Λειτουργικά Συστήματα

1η αναφορά

Ονοματεπώνυμο	**Εμμανουηλίδης Εμμανουήλ**	**Λίτσος Ιωάννης**
Αριθμός Μητρώου	**03119435**	**03119135**
Ομάδα	**oslab61**	

Άσκηση 1:

Εισαγωγή στο περιβάλλον προγραμματισμού

Σημείωση: οι κώδικες στην αναφορά είναι σε μορφή κειμένου και όχι print screen.

ΑΣΚΗΣΕΙΣ

1.1 Σύνδεση με αρχείο αντικειμένων

Αρχικά, αντιγράφουμε τα αρχεία zing.h και zing.o στον κατάλογο εργασίας μας με τις παρακάτω εντολές :

- cp /home/oslab/code/zing/zing.h zing.h
- cp /home/oslab/code/zing/zing.o zing.o

Στη συνέχεια, δημιουργούμε τη συνάρτηση main.c (η οποία καλεί τη συνάρτηση zing() χρησιμοποιώντας το αρχείο επικεφαλίδας zing.h) μέσω της οποίας παράγουμε το object file main.o . Ο κώδικας της main.c είναι ο εξής:

```c
#include "zing.h"

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

Συνδέουμε (linking) τα δύο object files main.o και zing.o μέσω της εντολής make η οποία χρησιμοποιεί το Makefile (το οποίο αναφέρεται παρακάτω) και έτσι παράγουμε το εκτελέσιμο αρχείο zing. Με την εντολή ./zing εκτελούμε το αρχείο zing το οποίο τυπώνει το παρακάτω μήνυμα :

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.1$ ./zing
Hello, oslaba61
```

1.1.1 Στην επικεφαλίδα γίνεται η δήλωση των συναρτήσεων (και των ορισμάτων τους) καθώς και των global μεταβλητών που πρόκειται να χρησιμοποιηθούν. Με την εντολή #include μπορούμε να ενσωματώσουμε αυτές τις συναρτήσεις και μεταβλητές σε άλλα κομμάτια κώδικα διαφορετικών αρχείων και έτσι ο compiler γνωρίζει τι θα χρειαστεί και από που. Με αυτό τον τρόπο, αφενός γλυτώνουμε χρόνο, αφού μπορούμε να χρησιμοποιούμε έτοιμα κομμάτια κώδικα σε διαφορετικά αρχεία και αφετέρου πετυχαίνουμε optimized compiling.

1.1.2 Makefile

Παραθέτουμε το αρχικό αρχείο Makefile για τη δημιουργία του εκτελέσιμου αρχείου zing.

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

1.1.3 Στο σημείο αυτό γράφουμε τη zing2.c και μέσω αυτής παράγουμε το object file zing2.o . Χρησιμοποιούμε τη συνάρτηση getlogin(3) , η οποία επιστρέφει έναν pointer στο string που περιέχει το username του χρήστη που συνδέθηκε (oslaba61).

zing2.c

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

void zing(void) {
	printf("Hello Manos and Giannis, your username is %s!\n", getlogin());
}
```

Κατόπιν , τροποποιούμε το αρχικό αρχείο Makefile έτσι ώστε να παράγονται δύο εκτελέσιμα αρχεία : ένα με το zing.o (zing) και ένα με το zing2.o (zing2) επαναχρησιμοποιώντας το κοινό object file main.o .

Ο κώδικας για το τροποποιημένο Makefile είναι ο ακόλουθος :

```
all: zing zing2
zing: zing.o main.o
	gcc -o zing zing.o main.o
zing2: zing2.o main.o
	gcc -o zing2 zing2.o main.o
zing2.o: zing2.c
	gcc -Wall -c zing2.c
main.o: main.c
	gcc -Wall -c main.c
```

Εκτελώντας το εκτελέσιμο αρχείο zing2 παίρνουμε το παρακάτω output :

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.1$ ./zing2
Hello Manos and Giannis, your username is oslaba61!
```

1.1.4 Αρχικά, είναι αναμενόμενο ο χρόνος μεταγλώττισης να είναι μεγάλος διότι κάθε φορά που αλλάζουμε μία συνάρτηση μεταγλωττίζουμε εκ νέου όλο το αρχείο το οποίο περιέχει και τις 500 συναρτήσεις . Μία λύση θα ήταν να γράψουμε κάθε συνάρτηση σε ξεχωριστό αρχείο.c έτσι ώστε κάθε φορά να μεταγλωττίζουμε μόνο τη συνάρτηση που τροποποιούμε. Ωστόσο αυτό δε θα ήταν ιδιαίτερα αποδοτικό διότι θα είχαμε πάρα πολλά αρχεία . Για το λόγο αυτό μία πιο αποτελεσματική λύση θα ήταν να διαμοιράσουμε τις συναρτήσεις σε διαφορετικά αρχεία.c τα οποία θα περιέχουν πχ.10 συναρτήσεις το καθένα. Έτσι, για μία αλλαγή σε μία μόνο συνάρτηση θα μεταγλωττίζουμε μόνο το αρχείο που περιέχει την τροποποιηθείσα συνάρτηση.

1.1.5 Η εντολή gcc –Wall –o foo.c foo.c παίρνει το πρώτο argument μετά το –o , δηλαδή το ‘’foo.c’’, και δημιουργεί ένα εκτελέσιμο με αυτό το όνομα , μεταγλωττίζοντας όποιο άλλο όρισμα έχει δοθεί μετά , που στη περίπτωσή μας είναι πάλι το ‘’foo.c’’ .Αυτό έχει ως συνέπεια , η παραπάνω εντολή να κάνει overwrite το αρχείο ‘’foo.c’’ που περιέχει το κώδικα και να δημιουργεί ένα εκτελέσιμο αρχείο με το όνομα ‘’foo.c’’ . Έτσι, μετά την κλήση της παραπάνω εντολής στον τρέχων κατάλογο θα υπάρχει μόνο το εκτελέσιμο αρχείο με όνομα ‘’foo.c’’.

1.2 Συνένωση δύο αρχείων σε τρίτο

Αρχικά, δημιουργούμε το αρχείο fconc_help.c μέσα στο οποίο υλοποιούνται οι συναρτήσεις doWrite και write_file . Ο κώδικας του αρχείου fconc_help.c είναι ο εξής:

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

int doWrite(int fd, const char *buff, int len) {
        size_t  idx = 0;
        ssize_t wcnt;
        do {
                wcnt = write(fd, buff + idx, len - idx);
                if(wcnt == -1) { /* error */
                   perror("write");
                   return 1;
                }
                idx += wcnt;
        } while (idx < len);
        return 0;
}

int write_file(int fd, const char *infile) {

        int o_fd;
        o_fd = open(infile, O_RDONLY);
        if (o_fd == -1) {
                perror("open");
                exit(1);
        }
        char buff[1024];
        ssize_t rcnt;
        for (;;) {
                rcnt = read(o_fd, buff, sizeof(buff)-1);
                if(rcnt == 0) /* end-of-file */
                        return 0;
                if(rcnt == -1) { /* error */
                        perror("read");
                        return 1;
                }
```

```
                buff[rcnt] = '\0';
                doWrite(fd, buff, rcnt);
        }
        close(o_fd);
}
```

Η doWrite(int fd , const char *buff , int len) κάνει εγγραφή στον file descriptor fd χρησιμοποιώντας τη κλήση συστήματος write . Η write δέχεται τα εξής τρία ορίσματα :

- fd το οποίο αντιστοιχεί στον file descriptor του αρχείου στο οποίο γίνεται εγγραφή
- το len – idx που είναι το άνω όριο των bytes που θέλουμε να γράψουμε από τον buffer
- το buff + idx το οποίο δείχνει σε ποιο σημείο του buffer βρισκόμαστε.

Σε περίπτωση επιτυχίας η write επιστρέφει τον αριθμό των bytes που γράφτηκαν , ενώ σε περίπτωση σφάλματος επιστρέφει -1 . Επισημαίνουμε ότι η μεταβλητή wcnt είναι μικρότερη ή ίση από το len – idx καθώς υπάρχει περίπτωση να γράψουμε λιγότερα από (len - idx) bytes.

Η write_file (int fd, const char *infile) διαβάζει τα περιεχόμενα του αρχείου με όνομα infile και τα γράφει στον file descriptor fd χρησιμοποιώντας την doWrite. Για το άνοιγμα και το διάβασμα του αρχείου με το όνομα infile η write_file καλεί τις κλήσεις συστήματος open και read αντίστοιχα, όπως επίσης και την close(o_fd) για το κλείσιμο του αρχείου.

Η open δέχεται τα ορίσματα infile (δηλαδή το αρχείο που θέλουμε να ανοίξει) και το flag 0_RDONLY το οποίο σημαίνει read-only ,δηλαδή άνοιγμα μόνο για ανάγνωση . Σε περίπτωση επιτυχίας η open επιστρέφει έναν ακέραιο (file descriptor) που λειτουργεί ως αναγνωριστικό για τις υπόλοιπες κλήσεις συστήματος (read, write , κλπ).

Η read δέχεται τα εξής τρία ορίσματα :

- o_fd το οποίο αντιστοιχεί στον file descriptor του αρχείου το οποίο θέλουμε να διαβάσουμε (αυτό που ανοίξαμε με την open)
- το sizeof(buff)-1 που είναι το άνω όριο των bytes που θέλουμε να διαβάσουμε στον buffer (buff)
- το buff

Σε περίπτωση επιτυχίας η open επιστρέφει τον αριθμό των bytes που διαβάστηκαν , ενώ σε περίπτωση σφάλματος επιστρέφει -1 . Επισημαίνουμε ότι η μεταβλητή rcnt είναι μικρότερη ή ίση από το sizeof(buff)-1 καθώς υπάρχει περίπτωση να διαβάσουμε λιγότερα bytes από όσα ζητούμε (sizeof(buff)-1) . Αυτό μπορεί να συμβεί αν για παράδειγμα βρισκόμαστε στο τέλος του αρχείου.

Στη συνέχεια, δημιουργούμε το αρχείο fconc.c στο οποίο αρχικά δηλώνουμε τη συνάρτηση write_file . Για να μπορεί το τελικό εκτελέσιμο fconc να χρησιμοποιεί την write_file φτιάχνουμε ένα Makefile στο οποίο γίνεται η σύνδεση των δύο object files fconc_help.o και fconc.o .

Ο κώδικας του αρχείου Makefile είναι ο εξής:

```
fconc: fconc_help.o fconc.o
	gcc -o fconc fconc_help.o fconc.o

fconc_help.o: fconc_help.c
	gcc -Wall -c fconc_help.c

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

Μετά τη δήλωση της write_file ακολουθεί η main :

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

int write_file(int fd, const char *infile);

int main(int argc, char **argv) {
	if(argc != 3 && argc != 4)
		printf("Usage: ./fconc infile1 infile2 [outfile
(default:fconc.out)] \n");
	else {
		int fd_out, oflags, mode;
		oflags = O_CREAT | O_WRONLY | O_TRUNC;
		mode = S_IRUSR | S_IWUSR;
		if(argc == 3) {
			fd_out = open("fconc.out", oflags, mode);
			if(fd_out == -1) {
				perror("open");
				exit(1);
			}
			write_file(fd_out, argv[1]);
			write_file(fd_out, argv[2]);
			close(fd_out);
			exit(0);
		}
		else {
			fd_out = open(argv[3], oflags, mode);
			if(fd_out == -1) {
```

```
                        perror("open");
                        exit(1);
                    }
                    write_file(fd_out, argv[1]);
                    write_file(fd_out, argv[2]);
                    close(fd_out);
                    exit(0);
                }
            }
            return 0;
}
```

Η main, επεξεργάζεται τα ορίσματα του προγράμματος με τις μεταβλητές argc και char **argv. Ο argc ισούται με τον αριθμό των ορισμάτων του προγράμματος, ενώ ο argv είναι πίνακας με τα ορίσματα (argv[0] αντιστοιχεί στο όνομα του προγράμματος). Η λειτουργία της main έχεις ως εξής:

- Ελέγχει αρχικά αν το πλήθος των ορισμάτων είναι έγκυρο και αν δεν είναι τυπώνει ένα κατάλληλο μήνυμα.
- Αν τα ορίσματα είναι τρία (2+1 το όνομα του προγράμματος) τότε καλείται η κλήση συστήματος open με πρώτο όρισμα ‘’fconc.out’’ προκειμένου να δημιουργήσει ένα αρχείο με όνομα fconc.out (ή να το ανοίξει σε περίπτωση που ήδη υπάρχει). Στη συνέχεια, καλείται η συνάρτηση write_file με παραμέτρους τον fd_out , δηλαδή το file descriptor που επέστρεψε η open , και στη πρώτη περίπτωση το argv[1] που αντιστοιχεί στο πρώτο αρχείο εισόδου και στη δεύτερη περίπτωση το argv[2] που αντιστοιχεί στο δεύτερο αρχείο εισόδου.
- Διαφορετικά ,δηλαδή στη περίπτωση που τα ορίσματα είναι 4(3 +1 το όνομα του προγράμματος) , ακολουθείται η ίδια διαδικασία με πριν με μόνη διαφορά ότι τώρα το πρώτο όρισμα της open είναι το argv[3] που αντιστοιχεί στο αρχείο εξόδου στο οποίο θέλουμε να γράψουμε.

Τα oflags επιτελούν τις εξής λειτουργίες :

- O_CREAT : Δημιουργία αρχείου αν δεν υπάρχει (πχ στην open όταν το πρώτο όρισμα είναι ‘’fconc.out’’)
- O_WRONLY: Εγγραφή μόνο
- O_TRUNC : Μηδενισμός αρχείου αν υπάρχει

Τα modes επιτελούν τις εξής λειτουργίες :

- S_IRUSR : Δικαίωμα ανάγνωσης στο κάτοχο
- S_IWUSR : Δικαίωμα εγγραφής στο κάτοχο

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$
```

Παρακάτω ακολουθούν ενδεικτικά παραδείγματα εκτέλεσης :

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ ./fconc A B
open: No such file or directory
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$
```

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat A
Goodbye,
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat B
and thanks for all the fish!
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ ./fconc A B C
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat C
Goodbye,
and thanks for all the fish!
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ ./fconc A B
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat fconc.out
Goodbye,
and thanks for all the fish!
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$
```

Ειδικές περιπτώσεις :

- ./fconc A B A : Σε αυτή τη περίπτωση εφόσον το αρχείο A υπάρχει ήδη , μηδενίζεται (O_TRUNC) οπότε γράφεται μόνο το περιεχόμενο του B στο A.
- ./fconc A A A : Ομοίως με πριν το A μηδενίζεται , επομένως δε γράφεται τίποτα στο A και μένει κενό.
- ./fconc A B B : Αρχικά, το B μηδενίζεται . Εφόσον το B είναι κενό η read που καλείται από την write_file επιστρέφει μηδέν (δεν διαβάστηκε κανένα byte). Άρα, και η μεταβλητή rcnt είναι μηδέν .Επομένως, κάθε φορά θα γράφεται το αποτέλεσμα του A στο B και θα επιστρέφουμε στην αρχή του αρχείου B . Οπότε, τελικά δε θα φτάσουμε ποτέ στο τέλος του αρχείου B με αποτέλεσμα να γράφεται το περιεχόμενο του αρχείου A στο αρχείο B ξανά και ξανά.

2.1 Εντολή strace

Εκτελούμε ένα παράδειγμα του fconc χρησιμοποιώντας την εντολή strace και τοποθετούμε την έξοδο της εντολής strace σε ένα αρχείο με το όνομα ‘’strace_fconc’’ μέσω της εντολής strace –o strace_fconc ./fconc A B C . Το περιεχόμενο της strace_fconc είναι το εξής:

```
execve("./fconc", ["./fconc", "A", "B"], [/* 18 vars */]) = 0
brk(0)                                  = 0x109b000
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) = 0x7f920b91d000
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=32730, ...}) = 0
mmap(NULL, 32730, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f920b915000
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) = 0x7f920b354000
mprotect(0x7f920b4f5000, 2097152, PROT_NONE) = 0
mmap(0x7f920b6f5000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7f920b6f5000
mmap(0x7f920b6fb000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f920b6fb000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f920b914000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f920b913000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f920b912000
arch_prctl(ARCH_SET_FS, 0x7f920b913700) = 0
mprotect(0x7f920b6f5000, 16384, PROT_READ) = 0
mprotect(0x7f920b91f000, 4096, PROT_READ) = 0
munmap(0x7f920b915000, 32730)           = 0
open("fconc.out", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY)                     = 4
read(4, "", 1023)                       = 0
open("B", O_RDONLY)                     = 5
read(5, "ok\n", 1023)                   = 3
write(3, "ok\n", 3)                     = 3
read(5, "", 1023)                       = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
~
~
```

Μέσα σε αυτό το αρχείο μπορούμε να διακρίνουμε διάφορες κλήσεις συστήματος που καλούνται από την fconc όπως για παράδειγμα:

open ("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
Ανοίγει το αρχείο C με σημαίες:

- O_WRONLY για εγγραφή
- O_CREAT για δημιουργία αν δεν υπάρχει
- O_TRUNC για μηδενισμό αν υπάρχει κάτι γραμμένο

Επιστρέφει έναν fd = 3

open ("A", O_RDONLY) = 4
Ανοίγει το αρχείο A μόνο για ανάγνωση και επιστρέφει fd = 4

read (4, "Goodbye, \n", 1023) = 10
Με βάση το fd = 4 διαβάζει "Goodbye, \n" από το αρχείο A και επιστρέφει 10 (= αριθμός των bytes που διάβασε)

write (3, "Goodbye, \n", 10) = 10
Με βάση το fd = 3 γράφει στο αρχείο C "Goodbye, \n" και επιστρέφει 10 (= αριθμός των bytes που γράφηκαν)

read(4, "", 1023) = 0
Ομοίως με νωρίτερα, αλλά τώρα επιστρέφει 0, δηλαδή δε διάβασε τίποτα, δηλαδή end of file.

Στη συνέχεια λειτουργεί με τον ίδιο τρόπο για το αρχείο B.

ΠΡΟΑΙΡΕΤΙΚΕΣ ΕΡΩΤΗΣΕΙΣ

3.1 Με την εντολή strace –o file strace αποθηκεύουμε την έξοδο της εντολής στο αρχείο file , του οποίου το περιεχόμενο είναι το εξής:

```
execve("/usr/bin/strace", ["strace"], [/* 18 vars */]) = 0
brk(0)                                  = 0x563fc69c4000
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) = 0x7fcdbcfda000
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=32730, ...}) = 0
mmap(NULL, 32730, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcdbcfd2000
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) = 0x7fcdbca11000
mprotect(0x7fcdbcbb2000, 2097152, PROT_NONE) = 0
mmap(0x7fcdbcdb2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7fcdbcdb2000
mmap(0x7fcdbcdb8000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fcdbcdb8000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdbcfd1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdbcfd0000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdbcfcf000
arch_prctl(ARCH_SET_FS, 0x7fcdbcfd0700) = 0
mprotect(0x7fcdbcdb2000, 16384, PROT_READ) = 0
mprotect(0x563fc5ab4000, 208896, PROT_READ) = 0
mprotect(0x7fcdbcfdc000, 4096, PROT_READ) = 0
munmap(0x7fcdbcfd2000, 32730)           = 0
rt_sigaction(SIGCHLD, {SIG_DFL, [CHLD], SA_RESTORER|SA_RESTART, 0x7fcdbca460e0}, {SIG_DFL, [], 0}, 8) = 0
getpid()                                = 6627
uname({sys="Linux", node="os-node1", ...}) = 0
brk(0)                                  = 0x563fc69c4000
brk(0x563fc69e5000)                     = 0x563fc69e5000
write(2, "usage: strace [-CdffhiqrtttTvVxx"..., 2227) = 2227
exit_group(1)                           = ?
+++ exited with 1 +++
~
~
~
~
```

Η εντολή strace υλοποιείται με τη κλήση συστήματος uname , η οποία επιστρέφει πληροφορίες του συστήματος . Σε περίπτωση επιτυχίας επιστρέφει μηδέν, ενώ σε περίπτωση σφάλματος επιστρέφει -1. Στο παραπάνω παράδειγμα μας λέει ότι sys =’’Linux’’ , node = ‘‘os-node1’’.

3.2 Για τη δημιουργία του εκτελέσιμου αρχείου zing έγινε σύνδεση (linking) των δύο object files main.o και zing.o . Η zing.o μας πληροφορεί για το πού θα βρούμε τη συνάρτηση zing και έτσι στο όρισμα της εντολής call μπαίνει η zing. Αντίθετα, στη main δεν έχουμε αυτή τη πληροφορία και έτσι στην εντολή call μπαίνει ως όρισμα η διεύθυνση main + 20 όπου θα βρει την zing. Επομένως, η αλλαγή αυτή οφείλεται στον linker και στον compiler.

3.3 Για το ζητούμενο πρόγραμμα τροποποιούμε την main στο αρχείο fconc.c ως εξής :

Ελέγχουμε αν έχουμε τέσσερα και πάνω ορίσματα (3 + 1 το όνομα του προγράμματος) . Σε περίπτωση που έχουμε φτιάχνουμε μία for loop μέσα στην οποία καλούμε την write_file με παραμέτρους τον fd_out που επιστρέφει η open του αρχείου εξόδου argv[argc -1] και το αρχείο εισόδου argv[i] .Επίσης, τροποποιήθηκε η Makefile προκειμένου να παράγει δύο εκτελέσιμα αρχεία, τα fconc και fconc2. Ακολουθούν η τροποποιημένη Makefile και ο τροποποιημένος κώδικας μαζί με ένα παράδειγμα εκτέλεσης:

Makefile:

```
all: fconc fconc2

fconc: fconc_help.o fconc.o
	gcc -o fconc fconc_help.o fconc.o

fconc_help.o: fconc_help.c
	gcc -Wall -c fconc_help.c

fconc.o: fconc.c
	gcc -Wall -c fconc.c

fconc2: fconc_help.o fconc2.o
	gcc -o fconc2 fconc_help.o fconc2.o

fconc2.o: fconc2.c
	gcc -Wall -c fconc2.c
```

fconc2.c:

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

int write_file(int fd, const char *infile);

int main(int argc, char **argv) {
	if(argc == 2 || argc == 1)
		printf("Usage: ./fconc infile1 infile2 [outfile
(default:fconc.out)] \n");
```

```
        else {
                int fd_out, oflags, mode;
                oflags = O_CREAT | O_WRONLY | O_TRUNC;
                mode = S_IRUSR | S_IWUSR;
                fd_out = open(argv[argc-1], oflags, mode);
                if(fd_out == -1) {
                        perror("open");
                                exit(1);
                        }
                int i;
                for(i = 1; i< argc-1; i++) {
                        write_file(fd_out, argv[i]);
                               }
                close(fd_out);
                exit(0);


                }
        return 0;
}
```

```
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat A B C D
-Goodbye,
and thanks for all the fish!
-Hope to see you again!
-Me too!

oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ ./fconc2 A B C D E
oslaba61@os-node1:~/exercises/1st_exercise/exercise_1.2$ cat E
-Goodbye,
and thanks for all the fish!
-Hope to see you again!
-Me too!
```

3.4 Όπως και στο ερώτημα 3.1 , χρησιμοποιούμε την εντολή strace για το εκτελέσιμο αρχείο whoops και αποθηκεύουμε την έξοδο στο αρχείο strace_whoops , παίρνοντας το παρακάτω περιεχόμενο:

```
execve("/home/oslab/code/whoops/whoops", ["/home/oslab/code/whoops/whoops"], [/* 18 vars */]) = 0
brk(0)                                  = 0x9981000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffffffff7779000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=32730, ...}) = 0
mmap2(NULL, 32730, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffffffffff7771000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib32/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\300\233\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1750708, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffffffff7770000
mmap2(NULL, 1755772, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xffffffffff75c3000
mmap2(0xf776a000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a7000) = 0xffffffffff776a000
mmap2(0xf776d000, 10876, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffffffffff776d000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffffffff75c2000
set_thread_area({entry_number:-1, base_addr:0xf75c2700, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0 (entry_number:12)
mprotect(0xf776a000, 8192, PROT_READ)   = 0
mprotect(0xf779e000, 4096, PROT_READ)   = 0
munmap(0xf7771000, 32730)               = 0
open("/etc/shadow", O_RDONLY)           = -1 EACCES (Permission denied)
write(2, "Problem!\n", 9)               = 9
exit_group(1)                           = ?
+++ exited with 1 +++
~
~
~
```

Το πρόβλημα εντοπίζεται στο εξής σημείο :

open ("/etc/shadow", O_RDONLY) = -1 EACCES (Permission denied)

Όπως βλέπουμε η open επιστρέφει error EACCES . Αν πάμε στο manual page της open θα δούμε ότι το error αυτό σημαίνει ότι δεν έχουμε δικαίωμα πρόσβασης σε αυτό το αρχείο .