

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΑΝΑΦΟΡΑ

Άσκηση 1

Αρετή Μέη 03120062

Τρις-Ελευθερία Παλιατσού 03120639

1.1 Σύνδεση με το αρχείο αντικειμένων

Ο πηγαίος κώδικας της άσκησης είναι ο εξής:

Αρχείο main.c για τη συνάρτηση main():

```
#include <stdio.h>
#include "zing.h"

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

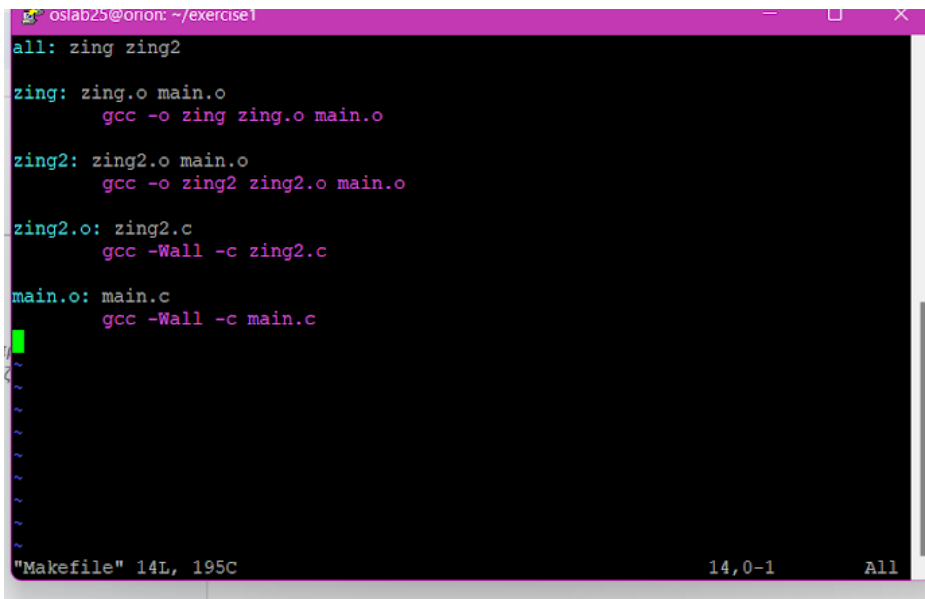
Ο κώδικας για το zing2.c:

```
oslab25@orion: ~/exercise1
#include <stdio.h>
#include <unistd.h>

void zing() {
    char *a = getlogin();
    printf("Nice to meet you %s! \n", a);
}
```

Η διαδικασία μεταγλώττισης και σύνδεσης:

Φτιάχνουμε ένα Makefile, στο οποίο κάνουμε και τη μεταγλώττιση και τη σύνδεση(αφορά στη δημιουργία 2 εκτελέσιμων, zing.o & zing2.o):



```
oslab25@orion: ~/exercise1
all: zing zing2

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

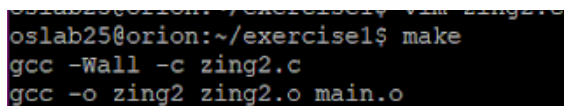
zing2: zing2.o main.o
    gcc -o zing2 zing2.o main.o

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

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

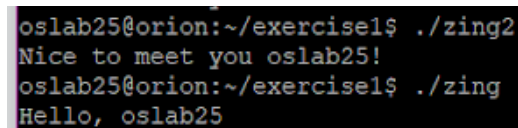
"Makefile" 14L, 195C                               14,0-1    All
```

Κάνουμε make για την αυτόματη δημιουργία του προγράμματος από τα αρχεία κώδικα, σύμφωνα με τις εντολές που έχουμε γράψει μέσα στο Makefile:



```
oslab25@orion: ~/exercise1$ make
gcc -Wall -c zing2.c
gcc -o zing2 zing2.o main.o
```

Η έξοδος εκτέλεσης των εκτελέσιμων zing2 & zing:



```
oslab25@orion:~/exercise1$ ./zing2
Nice to meet you oslab25!
oslab25@orion:~/exercise1$ ./zing
Hello, oslab25
```

Ερωτήσεις:

1. Στην επικεφαλίδα (header file) γίνεται η δήλωση μεταβλητών και συναρτήσεων, και χωρίζονται οι δηλώσεις μεταβλητών και συναρτήσεων από το σώμα τους. Έτσι, μπορούν οι συναρτήσεις αυτές να χρησιμοποιηθούν σε διάφορα σημεία στο πρόγραμμα (ή και σε διάφορα source files) και διευκολύνεται η διόρθωση, καθώς η αλλαγή γίνεται μόνο σε ένα σημείο του κώδικα.
- 2.. Το Makefile για τη δημιουργία του εκτελέσιμου zing.o:

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

3. Το δεύτερο Makefile στο οποίο γίνεται το compiling και το linking των zing.o και zing2.o:

```
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
```

4. Το πρόβλημα μπορεί να αντιμετωπισθεί με το σπάσιμο του κώδικα των συναρτήσεων και την προσθήκη header files στην main(), με τα σώματα των συναρτήσεων σε διαφορετικά Source files. Έτσι, μειώνεται ο χρόνος του compilation, μειώνεται ο χώρος, και διευκολύνεται η διόρθωση αφού οι αλλαγές γίνονται μόνο στο σώμα των συναρτήσεων και ισχύουν όπου χρησιμοποιούνται οι συναρτήσεις αυτές, χωρίς να χρειάζεται να αλλάξουμε πολλά σημεία του κώδικα.

5. Μετά το -o, θα έπρεπε να έχουμε το αρχείο του output file, ενώ εδώ έχουμε κατα λάθος βάλει το όνομα του input file, με αποτέλεσμα το εκτελέσιμο που δημιουργείται να γράφεται πάνω στο source code, να γίνεται δηλαδή ένα overwrite.

1.2 Συνένωση δύο αρχείων σε τρίτο

Ο πηγαίος κώδικας της άσκησης (αρχείο fconc.c):

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

void doWrite(int fd, const char *buff, int len) { //this is the function that copies the elements of buff into the output file
    int idx;
    idx = 0;

    ssize_t wcnt;
    do {
        wcnt = write(fd, buff+idx, len-idx); //wcnt: number of bytes written
        if(wcnt == -1) { //error
            perror("write");
            return 1;
        }
        idx += wcnt;
    } while(idx < len); //if idx == len, which means that every element of buff has been copied to fd(output file), this loop runs only 1 time
}

void write_file(int fd, const char *infile) { //this is the function that copies A and B input files to buffer (read only mode)
    char buff[1024];
    ssize_t rcnt; // we use ssize_t(signed integer), so that the variable rcnt can take the value -1, which indicates that something went wrong
    int fdop; //we will copy fdop, which is the infile to our buff

    fdop = open(infile, O_RDONLY); //file descriptor obtained by opening infile

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

    for(;;) { //we use an infinite loop, so that if our input file is bigger than our buffer size, the loop must repeat to copy the whole file to the buff
        rcnt = read(fdop, buff, sizeof(buff)-1); //read: return the number of bytes read
        if(rcnt == 0) //end of file
            break;
        if(rcnt == -1) { //error
            perror("read");
        }
    }
}
```

1,1

Top

```
}

int main(int argc, char **argv) {
    if((argc<3) || (argc>4)) //checking if the number of arguments is valid, argv[0]=program name
        printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");
    else {
        int fd, oflags, mode; //fd for output
        oflags = O_CREAT | O_WRONLY | O_TRUNC; //flags: create file if it doesn't exist, write only mode, truncate the file if it already exists*/
        mode = S_IRUSR | S_IWUSR; //read & write permission for the owner*/

        if(argc == 3) { // we dont have the output file C as a given argument
            fd = open("fconc.out", oflags, mode);
        }

        else {
            fd = open(argv[3], oflags, mode); //C is one of the arguments
        }

        if(fd == -1) { //error opening the output file
            perror("open");
            exit(1);
        }

        write_file(fd, argv[1]); //com line argument 1 (input file A)
        write_file(fd, argv[2]);

        close(fd); //ensures all data buffered in mem by the os is written to disk, frees up the system/file descriptors
    }
    return 0;
}

-- INSERT --
```

Ερωτήσεις:

```
oslab25@orion:~/exercisel/exercisel2$ echo 'Goodbye, ' > A
oslab25@orion:~/exercisel/exercisel2$ echo 'and thanks for all the fish!' > B
oslab25@orion:~/exercisel/exercisel2$ ./fconc A B C
oslab25@orion:~/exercisel/exercisel2$
```

1.

```

oslab25@orion:~/exercise1/exercise12$ strace ./fconc A B C
execve("./fconc", [ "./fconc", "A", "B", "C"], [/* 20 vars */]) = 0
brk(0) = 0xde9000
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) = 0x7f1fd7b1c000
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=29766, ...}) = 0
mmap(NULL, 29766, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1fd7b14000
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\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) = 0x7f1fd7553000
mprotect(0x7f1fd76f4000, 2097152, PROT_NONE) = 0
mmap(0x7f1fd78f4000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7f1fd78f4000
mmap(0x7f1fd78fa000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1fd78fa000
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fd7b13000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fd7b12000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fd7b11000
arch_prctl(ARCH_SET_FS, 0x7f1fd7b12700) = 0
mprotect(0x7f1fd78f4000, 16384, PROT_READ) = 0
mprotect(0x7f1fd7b1e000, 4096, PROT_READ) = 0
munmap(0x7f1fd7b14000, 29766) = 0
open("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY) = 4
read(4, "Goodbye, \n", 1023) = 10
write(3, "Goodbye, \n", 10) = 10
read(4, "", 1023) = 0
close(4) = 0
open("B", O_RDONLY) = 4
read(4, "and thanks for all the fish!\n", 1023) = 29
write(3, "and thanks for all the fish!\n", 29) = 29
read(4, "", 1023) = 0
close(4) = 0
close(3) = 0
exit_group(0) = ?
+++ exited with 0 +++

```

Γενικά , η strace, η οποία παίρνει ως όρισμα κάποια εντολή(εδώ το cat C), μας δείχνει όλα τα system calls που γίνονται από ένα πρόγραμμα και μας βοηθά στο debugging αυτών των system calls, ενώ παράλληλα μας επιτρέπει να δούμε τόσο τα ορίσματα όσο και τα return values αυτών των κλήσεων συστήματος.