Στοιχειώδης χρήση του GNU debugger (gdb)

Σ.Δ. Αγάθος, Β. Δημακόπουλος
http://paragroup.cse.uoi.gr

Έκδοση 0.1.0 — 27/10/2017

Βήματα βασικής αποσφαλμάτωσης κώδικα με χρήση gdb

1. Κάνω compile το πρόγραμμα με `-g`:

```
user@hostname$ gcc -g prog.c
```

2. Φορτώνω το πρόγραμμά μου στον gdb:

```
user@hostname$ gdb ./a.out
```

3. Με την εντολή `r` ή run ξεκινάμε την εκτέλεση του προγράμματος, αν το πρόγραμμα χρειάζεται ορίσματα (`arg1`, `arg2`, ..., `argN`) τα δίνουμε στην ίδια εντολή:

```
(gdb) r arg1 arg2 ... argN
```

4. Για να δούμε τις διαδοχικές κλήσεις συναρτήσεων του προγράμματος μέχρι να συμβεί το σφάλμα, εκτελούμε την εντολή `bt` ή `backtrace`

```
(gdb) bt
```

5. Για να βγούμε από τον debugger εκτελούμε την εντολή `q` ή `quit`

```
(gdb) q
```

Έστω για παράδειγμα ότι έχω το παρακάτω προβληματικό πρόγραμμα το οποίο κατά την εκτέλεσή του εμφανίζει segmentation fault.

```
1  #include <stdio.h>
2
3  void func(int *ptr) {
4    *ptr = 1;
5  }
6
7  void main() {
8    int *p = NULL;
9    func(p);
10 }
```

Για να βρούμε το λάθος τότε ακολουθώντας τα βήματα 1 έως 4 ο debugger θα μας δώσει το ακόλουθο μήνυμα:

```
(gdb) r
Starting program: ./a.out

Program received signal SIGSEGV, Segmentation fault.
0x080483e2 in func (ptr=0x0) at prog.c:4
4    *ptr = 1;
(gdb) bt
#0  0x080483e2 in func (ptr=0x0) at prog.c:4
#1  0x08048402 in main () at prog.c:9
```

Έτσι, εκτελώντας την εντολή `r`, ξεκινάει η εκτέλεση και πράγματι το πρόγραμμα προκαλεί segmentation fault. Όμως τώρα πληροφορούμαστε ότι η εντολή που προκάλεσε το σφάλμα είναι η εντολή στη γραμμή 4, η οποία ανήκει στη συνάρτηση `func()`. Η πληροφορία (`ptr=0x0`) είναι ιδιαίτερα σημαντική: ο debugger μας ενημερώνει ότι ο δείκτης με όνομα `ptr` έχει τιμή ίση με 0. Έτσι στην εντολή της γραμμής 4 όπου πάμε να προσβούμε τα δεδομένα της θέσης μνήμης 0 (η θέση 0 δεν μπορεί να προσπελαστεί από κανένα πρόγραμμα) προκαλούμε σφάλμα. Συνεχίζοντας, με την εντολή `bt` βλέπουμε όλες τις κλήσεις συναρτήσεων μέχρι να συμβεί το σφάλμα καθώς και τις τιμές που είχαν οι μεταβλητές που ήταν ορίσματα στις συναρτήσεις.

Breakpoints

Εάν θέλουμε να εξετάσουμε με λεπτομέρεια τί ακριβώς συμβαίνει σε μια συγκεκριμένη χρονική στιγμή του προγράμματός τότε μπορούμε να χρησιμοποιήσουμε breakpoints. Τα breakpoints είναι συγκεκριμένα σημεία στον κώδικα όπου όταν τα συναντήσει ο debugger τότε κάνει μια παύση στην εκτέλεση και περιμένει από εμάς εντολές. Για παράδειγμα, στην διάρκεια αυτής της παύσης (όπως θα δούμε με λεπτομέρεια παρακάτω) μπορούμε να δούμε τις τιμές που έχουν οι μεταβλητές του προγράμματος εκείνη την στιγμή. Τα breakpoints ορίζονται αφού έχει φορτωθεί ένα πρόγραμμα στον debugger. Για να ορίσουμε ένα breakpoint χρησιμοποιούμε την εντολή `b` ή `break`. Μπορούμε να θέσουμε breakpoints στην πρώτη εντολή του προγράμματος (`main()`), στην πρώτη εντολή μιας οποιασδήποτε συνάρτησης και φυσικά σε οποιαδήποτε γραμμή θέλουμε, για παράδειγμα:

```
(gdb) break main /* Breakpoint στην main */
Breakpoint 1 at 0x80483a5: file test.c, line 5.
(gdb) b test.c:6 /* Breakpoint στην γραμμή 6 */
Breakpoint 2 at 0x80483ac: file test.c, line 6.
(gdb) b func     /* Breakpoint στην συνάρτηση func */
Breakpoint 3 at 0x8048397: file test.c, line 2.
```

Για να δούμε την λίστα με όλα τα breakpoints που έχουν οριστεί τότε εκτελούμε την εντολή `info breakpoints`:

```
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x080483a5 in main at test.c:5
2 breakpoint keep y 0x080483ac in main at test.c:6
3 breakpoint keep y 0x08048397 in func at test.c:2
```

Μπορούμε να αφαιρέσουμε, να ενεργοποιήσουμε και να απενεργοποιήσουμε breakpoints με τις εντολές `delete` ή `del`, `enable` και `disable` αντίστοιχα. Για να προσδιορίσουμε το breakpoint που θέλουμε να διαγραφεί, να ενεργοποιηθεί ή να απενεργοποιηθεί χρησιμοποιούμε τον αύξων αριθμό του, που φαίνεται από την εντολή `info breakpoints`, στην πρώτη στήλη (`Num`):

```
(gdb) del 1
(gdb) disable 2
(gdb) info breakpoints
Num Type Disp Enb Address What
2 breakpoint keep n 0x080483ac in main at test.c:6
3 breakpoint keep y 0x08048397 in func at test.c:2
```

Μόλις θέσουμε τα breakpoints τότε ξεκινάμε την εκτέλεση του προγράμματος με την εντολή `r` ή `run`. Αυτό που θα δούμε είναι ότι η εκτέλεση κάνει παύση στο πρώτο breakpoint. Για να συνεχίσει η εκτέλεση πέρα από το breakpoint τότε χρησιμοποιούμε την εντολή `cont` ή `continue`. Η εκτέλεση του προγράμματος θα συνεχιστεί μέχρι το τέλος του προγράμματος ή μέχρι το επόμενο breakpoint:

```
Starting program: ./a.out

Breakpoint 1, main () at prog.c:25
25        i = 5;
(gdb) continue
Continuing.

Breakpoint 2, func1 (j=5) at prog.c:7
7         k = j + 5;
```

Αφού φτάσουμε σε ένα breakpoint μπορούμε να συνεχίσουμε την εκτέλεση του προγράμματος γραμμή-γραμμή ή εντολή-εντολή χρησιμοποιώντας τις εντολές `n` ή `next` και `s` ή `step`, αντίστοιχα:

```
Breakpoint 1, main () at prog.c:25
25        i = 5;
(gdb) next
26        j = 15;
(gdb) n
28        i = func1(i);
(gdb) s
func1 (j=5) at prog.c:7
7         k = j + 5;
(gdb) step
9         return k;
```

Επίσης, αφού φτάσουμε σε ένα breakpoint μπορούμε να συνεχίσουμε την εκτέλεση του προγράμματος μέχρι το τέλος μια συνάρτησης με την εντολή `finish`:

```
(gdb) r
Starting program: ./a.out

Breakpoint 1, func1 (j=5) at prog.c:7
7         k = j + 5;
(gdb) finish
Run till exit from #0  func1 (j=5) at prog.c:7
0x0000000000400556 in main () at prog.c:28
28        i = func1(i);
Value returned is $1 = 10
```

Στο σημείο αυτό μπορεί να παρατηρήσει κανείς ότι ο debugger εκτυπώνει και την τιμή της μεταβλητής που επιστρέφει η συνάρτηση που μόλις τελείωσε.

Εκτύπωση και ανάθεση τιμών μεταβλητών

Αφού φτάσουμε σε ένα breakpoint μπορούμε να εκτυπώσουμε την τιμή που έχει μια οποιαδήποτε μεταβλητή τη χρονική στιγμή του breakpoint. Σημείωση: Αν το breakpoint δείχνει σε μια γραμμή κώδικα, τότε οι τιμές που έχουν οι μεταβλητές θα είναι αυτές πριν εκτελεστεί η συγκεκριμένη γραμμή. Για να εκτυπώσουμε την τιμή χρησιμοποιούμε την εντολή `print`. Μπορούμε επίσης να θέσουμε νέα τιμή σε μια μεταβλητή με την εντολή `set var`:

```
(gdb) r
Starting program: ./a.out

Breakpoint 1, func1 (j=5) at prog.c:7
7         k = j + 5;
(gdb) print k
$4 = 0
(gdb) set var k=20
(gdb) print k
$5 = 20
```

Συνοπτικός πίνακας εντολών

Εντολή	Λειτουργία
`run` ή `r`	Εκκίνηση προγράμματος
`quit` ή `q`	Έξοδος από τον debugger
`backtrace` ή `bt`	Εμφάνιση κλήσεων συναρτήσεων
`continue` ή `cont`	Συνέχιση στο επόμενο breakpoint
`break` ή `b`	Εισαγωγή νέου breakpoint
`info breakpoints`	Εμφάνιση λίστας breakpoints
`next` ή `n`	Συνέχεια στην επόμενη γραμμή
`step` ή `s`	Συνέχεια στην επόμενη εντολή
`enable`	Ενεργοποίηση ενός breakpoint
`disable`	Απενεργοποίηση ενός breakpoint
`delete` ή `del`	Διαγραφή ενός breakpoint
`print`	Εκτύπωση τιμής μιας μεταβλητής
`set var`	Θέτει μια τιμή σε μεταβλητή

Επιπλέον λειτουργίες

Ο GNU debugger έχει πολλές επιπλέον δυνατότητες και χρήσιμες προχωρημένες λειτουργίες οι οποίες δεν είναι σκόπιμο να καλυφθούν εδώ. Εαν ενδιαφέρεστε, μπορείτε να ξεκινήσετε εκτελώντας τον `gdb` και δίνοντας την εντολή `help`.