ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Λειτουργικά Συστήματα

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2020-2021

Ονοματεπώνυμο	Αριθμός Μητρώου
Γουλόπουλος Δημήτριος	03107627
Σιαφάκας Ξενοφών	03115753

oslaba116

Άσκηση 1:Εισαγωγή στο περιβάλλον προγραμματισμού

Άσκηση1.1

Σύνδεση με αρχείο αντικειμένων

Στην άσκηση αυτή ζητείται η δημιουργία ενός εκτελέσιμου αρχείου (με όνομα zing) που θα καλεί τη συνάρτηση zing().

main.c

```
#include <stdio.h>

#include "zing.h"

#include "zing2.h"

int main (int argc , char **argv)

{

      zing();

      return 0;

}
```

Διαδικασία μεταγλώττισης και σύνδεσης:

Σε κάθε ενέργεια εφαρμόζουμε το flag -Wall, ώστε να εμφανίζονται όλα τα warnings.

Για να παραχθεί το εκτελέσιμο που ονομάζεται **zing**:

(a) Για την δημιουργία του object file του main.c, που ονομάζεται main.o
```
$ gcc -Wall -c   main.c
```

(b) Για να παραχθεί το εκτελέσιμο που ονομάζεται zing, από τα main.o και zing.o:
```
$ gcc   main.o   zing.o   -o zing
```

Για να παραχθεί το εκτελέσιμο που ονομάζεται **zing2**:

(a) Για την δημιουργία του object file του zing2.c, που ονομάζεται zing2.o:
```
$ gcc -Wall -c   zing2.c
```

(b) Για να παραχθεί το εκτελέσιμο που ονομάζεται zing2, από τα main.o και zing2.o:
```
$ gcc   main.o   zing2.o   -o zing2
```

Τρέχουμε το πρόγραμμα zing:

```
$ ./zing
Hello, oslaba116
```

το οποίο έχει σαν έξοδο ένα μήνυμα της μορφής "Hello, $USER" όπου $USER το όνομα του χρήστη που έχει κάνει login.

getlogin() returns a pointer to a string containing the name of the user logged in on the controlling terminal of the process.

Τρέχουμε το πρόγραμμα zing2:

```
$./zing2
zing2 here:  oslaba116
```

το οποίο έχει σαν έξοδο ένα μήνυμα της μορφής "zing2 here: $USER" όπου $USER το όνομα του χρήστη που έχει κάνει login.

Ερωτήσεις:

1. Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;

Η επικεφαλίδα (header file) είναι ένα αρχείο με επέκταση **.h** το οποίο περιέχει δηλώσεις συναρτήσεων και μακροεντολών, προκειμένου αυτές να μπορούν να διαμοιραστούν μεταξύ διαφόρων αρχείων πηγαίου κώδικα. Στην άσκηση είχαμε το αρχείο zing.h. Οι συναρτήσεις αυτές μπορεί να είναι είτε της standard library της γλώσσας C, είτε wrappers από system calls προς το λειτουργικό σύστημα, είτε συναρτήσεις φτιαγμένες από τον προγραμματιστή. Οι δηλώσεις που προσδιορίζουν το interface ενός τμήματος κώδικα τοποθετούνται στο header file ενώ η υλοποίηση των συναρτήσεων σε κάποιο άλλο αρχείο με κατάληξη .c

zing.h
```
void zing(void);
```

Χρησιμοποιούμε τα αρχεία επικεφαλίδας (header fles), ωστε να ορίσουμε τις απαραίτητες συναρτήσεις, δομές δεδομένων, σταθερές και preprocessor directives, που θα χρειαστεί το προγραμμά μας και γενικότερα που μπορούν να χρησιμοποιηθούν απο πληθώρα προγραμμάτων όπως για παράδειγμα η κλασσική βιβλιοθήκη της C – stdio.h. Κάνοντας include κάποιο αρχείο επικεφαλίδας σε κάποιο αρχείο κώδικα, είναι σαν να αντιγράφουμε τα περιεχόμενα του αρχείου αυτού στη θέση που κάνουμε include. Αυτό βοηθά τον προγραμματιστή να είναι πιο παραγωγικός, αφου μειώνει το χρόνο που χρειάζεται ώστε να κάνει αλλαγές στον κώδικα (αφου δεν θα χρειάζεται να κανει copy-paste σε όλα τα αρχεία που περιέχουν τον κώδικα που πρέπει να αλλαχτεί) καθώς επίσης η διαδικασία αποσφαλμάτωσης γίνεται πιο εύκολα.

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

Το αρχείο zing.o μας δόθηκε έτοιμο, άρα δε χρειάζεται να φτιαχτεί.

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

```
$  rm  main.o   zing
$  cp  Makefile--original  Makefile
```

Για να παραχθεί το εκτελέσιμο που ονομάζεται zing, αρκεί:

```
$  make
gcc -Wall -c main.c
gcc main.o zing.o -o zing
```

A Makefile is a file named "Makefile" containing a set of directives used by the **make** build automation tool to generate a target/goal.

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.

zing2.c

```
#include <stdio.h>
#include <unistd.h>       // includes  getlogin()
void  zing(char *name)
{
    name = getlogin();
    printf("zing2 here:  %s\n", name);
};
```

Το αρχείο zing.o μας δόθηκε έτοιμο, άρα δε χρειάζεται να φτιαχτεί.

Makefile-- 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
```

```
$  rm  main.o   zing  zing2.o    zing2

$  cp  Makefile--zing2   Makefile

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

```
make tries to achieve the first goal, which is "all".
"all" depends on   zing   and   zing2
"zing" depends on   main.o   and   zing.o
"main.o" depends on   main.c
      gcc -Wall -c main.c
      gcc main.o zing.o -o zing
"zing2" depends on   main.o   and   zing2.o
"zing2.o" depends on   zing2.c
      gcc -Wall -c zing2.c
      gcc main.o zing2.o -o zing2
```

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 ομάδα συναρτήσεων σε διαφορετικό αρχείο πηγαίου κώδικα μεταγλωττίζοντάς το ανεξάρτητα από τις άλλες συναρτήσεις που δεν έχουν υποστεί αλλαγές. Για κάθε αρχείο πηγαίου κώδικα, π.χ. το `abcd.c`, θα αντιστοιχεί ένα αρχείο `abcd.h` που περιλαμβάνει τα πρωτότυπα των συναρτήσεων, το οποίο πρέπει να γίνεται `#include` σε κάθε αρχείο `.c` που καλεί κάποια από τις συναρτήσεις που ορίζονται στο `abcd.c`. Στη συνέχεια, αν γίνουν αλλαγές στο `abcd.c` ή/και στο `abcd.h`, κάνουμε `compile` μόνο το αρχείο αυτό, και `re-link` το εκτελέσιμο από τα `object files`. Με τη χρήση Makefile αυτοματοποείται η διαδικασία compil`ing` και linking.

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 συνέβη 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;

Το argument `-o` ορίζει το αρχείο εξόδου που θα φτιάξει ο `gcc` ως το εκτελέσιμο αρχείο. Επομένως με την παραπάνω εντολή η μεταγλώτισση θα γίνει και θα αντικαταστήσει το αρχείο του πηγαίου κώδικα με το εκτελέσιμο που θα δημιουργηθεί.
Βέβαια, οι περισσότεροι compilers παρουσιάζουν σφάλμα τύπου **"fatal error"** στην περίπτωση αυτή. Συγκεκριμένα εμφανίζουν το μήνυμα:

```
gcc: fatal error: input file ‘foo.c’ is the same as output file
compilation terminated.
```

Άσκηση1.2

Συνένωση δύο αρχείων σε τρίτο

fconc.c

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

#ifndef BUFF_SIZE
#define BUFF_SIZE 1024
#endif

void  write_file(int fd, const char *infile);
void  doWrite(int fd, const char *buff, ssize_t len);

int main(int argc, char **argv)
{
    if (argc < 3 || argc > 4)    // if argc == 3 we have 2 arguments, if argc == 4 we have 3 arguments
    {
        printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)] \n");  // Wrong number of arguments
        exit(1);    // exit program
    }

    int fd, osFlags, filePerms;

    osFlags = O_CREAT | O_WRONLY | O_TRUNC;    // flags to create a file

    filePerms = S_IRUSR | S_IWUSR;             // permission read-write to owner of the file

    if (argv[3] == NULL) argv[3] = "fconc.out";    // 2 arguments case, output file is created with the name "fconc.out"
    fd = open(argv[3], osFlags, filePerms);        // open output file with descriptor "fd" with flags and permissions
    if (fd < 0)                                    // fd is a file descriptor, a FILE pointer, has always positive value
    {
        perror("Open");    // error message if open output file has been erroneous
        exit(1);
    }
    write_file(fd, argv[1]);    // argv[1]: the 1st argument, input file
    write_file(fd, argv[2]);    // argv[2]: the 2nd argument, input file
    if (close(fd) < 0)   // something happened when closing the output file
    {
        perror("Close");    // error message if close output file has been erroneous
        exit(2);
    }
    return 0;
}
```

```
void  write_file(int fd, const char *infile)  // Συνάρτηση που γράφει τα περιεχόμενα του αρχείου με όνομα infile
                                              // στον file descriptor fd. Χρησιμοποιεί την doWrite().
{
    int fd1;
    char buff[BUFF_SIZE];    // a buffer
    ssize_t rcnt;            // buffer block size

    fd1 = open(infile, O_RDONLY);    // open input file
    if (fd1 < 0)
    {
        perror(infile);  // error message if open input file has been erroneous
        exit(1);
    }

    while (((rcnt = read(fd1, buff, BUFF_SIZE)) != 0))    // while fd hasn't found end of file, read BUFF_SIZE bytes
    {
        if (rcnt == -1)
        {
            perror("Read");
            exit(1);
        }
        doWrite(fd, buff, rcnt);    // write buffer to output file
    }

    if (close(fd1) < 0)
    {
        perror("Close");     // error message if close input file has been erroneous
        exit(-1);
    }
}
```

```
void doWrite(int fd, const char *buff, ssize_t len)   // write buffer "buff" to output file and check if all bytes have been written
{
    if (write(fd, buff, len) != len)
    {
        perror("Couldn't write whole buffer!");
        exit(1);
    }
}
```

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.

Για να παραχθεί το εκτελέσιμο που ονομάζεται fconc:

```
gcc -o fconc fconc.c
```

Αν το πρόγραμμα κληθεί χωρίς τα κατάλληλα ορίσματα θα εμφανίζεται μήνυμα βοήθειας.

```
$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]
```

Σε περίπτωση που ένα από τα αρχείο εισόδου δεν υπάρχει, το πρόγραμμα θα πρέπει να εμφανίζει κατάλληλο μήνυμα λάθους.

```
$ ./fconc A B
A: No such file or directory
```

Δημιουργούμε 2 αρχεία κειμένου, A και B:

```
$ echo 'Goodbye,' > A
```

```
$ echo 'and thanks for all the fish!' > B
```

Τα περιεχόμενα του αρχείου εξόδου θα προκύπτουν συνενώνοντας τα περιεχόμενα δύο αρχείων εισόδου. Το πρόγραμμα (fconc) θα δέχεται δύο ή τρία ορίσματα.

- Το πρώτο και το δεύτερο όρισμα είναι τα αρχεία εισόδου.
- Το τρίτο όρισμα είναι προαιρετικό και είναι το αρχείο εξόδου.
- Η προεπιλεγμένη (default) τιμή για το αρχείο εξόδου είναι fconc.out

Εδώ η έξοδος πάει στο προεπιλεγμένο αρχείο εξόδου fconc.out:

```
$ ./fconc A B
```

```
$ cat fconc.out
Goodbye,
and thanks for all the fish!
```

Εδώ η έξοδος πάει στο αρχείο εξόδου C:

```
$ ./fconc A B C
```

```
$ cat C
Goodbye,
and thanks for all the fish!
```

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   result.txt   ./fconc A B C
```

```
$cat  result.txt
```

```
execve("./fconc", ["./fconc", "A", "B", "C"], [/* 18 vars */]) = 0
brk(0)                                  = 0x672000
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) = 0x7f54dab3c000
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=30952, ...}) = 0
mmap(NULL, 30952, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f54dab34000
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) = 0x7f54da573000
mprotect(0x7f54da714000, 2097152, PROT_NONE) = 0
mmap(0x7f54da914000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7f54da914000
mmap(0x7f54da91a000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f54da91a000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f54dab33000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f54dab32000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f54dab31000
arch_prctl(ARCH_SET_FS, 0x7f54dab32700) = 0
mprotect(0x7f54da914000, 16384, PROT_READ) = 0
mprotect(0x7f54dab3e000, 4096, PROT_READ) = 0
```

```
munmap(0x7f54dab34000, 30952)           = 0
open("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY)                     = 4
read(4, "Goodbye,\n", 1024)             = 9
write(3, "Goodbye,\n", 9)               = 9
read(4, "", 1024)                       = 0
close(4)                                = 0
open("B", O_RDONLY)                     = 4
read(4, "and thanks for all the fish!\n", 1024) = 29
write(3, "and thanks for all the fish!\n", 29) = 29
read(4, "", 1024)                       = 0
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```