ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Λειτουργικά Συστήματα

**Ενότητα:** ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:06

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

**Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών**

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

# 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 του Πανεπιστημίου 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.



# Περιεχόμενα

# 1. Σκοπός της άσκησης

- Compile πηγαίων αρχείων με το gcc.
- Αποσφαλμάτωση προγράμματος με gdb.
- Δημιουργία σύνθετων προγραμμάτων με το make.
- Δημιουργία χάρτη μνήμης. Τμήματα προγραμμάτων.
- Ανάλυση εκτελέσιμου & βελτιστοποίηση *(gprof, gcov).*

# 2. Παραδοτέα

**(A)** 61 ερωτήσεις
**(C)** 2 ασκήσεις

# 3. Compile πηγαίων αρχείων με το gcc

Χρησιμοποιώντας ένα οποιοδήποτε επεξεργαστή ASCII κειμένου στο Linux *(για παράδειγμα το pico)* γράψτε το παρακάτω πρόγραμμα:

```
/* hello.c Example code */
#include <stdio.h>
#include <stdlib.h>

void subroutine(const char *message)
{
printf(message);
}

int main(void)
{
const char *greeting = "Hello from subroutine\n";
printf("Hello World from main\n);
subroutine(greeting);
printf("And Goodbye from main\n\n");
return 0;
}
```

και αποθηκεύστε το ως `AM.c` όπου ΑΜ είναι ο αριθμός μητρώου σας.
ΠΡΟΣΟΧΗ: **Το 2015 το FreeBSD αποφάσισε να χρησιμοποιεί και να υποστηρίζει τον μεταγλωττιστή `clang` σε αντίθεση με το linux που χρησιμοποιεί το `gcc`. Σε περίπτωση που εργάζεστε σε FreeBSD μηχάνημα (όπως το zafora) θα πρέπει να αντικαταστήστε τις εντολές `gcc` με τις εντολές `clang` (οι παράμετροι παραμένουν ίδιοι). Σε περίπτωση που εργάζεστε σε Linux μηχάνημα (όπως το pleiades), δε χρειάζεται κάποια αλλαγή.**

Μεταφράστε το παραπάνω πρόγραμμα ως:
`gcc AM.c -o AM`
*(η παράμετρος -o δείχνει το όνομα αρχείου εξόδου)*

## 3.1 Ερωτήσεις Α1 έως Α5

Σας εμφανίζει κάποιο μήνυμα λάθους; Ποιο είναι αυτό; _____**(Α1)**

Κάντε τις απαραίτητες διορθώσεις και επαναλάβετε μέχρι να ολοκληρωθεί το compile με επιτυχία.

Στο τέλος εκτελέστε το ως
`./AM`
Τι κάνει το παραπάνω πρόγραμμα; _____ **(Α2)**
Δώστε `ls -l ./AM` και σημειώστε το μέγεθος του αρχείου: _____ **(Α3)**
Στο παραπάνω παράδειγμα δώστε επιπρόσθετα την παράμετρο -g κατά τη μετάφραση στο gcc και σημειώστε το νέο μέγεθος αρχείου *(εφόσον γίνει compile χωρίς λάθη ):*_____ **(Α4)**

Η παράμετρος `-g` προσθέτει πληροφορίες που θα βοηθήσουν στην αποσφαλμάτωση *(debugging)* του προγράμματος. Συνήθως χρησιμοποιούμε αυτήν την παράμετρο μέχρι να φτάσουμε σε μια ολοκληρωμένη μορφή του ενσωματωμένου προγράμματος και στη συνέχεια την απομακρύνουμε για να μειωθεί το μέγεθος.

Συνήθως τα προγράμματα μεταγλώττισης δέχονται παραμέτρους βελτιστοποίησης που μερικές φορές αυξάνουν το μέγεθος του αρχείου, αλλά αυξάνεται και η ταχύτητα εκτέλεσης.

Ποιες παραμέτρους δέχεται το GCC για βελτιστοποίηση *(χρησιμοποιήστε το **man**, οι παράμετροι ξεκινάνε με O από το **Optimization**)*; _____ **(Α5)**

Κάντε compile το πρόγραμμά σας με κάθε μια παράμετρο ξεχωριστά και σημειώστε το μέγεθος που προκύπτει. Εξηγείστε γιατί συμβαίνει αυτό και συμπληρώστε τον παρακάτω πίνακα:**(Α6)**

| **Παράμετρος βελτιστοποίησης** | **Μέγεθος εκτελέσιμου** | **Τι κάνει η συγκεκριμένη παράμετρος;** |
|---|---|---|
| | | |
| | | |

# 4. Αποσφαλμάτωση Προγράμματος με gdb

Στην προηγούμενη παράγραφο είδαμε πως γίνεται η συγγραφή ενός προγράμματος, το compile και η διόρθωση συντακτικών σφαλμάτων. Τώρα θα γνωρίσουμε το πρόγραμμα αποσφαλμάτωσης *(debugger)* gdb *(**G**nu **D**e**B**ugger)*.

Δημιουργήστε έναν κατάλογο εργασίας.
Τοποθετήστε μέσα στον κατάλογο το αρχείο **egdb.c** από το συγκεκριμένο εργαστήριο. Ανοίξτε τον κώδικα *(χωρίς να το μεταφορτώσετε).*

## 4.1 Ερωτήσεις Α7 έως Α12

Χωρίς να μεταφορτώσετε τον κώδικα τι σας φαίνεται να κάνει το συγκεκριμένο πρόγραμμα;_____**(Α7)**
Ποια είναι η λειτουργία του `-o` *(παύλα μικρό ο)*;_____ **(Α8)**
Κάντε μετάφραση το συγκεκριμένο πρόγραμμα χρησιμοποιώντας το πρόγραμμα gcc. Το εκτελέσιμο που θα προκύψει να ονομαστεί **egdb** *(να χρησιμοποιήσετε την κατάλληλη παράμετρο)*.
Δώστε την πλήρη εντολή που χρησιμοποιήσατε:_____**(Α9)**
Αν προσπαθήσετε να το εκτελέσετε ως egdb δε θα το καταφέρετε *(θα αναφέρει Command not found).* Αν προσπαθήσετε να το εκτελέσετε ως **./egdb** θα το καταφέρετε. Γιατί; _____ **(Α10)**
Όταν εκτελέσετε το πρόγραμμα θα δείτε ότι αναφέρει κάτι που ίσως δεν είναι αρκετά ενημερωτικό:

```
Segmentation fault
ή
Segmentation fault (coredump)
```

Όταν εμφανίζεται η λέξη **fault** κατά την εκτέλεση ενός προγράμματος, τότε θα πρέπει να ψάξουμε να βρούμε το πρόβλημα. 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.

Ποια είναι η πλήρης εντολή για τη μεταφόρτωση του egdb με την προσθήκη επιπρόσθετου κώδικα αποσφαλμάτωσης;_____ **(Α11)**

Αφού εκτελέσετε την προηγούμενη εντολή με επιτυχία, το επόμενο βήμα είναι να το φορτώσουμε στο αποσφαλματωτή. Αυτό γίνεται με την εντολή:

```
gdb egdb
```

Αφού τυπωθούν κάποια μηνύματα βρισκόμαστε στην προτροπή:
**(gdb)**

Από εδώ και πέρα θα δίνουμε εντολές προς τον debugger. Χρησιμοποιώντας το συνοδευτικό αρχείο **gdb-refcard.pdf** με τις συνοπτικές λειτουργίες του gdb, βρείτε την εντολή που εκκινεί το πρόγραμμα.

Ποια είναι η εντολή του gdb που θα ξεκινήσει (θα εκτελέσει) το πρόγραμμα που έχουμε ορίσει; _____**(Α12)**

Μόλις εκτελεστεί το πρόγραμμα *(αφού δώσετε την παραπάνω εντολή)* και δημιουργηθεί το σφάλμα, θα δούμε ένα μήνυμα παρόμοιο με:

```
Program received signal SIGSEGV, Segmentation fault.
0x080485a0 in transformMatrix (matrix=0xfff6a028) at
egdb.c:57
   57 matrix[i][j] = bimod(matrix[i][j]);
```

ή

```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040079f in transformMatrix (matrix=0x7fff101bf6f0) at egdb.c:57
57 matrix[i][j] = bimod(matrix[i][j]);
```

## 4.2 Ερωτήσεις Α13 έως Α28

Διαβάστε προσεκτικά το μήνυμα και απαντήστε. Ποιο σήμα στάλθηκε στη διεργασία μας; Ποιος έστειλε το σήμα; _____ **(Α13)**

Δώστε man signal για να δείτε τα σήματα που υποστηρίζει το λειτουργικό σύστημα. Το σήμα που στάλθηκε στη διεργασία μας τι αριθμό έχει;_____ **(Α14)**

Ποια είναι η προεπιλεγμένη ενέργεια για το συγκεκριμένο σήμα σύμφωνα με τη σελίδα βοήθειας *(στα ελληνικά);* _____**(Α15)**

Ποια είναι η περιγραφή για το συγκεκριμένο σήμα *(ελληνικά);*_____ **(Α16)**

Διαβάζοντας τη σελίδα βοήθειας *(man)* των σημάτων σημειώστε τουλάχιστον 2 σήματα τα οποία δε μπορούμε να τα χειριστούμε στη διεργασία μας ή να τοποθετήσουμε κατάλληλο κώδικα για να τα αγνοήσουμε: _____ **(Α17)**

Δώστε μια περιγραφή για τα 2 παραπάνω σήματα για το τι κάνουν στη διεργασία: _____ **(Α18)**

Διαβάζοντας τη σελίδα βοήθειας *(man)* των σημάτων σημειώστε τουλάχιστον 2 σήματα τα οποία μπορούμε να τα χειριστούμε στη διεργασία μας ή να τα αγνοήσουμε με κατάλληλο κώδικα:_____ **(Α19)**

Παρατηρώντας το gdb βλέπουμε ότι μας ενημερώνει για την εμφάνιση του fault στη γραμμή 57 του πηγαίου προγράμματος example.c στη διεύθυνση 0x080485a0 στη συνάρτηση transformMatrix *(ο αριθμός γραμμής ή η διεύθυνση μπορεί να είναι διαφορετικά στο δικό σας ΛΣ).*

Ποια είναι η εντολή του gdb που θα εμφανίσει τις 10 προηγούμενες γραμμές από τη γραμμή που προκάλεσε την κατάρρευση *(δείτε συνοπτικό οδηγό)*; _____ **(Α20)**

Ποια είναι η εντολή του gdb που θα εμφανίσει τις 10 επόμενες γραμμές από τη γραμμή που προκάλεσε την κατάρρευση *(δείτε συνοπτικό οδηγό);* _____ **(Α21)**

Εμφανίστε τις 10 επόμενες και τις 10 προηγούμενες γραμμές δίνοντας τις παραπάνω 2 εντολές.

Ποια είναι η εντολή του gdb που θα τοποθετήσει ένα σημείο παύσης (breakpoint) σε κάποια συνάρτηση; (δείτε συνοπτικό οδηγό)

_____ **(Α22)**

Ποια είναι η εντολή που θα τοποθετήσει ένα σημείο παύσης στη συνάρτηση που μας έχει εμφανίσει το fault;

_____ **(A23)**

Εκτελέστε την προηγούμενη εντολή. Αν γίνει με επιτυχία θα δείτε:

```
Breakpoint 1 at 0x40074e: file egdb.c
```

Εκτελέστε πάλι το πρόγραμμα, πατώντας το **`run`**.
Αν εμφανιστεί η ερώτηση ότι έχει ήδη ξεκινήσει η αποσφαλμάτωση και αν θέλουμε να αρχίσουμε τη διαδικασία από την αρχή πατήστε **`y`** και enter.
Μόλις κληθεί η συνάρτηση που έχουμε θέσει το breakpoint θα σταματήσει προσωρινά η εκτέλεση και θα εμφανιστεί η γραμμή εντολών *(gdb).*

Ποια είναι η εντολή του gdb που θα εκτελέσει την επόμενη γραμμή μαζί με τη συνάρτηση; *(δείτε συνοπτικό οδηγό)*

_____ **(A24)**

Εκτελέστε την ανωτέρω εντολή.
Αντί για την ανωτέρω εντολή μπορούμε να χρησιμοποιήσουμε και το **step** αλλά το **step** εκτελεί βηματικά και τις εντολές της καλούμενης συνάρτησης *(step into function)* ενώ η εντολή **(A24)** κάνει εκτέλεση της καλούμενης συνάρτησης χωρίς να εισέρχεται μέσα *(step over function).* Αν η επόμενη εντολή προς εκτέλεση ΔΕΝ είναι συνάρτηση η **(A24)** εντολή είναι ισοδύναμη με το step.

Ποια είναι η εντολή του gdb που θα απομακρύνει *(σβήσει)* το breakpoint που έχουμε τοποθετήσει; *(δείτε συνοπτικό οδηγό)*

_____ **(A25)**

Εκτελέστε την ανωτέρω εντολή αφού δε μας χρειάζεται πια το breakpoint, μια που έχουμε μπει μέσα στην κρίσιμη συνάρτηση.
Ποια είναι η εντολή του gdb που θα συνεχίσει την εκτέλεση από το σημείο που βρισκόμαστε;_____**(A26)**
Εκτελέστε την ανωτέρω εντολή για να φτάσουμε μέχρι το fault.
Μόλις εμφανιστεί το fault δώστε **print** για κάθε μεταβλητή για να εμφανιστεί η τιμή τους.

```
print matrix
print i
print j
```

Μπορείτε να διαπιστώσετε από τα παραπάνω αποτελέσματα ποιο είναι το πρόβλημα *(βοήθεια: δείτε τη δήλωση των αντίστοιχων μεταβλητών στο πηγαίο αρχείο);*

_____ **(A27)**

Επεξεργαστείτε το πηγαίο αρχείο και επιδιορθώστε το.
**(A28)** Μεταφορτώστε το και εκτελέστε το. Πρέπει να είναι OK. Στείλτε το διορθωμένο πηγαίο αρχείο.

Αυτή ήταν μια μικρή εισαγωγή στο gdb. Ο αποσφαλματωτής gdb είναι από τα πιο σημαντικά εργαλεία που έχει στη διάθεσή του ο προγραμματιστής συστημάτων.

# 5. Ανάπτυξη σύνθετων προγραμμάτων. Το πρόγραμμα make

Αν έχουμε ένα αρχείο C και θέλουμε να το μεταφράσουμε σε εκτελέσιμο, μπορούμε πολύ απλά να δώσουμε μια εντολή gcc και να το κάνουμε. Στον προγραμματισμό λειτουργικών συστημάτων όμως δεν υπάρχει μόνο ένα αρχείο source file. Σε ένα τυπικό Project θα συναντήσετε εκατοντάδες αρχεία. Το να κάθεται κάποιος προγραμματιστής και να εκτελεί μια-μια τις εντολές μετάφρασης και σύνδεσης ασφαλώς είναι χάσιμο χρόνου.

Για να βελτιστοποιηθεί αυτή η διαδικασία, ανακαλύφθηκε το 'Makefile'. Το Makefile είναι μια ανακάλυψη χρήσιμη σε όλους τους προγραμματιστές ή διαχειριστές συστημάτων. Αρκεί ο προγραμματιστής να κατασκευάσει ένα αρχείο με το όνομα 'Makefile', και να τοποθετήσει όλες τις συνδέσεις των αρχείων και το πώς πρέπει να γίνουν compile, κάτι που είναι μερικές φορές χρονοβόρο. Δύο είναι τα πλεονεκτήματα: Η χρήση του Makefile μας επιτρέπει να εξηγήσουμε στον υπολογιστή τις συνδέσεις μεταξύ των αρχείων. Για παράδειγμα, να του πούμε ότι πρέπει να γίνει πρώτα compile το αρχείο xxx1.c μετά το αρχείο xxx2.c και ούτω καθεξής. Επίσης σε περίπτωση που κάνουμε αλλαγή στο αρχείο xxx10.c δε θα χρειαστεί να γίνει πάλι compile στα προηγούμενα αρχεία αφού έχουμε καθορίσει μέσα στο Makefile τη σειρά compile των αρχείων. Τέλος, για να γίνει compile ένα project με εκατοντάδες αρχεία *(ή ακόμη και με ένα)* αρκεί να γράψουμε μια εντολή μόνο *(τη make)* και όχι να γράφουμε το gcc κάθε φορά.

Για περισσότερες πληροφορίες ψάξτε στο google για **"makefile tutorial".**

## 5.1 Ερωτήσεις Α29 έως Α36

Ποια είναι τα 2 πλεονεκτήματα χρήστης του Makefile *(συνοπτικά)*:

1ο _____

2ο _____ **(Α29)**

Ποιο είναι κατά τη γνώμη σας μειονέκτημα στη χρήση του Makefile;

_____ **(Α30)**

Δημιουργήστε έναν καινούργιο κατάλογο με το επίθετο σας *(π.χ. dasygenis)* και εισέλθετε.

Στον κατάλογο που βρισκόμαστε θα πρέπει να δημιουργήσουμε ένα αρχείο με το όνομα **`Makefile`** στο οποίο θα τοποθετήσουμε κάποιες εντολές *(το πρώτο γράμμα* **`M`** *είναι κεφαλαίο).*

Αρχικά κάντε download και τοποθετήστε στον κατάλογο εργασίας τα παρακάτω αρχεία:

```
bimod.c
examplegdb.c
examplegdb.h
transformMatrix.c
```

Τα αρχεία βρίσκονται στο αντίστοιχο εργαστήριο.

Επιβεβαιώστε ότι έχετε τοποθετήσει τα 4 αρχεία με την `ls`.

Δοκιμάστε να κάνετε compile τα αρχεία: Δώστε `gcc examplegdb.c`

Έγινε με επιτυχία; Γιατί; _____ **(A31)**

Δώστε `gcc bimod.c`

Έγινε με επιτυχία; Γιατί; _____ **(A32)**

Δώστε

```
gcc bimod.c transformMatrix.c examplegdb.h examplegdb.c
```

Επειδή λοιπόν υπάρχουν πολλές εξαρτήσεις που πρέπει να επιλυθούν, είναι προτιμότερο να κατασκευάσουμε το αρχείο Makefile.

Στο αρχείο Makefile τοποθετούμε μια ετικέτα για compile ενός ή περισσοτέρων αρχείων ακολουθούμενη από τις εξαρτήσεις. Χρησιμοποιώντας τον επεξεργαστή κειμένου και διαβάζοντας τις σημειώσεις που ακολουθούν κατασκευάστε ένα αρχείο με το όνομα Makefile με τις παρακάτω γραμμές:

```
#Linking source files
all: bimod.o transformMatrix.o examplegdb.o
gcc examplegdb.o transformMatrix.o bimod.o -o examplegdb

# Compiling source files
bimod.o: bimod.c
gcc -c bimod.c

transformMatrix.o: transformMatrix.c
gcc -c transformMatrix.c

examplegdb.o: examplegdb.c examplegdb.h
gcc -c examplegdb.c
```

Οι γραμμές που αρχίζουν από το **#** είναι σχόλια.

Όταν υπάρχει μια λέξη και μετά ακολουθεί το **:** τότε αυτή η λέξη είναι ετικέτα. Η κάθε ετικέτα ακολουθείται από 0 ή περισσότερα ονόματα αρχείων από τα οποία εξαρτάται. Για παράδειγμα το bimod.o εξαρτάται από το bimod.c. Το bimod.o θα γίνει compile

ΑΝ και ΜΟΝΟ ΑΝ τροποποιηθεί το αρχείο bimod.c . Κάτω από κάθε ετικέτα έχουμε πατήσει 2 φορές το πλήκτρο TAB **(ΟΧΙ το spacebar)**. Η παράμετρος **-c** στο gcc σημαίνει ότι να γίνει compile το αρχείο αλλά όχι Link, δηλαδή να δημιουργηθεί μόνο το Object file με κατάληξη .o.

Αυτό γίνεται, γιατί κάποια πηγαία αρχεία δε μπορούν να συνδεθούν μόνα τους, οπότε δημιουργούν μόνο τα αρχεία object.

Επιστρέψτε στη γραμμή εντολών και δώστε **make all**. Αυτή η εντολή έχει ως συνέπεια να εκτελεστεί η ετικέτα all: και να δει τι αρχεία χρειάζονται για να γίνει compile το πρόγραμμα. Αν πάνε όλα καλά ΔΕ θα δείτε μηνύματα σφάλματος και θα μπορέσετε να εκτελέσετε το πρόγραμμα με το να δώσετε **./examplegdb**

Εκτός από εντολές για compile το Makefile μπορεί να εκτελέσει και άλλες εντολές του Linux. Για παράδειγμα προσθέστε τις παρακάτω γραμμές στο Makefile και δώστε **make clean** ώστε να εκτελεστούν αυτές μόνο:

```
clean:
	rm examplegdb.o
	rm transformMatrix.o
	rm bimod.o
	rm examplegdb
```

Τι κάνουν αυτές οι εντολές; _____ **(A33)**

Τέλος, μπορούμε να τοποθετούμε και παραμέτρους στο Makefile προκειμένου να απλοποιήσουμε κάποια στοιχεία που επαναλαμβάνονται αρκετά συχνά. Για παράδειγμα τοποθετήστε ως πρώτη γραμμή την εντολή η οποία δημιουργεί μια μεταβλητή με το όνομα CC και τις δίνει την τιμή gcc: **CC=gcc**

Στη συνέχεια αντικαταστήστε στο Makefile την εντολή εκτέλεσης του compiler gcc *(όπου υπάρχει το gcc)* με την αναφορά στη μεταβλητή **$(CC)**. Επιβεβαιώστε με το να δώσετε **make all** και να μην εμφανιστεί κανένα πρόβλημα.

**Ποια είναι η νέα μορφή του Makefile;** Αντιγράψτε το Makefile στο αρχείο **A34** και παραδώστε το. _____**(A34)**

Τοποθετήστε στο Makefile μια ετικέτα all-optimized1 για να δημιουργεί το εκτελέσιμο **examplegdb** με την παράμετρο βελτιστοποίησης 1, μια ετικέτα all-optimized2 για να δημιουργεί το εκτελέσιμο **examplegdb** με την παράμετρο βελτιστοποίησης 2 και μια ετικέτα all-optimized3 για να δημιουργεί το εκτελέσιμο **examplegdb** με την παράμετρο βελτιστοποίησης 3. Αφού επιβεβαιώσετε τη σωστή λειτουργία του Makfile, αντιγράψτε το Makefile στο αρχείο **A35** και παραδώστε το. _____**(A35)**

Εκτός από την παράμετρο CC, συνήθως ορίζεται και μια μεταβλητή CFLAGS στην οποία τοποθετούνται οι παράμετροι που θέλουμε να περάσουμε στο gcc.

Να τοποθετήσετε τη μεταβλητή CFLAGS και να έχετε ως παράμετρο το -g. Κάντε `make clean ; make all` και δείτε κατά πόσο έχει ενεργοποιηθεί αυτή η παράμετρος. Περισσότερες πληροφορίες στο δεσμό που δόθηκε πριν. Αντιγράψτε το Makefile στο αρχείο **A35** και παραδώστε το. _____ **(A36)**

## 5.2 Παραδοτέο C1

**(C1)** Να κατασκευάσετε ένα Makefile, για ένα δικό σας project που αποτελείται από 5 τουλάχιστον αρχεία .c και 1 τουλάχιστον αρχείο .h. Τα αρχεία μπορείτε να βρείτε στο διαδίκτυο χρησιμοποιώντας το google, είτε μπορείτε να χρησιμοποιήσετε κάποιο δικό σας project *(όχι έτοιμο project που έχει δικό του Makefile)*. Μπορείτε για παράδειγμα αν έχετε ένα πρόγραμμα .C να τοποθετήσετε κάθε συνάρτηση σε άλλο αρχείο και έτσι από ένα .C να δημιουργήσετε αρκετά αρχεία .C και .H.

Στο Makefile θα πρέπει να έχετε για κάθε αρχείο το αντίστοιχο label.

Επίσης, θα πρέπει να έχετε και μια καταχώρηση clean: στην οποία θα διαγράφονται τα object files όπως και τα εκτελέσιμα. Τέλος, θα πρέπει να χρησιμοποιήσετε τις μεταβλητές CC και CFLAGS στην οποία θα πρέπει να υπάρχει η παράμετρος -g.

**Όταν κάποιος δίνει make all θα πρέπει να ΜΗΝ ΕΜΦΑΝΙΖΕΤΑΙ σφάλμα!**

➔ θα πρέπει να δημιουργήσετε ένα archive με το Makefile και όλα τα αρχεία .c/.h χρησιμοποιώνας το πρόγραμμα tar και τη σύνταξη:

```
tar -cvf /tmp/b1.tar κατάλογο_με_αρχεία
```

το οποίο δημιουργεί στο /tmp το archive αρχείο b1.tar με όλα τα αρχεία που βρίσκονται στον κατάλογο_με_αρχεία.

# 6. Δημιουργία χάρτη μνήμης. Τμήματα προγραμμάτων.

Κάποιες φορές μας είναι χρήσιμο κατά τη δημιουργία compilation η δημιουργία ενός αρχείου memory map *(χάρτης μνήμης),* το οποίο μας παρέχει πληροφορίες με το ποια σύμβολα και βιβλιοθήκες έχει συνδεθεί το πρόγραμμά μας και σε ποιες σχετικές διευθύνσεις μνήμης έχουν τοποθετηθεί. Αν θέλουμε να δημιουργήσουμε το χάρτη μνήμης ενός προγράμματος τότε θα τοποθετήσουμε αμέσως μετά το gcc τις παραμέτρους `-Wl,-Map -Wl,file.map` για να δημιουργηθεί το αρχείο **`file.map`** μέσα στον τρέχοντα κατάλογο.

Κάντε compile το αρχείο egdb.c με τους παραμέτρους δημιουργίας χάρτη μνήμη.

## 6.1 Ερωτήσεις A37 έως A47

Ποια είναι η πλήρης εντολή; _____**(A37)**

Ανοίξτε με έναν επεξεργαστή κειμένου το αρχείο file.map *(το παραδοθεί και το αρχείο file.map).*

Σε ποια διεύθυνση μνήμης έχει τοποθετηθεί η συνάρτηση bimod; _____ **(A38)**

Πόσα bytes καταλαμβάνει η συγκεκριμένη συνάρτηση στο δεκαδικό σύστημα *(βρείτε την επόμενη διεύθυνση μνήμης και κάντε την αφαίρεση)*;_____ **(A39)**

Κάθε εκτελέσιμο αποτελείται από πολλαπλά τμήματα. Τα πιο βασικά τμήματα είναι τα εξής:

- **.text** το οποίο περιέχει το τμήμα κώδικα *(code segment)*. Είναι μόνο για ανάγνωση και είναι ένα τμήμα το οποίο μπορεί να μοιραστεί με άλλες διεργασίες.
- **.data** το οποίο περιέχει όλες τις μεταβλητές που έχουν αρχικοποιηθεί.
- **.bss** το οποίο περιέχει όλες τις μεταβλητές που έχουν δηλωθεί αλλά δεν έχουν πάρει αρχικές τιμές.

Για να βρούμε το μέγεθος που καταλαμβάνουν αυτά τα τμήματα χρησιμοποιούμε το size. Σύνταξη: **`size όνομα_εκτελέσιμου`** , όπου `όνομα_εκτελέσιμου` είναι το εκτελέσιμο.

Ποιο είναι το μέγεθος των text, data, bss στο εκτελέσιμο egdb που έχετε δημιουργήσει; _____ **(A40)**

Η δομή των εκτελέσιμων αρχείων στο UNIX μπορεί να είναι διαφόρων τύπων.

Η τελευταία και πιο δημοφιλής μορφή είναι η ELF *(Executable and Linkable Format)*

Περισσότερα στο: http://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Αν θέλουμε να μας εμφανιστούν πληροφορίες για ένα ELF αρχείο θα χρησιμοποιήσουμε το πρόγραμμα `readelf` .

Σύνταξη: **`readelf -a όνομα_εκτελέσιμου`**

Βρείτε τις πληροφορίες elf για το εκτελέσιμο egdb που έχετε δημιουργήσει προηγουμένως και σημειώστε τις απαντήσεις *(κάντε copy paste ολόκληρη την αντίστοιχη γραμμή)*:

Ποια είναι η διεύθυνση εισόδου της διεργασίας *(η πρώτη εντολή που θα εκτελεστεί στο πρόγραμμα)*; _____ **(A41)**

Οι προσημασμένοι αριθμοί πως σημειώνονται *(με τι συμπλήρωμα;)* _____ **(A42)**

Δώστε ένα τμήμα το οποίο έχει σημειωθεί ότι είναι τμήμα εκτέλεσης *(X (execute))*
_____ **(A43)**

Σε ποια διεύθυνση μνήμης τοποθετείται το ανωτέρω τμήμα;_____ **(A44)**

Το compilation ενός αρχείου δημιουργεί ένα εκτελέσιμο το οποίο έχει εντολές assembly για τη συγκεκριμένη αρχιτεκτονική. Σε περίπτωση που θέλουμε να δούμε τις εντολές assembly που έχουν δημιουργηθεί *(για παράδειγμα για να βελτιστοποιήσουμε κάποιο κομμάτι αν θέλουμε)* χρησιμοποιούμε το πρόγραμμα **`objdump`**.

Χρησιμοποιώντας την κατάλληλη παράμετρο στο objdump να κάνετε dissasembly το εκτελέσιμο egdb και να εμφανίσετε τις 3 πρώτες γραμμές *(assembly)* του τμήματος .text *(Disassembly of section .text:)* **(A45)**

Πόσα Bytes στον κώδικα μηχανής καταλαμβάνει η πρώτη εντολή;_____ **(A46)**

Εφόσον έχετε κάνει compile με την παράμετρο **`-g`** στο gcc, μπορείτε να χρησιμοποιήσετε την παράμετρο **`-S`** στο objdump προκειμένου να εμφανίζεται η γραμμή της C και στη συνέχεια οι εντολές assembly που υλοποιούν το συγκεκριμένο κομμάτι κώδικα. Εκτελέστε **`objdump -S -d εκτελέσιμο`** και σημειώστε τις τρεις πρώτες εντολές της συνάρτησης int main()_____**(A47)**

**(C2)** Δώστε το screenshot που αντιστοιχεί στο **A47**.

# 7. Ανάλυση εκτελέσιμου και βελτιστοποίηση

## 7.1 Το πρόγραμμα gcov

Πολλές φορές υπάρχει η λανθασμένη εντύπωση ότι περισσότερος χρόνος απαιτείται για την ανάπτυξη ενός προγράμματος παρά για οτιδήποτε άλλο. Αυτή η ιδέα είναι λάθος επειδή συνήθως όσο χρόνο απαιτεί η ανάπτυξη ενός προγράμματος τόσο χρόνο απαιτείται για τον έλεγχο, την αποσφαλμάτωση και για τη βελτίωση. Η βελτίωση επιτυγχάνεται με μια πληθώρα εργαλείων. Σε αυτή την ενότητα θα γνωρίσουμε το εργαλείο ανάλυσης κάλυψης *(code coverage analysis)* **`gcov`**.

Το εργαλείο αυτό μας επιτρέπει να δούμε πόσες φορές εκτελέστηκε κάθε γραμμή, πόσες φορές έγινε αλλαγή ροής *(branch)* και αν υπάρχουν κομμάτια κώδικα που ποτέ δεν εκτελέστηκαν. Τα κριτήρια που σχετίζονται με την ανάλυση code coverage είναι τα παρακάτω:

- **function coverage:** πόσες φορές εκτελέστηκε η κάθε συνάρτηση.
- **statement coverage:** πόσες φορές εκτελέστηκε η κάθε εντολή.
- **decision coverage:** στις δομές ελέγχου ***(if)*** έχουν εκτελεστεί όλες οι περιπτώσεις *(if/elseif/else)*
- **condition coverage:** έχουν εκτελεστεί και οι true και οι false εκφράσεις bool;
- **path coverage:** έχει εκτελεστεί με όλες τις δυνατές πιθανότητες ροής ελέγχου;

- **entry/exit coverage:** έχει γίνει η είσοδος και η έξοδος από όλες τις δυνατές τοποθεσίες;

Το πρόγραμμα gcov είναι ένα εργαλείο coverage analysis που συνεργάζεται στενά με το εργαλείο gprof. Τα προγράμματα αυτά έρχονται μαζί με το μεταφραστή gcc και βρίσκονται *(σχεδόν)* σε κάθε σύστημα.

Για να χρησιμοποιηθούν τα εργαλεία βελτιστοποίησης θα πρέπει να ισχύουν τα παρακάτω:

- ο κώδικας θα πρέπει να έχει γίνει compile με το gcc
- θα πρέπει να έχουν χρησιμοποιηθεί οι παράμετροι
- -fprofile-arcs -ftest-coverage στο gcc
- δε θα πρέπει να χρησιμοποιείται καμία παράμετρος βελτιστοποίησης *(παράμετροι -O,-O1...)*
- να τοποθετήσετε μια εντολή C ανά γραμμή, επειδή τα στατιστικά αναφέρονται ανά γραμμή.

Θα δούμε τη διαδικασία ανάλυσης παρακάτω.

Κατασκευάστε ένα φάκελο coverage. Εισέλθετε μέσα στο φάκελο coverage. Δημιουργήστε μέσα στο φάκελο το πηγαίο αρχείο **test1.c** με τις παρακάτω εντολές:

```
#include <stdio.h>
int main (void)
{
int i;
for(i=1;i<13;i++)
{
if(i%2 == 0)
printf("%d is divisible by 2 \n",i);
if(i%5==0)
printf("%d is divisible by 5 \n",i);
if (i%14==0)
printf("%d is divisible by 14 \n",i);
}
return 0;
}
```

Κάντε compile το ανωτέρω αρχείο με τις παραμέτρους που ενεργοποιούν το **coverage analysis** στο εκτελέσιμο αρχείο test1 .

Εκτελέστε μια φορά το ανωτέρω αρχείο. Μόλις το εκτελέσετε θα δείτε ότι έχουν δημιουργηθεί κάποια επιπρόσθετα αρχεία που δεν υπήρχαν πριν.

### 7.1.1 Ερωτήσεις A48 έως A55

Ποια αρχεία έχουν δημιουργηθεί μετά την εκτέλεση;_____ **(A48)**
*(το ένα αρχείο έχει το δένδρο κλήσεων συναρτήσεων και το άλλο τους αριθμούς κλήσεων όλα σε κωδικοποιημένη μορφή).*

Εκτελέστε το gcov.

Σύνταξη: **`gcov όνομα_πηγαίου_αρχείου`** Δηλαδή, **`gcov test1`**

Μόλις το εκτελέσετε θα δείτε ότι δημιουργείται το αρχείο test1.c.gcov. Ανοίξτε το με έναν επεξεργαστή κειμένου. Στην αριστερή στήλη αναφέρεται πόσες φορές έχει εκτελεστεί η γραμμή *(αν έχει – δεν είναι γραμμή που εκτελείται, αν έχει #### τότε δεν εκτελέστηκε καθόλου. Μετά ακολουθεί ο αριθμός γραμμής του πηγαίου αρχείου).*

Πόσες φορές έχει εκτελεστεί η γραμμή:

```
printf("%d is divisible by 5 \n",i);
```

_____ **(A49)**

Ποια γραμμή έχει εκτελεστεί τις περισσότερες φορές;_____**(A50)**

Υπάρχει κάποια γραμμή που εκτελείται, η οποία δεν έχει εκτελεστεί καθόλου; _____ **(A51)**

Χρησιμοποιήστε την παράμετρο **-b** στο gcov προκειμένου να σας αναφερθούν οι πιθανότητες εκτέλεσης των διαφορετικών βρόχων *(branch probabilities)* και εκτελέστε πάλι το gcov.

Παρατηρήστε τι αναφέρει το gcov.

Πόσο ποσοστό των γραμμών εκτελέστηκαν;_____ **(A52)**

Πόσοι βρόχοι εκτελέστηκαν τουλάχιστον μια φορά *(Taken at least once)*; _____ **(A53)**

Ανοίξτε το αρχείο **test1.c.gcov.**

Υπάρχει κάποια κλήση συνάρτησης *(call)* που δεν έχει εκτελεστεί ποτέ; Σε ποια γραμμή είναι και ποια είναι αυτή; _____ **(A54)**

Παρατηρώντας τις διακλαδώσεις και τους αριθμούς φορών εκτέλεσης του κάθε branch, και γνωρίζοντας ότι στο pipeline μειώνεται η απόδοση όταν έχουμε αλλαγή ροής εκτέλεσης λόγω διαφορετικού branch, ποιο κομμάτι if θα προκαλέσει τη μεγαλύτερη επιβάρυνση στο pipeline;_____ **(A55)**

## 7.2 Το πρόγραμμα gprof

Μαζί με το gcov συνήθως χρησιμοποιείται το πρόγραμμα gprof το οποίο μας αναφέρει πόσο ποσοστό του χρόνου του επεξεργαστή ξοδεύτηκε σε κάθε συνάρτηση. Για να χρησιμοποιηθεί το gprof απαιτείται να χρησιμοποιηθεί η παράμετρος **-pg** στο gcc κατά τη δημιουργία του compile. Στη συνέχεια εκτελείται το αρχείο τουλάχιστον μια φορά και δημιουργείται ένα νέο αρχείο στον κατάλογο.

Κάντε compile το αρχείο collatz.c *(βρίσκεται στα αρχεία του εργαστηρίου)* χρησιμοποιώντας την παράμετρο ενεργοποίησης στατιστικών για το gprof και με την επιλογή να αποθηκευτεί το αρχείο που θα δημιουργηθεί στο collatz.

Ποια εντολή χρησιμοποιήσατε για το compile;_____ **(A56)**

Εκτελέστε μια φορά το collatz.

Δώστε την εντολή: **`gprof -l -u -a -b collatz | more`**

*(σε περίπτωση που κάποιοι παράμετροι δεν υποστηρίζονται στο σύστημα που βρίσκεστε αγνοήστε τις).*

Θα σας εμφανιστεί μια λίστα όπου η πρώτη στήλη θα είναι % (% time) του χρόνου του επεξεργαστή για τη συγκεκριμένη συνάρτηση (το όνομα της συνάρτησης αναφέρεται στο τέλος), η δεύτερη στήλη *(cumulative seconds)* το χρόνο που χρειάστηκε για να εκτελεστεί αυτή η συνάρτηση και οι κλήσεις που έγιναν από αυτή, η τρίτη στήλη *(self seconds)* ο χρόνος που δαπανήθηκε μόνο σε αυτή τη συνάρτηση, η τέταρτη στήλη *(calls)* πόσες φορές έγινε κλήση η συνάρτηση, ενώ η τελευταία στήλη είναι το όνομα της συνάρτησης.

Ποια συνάρτηση δαπάνησε τον περισσότερο χρόνο *(sec)* του επεξεργαστή;
_____ **(A57)**

Ποια συνάρτηση είχε τις περισσότερες κλήσεις; Πόσες ήταν αυτές;
_____ **(A58)**

Η παραπάνω έξοδος ονομάζεται flat-profile.

Αν αντί για την παράμετρο **`-l`** έχουμε **`-L`** τότε θα έχουμε ένα call-graph profile. Εκτελέστε το και δείτε την ιεραρχία κλήσεων, δηλαδή πως η μια συνάρτηση έκανε κλήση την επόμενη κ.ο.κ.

Στο διάγραμμα call-graph πόσο % του χρόνου του επεξεργαστή η _start κάλεσε την main η οποία κάλεσε την nseq και η οποία κάλεσε την printf; _____**(A59)**

Ποια συνάρτηση εκτελέστηκε το περισσότερο; _____ **(A60)**

Αφού ολοκληρωθεί το εργαστήριο απαντήστε: Ποια είναι η διαφορά των gprof/gcov;
_____ **(A61)**