Λειτουργικά Συστήματα – Άσκηση 1

Κυριακόπουλος Γιώργος – el18153
Τζελέπης Σεραφείμ – el18849

1.1

Πηγαίος κώδικας:

$cp -r ../code/zing .

$vim main.c

```
#include “zing.h”

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

$gcc -Wall -c main.c

$vim Makefile

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

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

$make

$./zing

```
Hello, oslaba40
```

Ερωτήσεις:

1. Η επικεφαλίδα αποτελεί τη διεπαφή προς άλλα κομμάτια κώδικα, η οποία περιέχει τη δήλωση συναρτήσεων και καθολικών μεταβλητών (global). Χρησιμεύει, επίσης, στο να ξέρουμε τι συναρτήσεις υπάρχουν διαθέσιμες, χωρίς απαραίτητα να έχουμε πρόσβαση στα .c files και στον έλεγχο μέσω του compiler της έγκυρης υλοποίησης και σχέσης μεταξύ .h και .c file.

2. Υπάρχει ήδη έτοιμο και υλοποιημένο στον πηγαίο κώδικα.

3. $vim zing2.c

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

void zing(void) {
        char *s = getlogin();
        printf("Welcome, %s\n", s);
}
```

$vim Makefile

```
all: zing zin2 //target: two executables

zing2: zing2.o main.o
        gcc -o zing2 zing2.o main.o //linking

zing: zing.o main.o
        gcc -o zing zing.o main.o //linking

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

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

$./zing2

```
Welcome, oslaba40
```

4. Αναλόγως με το εάν θέλουμε να πειράζουμε μόνο μία συγκεκριμένη συνάρτηση συνολικά ή παραπάνω από μία συνάρτηση, θα χωρίσουμε τον κώδικα των 500 συναρτήσεων, είτε σε δύο αρχεία με 499+1 συναρτήσεις, είτε αναδρομικά σε περισσότερα αρχεία με (498+1)+1 συναρτήσεις κ.ο.κ. με την τελική περίπτωση των 500 ξεχωριστών αρχείων με 1 συνάρτηση το καθένα. Έπειτα, με τη βοήθεια ενός Makefile θα ενώνουμε τα αρχεία αυτά για να παράγουμε το τελικό μας αρχείο και έτσι θα καταφέρουμε να γίνεται compile κάθε φορά μόνο το αρχείο (ή περισσότερα αρχεία, αλλά σίγουρα λιγότερα από 500) με τη 1 νέα αλλαγμένη συνάρτηση και να μην απαιτείται κάθε φορά χρόνος για compile και των 500 συναρτήσεων του κώδικα.

5. Το foo.c αρχείο έχει πλέον το output του αρχικού foo.c λόγω της τελευταίας εντολής. Επομένως, το αρχικό foo.c που περιείχε τον κώδικα c έχει χαθεί. Θα έπρεπε η τελευταία εντολή να είναι της μορφής: gcc -Wall foo.c -o foo ή gcc -Wall -o foo foo.c, όπου το foo είναι το destination (executable) και το foo.c το source file και στις δύο περιπτώσεις.

1.2

Πηγαίος κώδικας:

$vim fconc.c

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

void doWrite(int fd, const char *buff, int len) {
        size_t idx = 0;
        ssize_t wcnt;

        do {
                //write to output file
                wcnt = write(fd, buff + idx, len - idx);

                if(wcnt == -1) {
                        perror("Can't write to output file.");
                        exit(1);
                }

                idx += wcnt;
        } while (idx < len);
}

void write_file(int fd, const char *infile) {
        //open input file
        int fd1 = open(infile, O_RDONLY);
        if(fd1 == -1) {
                perror("Can't open given output file.");
                exit(1);
        }

        ssize_t rcnt;
        int buffsize = 1024;
```

```
        for(;;) {
                //malloc buffsize characters
                char *buff = (char *)malloc(buffsize*sizeof(char));
                if(!buff) {
                        perror("Failed to allocate memory.");
                        exit(1);
                }

                //read at most buffsize-1 characters from input file
                rcnt = read(fd1, buff, buffsize - 1);
                //if end of file is read
                if(rcnt == 0) {
                        break;
                } else if(rcnt == -1) {
                        perror("Can't read given input file.");
                        exit(1);
                }

                //if less than buffsize-1 characters are read, realloc less memory
                if(rcnt + 1 != buffsize) {
                        buff = realloc(buff, rcnt*sizeof(char));

                        if(!buff) {
                                perror("Failed to reallocate memory.");
                                exit(1);
                        }
                }

                doWrite(fd, buff, strlen(buff));
                free(buff);
        }
        close(fd1);
}

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

        //check for valid number of arguments
        if(argc < 3 || argc > 4) {
                printf("Usage: ./fconc infile1 infile 2 [outfile (default: fconc.out)].");
                exit(1);
        }
```

```
        int fd, oflags, mode;
        oflags = O_CREATE | O_WRONLY | O_TRUNC;
        mode = S_IRUSR | S_IWUSR;

        //select output file depending on if its given
        char *outputFile = “fconc.out”;
        if(argc == 4) {
                outputFile = argv[3];
        }

        //open output file
        fd = open(outputFile, oflags, mode);
        if(fd == -1) {
                perror(“Can’t open output file.”);
                exit(1);
        }

        write_file(fd, argv[1]);
        write_file(fd, argv[2]);

        close(fd);
        return 0;
}
```

Ερωτήσεις:

$echo ‘Hello everybody, this is the first sample file of the exercise.’ > A

$echo ‘This is the second example file of the exercise, goodbye everyone!’ > B

$strace ./fconc A B C

```
execve("./fconc", ["./fconc", "A", "B", "C"], [/* 18 vars */]) = 0

brk(0)                          = 0x236a000

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) = 0x7f17efada000

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) = 0x7f17efad2000

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) = 0x7f17ef511000

mprotect(0x7f17ef6b2000, 2097152, PROT_NONE) = 0

mmap(0x7f17ef8b2000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7f17ef8b2000

mmap(0x7f17ef8b8000, 14880, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f17ef8b8000

close(3)                                = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f17efad1000

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f17efad0000

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f17efacf000

arch_prctl(ARCH_SET_FS, 0x7f17efad0700) = 0

mprotect(0x7f17ef8b2000, 16384, PROT_READ) = 0

mprotect(0x7f17efadc000, 4096, PROT_READ) = 0

munmap(0x7f17efad2000, 30952)           = 0

open("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3

open("A", O_RDONLY)                     = 4
```

```
brk(0)                          = 0x236a000
brk(0x238b000)                  = 0x238b000
read(4, "Hello everybody, this is the fir"..., 1023) = 64
write(3, "Hello everybody, this is the fir"..., 64) = 64
read(4, "", 1023)               = 0
close(4)                        = 0
open("B", O_RDONLY)             = 4
read(4, "This is the second sample file o"..., 1023) = 66
write(3, "This is the second sample file o"..., 66) = 66
read(4, "", 1023)               = 0
close(4)                        = 0
close(3)                        = 0
exit_group(0)                   = ?
+++ exited with 0 +++
```

Μέσω της strace βλέπουμε ότι τα αναμενόμενα system calls καλούνται, με τις σωστές παραμέτρους και επιστρέφουν τα αποτελέσματα που περιμέναμε. Για παράδειγμα, ανοίγει το αρχείο output με fd = 3, στη συνέχεια ανοίγει το πρώτο input file με fd = 4, γίνεται κάποιο memory allocation και ακολουθούν τα read από το αρχείο εισόδου με fd = 4 στον buffer και τα write από τον buffer στο αρχείο εξόδου με fd = 3, χωρίς κάποιο πρόβλημα, το διάβασμα του EOF, το κλείσιμο του πρώτου αρχείου input με fd = 4 και η όμοια διαδικασία για το δεύτερο αρχείο input με τελικά system calls το κλείσιμο του output αρχείου και την έξοδο της διαδικασίας.