ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2017-2018

Άσκηση 1: Εισαγωγή στο περιβάλλον προγραμματισμού

	Team: oslaba27
Ονοματεπώνυμο	Αριθμός Μητρώου
Κερασιώτης Ιωάννης	03114951
Ραφτόπουλος Ευάγγελος	03114743

Άσκηση 1.1

Πηγαίος Κώδικας:

main.c

```
/*****************************************************************/
/*                          Operating Systems                    */
/*                          First Lab Exercise                   */
/*                                                               */
/* Exercise 1.1 Main program                                     */
/*                                                               */
/* Team: oslaba27                                                */
/*                                                               */
/* Full Name: Kerasiotis Ioannis, Student ID: 03114951           */
/* Full Name: Raftopoulos Evangelos, Student ID: 03114743        */
/*                                                               */
/*****************************************************************/


#include "zing.h"  // include the header with the declaration of zing() function.

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

	zing();  // calling zing() function

	return (0);

}
```

zing2.c

```
/*****************************************************************/
/*                          Operating Systems                    */
/*                           First Lab Exercise                  */
/*                                                               */
/* Exercise 1.1 Zing2 source code for zing function              */
/*                                                               */
/* Team: oslaba27                                                */
/*                                                               */
/* Full Name: Kerasiotis Ioannis, Student ID: 03114951           */
/* Full Name: Raftopouloes Evangelos, Student ID: 03114743       */
/*                                                               */
/*****************************************************************/


#include <stdio.h> // include stdio.h header for performing input and output
#include <unistd.h> // include unistd.h header for access to operating system

void zing(){

        char *name; // a string buffer which stores pointer that returns from getlogin()
        name = getlogin(); // calling getlogin() function
        if (name == NULL) // if function return null pointer
                perror("getlogin() error"); // return error message
        else
                printf("Hi, %s !\n", name); // else print this message with login name
}
```

Διαδικασία μεταγλώτησης και σύνδεσης:

- Για την δημιουργία του object file του main.c

  $ gcc -Wall -c main.c

- Για την zing:

  $ gcc main.o zing.o -o zing

- Για την zing2:

  $ gcc -Wall -c zing2.c
  $ gcc main.o zing2.o -o zing

Έξοδος εκτέλεσης προγράμματος:

Το πρόγραμμα zing έχει σαν έξοδο ένα μήνυμα της μορφής “Hello, $name” όπου $name το όνομα του του χρήστη που έχει κάνει login, στην συγκεκριμένη περίπτωση του VM που έχει συνδεθεί η κάθε ομάδα. Αντίστοιχα το zing2 βγάζει μήνυμα “Hi, $name !”

Πιο συγκεκριμένα:

1. Για το zing:

   $ ./zing
   Hello, oslaba27

2. Για το zing2:

   $./zing2
   $Hi, oslaba27 !

Ερωτήσεις:

1. Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;

Η επικεφαλίδα είναι ένα αρχείο με επέκταση **.h** το οποίο περιέχει δηλώσεις συναρτήσεων και μακροεντολών της γλώσσας προγραμματισμού C, προκειμένου να μοιραστούν μεταξύ διαφόρων αρχείων πηγαίου κώδικα. Οι επικεφαλίδες μπορεί να είναι είτε του συστήματος, γνωστές ως βιβλιοθήκες, είτε να περιέχουν δηλώσεις συναρτήσεων και μακροεντολών φτιαγμένες από τον προγραμματιστή. Η εισαγωγή και ενσωμάτωση πληροφοριών της επικεφαλίδας γίνεται με την "#include". Στο δικό μας πρόγραμμα η επικεφαλίδα zing.h έχει σκοπό να ορίσει την συνάρτηση void zing().

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

Makefile

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

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

3. Γράψτε το δικό σας zing2.o, το οποίο θα περιέχει zing() που θα εμφανίζει διαφορετικό αλλά παρόμοιο μήνυμα με τη zing() του zing.o. Συμβουλευτείτε το manual page της getlogin(3). Αλλάξτε το Makefile ώστε να παράγονται δύο εκτελέσιμα, ένα με το zing.o, ένα με το zing2.o, επαναχρησιμοποιώντας το κοινό object file main.o.

Στο κομμάτι με το πηγαίο κώδικα το αρχείο zing2.c είναι ο πηγαίος κώδικας που ζητείται. Για την μεταγλώττιση το αρχείο Makefile πρέπει να αλλάξει ως εξής:

Makefile

```
zing: main.o zing.o zing2.o
	gcc main.o zing.o -o zing
	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. Έστω ότι έχετε γράψει το πρόγραμμά σας σε ένα αρχείο που περιέχει 500 συναρτήσεις. Αυτή τη στιγμή κάνετε αλλαγές μόνο σε μία συνάρτηση. Ο κύκλος εργασίας είναι: αλλαγές στον κώδικα, μεταγλώττιση, εκτέλεση, αλλαγές στον κώδικα, κ.ο.κ. Ο χρόνος μεταγλώττισης είναι μεγάλος, γεγονός που σας καθυστερεί. Πώς μπορεί να αντιμετωπισθεί το πρόβλημα αυτό;

Για να γλιτώσουμε χρόνο στην μεταγλώττιση θα μπορούσαμε να γράφαμε κάθε συνάρτηση ή set συναρτήσεων σε διαφορετικό αρχείο πηγαίου κώδικα μεταγλωττίζοντας ανεξάρτητα με τις άλλες συναρτήσεις που δεν έχουν υποστεί αλλάγές. Αυτό όμως προϋποθέτει την δημιουργία μιας ή παραπάνω επικεφαλίδας (header file), στην οποία θα δηλώνονται οι συναρτήσεις που καλούνται από άλλες συναρτήσεις προκειμένου να εκτελέση μια εργασία.

5. Ο συνεργάτης σας και εσείς δουλεύατε στο πρόγραμμα foo.c όλη την προηγούμενη εβδομάδα. Καθώς κάνατε ένα διάλειμμα και ο συνεργάτης σας δούλευε στον κώδικα, ακούτε μια απελπισμένη κραυγή. Ρωτάτε τι συνέβει και ο συνεργάτης σας λέει ότι το αρχείο foo.c χάθηκε! Κοιτάτε το history του φλοιού και η τελευταία εντολή ήταν η:

gcc -Wall -o foo.c foo.c

Τι συνέβη;

Αναζητώντας στο manual του gcc (με χρήση της εντολής *man gcc)*, βλέπουμε πως το argument -o ορίζει το αρχείο εξόδου. Επομένως με την παραπάνω εντολή η μεταγλώτισση θα γίνει και θα αντικαταστήσει το αρχείο του πηγαίου κώδικα με το εκτελέσιμο που θα δημιουργηθεί. Βέβαια, πλέον οι περισσότεροι compilers με την εντολη παρουσιάζουν σφάλμα (**fatal error**). Συγκεκριμένα εμφανίζουν το μήνυμα.

$ **gcc: fatal error:** input file '**foo.c**' is the same as output file
compilation terminated.

Άσκηση 1.2

Πηγαίος Κώδικας:

fconc.c

```
/*****************************************************************/
/*                          Operating Systems                    */
/*                          First Lab Exercise                   */
/*                                                               */
/* Exercise 1.2 fconc program                                    */
/*                                                               */
/* Team: oslaba27                                               */
/*                                                               */
/* Full Name: Kerasiotis Ioannis, Student ID: 03114951           */
/* Full Name: Raftopoulos Evangelos, Student ID: 03114743        */
/*                                                               */
/*****************************************************************/


/************************ Libraries ****************************/

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

/*****************************************************************/

#ifndef BUFF_SIZE // allows "gcc -D" to override definition
#define BUFF_SIZE 1024 // define a constant value at BUFF SIZE
#endif

/************************ Functions ****************************/

void write_file(int fd, const char *infile); //  a function which writes the content of file with the "infile" name to fd file
void doWrite(int fd, const char *buff, ssize_t len); // a function which figurate the writing of file

/*****************************************************************/

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

        if (argc<3 || argc>4){                  // Check if the number of arguments is acceptable
```

```
		printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n"); // if not print this
                                                                                  //message
		exit(1);                                                          // and end the program
	}

	int fd, osFlags, filePerms;

	osFlags = O_CREAT | O_WRONLY | O_TRUNC; // flags for creating a file (if it does exist),
                                        //writing (not reading) and truncate the existing file to zero
	filePerms = S_IRUSR | S_IWUSR;          // Read(0400)  and write(0200) permission bit for
                                        //the owner of the file

	if (argv[3] == NULL) argv[3] = "fconc.out"; // in case there is not a third argument give the by
                                                //default value "fconc.out"
	fd = open(argv[3], osFlags, filePerms);    // open (or create) a file identified by a path name and
                                               //return a file descriptor (fd)
	if (fd < 0){
		perror("Open"); //in case of error display an error message and terminate the program
		exit(1);
	}

	write_file(fd, argv[1]); // call function write_file(2) for first argument
	write_file(fd, argv[2]); // call function write_file(2) for second argument

	if(close(fd) < 0){              //close file descriptor
		perror("Close");        //in case of wrong
		exit(1);
	}

	return (0);

}

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

	int fd1; // a file descriptor for the file which is opened and read
	char buff[BUFF_SIZE]; // a current buffer for reading and writing the content
	ssize_t rcnt; //This data type is used to represent the sizes of blocks that can be read or written in
                  //a single operation

	fd1 = open(infile, O_RDONLY); // opening the file with pathname infile (only for reading) and
                                  //returning at file descriptor fd1
	if (fd1 < 0){                 // in case of wrong opening (for example, if there is not existing file)
		perror(infile);     // display error message
		exit(1);   // and terminate program
	}
```

```
        while(((rcnt = read(fd1, buff, BUFF_SIZE)) != 0 )) { // if the file descriptor is not at the end of
                                                              //file (if fd==0 it is at end)
                if (rcnt == -1){                              // if there is wrong with reading
                        perror("Read");                       // display error
                        exit(1);                      // and terminate the program
                }
                doWrite(fd,buff,rcnt); //calling the function to write the temporary content of the buffer
                                       //to file

        }

        if (close(fd1) < 0){                //close file descriptor
           perror("Close");       //in case of wrong
           exit(-1);            // and terminate
      }

}

void doWrite(int fd, const char *buff, ssize_t len){
        if (write(fd,buff,len) != len){             // write the content and check if it is right
                perror("Could not write whole buffer!");  // else display error
                exit(1);                            // and terminate
        }
}
```

Ερωτήσεις:

1. Εκτελέστε ένα παράδειγμα του fconc χρησιμοποιώντας την εντολή strace. Αντιγράψτε το κομμάτι της εξόδου της strace που προκύπτει από τον κώδικα που γράψατε.

Εκτελώντας τις εντολές:

```
$ man 2 read > A
$ man 2 write > B
```

αποθηκεύομαι το περιέχομενο του manual των συναρτήσεων read(2) και write(2) στα αρχεία A και B αντίστοιχα, προκειμένου να τα χρησιμοποιήσουμε ως ορίσματα στην κλήση του προγράμματος fconc.

Η εντολή strace παρακολουθεί και καταγράφει όλες τις κλήσεις του συστήματος από μια διεργασία και τα σήματα που λαμβάνει από το σύστημα μέχρι να τερματιστεί το πρόγραμμα. Η εντολή strace αποτελεί χρήσιμο εργαλείο για διάγνωση και εντοπισμό σφαλμάτων.

Το αποτέλεσμα της εντολής

```
$ strace -o str_fconc ./fconc A B C
```

Είναι ένα αρχείο με την έξοδο της παραπάνω εντολής.

$ cat str_fconc

```
execve("./fconc", ["./fconc", "a", "b", "c"], [/* 18 vars */]) = 0
brk(0)                                  = 0x15bb000
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) = 0x7fc775fae000
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=30868, ...}) = 0
mmap(NULL, 30868, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc775fa6000
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"..., 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) = 0x7fc7759e5000
mprotect(0x7fc775b86000, 2097152, PROT_NONE) = 0
mmap(0x7fc775d86000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7fc775d86000
mmap(0x7fc775d8c000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc775d8c000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc775fa5000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc775fa4000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc775fa3000
arch_prctl(ARCH_SET_FS, 0x7fc775fa4700) = 0
mprotect(0x7fc775d86000, 16384, PROT_READ) = 0
mprotect(0x7fc775fb0000, 4096, PROT_READ) = 0
munmap(0x7fc775fa6000, 30868)           = 0
open("c", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("a", O_RDONLY)                     = 4
read(4, "READ(2)                  Linux"..., 1024) = 1024
write(3, "READ(2)                  Linux"..., 1024) = 1024
read(4, "f\n       file), and the file pos"..., 1024) = 1024
write(3, "f\n       file), and the file pos"..., 1024) = 1024
read(4, "ned for this case,\n              "..., 1024) = 1024
write(3, "ned for this case,\n              "..., 1024) = 1024
read(4, "group, tries to read from  its  "..., 1024) = 1024
write(3, "group, tries to read from  its  "..., 1024) = 1024
read(4, "st_atime updates on the server a"..., 1024) = 1024
write(3, "st_atime updates on the server a"..., 1024) = 1024
read(4, "ere not atomic with respect upda"..., 1024) = 762
write(3, "ere not atomic with respect upda"..., 762) = 762
read(4, "", 1024)                       = 0
close(4)                                = 0
open("b", O_RDONLY)                     = 4
read(4, "WRITE(2)                 Linux"..., 1024) = 1024
write(3, "WRITE(2)                  Linux"..., 1024) = 1024
read(4, "END, the file offset is\n       f"..., 1024) = 1024
write(3, "END, the file offset is\n       f"..., 1024) = 1024
read(4, "      has been marked nonblockin"..., 1024) = 1024
write(3, "      has been marked nonblockin"..., 1024) = 1024
read(4, "tion-defined maximum file size o"..., 1024) = 1024
write(3, "tion-defined maximum file size o"..., 1024) = 1024
read(4, "gnal.)\n\n       Other errors may "..., 1024) = 1024
write(3, "gnal.)\n\n       Other errors may "..., 1024) = 1024
read(4, "      All of the following funct"..., 1024) = 1024
write(3, "      All of the following funct"..., 1024) = 1024
read(4, " release 3.74 of the Linux man-p"..., 1024) = 334
write(3, " release 3.74 of the Linux man-p"..., 334) = 334
read(4, "", 1024)                       = 0
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```