ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολη ΗΜ&ΜΥ
Λειτουργικά Συστήματα
Άσκηση 1
Ακ. Έτος 2012-13

Παπαδιάς Σεραφείμ | ΑΜ: 03109193
Παπαδημητρίου Κων/νος | ΑΜ: 03108769
Ημερομηνία: 29/11/2012

<u>Άσκηση 1.1</u> Σύνδεση με αρχείο αντικειμένων

Αρχικά αντιγράφουμε το αρχείο επικεφαλίδας zing.h και το αρχείο αντικειμένου zing.o στον κατάλογο εργασίας μας.Η επικεφαλίδα είναι:

```
// FILE: zing.h
#ifndef ZING_H__
#define ZING_H__

void zing(void);

#endif
```

Το αρχείο κώδικα main.c που γράψαμε για να καλεί την zing() περιέχει:

```
// FILE: main.c
#include <stdio.h>
#include "zing.h"

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

Για να μεταγλώττισουμε το αρχείο main.c εκτελούμε την εντολή:

gcc -Wall -c main.c zing.h

με τη οποία ο GCC δημιουργεί το object file main.o (χρησιμοποιώντας την επικεφαλίδα της zing.h)
Η σύνδεση (linking) των δυο αρχείων έγινε με την εντολή:

gcc main.o zing.o -Wall -o zing

Μετά εκτελούμε το εκτελέσιμο που δημιουργήθηκε μετά το linking και μας τυπώνει:
Hello oslaba04!

<u>Έρωτησεις</u>

1.*<u>Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;</u>*

Όταν μία συνάρτηση έχει οριστεί σε ένα αρχείο (file1.c) και εμείς θέλουμε να τη χρησιμοποιήσουμε σε ένα άλλο αρχείο (file2.c) θα πρέπει αυτή η συνάρτηση να έχει δηλωθεί σε ένα ξεχωριστό αρχείο (header.h) το οποίο θα γίνει include στο 2ο αρχείο (file2.c).
Στη περίπτωσή μας, η συνάρτηση zing() έχει οριστεί στο object file zing.o, οπότε για να τη χρησιμοποιήσουμε στο αρχείο main.c πρέπει να τη δηλώσουμε σε ένα header (το zing.h) και στη συνέχεια το header αυτό να το κάνουμε include στο main.c

2.*Ζητείται κατάλληλο Makefile για τη δημιουργία του εκτελέσιμου της άσκησης.*
Το Makefile που φτιάξαμε είναι:

```
all: zing1
zing1: zing.o main.o
	gcc -o zing zing.o main.o -Wall

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

clean:
	rm zing main.o
```

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

Φτιάξαμε το αρχείο zing2.c:

```
// FILE: zing2.c
#include <unistd.h>		//getlogin
#include <stdio.h>
#include <string.h>		//strcat
#include "zing.h"

void zing(void) {
	char str[BUFSIZ] = "Good bye ";
	strcat(str, getlogin());
	strcat(str, "!\n");
	write(1, str, BUFSIZ);
}
```

Αλλάζουμε και το Makefile σε:

```
all: zing1 zing2

zing1: 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
```

```
clean:
	rm main.o zing zing2 zing2.o
```

Τώρα το Makefile παράγει δύο εκτελέσιμα αρχεία, τα zing και zing2 επαναχρησιμοποιώντας κοινό object file, το main.o.

Και κάνοντας make και εκτελώντας
./zing2
παίρνουμε:
Goodbye oslaba04!

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;*

Ομαδοποιούμε τις συναρτήσεις σε περισσότερα του ενός αρχεία. Μπορούμε να μεταλωττίζουμε κάθε αρχείο ξεχωριστά, παράγοντας object files. Έτσι κάθε φορά που αλλάζουμε τον κώδικα ενός αρχείου μπορούμε να μεταγλωττίσουμε μόνο αυτό και όχι όλο το πρόγραμμα από την αρχή. Με τη χρήση ενός Makefile μπορούμε να αυτοματοποιήσουμε αυτή τη διαδικασία και κάθε φορά να μεταγλωττίζονται μόνο τα αρχεία που έχουν υποστεί κάποια αλλαγή.

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;

Προσπάθησε να παράξει ένα εκτελέσιμο αρχείο από το foo.c όμως έδωσε το ίδιο όνομα εκτελέσιμου με το αρχείο πηγαίου κώδικα.Επομένως στο αρχείο foo.c που είχαμε τον πηγαίο κώδικα γράφτηκε το εκτελέσιμο.Άρα έχουμε εκτελέσιμο που μπορούμε να το τρέξουμε , αλλά χάσαμε τον πηγαίο κώδικα.

Άσκηση 1.2 Συνένωση δύο αρχείων σε τρίτο

Ο πηγαίος κώδικας για αυτήν την άσκηση βρίσκεται στο αρχείο fcon.c, ενώ έχουμε και το fcon.h το οποίο περιέχει τις δηλώσεις των συναρτήσεων, καθώς και διάφορες βιβλιοθήκες που χρησιμοποιούμε. Τα 2 αρχεία περιέχουν:

```
//  FILE: fcon.h
#include <fcntl.h>
#include <unistd.h>
```

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

// PERM: όταν δημιουργείται ένα αρχείο, προσπάθησε να δώσεις PERM permissions
#define PERM S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH
#define OP_FLAGS O_CREAT | O_WRONLY | O_TRUNC

void doWrite(int, const char *, int);
void write_file(int, const char *);
void fcon(const char *, const char *, const char *);
void move(const char *, const char *);
```

To fcon.c:

```
//  FILE: fcon.c
#include "fcon.h"

void fcon(const char *a, const char *b, const char *c)
{
        int fd;
        if ((fd = open(c, OP_FLAGS, PERM)) < 0){     // άνοιξε το output_file
                perror("open");                      // error handling
                exit(1);
        }

        write_file(fd, a);                           // γράψε στο output_file το inp_a
        write_file(fd, b);                           // γράψε στο output_file το inp_b

        if (close(fd) < 0) {                         // κλείσε το output_file
                perror("close");
                exit(1);
        }
}


/* Η move() ακολουθεί την ίδια λογική με την fcon(), μόνο που έχει μόνο ένα αρχείο για
 * είσοδο.
 */
void move(const char *a, const char *b)
{
        int fd;
        if ((fd = open(b, OP_FLAGS, PERM)) < 0){
                perror("open");
                exit(1);
        }
```

```
        write_file(fd,a);

        if (close(fd) < 0){
                perror("close");
                exit(1);
        }

        if (unlink(a) < 0)      //διέγραψε το tmp file
                printf("Please delete /tmp/fcon.out.tmp manually");
}



void
write_file(int fd, const char *infile)
{
        int in;
        int r;
        char buf[BUFSIZ];
        if ((in = open(infile, O_RDONLY)) < 0){                 // άνοιξε το input_file
                perror("open");                                 // error handling
                exit (1);
        }
        //   while !EOF doWrite()
        while ((r = read(in, buf, BUFSIZ)) > 0){        // διάβαζε το input_file και γράφε
                doWrite(fd, buf, r);                    // στο output_f (όσα bytes διάβασες)
        }                                               // μέχρι να φτάσεις στο τέλος

        if (r < 0){
                perror("read");
                exit(1);
        }
        if (close(in) < 0){
                perror("close");
                exit(1);
        }
}

/* με τη doWrite(), γράφουμε το buf στο αρχείο (που έχει f descriptor fd).
 * Επειδή η κλήση συστήματος write() μπορεί να μην καταφέρει να γράψει ολόκληρο
 * το buf με τη μια, την καλούμε όσες φορές χρειάζεται, μέχρι να ολοκληρωθεί το έργο μας.
 */
void
doWrite(int fd, const char *buf, int len)
{
        int rem = len;
        do {
```

```
                rem -= write(fd, buf + len-rem, rem);
                if ((rem -= write(fd, buf + len-rem, rem)) < 0){
                        perror("write");
                        exit(1);
                }
        }while (rem > 0);
}

int
main(int argc, char **argv)
{
        const char *out;
        int i;
        int same_name=0;
  // parsing main arguments.
        if (argc == 3)
                out = "fcon.out";               // αν δεν δώσουμε το output_file σαν όρισμα στην
        else{                                   // κλήση του προγράμματος δουλεύουμε στο fcon.out
                if (argc == 4)
                        out = argv[3];
                else{                           // αν δώσουμε λάθος ορίσματα τύπωσε λάθος
                        printf("Usage: ./fcon input1 input2 [output]\n");
                        exit(1);
                }
        }

/* σε περίπτωση που ένα από τα input_files το χρησιμοποιούμε και για output_file,
 * δουλεύουμε πρώτα σε ένα temporary file και μετά το αντιγράφουμε στο αρχικό.
 */
  // if input1 or input2 matches output work on fcon.out.tmp
        for (i=1; i<3; i++){
                if (strcmp(argv[i], out) == 0){
                        same_name = 1;
                        break;
                }
        }
        if (same_name){
                fcon(argv[1], argv[2], "/tmp/fcon.out.tmp");
                move("/tmp/fcon.out.tmp",out);
        }
        else
                fcon(argv[1], argv[2], out);

        exit(0);
}
```

Και το Makefile:

```
CCFLAGS = -Wall -O3
all: fcon
fcon: fcon.o
	gcc -o fcon fcon.o ${CCFLAGS}
fcon.o: fcon.c
	gcc -c fcon.c ${CCFLAGS}
clean:
	rm fcon.o
	rm fcon
```

Ερωτησεις

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

Η εντολή strace εντοπίζει (εκτός άλλων) τις κλήσεις συστήματος που γίνονται κατα την εκτέλεση ενός προγράμματος. Στη δική μας περίπτωση, το κομμάτι της εξόδου που αφορά τον κώδικα που γράψαμε είναι:

```
open("fcon.out", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("A", O_RDONLY)                     = 4
read(4, "Poli\n", 8192)                 = 5
write(3, "Poli\n", 5)                   = 5
write(3, "", 0)                         = 0
read(4, "", 8192)                       = 0
close(4)                                = 0
open("B", O_RDONLY)                     = 4
read(4, "kalhspera sas!\n", 8192)       = 15
write(3, "kalhspera sas!\n", 15)        = 15
write(3, "", 0)                         = 0
read(4, "", 8192)                       = 0
close(4)                                = 0
close(3)                                = 0
exit_group(0)                           = ?
```

Είναι οι κλήσεις συστήματος (και οι τιμές που επιστρέφουν) που πραγματοποιήθηκαν κατά την εκτέλεση:
./fcon A B
με το Α να περιέχει το string:
"Poli\n"
και το Β το string:
"kalhspera sas!\n"
Σε περίπτωση που είχαμε εκτελέσει το πρόγραμμα με διαφορετικά arguments προφανώς κάποιες κλήσεις θα ήταν διαφορετικές.

Σημείωση: *Επειδή κάποια σημεία ίσως δεν φαίνονται αρκετά καθαρά, επισυνάπτουμε τον κώδικα που γράψαμε καθώς και ένα αρχείο με την έξοδο της εντολής strace.*

3. Προαιρετικές ερωτήσεις

1. *Χρησιμοποιήστε την εντολή strace για να εντοπίσετε με ποια κλήση συστήματος*

υλοποιείται η εντολή strace. Υπόδειξη: με strace -o file η έξοδος της εντολής τοποθετείται στο αρχείο file.

Εκτελώντας την εντολή:
strace -o strace.out strace
στο αρχείο strace.out βλέπουμε ποια κλήση συστήματος χρησιμοποιείται για από το πρόγραμμα strace. Αυτή είναι η ptrace.
Πρόκειται για μια κληση η οποία επιτρέπει σε μία διεργασία να ελέγχει μία άλλη.
Χρησιμοποιείται από διάφορα προγράμματα όπως το strace και το ltrace καθώς και από αρκτετούς debuggers όπως ο gdb.

2. *(...)*

Την αλλαγή αυτή την κάνει ο linker σε στάδιο μετά τη μεταγλώττιση. Συγκεκριμένα, οφείλεται στο ότι ο linker θα αποτιμήσει την τιμή της διεύθυνσης που βρίσκεται η συνάρτηση, αφού πάρει το αρχείο zing.o, όπου και θα μας δώσει το τελικό εκτελέσιμο zing.

3. *Γράψτε το πρόγραμμα της άσκησης 1.2, ώστε να υποστηρίζει αόριστο αριθμό αρχείων εισόδου (π.χ. 1, 2, 3, 4, ...). Θεωρήστε ότι το τελευταίο όρισμα είναι πάντα το αρχείο εξόδου.*

(Ο τροποποιημένος κώδικας επισυνάπτεται)

4. *Εάν τρέξετε το εκτελέσιμο /home/oslab/code/whoops/whoops θα δώσει:*
$ /home/oslab/code/whoops/whoops
Problem!
Θεωρήστε ότι πραγματικά υπάρχει πρόβλημα. Εντοπίστε το.

Εκτελώντας την εντολή
strace -o whoops.out /home/oslab/oslaba04/ask1/whoops/whoops
βλέπουμε στη γραμμή 22 του αρχείου whoops.out, πως το πρόγραμμα προσπαθεί να ανοίξει το αρχείο
/etc/shadow
χωρίς να έχει τα κατάλληλα δικαιώματα. Έτσι τυπώνει (αφού ενημερώσει το errno) στην οθόνη
Problem! (γραμμή 23)