ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα Υπολογιστών

1η εργαστηριακή αναφορά: Εισαγωγή στο περιβάλλον προγραμματισμού

Διδάσκοντες:

Ν. Κοζύρης
Γ. Γκούμας

oslaba84:

Ειρήνη Δόντη
ΑΜ 03119839

6ο εξάμηνο

Αθήνα 2022

Περιεχόμενα:

1.1 Σύνδεση με αρχείο αντικειμένων

Βήματα:

1. Τα αρχεία zing.h και zing.o βρίσκονται στο μονοπάτι /home/oslab/code/zing. Μεταβαίνουμε σε αυτό το μονοπάτι με την εντολή **ls /home/oslab/code/zing**. Αντιγράφουμε τα δύο αρχεία στον κατάλογο της εργασίας μας κάνοντας χρήση της εντολής **cp -r /home/oslab/code/zing .** με την οποία παίρνουμε το περιεχόμενο του φακέλου zing και το αντιγράφουμε στον τωρινό μας κατάλογο (υποδηλώνεται με την χρήση της τελείας .).

   Όταν τρέχουμε τον κώδικα του εκτελέσιμου αρχείου zing λαμβάνουμε το εξής αποτέλεσμα:

```
oslaba84@os-node1:~/myzing/zing$ ./zing
Hello, oslaba84
```

2. Με την εντολή vim main.c δημιουργούμε ένα αρχείο με όνομα main, το οποίο περιέχει τον παρακάτω κώδικα:

```
#include"zing.h"

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

   Στην συνέχεια μεταγλωττίζουμε το αρχείο με χρήση της εντολής **gcc -Wall -c main.c** και προκύπτει το αρχείο αντικειμένων main.o.

   Για δική μας ευκολία δημιουργούμε έναν κατάλογο ονόματι myzing (**mkdir myzing**) όπου μεταφέρουμε όλα τα αρχεία της άσκησης.

3. Συνδέουμε τα δύο αρχεία αντικειμένων zing.o και main.o στο τελικό εκτελέσιμο αρχείο ονόματι zing κάνοντας χρήση της εντολής:

   gcc main.o myzing/zing/zing.o -o zing.

Ερωτήσεις:

1. Σκοπός της επικεφαλίδας είναι η δήλωση συναρτήσεων και των ορισμάτων τους που θα χρησιμοποιηθούν στον κεντρικό κώδικα. Με αυτόν τον τρόπο, ο compiler θα γνωρίζει τι μεταβλητές ορίζονται για κάποια συνάρτηση και σε ποιο σημείο του κώδικα θα τις βρει. Επίσης, η επικεφαλίδα είναι χρήσιμη, καθώς οι συναρτήσεις που ορίζει, μπορεί να χρησιμοποιούνται και σε άλλα αρχεία με κώδικα και συνεπώς να γίνεται οικονομία χώρου και χρόνου.
2. **Makefile -** Κάνοντας χρήση της εντολής *vim Makefile* δημιουργούμε αρχείο όπου γράφουμε τον παρακάτω κώδικα (κανόνες):

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

Στη συνέχεια καλούμε την εντολή *make* για την παραγωγή του προγράμματος.Το αρχείο Makefile καθώς και την κλήση της εντολής make τα κάνουμε μέσα στον ίδιο φάκελο myzing/zing.

Παρατηρήσεις: Το zing είναι το αρχείο-στόχος, το zing.o και το main.o είναι τα αρχεία-απαίτηση του zing με το δεύτερο. Το αρχείο main.o γίνεται, εν συνεχεία αρχείο στόχος με αρχείο-απαίτηση το main.c. Οι εντολές gcc -o zing zing.o main.o και gcc -Wall -c main.c είναι οι εντολές παραγωγής των στόχων από τα απαιτούμενα αρχεία.

3. Δημιουργούμε αρχείο zing2.c, που περιέχει τη συνάρτηση zing(), με χρήση της εντολής **vim zing2.c**, που περιέχει τον ακόλουθο κώδικα:

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

void zing(){
printf("No more war, %s\n", getlogin());
}
```

Τρέχουμε τον κώδικα του αρχείου zing2 και λαμβάνουμε το εξής αποτέλεσμα:

```
oslaba84@os-node1:~/myzing/zing$ ./zing2
No more war, oslaba84
```

Μεταγλωττίζουμε το zing2.c μέσω της εντολής **gcc -Wall -c zing2.c** και λαμβάνουμε το αρχείο αντικειμένων zing2.o.

Στη συνέχεια αλλάζουμε το makefile ώστε να παράγονται δύο εκτελέσιμα, ένα με το zing.o και ένα με το zing2.o επαναχρησιμοποιώντας το κοινό object file main.o. Το νέο makefile που προκύπτει είναι το ακόλουθο:

Makefile

```
all: zing zing2 zing2.o main.o
zing: main.o zing.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
```

Την πρώτη φορά που εκτελούμε την εντολή make, λαμβάνουμε τα παρακάτω:

```
oslaba84@os-node1:~/myzing/zing$ vim Makefile
oslaba84@os-node1:~/myzing/zing$ make
gcc -Wall -c zing2.c
gcc -o zing2 zing2.o main.o
```

Αν εκτελέσουμε την εντολή κάποια επόμενη φορά, θα λάβουμε το κάτωθι:

```
oslaba84@os-node1:~/myzing/zing$ make
make: Nothing to be done for 'all'.
```

Αυτό είναι λογικό, καθώς το αρχείο-στόχος εξαρτάται από τα object files (αρχεία-απαίτηση). Αν το αρχείο-στόχος και τα object files δεν μεταβληθούν, τότε το linux θεωρεί ότι δεν χρειάζεται να κάνει κάτι για όλα τα αρχεία.

4. Όταν θέλουμε να κάνουμε αλλαγές μόνο σε μία από τις 500 συναρτήσεις, ο χρόνος μεταγλώττισης θα είναι μεγάλος και θα έχουμε καθυστέρηση. Για να αντιμετωπίσουμε αυτό το πρόβλημα μπορούμε να διαχωρίσουμε τις συναρτήσεις σε διαφορετικά αρχεία τύπου .c και για καθένα από αυτά τα αρχεία θα κατασκευάσουμε ένα αρχείο τύπου .o που θα βρίσκεται στο Makefile. Εκεί θα βρίσκονται επίσης τα αρχεία που χρησιμοποιούν κάποια συνάρτηση που περιέχει το αντίστοιχο αρχείο τύπου .o. Συνεπώς, όταν θέλουμε να κάνουμε αλλαγή σε μία μόνο συνάρτηση, θα εκτελεστούν ξανά μόνο τα αρχεία που καλούν την συνάρτηση που αλλάξαμε .
5. Η εντολή gcc -Wall -o foo.c foo.c σημαίνει ότι, από το αρχείο foo.c, παράγεται ένα εκτελέσιμο με όνομα foo.c. Όταν εκτελέσουμε αυτήν την εντολή, θα χαθεί το περιεχόμενο του αρχείου κώδικα. Αυτό οφείλεται στο γεγονός ότι θα γίνει overwrite το όνομα του αρχείου τύπου .c με αυτό του εκτελέσιμου. Οπότε, το νέο αρχείο επιβάλλει το περιεχόμενο του στο παλιό, με αποτέλεσμα, αφού το νέο είναι κενό, να γίνει κενό και το παλιό αρχείο. Δηλαδή, το περιεχόμενό του αρχείου foo.c διαγράφεται.

1.2 Συνένωση δύο αρχείων σε τρίτο

Κάνοντας χρήση της εντολής **vim fconc.c**, δημιουργούμε αρχείο ονόματι fconc που περιέχει τη συνάρτηση **void doWrite(int fd, const char *buff, int len)**, τη συνάρτηση **void write_file(int fd, const char *infile)** και μια κύρια συνάρτηση **main(int argc, char **argv)**.

Παρακάτω, παρατίθεται ο ζητούμενος κώδικας ο οποίος περιέχει τις συναρτήσεις που αναφέραμε παραπάνω:

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

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

        do{
                wcnt = write(fd, buff + idx, len - idx);
                if(wcnt == -1) {
                        perror("write");
                        return;
                }
                idx += wcnt;
        } while(idx < len);
}
```

```c
void write_file(int fd, const char *infile){

        char buff[1024];
        ssize_t rcnt;

        ssize_t open_fd = open(infile, O_RDONLY);

        if(open_fd == -1){
                perror("open");
                exit(1);

        }

        for(;;){
                rcnt = read(open_fd, buff, sizeof(buff)-1);
                if(rcnt == 0){
                        return;
                }
                if(rcnt == -1){  /*error*/
                        perror("read");
                        return;
                }
                doWrite(fd, buff, rcnt);
        }

        close(open_fd);
}
```

```c
int main(int argc, char **argv){
        if(argc <3 || argc >4){ /* if no suitable arguments */
                printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.
out)]");
        }
        else if(argc>=3 && argc<=4){
                int exit_fd, oflags, mode;
                oflags = O_CREAT | O_WRONLY | O_TRUNC;
                mode = S_IRUSR | S_IWUSR;

                if(argc == 3){ /* if .out exists */
                        exit_fd = open("fconc.out", oflags, mode);
                }
                else { /* if .out doesn't exist */
                        exit_fd = open(argv[3], oflags, mode);
                }

                if(exit_fd == -1){
                        perror("open");
                        exit(1);
                }
                else{
                        write_file(exit_fd, argv[1]);
                        write_file(exit_fd, argv[2]);
                }
                close(exit_fd);
        } return 0;
}
```

Στη συνέχεια, εκτελούμε τις εντολές **gcc -Wall -c fconc.c** και **gcc fconc.o -o fconc**, για να μεταγλωττίσουμε το αρχείο και να παράξουμε το αρχείο τύπου .o.

Δοκιμάζουμε το παράδειγμα που μας δίνεται στην εκφώνηση και λαμβάνουμε τα κάτωθι αποτελέσματα:

```
oslaba84@os-node1:~$ vim fconc.c
oslaba84@os-node1:~$ gcc -Wall -c  fconc.c
oslaba84@os-node1:~$ gcc fconc.o -o fconc
oslaba84@os-node1:~$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]oslaba84@os-node1:~
$ ./fconc A B
oslaba84@os-node1:~$ echo 'Goodbye,' > A
oslaba84@os-node1:~$ echo 'and thanks for all the fish!' > B
oslaba84@os-node1:~$ ./fconc A B
oslaba84@os-node1:~$ cat fconc.out
Goodbye,
and thanks for all the fish!
oslaba84@os-node1:~$ ./fconc A B C
oslaba84@os-node1:~$ cat C
Goodbye,
and thanks for all the fish!
oslaba84@os-node1:~$
```

Σημείωση: Την πρώτη φορά που δοκιμάσαμε να τρέξουμε τις εντολές **./ fconc A** και **./ fconc A B**, τα μηνύματα που λαμβάναμε ήταν:

```
oslaba84@os-node1:~$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]oslaba84@os-node1:~
```

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

Αυτό είναι λογικό, καθώς δεν είχαμε ορίσει το μήνυμα που αντιστοιχούσε στο A. Αν ορίσουμε τα A και B και δοκιμάσουμε ξανά να εκτελέσουμε τις ίδιες εντολές, θα λάβουμε μήνυμα ότι τα A και B είναι φάκελοι.

Ερωτήσεις:

1. Εκτελούμε την εντολή **strace ./fconc A B** και λαμβάνουμε τα παρακάτω αποτελέσματα:

```
oslaba84@os-node1:~$ strace ./fconc A B
execve("./fconc", ["./fconc", "A", "B"], [/* 26 vars */]) = 0
brk(0)                                  = 0x1871000
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) = 0x7f
37e3620000
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=32730, ...}) = 0
mmap(NULL, 32730, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f37e3618000
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"..., 83
2) = 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) = 0x7
f37e3057000
mprotect(0x7f37e31f8000, 2097152, PROT_NONE) = 0
mmap(0x7f37e33f8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DEN
YWRITE, 3, 0x1a1000) = 0x7f37e33f8000
mmap(0x7f37e33fe000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANO
NYMOUS, -1, 0) = 0x7f37e33fe000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f
37e3617000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f
37e3616000
```

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f
37e3615000
arch_prctl(ARCH_SET_FS, 0x7f37e3616700) = 0
mprotect(0x7f37e33f8000, 16384, PROT_READ) = 0
mprotect(0x7f37e3622000, 4096, PROT_READ) = 0
munmap(0x7f37e3618000, 32730)           = 0
open("fconc.out", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY)                     = 4
read(4, "Goodbye,\n", 1023)             = 9
write(3, "Goodbye,\n", 9)               = 9
read(4, "", 1023)                       = 0
open("B", O_RDONLY)                     = 5
read(5, "and thanks for all the fish!\n", 1023) = 29
write(3, "and thanks for all the fish!\n", 29) = 29
read(5, "", 1023)                       = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```