Πανεπιστήμιο Δυτικής Μακεδονίας
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα

Ενότητα 6: Compile πηγαίων αρχείων με gcc. Αποσφαλμάτωση προγράμματος με gdb. Δημιουργία σύνθετων προγραμμάτων με το make Ανάλυση εκτελέσιμου & βελτιστοποίηση (gprof, gcov).

Επιβλέπων: Δασυγένης Μηνάς
Παυλίδου Ελένη

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Λειτουργικών Συστημάτων
http://arch.ece.uowm.gr/courses/os/

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Compile πηγαίων αρχείων με gcc (1/2)

- Ας υποθέσουμε ότι κατεβάσαμε ένα tarball, το αποσυμπιέσαμε και τελικά φτάσαμε σε κάποιο README αρχείο που λέει ότι πρέπει να το κάνουμε compile, δηλαδή να μεταγλωττίσουμε το πρόγραμμα από τον κώδικά του για να παραχθεί ένα εκτελέσιμο αρχείο.
  Αν δεν το κάνουμε, απλά το πρόγραμμα δεν τρέχει.
  Γι' αυτό, θα χρειαστούμε ένα μεταγλωττιστή (compiler) στο Linux. Και οι πιο γνωστοί και καλύτεροι compilers στο Linux είναι η συλλογή μεταγλωττιστών GNU (GNU Compiler Collection), πιο γνωστή ως GCC.

Compile πηγαίων αρχείων με gcc (2/2)

- Ένας μεταγλωττιστής (compiler) δρα σαν μεταφραστής ανάμεσα στον άνθρωπο και την μηχανή, γι' αυτό και λέγεται μετα-γλωττιστής. Οι μεταγλωττιστές μπορούν συνήθως να υποστηρίζουν αρκετές γλώσσες προγραμματισμού (με τα front-ends) και διαφορετικές αρχιτεκτονικές (με τα λεγόμενα back-ends). Για παράδειγμα, ο GCC έχει front-ends για C, C++, Java, Objective C, ADA και Fortran, ενώ έχει back-ends για αναρίθμητες αρχιτεκτονικές και σχεδόν όλα τα γνωστά λειτουργικά συστήματα: Linux, Mac OS X, BSD, Unix παράγωγα αλλά και τις σύγχρονες εκδόσεις των Windows.

Μεταγλώττιση πηγαίων αρχείων (1/2)

- Για τη μεταγλώττιση απλά καλούμε το μεταγλωττιστή με την εντολή `gcc` ακολουθούμενη από το όνομα του προς μεταγλώττιση προγράμματος: `gcc program.c` όπου *`program.c`* είναι το όνομα του αρχείου κειμένου του προγράμματος.

- Αν υπάρχουν προφανή συντακτικά λάθη στο πρόγραμμα, ο μεταγλωττιστής τα αναγνωρίζει και τα εμφανίζει στην πρότυπη έξοδο. Φυσικά, υπάρχουν σφάλματα που ο μεταγλωττιστής δεν μπορεί να αναγνωρίσει, όπως τα λογικά σφάλματα.

Μεταγλώττιση πηγαίων αρχείων (2/2)

- Αν η μεταγλώττιση είναι επιτυχής, το εκτελέσιμο αποθηκεύεται στον τρέχοντα κατάλογο με όνομα *a.out* εκτός αν χρησιμοποιηθεί η επιλογή *-o* : τότε χρησιμοποιείται το όνομα που ακολουθεί την επιλογή *-o*. Είναι πιο βολικό να χρησιμοποιούμε την επιλογή *-o* όπως

```
gcc -o program program.c
```

που θέτει το εκτελέσιμο στο αρχείο *program* (ή ότι όνομα βάλετε μετά το όρισμα "-o") **αντί** για το αρχείο *a.out*.

Εκτέλεση πηγαίων αρχείων (1/2)

- Το επόμενο στάδιο είναι η εκτέλεση του εκτελέσιμου. Στο UNIX / Linux απλά εισάγουμε στη γραμμή εντολής το όνομα του εκτελέσιμου *program* (ή *a.out*). (Σιγουρευόμαστε ότι έχουμε δικαιώματα εκτέλεσης στο πρόγραμμα, αν όχι μεταβάλλουμε τα σχετικά δικαιώματα με την εντολή *chmod)*. Σε περίπτωση που ο τρέχον κατάλογος δεν είναι δηλωμένος στη μεταβλητή περιβάλλοντος *PATH,* δηλαδή θα προσθέσουμε την τρέχουσα διαδρομή **.**, ο φλοιός δεν θα βρει το εκτελέσιμο αρχείο.

Εκτέλεση πηγαίων αρχείων (2/2)

- Έχουμε δύο λύσεις, είτε θα εκτελέσουμε το πρόγραμμα γράφοντας `./program ή ./a.out` είτε θα ενημερώσουμε τη μεταβλητή περιβάλλοντος *PATH*.
  Κατά την εκτέλεση του προγράμματος τα αποτελέσματα εμφανίζονται στην οθόνη.
  Σε περίπτωση σφαλμάτων θα λάβουμε λάθος ή καθόλου έξοδο.
  Τότε επιστρέφουμε στο κείμενο, διορθώνουμε, μεταγλωττίζουμε και εκτελούμε.

Χρήσιμες Επιλογές Μεταγλώττισης (1/2)

- Τώρα που έχουμε μια βασική κατανόηση της διαδικασίας μεταγλώττισης μπορούμε να εισάγουμε μερικές χρήσιμες επιλογές μεταγλώττισης.

- `-c`: Ακυρώνει τη διαδικασία σύνδεσης και παράγει ένα αρχείο αντικείμενου κώδικα `.o` για κάθε αρχείο πηγαίου κώδικα που αναφέρεται. Στη συνέχεια, πολλά αρχεία αντικείμενου κώδικα μπορούν να συνδεθούν με την εντολή `gcc`, π.χ.:
  ```
  gcc file1.o file2.o ... -o executable
  ```

- `-g`: Επιλογή εκσφαλμάτωσης. Δίνει εντολή στον μεταγλωττιστή να παράξει πρόσθετη πληροφορία για τον πίνακα συμβόλων που είναι χρήσιμη στις λειτουργίες εκσφαλμάτωσης.

Χρήσιμες Επιλογές Μεταγλώττισης (2/2)

- `-pg`: Επιλογή εκσφαλμάτωσης και profiling.

- `-D`: Ορισμός συμβόλων ως ονόματα (-D*identifer*) ή ως τιμές (-D*symbol=value*) παρόμοια όπως η οδηγία προεπεξεργαστή `#define`.

- `-llibrary`: Σύνδεση αντικείμενων βιβλιοθηκών.
  Αυτή η επιλογή πρέπει να ακολουθεί τα ορίσματα του πηγαίου κώδικα. Η πιο συνηθισμένη βιβλιοθήκη είναι των μαθηματικών συναρτήσεων (math.h). Πρέπει να τη συνδέσουμε ρητά (μην ξεχνάτε την οδηγία `#include <math.h>`)
  π.χ.: `gcc calc.c -o calc -lm`

Segmentation fault (1/2)

- Ένα σφάλμα κατάτμησης ή segfault είναι ένα σφάλμα μνήμης στο οποίο ένα πρόγραμμα προσπαθεί να αποκτήσει πρόσβαση σε μια διεύθυνση μνήμης που δεν υπάρχει ή το πρόγραμμα δεν έχει δικαιώματα πρόσβασης. Είναι ένα κοινό σφάλμα σε κακά γραπτά προγράμματα C και C ++. Όταν ένα πρόγραμμα χτυπά ένα σφάλμα κατάτμησης, συχνά συντρίβεται με τη φράση λάθους "Segmentation fault".

- Προκειμένου να μπούμε στην κατάσταση αποσφαλμάτωσης τότε θα πρέπει να κάνουμε μετάφραση τον πηγαίο κώδικα με το να δώσουμε στο `gcc` επιπρόσθετα την παράμετρο `-g`.

Segmentation fault (2/2)

- Η πλήρης εντολή για τη μεταφόρτωση του `egdb` με την προσθήκη επιπρόσθετου κώδικα αποσφαλμάτωσης είναι:

```
gcc -g egdb.c -o egdb
```

- Αφού εκτελέσουμε την προηγούμενη εντολή με επιτυχία, το επόμενο βήμα είναι να το φορτώσουμε στον αποσφαλματωτή. Αυτό γίνεται με την εντολή:

```
gdb egdb
```

Εντολές του gdb (1/3)

- Ο GNU debugger έχει πολλές δυνατότητες και χρήσιμες λειτουργίες. Μερικές από αυτές αναφέρονται παρακάτω.

- `run` ή `r` : Εκκίνηση προγράμματος.

- `run –v` : Εκτελεί το πρόγραμμα που έχει ήδη φορτωθεί, με δεδομένες παραμέτρους.

- `quit` ή `q` : Έξοδος από τον debugger.

- `backtrace` ή `bt` : Εμφάνιση κλήσεων συναρτήσεων.

Εντολές του gdb (2/3)

- `continue` ή `cont` : Συνέχιση στο επόμενο 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.

Εντολές του gdb (3/3)

- `break` ή `b` : Εισαγωγή νέου breakpoint.
- `delete` ή `del` : Διαγραφή ενός breakpoint.
- `print` : Εκτύπωση τιμής μιας μεταβλητής.
- `set var` : Θέτει μια τιμή σε μεταβλητή.
- `info registers` : Εμφανίζει όλους τους καταχωρητές.
- `bt` : Ανίχνευση κλήσεων (backtrace) όταν το πρόγραμμα έχει τερματιστεί με σφάλμα.

Signals (1/5)

- Μέσω των signals μία διεργασία μπορεί να ειδοποιηθεί για διάφορα γεγονότα Τα signals ονομάζονται και “software interrupts”.

- Signals μπορούν να σταλούν από μία διεργασία σε μία άλλη διεργασία ή από τον kernel σε μία διεργασία.

- Τα signals μπορούν να σταλούν με την εντολή `kill`:

- Από τη γραμμή εντολών: `kill –sigid –pid`
- Σε ένα C πρόγραμμα:

```
int kill(pid_t pid, int sigid);
```

Signals (2/5)

- Κάθε signal έχει εξ'ορισμού του μία (προ)καθορισμένη συμπεριφορά.

Signal	id	default
SIGKILL	9	Terminate
SIGALRM	14	Terminate
SIGSYS	12	Core
SIGCHLD	20	Ignore

Signals (3/5)

- Στο πρόγραμμά μας μπορούμε να καθορίσουμε επ'ακριβώς την συμπεριφορά μίας διεργασίας, ύστερα από την λήψη ενός signal:

  - Εκτέλεση μίας συνάρτησης (signal handler).
  - Καμία ενέργεια (`SIG_IGN`).
  - Προκαθορισμένη συμπεριφορά (`SIG_DFL`).

- Τα σήματα `SIGKILL` και `SIGSTOP` δεν μπορούν να παγιδευτούν, να μπλοκαριστούν ή να αγνοηθούν.

Signals (4/5)

- Το σήμα `SIGKILL` αποστέλλεται σε μια διαδικασία που το αναγκάζει να τερματιστεί αμέσως. Σε αντίθεση με το `SIGTERM` και το `SIGINT` αυτό το σήμα δεν μπορεί να ληφθεί ή να αγνοηθεί και η διαδικασία λήψης δεν μπορεί να πραγματοποιήσει κανένα καθαρισμό κατά τη λήψη αυτού του σήματος.

- Το σήμα `SIGSTOP` δίνει εντολή στο λειτουργικό σύστημα να σταματήσει μια διαδικασία για μεταγενέστερη επανάληψη.

- Το `SIGTERM` είναι ένα γενικό σήμα που χρησιμοποιείται για τον τερματισμό του προγράμματος. Αντίθετα με το `SIGKILL`, αυτό το σήμα μπορεί να αποκλειστεί, να αντιμετωπιστεί και να αγνοηθεί.

Signals (5/5)

- Το `SIGHUP` ("hang-up") χρησιμοποιείται για την αναφορά αποσύνδεσης του τερματικού του χρήστη, ίσως επειδή έχει διακοπεί η σύνδεση δικτύου ή τηλεφώνου.

- Αυτό το σήμα χρησιμοποιείται επίσης για την αναφορά του τερματισμού της διαδικασίας ελέγχου σε ένα τερματικό στις εργασίες που σχετίζονται με αυτή τη σύνοδο. Αυτός ο τερματισμός αποσυνδέει αποτελεσματικά όλες τις διαδικασίες στη συνεδρία από το τερματικό ελέγχου.

Το εργαλείο make

- Το εργαλείο *make* χρησιμοποιείται για τη διαχείριση μεγάλων εργασιών με πολλά αρχεία. Το *make* ελέγχει ποια αρχεία έχουν αλλάξει και τα μεταγλωττίζει αυτόματα όταν χρειάζεται.

- Οι πληροφορίες για τον τρόπο που το *make* μεταγλωττίζει τα διάφορα αρχεία περιέχονται σε ένα αρχείο που ονομάζεται *Makefile*.
  Το *Makefile* περιέχει κανόνες, μεταβλητές, σχόλια, γενικούς κανόνες και οδηγίες.

Κανόνες (1/2)

- Οι κανόνες είναι ο κορμός του *Makefile*. Η μορφή κάθε κανόνα είναι η παρακάτω :

```
<target> : <dependencies>
<action1>
<action2>...
```

- Ο κανόνας ορίζει τρία πράγματα. Το όνομα του αρχείου που πρέπει να ανανεωθεί (*στόχος - target*), πότε πρέπει να ανανεωθεί (*εξαρτήσεις - dependencies*) και πως θα ανανεωθεί (*ενέργειες - actions*).

Κανόνες (2/2)

- Έστω ότι έχουμε τα αρχεία `list.C` και `list.h`.
  Ένας κανόνας για να δημιουργούμε το list.o είναι:

```
list.o : list.C list.h
    gcc -c list.C
```

Αν το εκτελέσιμο της εργασίας ονομάζεται `test_data_structures` και περιέχει ένα επιπλέον αρχείο `main.C`, τότε για να το δημιουργήσουμε αρκεί να προσθέσουμε στην αρχή του αρχείου *`Makefile`* τους κανόνες:

test_data_structures : `list.o main.o`

```
    gcc -o test_data_structures list.o main.o
main.o : main.C g++ -c main.C
```

Έτσι όταν δίνουμε την εντολή `make`, θα μεταγλωττίζεται μόνο όποιο αρχείο είναι νεότερο από το εκτελέσιμο.

Σχόλια

- Τα σχόλια στο *Makefile* εισάγονται με τη χρήση του χαρακτήρα *#*.
- Ο χαρακτήρας *#* και όλοι οι χαρακτήρες μέχρι το τέλος της γραμμής αγνοούνται.
- Τα σχόλια μπορούν να εμφανίζονται οπουδήποτε στο *Makefile*.

- Παραδειγμα: `LEDA_LIBS = -lL -lm # Link with the LEDA library of basic data types`

Ειδικοί κανόνες και ορίσματα γραμμής εντολών.

- Το όνομα ενός *στόχου (target)* δεν είναι ανάγκη να είναι κάποιο αρχείο. Μπορούμε να ορίσουμε έναν ειδικό κανόνα που όταν κληθεί θα κάνει μια εργασία διαφορετική από μεταγλώττιση. Ο συνηθέστερος τέτοιος κανόνας είναι αυτός που ορίζουμε για να διαγραφούν ορισμένα αρχεία όταν δεν χρειάζονται `clean`:

```
rm -f *.o test_data_structures core
```

- Τέτοιους κανόνες τους καλούμε δίνοντας το όνομα του στόχου σαν όρισμα στη γραμμή εντολών αμέσως μετά το *`make`*, π.χ.: `make clean`

Μεταβλητές

- Στην αρχή κάθε `Makefile` μπορούμε να ορίσουμε μεταβλητές για να αποφεύγουμε περιττές επαναλήψεις και να το κάνουμε πιο ευανάγνωστο. Οι ορισμοί των μεταβλητών είναι της μορφής:

  `<ονομα_μεταβλητης> = <κειμενο>`

- Αν υπάρχουν κενά στην αρχή του κειμένου αγνοούνται. Οι μεταβλητές μπορούν να χρησιμοποιηθούν οπουδήποτε (ακόμη και για τον ορισμό άλλων μεταβλητών). Στην τιμή μιας μεταβλητής αναφερόμαστε με τη σύνταξη:

  `$(<ονομα>)` ή `${<ονομα>}`

gcov και gprof (1/4)

- Το `gcov` παράγει μια «coverage analysis» ενός ειδικά εγκεκριμένου προγράμματος .Οι επιλογές `-fprofile-arcs` και `-ftest-coverage`, χρησιμοποιούνται για την κατάρτιση του προγράμματος για ανάλυση κάλυψης (η πρώτη επιλογή για την καταγραφή των στατιστικών κλάδων και η δεύτερη για την αποθήκευση του αριθμού εκτέλεσης γραμμών). Η επιλογή `-fprofile-arcs` χρησιμοποιείται επίσης για τη σύνδεση του προγράμματος. Για να ενεργοποιήσουμε τη δοκιμή κάλυψης πληκτρολογούμε:

```
gcc -Wall -fprofile-arcs -ftest-coverage cov.c
```

όπου `cov.c` είναι το όνομα του αρχείου προγράμματος.

gcov και gprof (2/4)

- Μετά την εκτέλεση αυτού του προγράμματος, θα δημιουργηθούν αρκετά αρχεία με επεκτάσεις "`.bb`" "`.bbg`" και "`.da`", τα οποία αναλύονται από το `gcov`. Παίρνει αρχεία προέλευσης και παράγει μια λίστα σχολιασμών. Κάθε γραμμή πηγαίου κώδικα έχει προθέματα με τον αριθμό των φορών που έχει εκτελεστεί. Οι γραμμές που δεν έχουν εκτελεστεί έχουν πρόθεμα "`#####`". Το `gcov` δημιουργεί ένα αρχείο καταγραφής που ονομάζεται *`sourcefile.gcov`* και υποδεικνύει πόσες φορές έχει εκτελεστεί κάθε γραμμή ενός *`sourcefile.c`*. Αυτό το σχολιασμένο αρχείο προέλευσης μπορεί να χρησιμοποιηθεί από το `gprof`.

gcov και gprof (3/4)

Μερικές επιλογές γραμμής εντολών `gcov`.

-*h* (*--help*) : Εμφανίζει βοήθεια σχετικά με τη χρήση του `gcov` και πραγματοποιεί έξοδο χωρίς να κάνει περαιτέρω επεξεργασία.

-b (branch - probabilities) : Καταγράφει τις συχνότητες βρόχων στο αρχείο εξόδου και γράφει τις πληροφορίες περί συνόλου των βρόχων στην τυπική έξοδο. Αυτή η επιλογή επιτρέπει να δούμε πόσο συχνά έχει ληφθεί κάθε βρόχος στο πρόγραμμά μας.

gcov και gprof (4/4)

- Λίγες πληροφορίες για το `gprof`:

- Το `gprof` είναι ένα performance analysis εργαλείο.
  Χρησιμοποίησε έναν συνδυασμό δειγμάτων και δημιουργήθηκε ως εκτεταμένη έκδοση του παλαιότερου εργαλείου "`prof`".
  Σε αντίθεση με το `prof`, το `gprof` είναι ικανό να συλλέγει και να εκτυπώνει περιορισμένα call graphs.

Τέλος Ενότητας

Πανεπιστήμιο Δυτικής Μακεδονίας