Εισαγωγή στο debugging με τον gdb

Λίγα λόγια για τον compiler gcc

Η σχέση C και UNIX

Η ανάπτυξη της γλώσσας C είναι από την αρχή άρρηκτα συνδεδεμένη με το λειτουργικό σύστημα UNIX καθώς τα έφτιαξε και τα δύο η ίδια μικρή ομάδα επιστημόνων των Bell Labs, με βασικότερους τον Dennis Ritchie και τον Ken Thompson.

Το UNIX άρχισε να αναπτύσσεται την δεκαετία του ‘60 στα προαναφερθέντα Bell Labs και ήταν γραμμένο σε Assembly. Το πρώτο διαθέσιμο version UNIX εκτός του εργαστηρίου, το version 1.0, βγήκε το 1971.

Το 1972 ο Thompson ήθελε μια γλώσσα για να φτιάξει εφαρμογές (utilities) για τα τότε νεότερα μηχανήματα του lab σε σχέση με αυτά που είχαν τη δεκαετία του 60. Δοκίμασε τον compiler της Fortran αλλά δεν έμεινε ικανοποιημένος. Ο Richie, έχοντας τον ίδιο σκοπό, ξεκίνησε τις δοκιμές από τη γλώσσα BCPL, την οποία απλοποίησε και έφτιαξε την “B”. Επειδή ήταν αργή ο Richie άρχισε να την τροποποιεί ώστε να εκμεταλλευτεί τις δυνατότητες των νέων μηχανημάτων που είχαν στο lab και ιδιαίτερα το byte addressability, δηλαδή το να μπορείς να αποθηκεύσεις τιμές σε οποιοδήποτε τυχαίο σημείο της μνήμης (RAM). Έτσι δημιουργήθηκε η πρώτη version της C. Ο C compiler ήταν παρών στην version 2 του UNIX που κυκλοφόρησε το 1972.

Το 1973 στη version 3 του UNIX προστέθηκε ο C debugger και το 1974 στη version 4 τού ολόκληρο το UNIX ξαναγράφτηκε πλήρως σε C, όπως και παραμένει. Καθώς το λειτουργικό είναι γραμμένο σε C, ο ίδιος πηγαίος κώδικας μπορεί να γίνει compile σε διαφορετικά συστήματα (σε αντίθεση με την Assembly) και για τον λόγο αυτό το UNIX είναι το πρώτο “portable” λειτουργικό σύστημα.

gcc, g++ και GNU

Το καθεαυτό πρόγραμμα του compiler στα mostly POSIX compliant λειτουργικά είναι το GCC δηλαδή το το **G**NU **C**ompiler **C**ollection του GNU project.

Τα gcc και g++ είναι διαφορετικά “compiler invocation commands” ή “compiler-drivers” του GCC, δηλαδή ουσιαστικά καλούν τον gcc με διαφορετικά options.

Με g++ μπορούμε να κάνουμε compile οποιοδήποτε αρχείο γραμμένο σε C ή C++ αλλά θα το θεωρήσει και στις δύο περιπτώσεις προγράμματα C++ που σημαίνει ότι και τα δύο θα τα συνδέσει στο linking με τα libraries της C++ (και όχι της C για τα προγράμματα σε C).

Ως παραγόμενο αποτέλεσμα το g++ παράγει ακριβώς το ίδιο με το
gcc -xc++ -lstdc++ -shared-libgcc
Το -x καθορίζει τη γλώσσα (c++), το -l την βιβλιοθήκη με την οποία γίνεται το link (stdc++, standard c++ δηλαδή). -shared-lib σημαίνει ότι κάνουμε dynamic linking αντί static.

Άρα αν θες να χρησιμοποιήσεις τις βιβλιοθήκες της C και όχι C++ στο linking για ένα πρόγραμμα C πρέπει να χρησιμοποιήσεις την εντολή gcc και όχι την g++.

Το GNU GCC είναι ένα από τα πιο σημαντικά και ευρύτερα χρησιμοποιούμενα λογισμικά του κόσμου και έχει γίνει port πρακτικά σε όλες τις αρχιτεκτονικές υπολογιστών.

Το GCC υποστηρίζει και άλλες γλώσσες εκτός C (εξού και το “collection”) και είναι ένας optimizing compiler δηλαδή προσπαθεί να βελτιστοποιήσει τον παραγόμενο κώδικα μηχανής γενικά αλλά και για κάθε ειδική αρχιτεκτονική. Ως εκ τούτου ο gcc είναι ένα πολύ σύνθετο πρόγραμμα, δείτε για παράδειγμα μια περίληψη των options του.

Linux kernel

Ο πυρήνας του Linux είναι γραμμένος αποκλειστικά C. Τα options του gcc για το compilation του είναι

gcc -std=gnu89

-std ο gcc υποστηρίζει διάφορες διαλέκτους της γλώσσας που αντιστοιχούν σε διάφορα standards της γλώσσας. Εδώ το option έχει τιμή gnu89 που είναι η διάλεκτος C90 της γλώσσας.

Grader

Ο grader του μαθήματος κάνει compile ως εξής:

c++ -std=c++11 -O2 -DCONTEST -s -static

“c++” είναι ένα alias για τον c++ compiler του συστήματος που στην περίπτωση του συγκεκριμένου συστήματος είναι βέβαια ο GCC (το c++ είναι ίδιο με το g++ δηλαδή).

-std=c++11 η διάλεκτος αντιστοιχεί στο standard C++11

-O2 Optimization level 2. Είναι το τυπικό optimization level για release builds. Μια σύντομη εξήγηση για τα optimizations. Βλέπε και πιο κάτω σχετικά με τα debug και release builds.

-DCONTEST είναι ένα macro για τον προεπεξεργαστή (παράδειγμα)

-s μην βάλεις καθόλου σύμβολα debugging (βλ. πιο κάτω στο debugging)

-static στατική σύνδεση με τις βιβλιοθήκες (αντί δυναμικής) - υπενθύμιση.

Compiling and debugging

Log-message levels

Σε όλα τα UNIX-like συστήματα, ο πυρήνας (kernel) και οι βασικές εφαρμογές κατά την εκτέλεσή τους παράγουν μηνύματα εξόδου (logging messages) προκειμένου να ενημερώνουν για την λειτουργία τους (το σύστημα, άλλες εφαρμογές, τους χρήστες). Όταν τα μηνύματα αυτά καταγράφονται σε αρχεία κειμένου δημιουργούνται τα αρχεία καταγραφής (logfiles).

Τα log messages έχουν 7 επίπεδα σοβαρότητας (severity levels)

Value	Severity	Keyword	Deprecated keywords	Description
0	Emergency	`emerg`	`panic` [9]	System is unusable
1	Alert	`alert`		Action must be taken immediately
2	Critical	`crit`		Critical conditions
3	Error	`err`	`error` [9]	Error conditions
4	Warning	`warning`	`warn` [9]	Warning conditions
5	Notice	`notice`		Normal but significant conditions
6	Informational	`info`		Informational messages
7	Debug	`debug`		Debug-level messages

Μπορούμε να ζητήσουμε ένα πρόγραμμα να λειτουργήσει σε οποιοδήποτε log level. Αν πούμε “err” θα τυπώνει μόνο από σφάλματα και πάνω (0-3, το 0 είναι το υψηλότερο επίπεδο). Αν βάλουμε “warning” θα βγάζει και τις προειδοποιήσεις (0-4). Αν βάλουμε και notice θα τυπώνει τα 0-5. Παράδειγμα notice: “η συνάρτηση αυτή θα καταργηθεί σε επόμενο version” δηλαδή τώρα λειτουργεί κανονικά χωρίς warning αλλά πρέπει να λάβεις κάτι υπόψη σου στη συνέχεια για τη συντήρηση του προγράμματος. Τα info συνήθως ενημερώνουν ότι κάτι εκτελείται κανονικά πχ “τελείωσα αυτή τη δουλειά”. Τελος το debug, είναι ειδικό επίπεδο όπου εξάγονται πρόσθετα μηνύματα που βοηθούν στο debugging.

Warning options και printing μεταβλητών

Γνωρίζουμε ότι οταν ο compiler πέφτει σε errors πχ συντακτικά σφάλματα μας τα τυπώνει στην οθόνη και δεν παράγεται εκτελέσιμο αρχείο.

Ωστόσο, στην ανάπτυξη κώδικα βοηθάει πολύ να λαμβάνουμε υπόψη μας και μηνύματα για πράγματα που έχουμε στον κώδικά μας που ναι μεν δεν οδηγούν σε σφάλμα και το πρόγραμμα κάνει compile αλλά που μπορεί να είναι σοβαρές ενδείξεις επικείμενου λάθους, λογικού λάθους και τα λοιπά. Για το λόγο αυτό συνηθίζουμε να κάνουμε compile με το option -Wall δηλαδή -Warning(all,) δηλαδή όλα τα warnings.

Με -Wall και βάζοντας διάφορες εντολές εκτύπωσης συνήθως μεταβλητών μέσα στο πρόγραμμα είναι δυνατό να κάνεις debugging αλλά θα διαπιστώσουμε ότι συχνά μας εμφανίζονται προβλήματα για τα οποία τα warnings και το debugging via printing δεν μπορούν να μας βοηθήσουν.

Debugging

Το debugging είναι compiling όπου όμως ο gcc συμπεριφέρεται σε επίπεδο “debug”. Debug κάνουμε με το option -g (από το debu**g**). Παράδειγμα:

g++ -Wall -g

Εδώ κατά το compilation θα λάβουμε τα ίδια μηνύματα warnings απότι και χωρίς -g αλλά η σημαντική διαφορά είναι ότι θα παραχθεί ένα διαφορετικό executable που είναι το λεγόμενο debug version του εκτελέσιμου.

Το debug version θα είναι πιο αργό και πιο μεγάλο σε bytes αλλά θα περιέχει περισσότερες πληροφορίες και συγκεκριμένα θα περιλαμβάνει τα λεγόμενα debug symbols. Τα debug symbols είναι επιπλέον πληροφορίες που μπαίνουν στον πίνακα συμβόλων. Αυτός ο επαυξημένος πίνακας συμβόλων δεν χρησιμοποιείται μόνο κατά τη διάρκεια του compilation από τον assembler αλλά ενσωματώνεται και στα object files. Έτσι μπορούμε να χρησιμοποιήσουμε ένα ειδικό πρόγραμμα, τον debugger (στην περίπτωσή μας τον gdb) για να κάνουμε interactive debugging με το εκτελέσιμο αρχείο και να εντοπίσουμε σφάλματα που ξεφεύγουν από τα warnings και το print των μεταβλητών.

Debug and release builds

Τυπικά στην ανάπτυξη ενός προγράμματος δουλεύουμε με δύο versions του προγράμματος, το debug version και το production ή release build. Ο πηγαίος κώδικας είναι ο ίδιος αλλά αλλάζουν τα options που περνάμε στον g++. Στο debug version περνάμε το -g ενώ στο release δεν το περνάμε - αντίθετα περνάμε γενικά optimization options πχ -O2 ή optimization options για την στοχευόμενη αρχιτεκτονική μας (target architecture)..

Τυπικά κατά την ανάπτυξη δουλεύουμε κυκλικά και με τα δύο versions: ανάπτυξη -> debug -> release -> ανάπτυξη -> debug -> release κοκ

gdb intro

Ο gdb είναι λοιπόν ο συμβολικός interactive debugger του GNU Project. Υπενθυμίζουμε ότι δεν εγκαθίσταται μαζί με τον compiler αλλά με sudo apt install gdb.Το full documentation του είναι μεγάλης έκτασης.

Για μια γρήγορη εισαγωγή προτείνουμε αυτό το σύντομο tutorial. Κατεβάστε τον κώδικα (main.cpp) του “An Example Debugging Session” από εδώ. Φυσικά θα κάνετε compile με -Wall -g. Το πρόγραμμα δεν θα έχει κανένα warning αλλά δεν θα λειτουργεί σωστά.

Αν στο προηγούμενο παράδειγμα μπορεί κάποιος να πει ότι θα έβρισκε το πρόβλημα με εκτύπωση των μεταβλητών μέσα στο πρόγραμμα, αυτό δεν ισχύει για το crash.cpp, το πρώτο πρόγραμμα που μπορείτε να βρείτε και να τρέξετε από εδώ.

Τρέξτε το και με -Wall -g. Το πρόγραμμα αυτό προκαλεί core dump. Core dump σημαίνει ότι κάνει κάτι πολύ λάθος οπότε το λειτουργικό σύστημα διακόπτει την εκτέλεσή του με κάποιο signal και κάνει “dump” δηλαδή πετάει (γράφει) τα περιεχόμενα της RAM εκείνη τη στιγμή σε ένα αρχείο το dump file προκειμένου να γίνει debugging. Το ίδιο γίνεται και όταν προκύπτουν τα γνωστά segmentation faults που αφορούν σε λάθος χρήσεις των διευθύνσεων της μνήμης.

Αφού ακολουθήσετε τον οδηγό για το πρώτο παράδειγμα δοκιμάστε να αλλάξετε το crash.cpp βάζοντας κάτω από τη γραμμή 10 (x=3; y=0;) ένα cout << y;. Το y είναι ο “φταίχτης” που προκαλεί το divide by zero όμως λόγω του termination signal δεν εκτυπώνεται τίποτα στην έξοδο. Συνεπώς, αν το σφάλμα δεν ήταν τόσο προφανές και είχαμε core dump δεν θα μπορούσαμε να βρούμε το bug μόνο με cout μεταβλητών.

Το δεύτερο παράδειγμα δεν δημιουργεί core dump, είναι λανθασμένο αυτό που γράφει η ιστοσελίδα. Μάλιστα αν προσθέσεις -Wall τα warnings αρκούν για να λύσεις το πρόβλημα.