niveau dans la liste

1.9 Divers

Aide en ligne : help <*nom_ commande*>
Abréviations : run (r), break (b), step (s), next (n), continue (c), print (p)
Arrêt : quit (confirmation demandée si le programme est en cours d'exécution)

2 Exercice 1

- Copier dans votre dossier les fichiers defs.h, global.c, mult.c, produit.c, valeur.c qui se trouvent dans /users/prof/quirina/GL/TP4.
- Trouvez l'option de gcc (man gcc) pour permettre l'utilisation de gdb.
- Génerer le code executable a l'aide du makefile correspondant. Lors de l'execution du programme, la multiplication de 2 polynomes ne donne pas le resultat espere. Trouver la raison de cette erreur a l'aide de gdb.

3 Exercice 2

- Ecrivez un programme factorielle.c qui calcule une factorielle à l'aide de la fonction suivante :

```
unsigned int factor (unsigned int n)
{
  unsigned int i = 1, f = 1;

  while (i <= n)
      f = f * i++;
  return f;
}
```

- Placez un point d'arrêt avant l'appel à la fonction factor().
- Exécutez alors le programme en mode pas à pas.
- Affichez les valeurs des variables f et i de manière continue : leurs valeurs sont quelconques car elles n'ont encore pas été initialisées. Les variables locales sont en effet allouées automatiquement dans la pile, qui à cet instant de l'exécution peut contenir n'importe quoi. En continuant à exécuter en mode pas à pas, vous verrez les variables f et i évoluer au cours des itérations de la boucle.
- Relancez l'exécution du programme avec le même point d'arrêt. Modifier la valeur du paramètre passé à la fonction factor() par la fonction main() pendant une session gdb (par exemple à 12) et terminez son exécution (cont). Le programme a bien calculé la factorielle de 12.
- Recommencez avec n=13, et vérifiez que le résultat n'est pas égal à 13 fois le précédent. Pourquoi ?

4 Suite

- En vue de l'approche du contrôle continu, terminez le reste des TPs précédents.