**1η Εργασία**

**στα**

***Λειτουργικά Συστήματα***

Ομάδα D12

των: Καραμπάση Αικατερίνη 03112517

Μανδηλαράς Νικηφόρος 03112012

****

Σ.Η.Μ.Μ.Υ.

Ε.Μ.Π.

2015-2016

**Άσκηση 1.1**

Βήματα:

1. Αντιγράφουμε τα αρχεία zing.h και zing.o στον κατάλογο εργασίας μας δίνοντας τις εντολές:

cp /home/oslab/code/zing/zing.h home/oslab/oslabd12/zing.h

cp /home/oslab/code/zing/zing.o home/oslab/oslabd12/zing.o

1. Στη συνέχεια συντάσσουμε τον πηγαίο κώδικα της main() η οποία καλεί τη συνάρτηση zing(). Η συνάρτηση zing δε δέχεται ορίσματα, οπότε έχουμε την παρακάτω μορφή:

#include "zing.h"

int main(){

zing();

return 0;

}

Για τη δημιουργία του object file της main() δίνουμε την εξής εντολή στο terminal:

gcc –Wall –c main.c

Μόλις εκτελεστεί η εντολή αυτή, δημιουργείται στο directory που βρισκόμαστε και είναι και η main(), το αρχείο main.o.

1. Για να δημιουργήσουμε το εκτελέσιμο αρχείο μας πρέπει να συνδέσουμε τα object files και να ονοματίσουμε το αρχείο μας. Οπότε γράφουμε στο terminal την εντολή:

gcc main.o zing.o –o zing

διότι θέλουμε το εκτελέσιμο αρχείο μας να ονομάζεται zing. Η εντολή που δώσαμε μόλις στο terminal ονομάζεται linking των δύο αρχείων, δηλαδή η main() που χρειάζεται τη zing(), με τη δεύτερη να βρίσκεται σε ξεχωριστό αρχείο, μετά το linking ξέρει σε ποια διεύθυνση μνήμης μπορεί να τη βρει.

Για να δοκιμάσουμε το πρόγραμμα που φτιάξαμε, εκτελούμε τη ./zing και στο terminal εμφανίζεται: “Hello oslabd12!”

**Ερωτήσεις:**

1. Τη στιγμή που καλούμαστε να γράψουμε ένα πρόγραμμα είναι καλό να το χωρίσουμε σε μικρά κομμάτια, αυτοτελή. Αυτό μας βοηθάει αρχικά στο χρόνο μεταγλώττισης του εκάστοτε προγράμματος κι έπειτα στην επαναχρησιμοποίηση κάποιων συναρτήσεων σε άλλα προγράμματα, κάτι το οποίο μας γλιτώνει κόπο και χρόνο στη συνολική δουλειά μας. Ωστόσο, για να τρέξει μία συνάρτηση που δεν είναι γραμμένη στο ίδιο αρχείο με εκείνο της main() θα χρειαστεί η main() να βρει πού είναι αυτή η συνάρτηση, ώστε όταν έρθει η ώρα να την καλέσει. Αυτό, όταν δεν έχουμε τον κώδικα της συνάρτησης αλλά το header file της, γίνεται μέσω της εντολής #include file.h στον πηγαίο κώδικα της main(). Επιπλέον, αν χρειαστεί να κάνουμε κάποια αλλαγή σε μία συνάρτηση, δεν είναι απαραίτητο να κάνουμε compile όλες τις συναρτήσεις που το πρόγραμμά μας καλεί (κάτι το οποίο θα μπορούσε να ήταν αρκετά χρονοβόρο) αλλά να κάνουμε compile τη συγκεκριμένη συνάρτηση και linking με το κυρίως πρόγραμμα.
2. Τα Makefiles αποτελούν έναν τρόπο εξυπηρέτησης ώστε να μη γράφουμε συνεχώς τις εντολές για το compiling και το linking. Έτσι, σε ένα κοινό Makefile μπορούμε να οργανώσουμε τις εντολές αυτές για κάθε πρόγραμμα που θέλουμε να παράγουμε και με την εντολή make target να δημιουργείται αυτό. Για να καταλάβουμε τη χρησιμότητά του ας σκεφτούμε να έχουμε 15 συναρτήσεις οι οποίες αλληλοεξαρτώνται και καταλήγουν στον βασικό πρόγραμμα και τι θα έπρεπε να γράφαμε κάθε φορά που π.χ διορθώναμε ένα λάθος ή βελτιώναμε τον κώδικα μίας εκ των συναρτήσεων.

Για το Makefile της zing() γράφουμε:

zing:main.o zing.o

gcc main.o zing.o –o zing

main.o:

gcc –Wall –c main.c

1. Είδαμε πως το zing() μας εμφανίζει ένα μήνυμα χαιρετισμού, οπότε καλούμαστε να γράψουμε ένα πρόγραμμα που θα εμφανίζει κάτι παρόμοιο. Για να εμφανίσουμε το όνομα του user μας θα αξιοποιήσουμε τη συνάρτηση getlogin() η οποία μας επιστρέφει έναν δείκτη σε θέση μνήμης που είναι το username μας. Ο κώδικας της zing2.c είναι:

#include "stdio.h"

#include "unistd.h"

int zing(){

char \*msg;

msg = getlogin();

printf("It's us, %s\n", msg);

return 0;

}

Ονομάσαμε τη συνάρτησή μας zing() ώστε να χρησιμοποιήσουμε το κοινό main.o για την παραγωγή των εκτελέσιμων αρχείων μας χωρίς να μας βγάζει κάποιο σφάλμα. Επομένως, τροποποιώντας το Makefile που γράψαμε προηγουμένως έχουμε:

all:p1 p2

p1:main.o zing.o

gcc main.o zing.o -o p1

p2:zing2.o

gcc main.o zing2.o -o p2

zing2.o:

gcc -Wall -c zing2.c

main.o:

gcc -Wall -c main.c

Τρέχοντας στο terminal το make all μας εμφανίζει:

Hello oslabd12!

It’s us, oslabd12

1. Όπως ειπώθηκε και προηγουμένως, είναι πολύ πιο συμφέρον για το χρόνο μας αλλά και για τη δουλειά μας να γράφουμε συναρτήσεις που κάνουν διαφορετικές δουλειές, σε διαφορετικά αρχεία. Κάτι τέτοιο, μας σώζει χρόνο την επόμενη φορά που θα θέλουμε να αξιοποιήσουμε μία συνάρτηση σε ένα άλλο πρόγραμμα. Εκτός αυτού, ο χρόνος μεταγλώττισης και ένωσης την πρώτη φορά θα είναι και στις δύο περιπτώσεις περίπου ο ίδιος. Ωστόσο, τις φορές που θα αλλάζουμε τον κώδικα σε μία συνάρτηση και θα χρειάζεται πάλι compiling και linking η σύγκριση είναι 500 προς 1, διότι όταν θα τα έχουμε όλα μαζί θα πρέπει να τα ξανακάνει όλα, ενώ όταν θα τα έχουμε σε ξεχωριστά αρχεία, θα κάνουμε compile το μοναδικό αρχείο στο οποίο έγιναν οι αλλαγές και linking όλα μαζί πάλι, πράγμα που δε θα κοστίσει, ειδικά αν σκεφτούμε το κόστος του compiling 500 συναρτήσεων.
2. Η εντολή που έγραψε ο συνεργάτης δεν περιείχε αρχείο προς μεταγλώττιση, λόγω της λανθασμένης σύνταξης της εντολής. Έτσι, με αυτήν την εντολή γράφουμε πάνω στο αρχείο foo.c ένα κενό αρχείο που του δίνουμε ίδιο όνομα με τον κώδικά μας, κι έτσι χάνουμε τον κώδικά μας.

**Άσκηση 1.2**

Το πρόγραμμα το οποίο συντάξαμε, σύμφωνα και με τον προτεινόμενο σκελετό υλοποίησης των συναρτήσεων ήταν το εξής:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

void doWrite (int fd, const char \*buff, int len)

{

int wcnt, idx=0;

do{

wcnt = write(fd, buff+idx, len-idx);

if (wcnt == -1) //error

{

perror("write");

return;

}

idx += wcnt;

} while (idx<len);

}

void write\_file(int fd, const char \*infile)

{

int fd\_infile, rcnt;

char buff[1024];

fd\_infile = open(infile, O\_RDONLY); //open input file only for read

if (fd\_infile == -1)

{

perror("open");

exit(1);

}

for (;;)

{

rcnt = read(fd\_infile, buff, sizeof(buff)-1);

if (rcnt == 0) //end-of-file

return;

if (rcnt == -1) //error

{

perror("read");

return;

}

doWrite(fd, buff, rcnt);

}

}

int main(int argc, char \*argv[])

{

if (argc < 3) //Zero or one input files

{

fprintf(stdout, "Usage: ./fconc infile1 infile2 [outifle (default:fconc.out)]\n");

return 0;

}

int fd, oflags, mode;

oflags = O\_CREAT | O\_WRONLY | O\_TRUNC ;

mode = S\_IRUSR | S\_IWUSR ;

if (argc ==3) //Two input files

{

fd = open("fconc.out", oflags, mode); //there is no output file, we create fconc.out

if (fd == -1)

{

perror("open");

exit(1);

}

write\_file(fd , argv[1]);

write\_file(fd , argv[2]);

}

else if (argc == 4) //Two input files, one output file

{

fd = open(argv[3], oflags, mode); //There is already one output file, we just open it

if (fd == -1)

{

perror("open");

exit(1);

}

write\_file(fd , argv[1]);

write\_file(fd , argv[2]);

}

else //More than three files

{

fprintf(stdout, "Usage: ./fconc infile1 infile2 [outifle (default:fconc.out)]\n");

return 0;

}

close(fd);

return 0;

}

Δημιουργούμε το εκτελέσιμο αρχείο μας δίνοντας όνομα fconc και εξετάζουμε τη λειτουργία του δίνοντας διαφορετικό αριθμό ορισμάτων κάθε φορά:

./fconc A

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

./fconc A B

A: No such file or directory

echo “Hello and have a wonderful day”> A

echo “The sun hasn’t come out yet and yet we dance waiting for the rain”> B

./fconc A B

cat fconc.out

Hello and have a wonderful day

The sun hasn’t come out yet and yet we dance waiting for the rain

./fconc A B C

cat C

Hello and have a wonderful day

The sun hasn’t come out yet and yet we dance waiting for the rain

echo “And when is it gonna be at last?”> D

./fconc A B D fconc.out

cat fconc.out

Hello and have a wonderful day

The sun hasn’t come out yet and yet we dance waiting for the rain

And when is it gonna be at last?

**Ερωτήσεις:**

1. Εκτελώντας την εντολή strace για το αρχείο fconc που δημιουργήσαμε γράφουμε strace ./fconc και στο terminal εμφανίζονται τα εξής:

execve("./fconc", ["./fconc"], [/\* 27 vars \*/]) = 0

brk(0) = 0x804a000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

mmap2(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb78a0000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

open("/etc/ld.so.cache", O\_RDONLY) = 3

fstat64(3, {st\_mode=S\_IFREG|0644, st\_size=67766, ...}) = 0

mmap2(NULL, 67766, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0xb788f000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

open("/lib/i686/cmov/libc.so.6", O\_RDONLY) = 3

read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\260l\1\0004\0\0\0\34"..., 512) = 512

fstat64(3, {st\_mode=S\_IFREG|0755, st\_size=1331684, ...}) = 0

mmap2(NULL, 1337704, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0xb7748000

mmap2(0xb7889000, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x141) = 0xb7889000

mmap2(0xb788c000, 10600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0xb788c000

close(3) = 0

mmap2(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb7747000

set\_thread\_area({entry\_number:-1 -> 6, base\_addr:0xb77476c0, limit:1048575, seg\_32bit:1, contents:0, read\_exec\_only:0, limit\_in\_pages:1, seg\_not\_present:0, useable:1}) = 0

mprotect(0xb7889000, 8192, PROT\_READ) = 0

mprotect(0xb78be000, 4096, PROT\_READ) = 0

munmap(0xb788f000, 67766) = 0

fstat64(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 0), ...}) = 0

mmap2(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb789f000

write(1, "Usage: ./fconc infile1 infile2 [o"..., 61Usage: ./fconc infile1 infile2 [outifle (default:fconc.out)]

) = 61

exit\_group(0) = ?

**Προαιρετικές ερωτήσεις:**

1. Για να δούμε με ποια κλήση συστήματος υλοποιείται η strace γράφουμε στο terminal: strace –o file1 ώστε να μας εμφανίσει το αποτέλεσμα στο file1, οπότε έχουμε:

usage: strace [-CdffhiqrtttTvVxxy] [-I n] [-e expr]...

[-a column] [-o file] [-s strsize] [-P path]...

-p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]

or: strace -c[df] [-I n] [-e expr]... [-O overhead] [-S sortby]

-p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]

-c -- count time, calls, and errors for each syscall and report summary

-C -- like -c but also print regular output

-d -- enable debug output to stderr

-D -- run tracer process as a detached grandchild, not as parent

-f -- follow forks, -ff -- with output into separate files

-i -- print instruction pointer at time of syscall

-q -- suppress messages about attaching, detaching, etc.

-r -- print relative timestamp, -t -- absolute timestamp, -tt -- with usecs

-T -- print time spent in each syscall

-v -- verbose mode: print unabbreviated argv, stat, termios, etc. args

-x -- print non-ascii strings in hex, -xx -- print all strings in hex

-y -- print paths associated with file descriptor arguments

-h -- print help message, -V -- print version

-a column -- alignment COLUMN for printing syscall results (default 40)

-b execve -- detach on this syscall

-e expr -- a qualifying expression: option=[!]all or option=[!]val1[,val2]...

options: trace, abbrev, verbose, raw, signal, read, write

-I interruptible --

1: no signals are blocked

2: fatal signals are blocked while decoding syscall (default)

3: fatal signals are always blocked (default if '-o FILE PROG')

4: fatal signals and SIGTSTP (^Z) are always blocked

(useful to make 'strace -o FILE PROG' not stop on ^Z)

-o file -- send trace output to FILE instead of stderr

-O overhead -- set overhead for tracing syscalls to OVERHEAD usecs

-p pid -- trace process with process id PID, may be repeated

-s strsize -- limit length of print strings to STRSIZE chars (default 32)

-S sortby -- sort syscall counts by: time, calls, name, nothing (default time)

-u username -- run command as username handling setuid and/or setgid

-E var=val -- put var=val in the environment for command

-E var -- remove var from the environment for command

-P path -- trace accesses to path

1. Ο κώδικας σε assembly της main() δημιουργήθηκε από το main.o ενώ του εκτελέσιμου αρχείου από τα main.o και zing.o. Δηλαδή, πριν τη σύνδεση της zing με τη main η main δεν ξέρει πού βρίσκεται η zing, ενώ μετά το linking ξέρει τη διεύθυνσή της, ώστε να την καλέσει όταν έρθει η ώρα. Αυτή η αλλαγή, επομένως, έγινε κατά το linking από τον compiler.
2. Οι συναρτήσεις doWrite() και write\_file() παραμένουν ακριβώς οι ίδιες. Αυτό που θα αλλάξει είναι η main() και ο έλεγχος των αρχείων εισόδου και εξόδου. Τροποποιώντας την έχουμε:

int main(int argc, char \*argv[])

{

if (argc < 3) //Zero or one input files

{

fprintf(stdout, "Usage: ./fconc infile1 infile2 [outifle (default:fconc.out)]\n");

return 0;

}

int fd, oflags, mode;

oflags = O\_CREAT | O\_WRONLY | O\_TRUNC ;

mode = S\_IRUSR | S\_IWUSR ;

if (argc ==3) //Two input files

{

fd = open("fconc.out", oflags, mode); //there is no output file, we create fconc.out

if (fd == -1)

{

perror("open");

exit(1);

}

Write\_file(fd , argv[1]);

Write\_file(fd , argv[2]);

}

else if (argc >= 4) //Two or more input files, one output file

{

int i;

fd = open(argv[argc-1], oflags, mode); //There is already one output file, we just open it

if (fd == -1)

{

perror("open");

exit(1);

}

for(i=1; i<=argc-2; i++)

{

if (\*argv[i]!=\*argv[argc-1])

{

Write\_file(fd , argv[i]);

}

else

{

fprintf(stdout, "Usage: ./fconc infile1 infile2 [outifle infile3 different from infile1 or infile2]\n");

return 0;

}

}

}

close(fd);

return 0;

}

1. Δίνοντας τη συγκεκριμένη εντολή στο terminal όντως μας εμφανίζει Problem! Για να βρούμε το πρόβλημα μπορούμε να αντιγράψουμε το αρχείο στο φάκελο εργασίας μας και να αξιοποιήσουμε την strace η οποία θα μας δώσει αρνητική τιμή εκεί που θα βρίσκεται το σφάλμα που συνέβη. Οπότε έχουμε:

execve("./whoops", ["./whoops"], [/\* 27 vars \*/]) = 0

brk(0) = 0x804a000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

mmap2(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb7866000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

open("/etc/ld.so.cache", O\_RDONLY) = 3

fstat64(3, {st\_mode=S\_IFREG|0644, st\_size=67766, ...}) = 0

mmap2(NULL, 67766, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0xb7855000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

open("/lib/i686/cmov/libc.so.6", O\_RDONLY) = 3

read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\260l\1\0004\0\0\0\34"..., 512) = 512

fstat64(3, {st\_mode=S\_IFREG|0755, st\_size=1331684, ...}) = 0

mmap2(NULL, 1337704, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0xb770e000

mmap2(0xb784f000, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x141) = 0xb784f000

mmap2(0xb7852000, 10600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0xb7852000

close(3) = 0

mmap2(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb770d000

set\_thread\_area({entry\_number:-1 -> 6, base\_addr:0xb770d6c0, limit:1048575, seg\_32bit:1, contents:0, read\_exec\_only:0, limit\_in\_pages:1, seg\_not\_present:0, useable:1}) = 0

mprotect(0xb784f000, 8192, PROT\_READ) = 0

mprotect(0xb7884000, 4096, PROT\_READ) = 0

munmap(0xb7855000, 67766) = 0

open("/etc/shadow", O\_RDONLY) = -1 EACCES (Permission denied)

write(2, "Problem!\n"..., 9Problem!

) = 9

exit\_group(1) = ?

Παρατηρούμε πως η κλήση συστήματος open του αρχείου shadow για ανάγνωση αποτυγχάνει δίνοντας σφάλμα open("/etc/shadow", O\_RDONLY) = -1. Άρα υπάρχει σφάλμα στην κλήση συστήματος open.