

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

**1η εργαστηριακή αναφορά**

στο μάθημα **“Λειτουργικά συστήματα”** 6ου Εξαμήνου

των φοιτητών

Άγγελου Γκίκα , ΑΜ: 03118218

Νικήτα Τσίννα, ΑΜ: 03118187

**Άσκηση 1.1**

**α) Πηγαίος κώδικας:**

**main.c:**

#include "zing.h"

int main(int argc, char \*\*argv){

zing();

return 0;

}

**zing.h:**

#ifndef ZING\_H\_\_

#define ZING\_H\_\_

void zing(void);

#endif

**zing2.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

void zing() {

char\* usr;

usr = getlogin();

printf("%s, to the moon!!\n", usr);

}

**Makefile:**

all: zing zing2

zing: main.o zing.o

gcc main.o zing.o -o zing

main.o: main.c

gcc -c main.c

zing2: main.o zing2.o

gcc main.o zing2.o -o zing2

zing2.o: zing2.c

gcc -c zing2.c

**β) Διαδικασία μεταγλώττισης και σύνδεσης:**

Η διαδικασία αυτή επιτυγχάνεται κάνοντας χρήση του αρχείου Makefile, εκτελώντας την εντολή make. Πιο συγκεκριμένα, αν είναι η πρώτη φορά που κάνουμε κλήση, συμβαίνουν τα εξής:

Ο σκοπός είναι να παράξουμε τα δύο εκτελέσιμα, zing και zing2. Ξεκινάμε με σκοπό να παράξουμε το zing. Ο κώδικας που θα παράξει το zing βρίσκεται στην main.c. Επομένως εκτελούμε την εντολή gcc -c main.c ώστε να παράξουμε το object file main.o. Στην main.c όμως γίνεται χρήση της συνάρτησης zing(), στην οποία έχουμε πρόσβαση μέσω της επικεφαλίδας #include "zing.h", μέσω της οποίας συνδέεται το main.c με το αρχείο zing.h στο οποίο βρίσκεται η δήλωση της zing(). Χρειαζόμαστε επομένως ένα object file στο οποίο να υπάρχει η υλοποίηση της zing. Αυτό προς το παρόν είναι μόνο το zing.o, επομένως εκτελούμε την εντολή gcc main.o zing.o -o zing για να παράξουμε το εκτελέσιμο αρχείο zing. Στη συνέχεια επιθυμούμε να παράξουμε το εκτελέσιμο αρχείο zing2. Για να το κάνουμε αυτό θα χρειαστούμε ένα object file το οποίο θα περιέχει την υλοποίηση της zing, η οποία αυτή τη φορά θα είναι διαφορετική. Για αυτό το λόγο, αφού έχουμε δημιουργήσει το zing2.c το οποίο περιέχει σε κώδικα c την επιθυμητή αυτή υλοποίηση, εκτελούμε την εντολή gcc -c zing2.c ώστε να παραχθεί το επιθυμητό object file με όνομα zing2.o. Στη συνέχεια, αφού έχουμε το επιθυμητό object file που χρειάζεται η main.c για να εκτελεστεί, εκτελούμε την εντολή gcc main.o zing2.o -o zing2, όπου ο linker αντιλαμβάνεται ότι το zing2.o περιέχει την υλοποίηση της zing() και έτσι παράγεται το εκτελέσιμο zing2.

**γ) Έξοδος εκτέλεσης του προγράμματος:**

Εκτελώντας τις εντολές έχουμε τις αντίστοιχες εξόδους:

oslaba14@orion:~/ex1\_1$ ./zing

Hello, oslaba14

oslaba14@orion:~/ex1\_1$ ./zing2

oslaba14, to the moon!!

**δ) Σύντομες απαντήσεις στις ερωτήσεις:**

1. Η επικεφαλίδα #include "zing.h" χρησιμοποιείται ώστε να υποδείξουμε στον compiler σε ποιο header file να ανατρέξει κατά την μεταγλώττιση, στην συγκεκριμένη περίπτωση στο zing.h, ώστε μέσα σε αυτό να βρει την δήλωση της συναρτησης zing(), η οποία καλείται μέσα στο body της main.c .  
   Τα header files χρησιμοποιούνται για να οριστούν οι απαραίτητες συναρτήσεις (function definitions) και όταν περιέχονται σε ένα αρχείο κώδικα, αυτό ισοδυναμεί με το να έχουμε γράψει τα περιεχόμενα του .h αρχείου στην θέση που το κάναμε #include. Αυτό βοηθά στην απλότητα του κώδικα αλλά και στην παραγωγικότητα του προγραμματιστή, αφού του δίνει την δυνατότητα να περιέχει πολλούς ορισμούς συναρτήσεων ή βιβλιοθηκών σε μία μόνο γραμμή τύπου #include.
2. Αρχικά το Makefile το οποίο δημιουργεί μόνο το εκτελέσιμο πρόγραμμα zing (όχι ακόμα το zing2) περιέχει:

all: zing

zing: main.o zing.o

gcc main.o zing.o -o zing

main.o: main.c

gcc -c main.c

1. Έπειτα έχοντας ως σκοπό να παράγονται δύο αρχεία, τα zing και zing2, το Makefile διαμορφώνεται όπως έχει παρουσιαστεί στο ερώτημα (α). Οι διαδικασίες που ζητούνται να πραγματοποιηθούν, περιγράφονται στο ερώτημα (β)
2. Με σκοπό την επίλυση του προβλήματος, “σπαμε” τις συναρτήσεις σε ομάδες, τοποθετώντας κάθε ομάδα σε ένα αρχείο κώδικα, ή και ακόμα τοποθετούμε μία μόνο συνάρτηση σε κάθε file (την επιλογή ή τον συνδυασμό των παραπάνω επιλογών την κάνει ο προγραμματιστής ανάλογα το πρόγραμμα). Με τον τρόπο αυτό, και αφού έχουμε δημιουργήσει Makefile, κάθε φορά που κάνουμε μία αλλαγή σε μία συνάρτηση μεμονωμένα, καλούμε έπειτα την make, η οποία κάνει την μεταγλώττιση και εκτελεί ξανά μόνο τις εντολές που έχουν άμεση σχέση με το αρχείο της συνάρτησης στην οποία έγινε η αλλαγή εφόσον το makefile έχει συνταχθεί κατάλληλα, έπειτα συνδέει (linking) τα κατάλληλα .ο αρχεία για να παραχθεί το τελικό εκτελέσιμο. Αυτό έχει το σημαντικό αποτέλεσμα να μειώνεται σημαντικά ο χρόνος μεταγλώττισης.
3. Με την εντολή gcc -Wall -o foo.c foo.c ουσιαστικά δίνουμε την εντολή στον compiler να παράξει το εκτελέσιμο αρχείο από το αρχείο foo.c, και να του δώσει όνομα foo.c. Όμως επειδή υπάρχει ήδη αρχείο με το όνομα αυτό (που αποτελεί τον κώδικά μας), το περιεχόμενο του πρωταρχικού αρχείου foo.c, αντικαθίσταται από το περιεχόμενο του εκτελέσιμου foo.c που παράχθηκε. Ως αποτέλεσμα, πλέον υπάρχει μόνο ένα αρχείο foo.c, το οποίο είναι εκτελέσιμο, και ο κώδικάς μας σε c έχει χαθεί.

**Άσκηση 1.2**

**α) Πηγαίος κώδικας:**

**main.c:**

#include <stdio.h>

#include <unistd.h>

#include "fun.h"

int main(int argc, char \*\*argv)

{

if (argc <3 || argc >4) { //if arguments are wrong

printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");

return 1;

}

int fd, oflags, mode; //opening write-file...

oflags = O\_CREAT | O\_WRONLY | O\_TRUNC;

mode = S\_IRUSR | S\_IWUSR;//permission bits

if (argc == 3) fd = open("fconc.out", oflags, mode); //default output file

else fd = open(argv[3], oflags, mode); //output file determined by the usr

if (fd == -1) {

perror("infile");

exit(1);

}

write\_file(fd, argv[1]);

write\_file(fd, argv[2]);

close(fd); //closing file

}

**fun.c:** //Περιέχει την υλοποίηση των συναρτήσεων doWrite και write\_file

#include "fun.h"

void doWrite(int fd, const char \*buff, int len)

{

if(buff == NULL) return;

size\_t idx = 0;

ssize\_t wcnt;

do {

wcnt = write(fd, buff+idx, len-idx);

if(wcnt == -1) {

perror("write");

exit(1);

}

idx += wcnt;

} while (idx < len);

}

void write\_file(int fd, const char \*infile)

{

if (fd == -1) {

perror("write");

exit(1);

}

int fdr; //fd of read file

fdr = open(infile, O\_RDONLY);

if (fdr == -1) {

perror("infile");

exit(1);

}

char buff[1024]; //reading file...

ssize\_t rcnt; //read counter

for(;;){

rcnt = read(fdr, buff, sizeof(buff)-1);

if (rcnt == 0) //means end of file

return;

if (rcnt == -1) {

perror("read");

close(fdr); //closing read-file...

exit(1);

}

buff[rcnt] = '\0';//putting end-of-file char in buffer

doWrite(fd,buff,strlen(buff)); //writing to write-file

}

close(fdr); //closing read-file

}

**β) Σύντομες απαντήσεις στις ερωτήσεις**

Εκτελώντας την εντολή $ strace ./fconc A B

πήραμε ως έξοδο τα παρακάτω:

execve("./fconc", ["./fconc", "A", "B"], [/\* 20 vars \*/]) = 0

brk(0) = 0x2176000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb 4ffd33000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

open("/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=29766, ...}) = 0

mmap(NULL, 29766, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fb4ffd2b000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

open("/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\34\2\0\0\0\0\0"..., 832 ) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=1738176, ...}) = 0

mmap(NULL, 3844640, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f b4ff76a000

mprotect(0x7fb4ff90b000, 2097152, PROT\_NONE) = 0

mmap(0x7fb4ffb0b000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENY WRITE, 3, 0x1a1000) = 0x7fb4ffb0b000

mmap(0x7fb4ffb11000, 14880, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANON YMOUS, -1, 0) = 0x7fb4ffb11000

close(3) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb 4ffd2a000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb 4ffd29000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb 4ffd28000

arch\_prctl(ARCH\_SET\_FS, 0x7fb4ffd29700) = 0

mprotect(0x7fb4ffb0b000, 16384, PROT\_READ) = 0

mprotect(0x7fb4ffd35000, 4096, PROT\_READ) = 0

munmap(0x7fb4ffd2b000, 29766) = 0

open("fconc.out", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 3

open("A", O\_RDONLY) = 4

read(4, "Hello there,\n", 1023) = 13

write(3, "Hello there,\n", 13) = 13

read(4, "", 1023) = 0

open("B", O\_RDONLY) = 5

read(5, "my name is Daizy.\n", 1023) = 18

write(3, "my name is Daizy.\n", 18) = 18

read(5, "", 1023) = 0

close(3) = 0

exit\_group(0) = ?

+++ exited with 0 +++

Παραπάνω μπορούμε να παρατηρήσουμε όλες τις κλήσεις του συστήματος που γίνονται με την σειρά κατά την εκτέλεση του προγράμματος.