Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Λειτουργικά Συστήματα

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Ημερομηνία Παράδοσης : 03/11/2017

Ονοματεπώνυμο	**Αριθμός Μητρώου**
Αλεξάκης Κωνσταντίνος	03114086
Βασιλάκης Εμμανουήλ	03114167

Εξάμηνο	**Ακαδημαϊκό Έτος**
7ο	2017-2018

1.1 Σύνδεση με αρχείο αντικειμένων

1.1 Πηγαίος κώδικας

Στην πρώτη άσκηση, ζητήθηκε να δημιουργηθεί object main.o, το οποίο να συνδεθεί με το object zing.o, αφού αυτό έχει πρώτα αντιγραφεί στο directory μας. Ακολουθεί ο κώδικας της main.c :

```
#include "zing.h"

int main (int argc, char *argv[])
{
    zing();
    return 0;
}
```

1.2 Διαδικασία μεταγλώττισης και σύνδεσης

Αρχικά, πρέπει να γίνει το compilation, ώστε να παραχθεί το object main.o. Αυτό επιτυγχάνεται με την παρακάτω εντολή :

gcc –Wall –c main.c

Ακολούθως, πρέπει να γίνει το linking μεταξύ των δύο object. Αυτό επιτυγχάνεται με την ακόλουθη εντολή :

gcc main.o zing.o -o zing

Μετά την εκτέλεση των παραπάνω εντολών, έχει δημιουργηθεί ένα εκτελέσιμο zing.

1.3 Έξοδος εκτέλεσης του προγράμματος

Αν τώρα, τρέξουμε το εκτελέσιμο μας, θα λάβουμε την ακόλουθη έξοδο:

./zing *// εκτελώ το εκτελέσιμο*

Hello, oslabc20 *// έξοδος του προγράμματος*

1.4 Ερωτήσεις

1. *Ποιο σκοπό εξυπηρετεί η επικεφαλίδα ;*

   Τα header files ή αλλιώς επικεφαλίδα, χρησιμοποιούνται ώστε να είναι δυνατή η κλήση συναρτήσεων που έχουν δηλωθεί σε κάποιο αρχείο ".h", από πολλά διαφορετικά source files. Στη περίπτωσή μας, στη συνάρτηση main() έχουμε κλήση της συνάρτησης zing(), η οποία έχει δηλωθεί εντός του αρχείου κεφαλίδας zing.h. Για το λόγο αυτό, κατά την μεταγλώττιση του προγράμματος main.c, ο compiler περιμένει στη κορυφή του κώδικα να βρει την αντίστοιχη επικεφαλίδα-header file #include "zing.h" ,έτσι ώστε να μην δημιουργηθεί κάποιο πρόβλημα κατά την εκτέλεση του προγράμματος, αφού η δήλωση της συνάρτησης zing() γίνεται μέσα σε εκείνο το αρχείο.

2. *Ζητείται κατάλληλο Makefile για τη δημιουργία του εκτελέσιμου της άσκησης.*

   Δημιουργούμε ένα νέο αρχείο με όνομα “Makefile” στο ίδιο directory. Το περιεχόμενο του αρχείου μας για την μεταγλώττιση και το linking των αρχείων αντικειμένων καθορίζεται ως εξής:

```
zing : main.o zing.o                //linking και δημιουργία του εκτελέσιμου
    gcc-o zing main.o zing.o

main.o : main.c                     //δημιουργία του main.o (μεταγλώττιση)
    gcc-Wall-c main.c

clean:                              //εκκαθάριση του main.o και του
    rm main.o zing                  //εκτελέσιμου zing
```

3. *Να δημιουργηθεί zing2 εκτελέσιμο που εμφανίζει παρόμοιο μήνυμα με την zing. Να τροποποιηθεί κατάλληλα το Makefile, ώστε να παράγονται και δύο εκτελέσιμα, ένα με το zing.o, ένα με το zing2.o, επαναχρησιμοποιώντας το κοινό object file main.o.*

   Ακολουθεί αρχικά ο κώδικας του zing2.c :

```
#include <unistd.h>
#include <stdio.h>

void zing() {
    char *name;
    name = getlogin();
    if (name == NULL) name = "unknown";
    printf("Hi, %s\n", name);
}
```

   Τώρα, πρέπει να ενημερώσουμε το Makefile, ώστε να μεταγλωττίζει το zing2.c και επιπλέον να κάνει link μεταξύ των main.o και των zing.o και zing2.o. Έτσι το καινούριο μας Makefile είναι το ακόλουθο:

```
final : zing zing2                  //δημιουγία των δύο εκτελέσιμων

zing : main.o zing.o                //link main.o με zing.o, δημιουργία zing
    gcc-o zing main.o zing.o

zing2 : main.o zing2.o              //link main.o με zing2.o, δημιουργία zing2
    gcc-o zing2 main.o zing2.o

main.o : main.c                     //δημιουργία του main.o
    gcc-Wall-c main.c

zing2.o : zing2.c                   //δημιουργία του zing2.o
    gcc-Wall-c zing2.c

clean:                              //εκκαθάριση των εκτελέσιμων και των object
    rm zing2.o main.o zing zing2
```

4. *Έστω ότι έχετε γράψει το πρόγραμμά σας σε ένα αρχείο που περιέχει 500 συναρτήσεις. Πώς μπορεί να μειωθεί ο χρόνος που καταναλώνεται κατά την μεταγλώττιση του προγράμματος μετά από κάθε αλλαγή στον κώδικα ;*

   Το πρόβλημα αυτό λύνεται με τη δημιουργία ενός Makefile. Ουσιαστικά, έχοντας και τις 500 συναρτήσεις μέσα σε ένα αρχείο, κάνοντας αλλαγές μόνο σε μία από αυτές τις συναρτήσεις για να γίνει η εκτέλεση του προγράμματος, θα πρέπει να γίνει μεταγλώττιση ολόκληρου του αρχείου και των 500 συναρτήσεων (πράγμα ασύμφορο, χρονοβόρο και ανούσιο). Το παραπάνω ζήτημα μπορεί εύκολα να λυθεί εάν κάθε μία από τις 500 συναρτήσεις βρίσκεται σε διαφορετικό αρχείο. Επομένως, σε αυτήν την περίπτωση με τη δημιουργία ενός κατάλληλου Makefile, θα είναι εφικτή η τροποποίηση ενός μόνου αρχείου, χωρίς να χρειάζεται η μεταγλώττιση των υπολοίπων συναρτήσεων. Η μεταγλώττιση θα γίνει μονάχα στο αρχείο που έχει τροποποιηθεί και στην συνέχεια με κατάλληλο linking θα παραχθεί το τελικό εκτελέσιμο πρόγραμμα, μειώνοντας έτσι στο ελάχιστο το χρόνο του compilation.

5. *Γιατί χάνεται το αρχείο foo.c αν εκτελέσουμε την εντολή :*
   gcc –Wall –o foo.c foo.c ;

   Η εντολή **gcc -Wall -o foo.c foo.c** ουσιαστικά κάνει απευθείας compile και δημιουργεί το εκτελέσιμο πρόγραμμα χωρίς ενδιάμεσα να παραχθεί object file foo.o. Όμως, με την παραπάνω εντολή δίνεται στο εκτελέσιμο πρόγραμμα το ίδιο όνομα με το αρχείο του πηγαίου προγράμματος. Επομένως, το εκτελέσιμο αντιγράφεται πάνω στο αρχείο του πηγαίου κώδικα, επομένως λογικό είναι το τελευταίο να χάνεται.

1.2 Συνένωση δύο αρχείων σε τρίτο

1. *Πηγαίος κώδικας*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void doWrite(int fd, const char *buff, int length)
{
	size_t len, idx;
	ssize_t wcnt;
	idx=0;
	len=length;
	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)     //eggrafh apo ton infile
{                                                //ston perihghth fd
	ssize_t  rec;                            //rec= plithos bytes poy diabasa
	int fda;                                 //fda= perihghths toy arxeioy poy anoi3a
	char buff[30];
	fda=open(infile, O_RDONLY);
	if(fda ==-1) {
		perror (infile);
		exit(1);
	}
	rec=read(fda, buff, 30);
	while(rec > 0)
	{
		doWrite(fd, buff, rec);
		rec = read(fda, buff, 30);
	}
	if(close(fda) ==-1)
```

```
    {
        perror("ERROR : cannot close input file\n");
        exit(1);
    }
    if(rec < 0)
    {
        perror("read");
        exit(1);
    }
    return;
}

int main (int argc, char **argv) {
    int fdc, oflags, mode;
    if(argc < 3 || argc > 4) {
        printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)] \n");
        exit(1);
    }
    if(argc > 3) {
        if(strcmp(argv[1], argv[3]) == 0 || strcmp(argv[2], argv[3]) == 0) {
            printf("ERROR : input file and output file cannot be the same\n");
            exit(1);
        }
        oflags = O_CREAT | O_WRONLY| O_TRUNC ;
        mode = S_IRUSR | S_IWUSR;
        fdc = open(argv[3], oflags, mode);
    }
    else {
        oflags = O_CREAT | O_WRONLY| O_TRUNC ;
        mode = S_IRUSR | S_IWUSR;
        fdc = open("fconc.out", oflags, mode);
    }
    if(fdc ==-1) {
        printf("ERROR : cannot open output file\n");
        exit(1);
    }
    write_file(fdc, argv[1]);
    write_file(fdc, argv[2]);
    if(close(fdc) ==-1) {
        printf("ERROR : cannot close output file\n");
        exit(1);
    }
    return 0;
}
```

2. *Ερωτήσεις*

Το κομμάτι της εξόδου της εκτέλεσης της εντολής **strace ./fconc A B** που προκύπτει από τον κώδικα που γράψαμε παρατίθεται παρακάτω :

```
open("fconc.out", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY)                     = 4
read(4, "periexomena arxeioy A\n", 30)  = 22
write(3, "periexomena arxeioy A\n", 22) = 22
read(4, "", 30)                         = 0
close(4)                                = 0
open("B", O_RDONLY)                     = 4
read(4, "periexomena arxeioy B\n", 30)  = 22
write(3, "periexomena arxeioy B\n", 22) = 22
read(4, "", 30)                         = 0
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```