Λειτουργικά Συστήματα Υπολογιστών (τμήμα 1)

1η εργαστηριακή αναφορά

Ομάδα oslaba24	
Νικόλαος Παγώνας	el18175
Αναστάσιος Παπαζαφειρόπουλος	el18079

Άσκηση 1.1: Σύνδεση με αρχείο αντικειμένων

Πηγαίος κώδικας:

```
#include "zing.h"

int main() {
      zing();
      return 0;
}
```

Διαδικασία μεταγλώττισης και σύνδεσης:

1. Αντιγραφή αρχείων zing.h και zing.o στον κατάλογο εργασίας:

```
$ cp /home/oslab/code/zing/zing.h /home/oslab/code/zing/zing.o ~/1/1.1
```

   (1.1 είναι ο κατάλογος που περιέχει τα αρχεία που αφορούν την άσκηση 1.1)

2. Δημιουργία αρχείου αντικειμένων main.o για τη συνάρτηση main():

```
$ gcc -Wall -c main.c
```

3. Σύνδεση (linking) των δύο αρχείων αντικειμένων:

```
$ gcc main.o zing.o -o zing
```

Έξοδος εκτέλεσης του προγράμματος:

```
Hello, oslaba24
```

Ερωτήσεις:

1. Η επικεφαλίδα περιέχει τις δηλώσεις των συναρτήσεων, χωρίς όμως να περιέχει την υλοποίησή τους. Έτσι μειώνεται σημαντικά ο χρόνος της μεταγλώττισης, αφού κατά τη διάρκειά της ο compiler χρειάζεται μόνο τις δηλώσεις των συναρτήσεων.

2. Makefile:

```
zing: main.o zing.o
  gcc main.o zing.o -o zing

main.o: main.c
  gcc -Wall -c main.c
```

3. Η συνάρτηση zing() τυπώνει “Hello, <user>”, όπου <user> είναι το username του χρήστη που έχει κάνει login στο terminal. Διαβάζοντας το manual page της getlogin(3), βλέπουμε ότι η συνάρτηση getlogin() επιστρέφει ακριβώς αυτό. Επομένως, γράφουμε το δικό μας αρχείο zing2.c ως εξής:

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

void zing(void) {
      char *username = getlogin();
      printf("%s sucks at coding.\n", username); // ψέμα!
}
```

Και για να αυτοματοποιήσουμε την διαδικασία, τροποποιούμε το Makefile, συμπεριλαμβάνοντας τις προσθήκες που αφορούν το zing2:

```
all: zing zing2

zing: main.o zing.o
      gcc main.o zing.o -o zing

zing2: main.o zing2.o
      gcc main.o zing2.o -o zing2

main.o: main.c
      gcc -Wall -c main.c

zing2.o: zing2.c
      gcc -Wall -c zing2.c
```

4. Εφόσον για να γίνει compile ένα αρχείο που χρησιμοποιεί μια συνάρτηση, το μόνο που απαιτείται από τον compiler είναι η δήλωση της συνάρτησης, μπορούμε να φτιάξουμε ένα header file με όλες τις δηλώσεις των συναρτήσεων που δεν μεταβάλλονται και ένα αρχείο με την υλοποίησή τους. Έτσι, το αρχείο με την υλοποίηση των 499 συναρτήσεων γίνεται compile μία

μόνο φορά και χρησιμοποιούμε το header file για να γίνουν οι απαραίτητες αλλαγές στη συνάντηση που θέλουμε.

5. Μετά την εκτέλεση της εντολής: `gcc -Wall -o foo.c foo.c` γίνεται overwrite το αποτέλεσμα της μεταγλώττισης (εκτελέσιμο αρχείο) στο αρχείο foo.c, το οποίο μέχρι πρότινος περιείχε τον πηγαίο κώδικά μας. Οπότε έτσι ο κώδικας χάνεται.

Άσκηση 1.2: Συνένωση δύο αρχείων σε τρίτο

Πηγαίος Κώδικας

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

void doWrite(int fd, const char *buff, int len, const char *outname) {
	// Συνάρτηση που αναλαμβάνει την εγγραφή στον περιγραφητή αρχείου fd.
	ssize_t wcnt;
	int idx = 0; // το idx θα χρησιμοποιηθεί σε περίπτωση σφάλματος κατά την εγγραφή,
		     // ώστε να μην ξαναπροσπελάσουμε τα στοιχεία του buffer που ήδη έχουν
		     // εγγραφεί
	do {
		wcnt = write(fd, buff + idx,len - idx);
		if (wcnt == -1) {
			perror(outname);
			exit(1);
		}
		idx += wcnt;
	} while(idx < len);
}
```

```c
void write_file(int fd, const char *infile, const char *outname) {
	// Συνάρτηση που γράφει τα περιεχόμενα του αρχείου με όνομα infile στον περιγραφητή
	// αρχείου fd. Χρησιμοποιεί την doWrite().

	// open file for reading
	int  fd_read = open(infile, O_RDONLY);

	if (fd_read == -1) {
		perror(infile);
		exit(1);
	}

	char buff[1024];
	ssize_t rcnt;

	for (;;) {
		rcnt = read(fd_read, buff, sizeof(buff));
		if (rcnt == 0) break;
		if (rcnt == -1) {
			perror(infile);
			exit(1);
		}
		doWrite(fd, buff, rcnt, outname);
	}
	close(fd_read);
}
```

```c
int main(int argc, char **argv) {
	// if wrong number of arguments, print appropriate message
	if (argc < 3 || argc > 4) {
		fprintf(stderr, "Usage: ./fconc infile1 infile2 [outfile
(default:fconc.out)]\n");
		exit(1);
	}

	// default output name: fconc.out
	char *outname = (argc == 3) ? "fconc.out" : argv[3];

	// open file for writing
	int oflags = O_CREAT | O_WRONLY | O_TRUNC;
	int mode = S_IRUSR | S_IWUSR;
	int fd_write = open(outname, oflags, mode);
	if (fd_write == -1) {
		perror(outname);
		exit(1);
	}

	// write files 1 and 2 back to back
	write_file(fd_write, argv[1], outname);
	write_file(fd_write, argv[2], outname);

	close(fd_write);

	return 0;
}
```

Ερωτήσεις:

1. Έξοδος της strace:

```
execve("./fconc", ["./fconc", "hello", "world"], 0x7ffd48894ab0 /* 66 vars */) = 0
brk(NULL)                               = 0x5629e2aa8000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=85033, ...}) = 0
mmap(NULL, 85033, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcf55421000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/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\0\20\35\2\0\0\0\0\0"..., 832) =
832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fcf5541f000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fcf54e1c000
mprotect(0x7fcf55003000, 2097152, PROT_NONE) = 0
mmap(0x7fcf55203000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1e7000) = 0x7fcf55203000
mmap(0x7fcf55209000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fcf55209000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fcf554204c0) = 0
mprotect(0x7fcf55203000, 16384, PROT_READ) = 0
mprotect(0x5629e13c8000, 4096, PROT_READ) = 0
mprotect(0x7fcf55436000, 4096, PROT_READ) = 0
munmap(0x7fcf55421000, 85033)           = 0
--------------------------------------------------------------------------------
-----------------Από εδώ και κάτω η έξοδος αντιστοιχεί στον κώδικά μας-----------------
--------------------------------------------------------------------------------
openat(AT_FDCWD, "fconc.out", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
openat(AT_FDCWD, "hello", O_RDONLY)     = 4
read(4, "Hello\n", 1024)                = 6
write(3, "Hello\n", 6)                  = 6
read(4, "", 1024)                       = 0
close(4)                                = 0
openat(AT_FDCWD, "world", O_RDONLY)     = 4
read(4, "World!\n", 1024)               = 7
```

```
write(3, "World!\n", 7)                = 7
read(4, "", 1024)                      = 0
close(4)                               = 0
close(3)                               = 0
exit_group(0)                          = ?
+++ exited with 0 +++
```

Για την παραπάνω εκτέλεση της strace χρησιμοποιήθηκαν δύο αρχεία ως ορίσματα της fconc, το πρώτο περιείχε: “Hello”, ενώ το δεύτερο: “World!”. Όπως, παρατηρούμε η fconc πρώτα δημιουργεί ένα αρχείο, το fconc.out για να γράψει σε αυτό, καταχωρώντας του fd_write=3. Στη συνέχεια ανοίγει το πρώτο αρχείο καταχωρώντας του fd_read=4 [`openat(AT_FDCWD, "hello", O_RDONLY) = 4`]. Στη συνέχεια, το διαβάζει και τοποθετεί σε έναν buffer 1024 θέσεων τους χαρακτήρες που διάβασε επιστρέφοντας το πλήθος τους, εδώ 6 [`read(4, "Hello\n", 1024) = 6`]. Το γράφει από τον buffer στο νέο αρχείο, επιστρέφοντας το πλήθος χαρακτήρων που έγραψε, εδώ πάλι 6 [`write(3, "Hello\n", 6) = 6`]. Έπειτα, αφού δεν υπάρχει κάτι άλλο να διαβάσει από το συγκεκριμένο αρχείο, το κλείνει. Εκτελεί ακριβώς την ίδια διαδικασία για το δεύτερο αρχείο. Τέλος, κλείνει το νέο αρχείο προς εγγραφή και τερματίζει. Η διαδικασία που ακολουθεί ήταν η αναμενόμενη με βάση τον κώδικά μας.