Λειτουργικά Συστήματα, 6ο Εξάμηνο, 2022-2023

Δημουλάς Ιωάννης, 03120083,
Ματζόρι Ενρίκα Ηλιάνα, 03120143
oslab20

Άσκηση 1η

Άσκηση 1.1

Στην εν λόγω άσκηση, καλούμαστε να δημιουργήσουμε ένα εκτελέσιμο αρχείο *'zing'* το οποίο θα καλεί την συνάρτηση *'zing()'* δηλωθείσα στο αρχείο *'zing.h'*.
Αφού, λοιπόν, αντιγράψουμε με την βοήθεια της εντολής *cp* τα αρχεία *'zing.h'* και *'zing.o'* στον κατάλογο εργασίας μας, προχωρούμε στην καταγραφή του κώδικα για την *'main.c'*:

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

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

Εφόσον έχουμε γράψει τον πηγαίο κώδικα, μεταγλωττίζουμε το αρχείο *'main.c'* μέσω της εντολής 'gcc -Wall -c main.c' (το αρχείο *'zing.o'* ήδη υπάρχει οπότε δεν υπάρχει λόγος μεταγλώττισης τυχόντος αρχείο *'zing.c'*). Κατόπιν εφαρμόζουμε την διαδικασία linking των δύο object files μέσω της εντολής 'gcc main.o zing.o -o zing'. Τέλος γράφουμε στον terminal την εντολή './zing' για την εμφάνιση της εξόδου της εκτέλεσης του προγράμματος. Η παραπάνω διαδικασία διακρίνεται στην παρακάτω εικόνα:

```
oslab20@orion:~$ gcc -Wall -c main.c
oslab20@orion:~$ gcc main.o zing.o -o zing
oslab20@orion:~$ ./zing
Hello, oslab20
```

1. *Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;*

   Η χρήση της εντολής '#include "zing.h"' είναι απαραίτητη όταν καλούμε μια συνάρτηση (στην προκειμένη περίπτωση την *'zing()'*), η οποία είναι ορισμένη σε διαφορετικό .c αρχείο απ' αυτό στο οποίο εργαζόμαστε. Καθ' αυτόν τον τρόπο έχουμε δήλωση της εκάστοτε συνάρτησης πριν την κλήση της. Γενικά η χρήση header files, που, ουσιαστικά, μοιράζει τον κώδικα σε κομμάτια, είναι πολύ σημαντική για μεγάλα προγράμματα, για λόγους που θα γίνουν περισσότερο εμφανείς στην ερώτηση 4.

2. *Ζητείται κατάλληλο Makefile για τη δημιουργία του εκτελέσιμου της άσκησης.*

   Το Makefile για το εκτελέσιμο αρχείο *'zing'* φαίνεται στην παρακάτω εικόνα:

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

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

Δημιουργούμε καινούριο αρχείο *'zing2.c'* στο οποίο ορίζουμε νέα συνάρτηση *'zing'* η οποία θα εκτυπώνει "Welcome oslab20", όπου το oslab20 είναι το username της ομάδας μας και το οποίο θα λάβουμε μέσα από την συνάρτηση getlogin(). Ακολουθεί ο πηγαίος κώδικας της *'zing2'*:

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

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

Κατόπιν οφείλουμε να μεταγλωττίσουμε τον κώδικα γράφοντας στο terminal την εντολή 'gcc -Wall -c zing2.c' παράγοντας, έτσι, το αρχείο *'zing2.o'*. Το αρχείο *'main.o'* υπάρχει ήδη από πριν. Κατόπιν, προχωράμε στην σύνδεση των δύο αρχείων με της εξής εντολή 'gcc main.o zing2.o -o zing2'. Δαχτυλογραφώντας, εν συνεχεία, την εντολή './zing2' φανίζεται η ζητούμενη έξοδος. Η παραπάνω διαδικασία διακρίνεται στην παραπάνω εικόνα:

```
oslab20@orion:~$ gcc -Wall -c zing2.c
oslab20@orion:~$ gcc main.o zing2.o -o zing2
oslab20@orion:~$ ./zing2
Welcome oslab20
```

Στην συνέχεια συνθέτουμε εκ νέου το Makefile για τα εκτελέσιμα αρχεία *'zing'* και *'zing2'*. Το Makefile, τώρα παίρνει την εξής μορφή:

```
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

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

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

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;*

   Ένα πρόγραμμα στο οποίο είναι ορισμένο μεγάλο πλήθος συναρτήσεων είναι προφανές ότι θα χρειάζεται και μεγαλύτερο χρόνο μεταγλώττισης. Για να αντιμετωπίσουμε αυτό το πρόβλημα θα μπορούσαμε να ορίσουμε τις διάφορες συναρτήσεις σε διαφορετικά .c files και να αντικαταστήσουμε τις δηλώσεις τους με τα κατάλληλα prepocessor directives (#include "header.h"). Έτσι, κάθε φορά που θέλουμε να κάνουμε κάποια αλλαγή σε κάποια απ' αυτές τις συναρτήσεις, θα χρειάζεται να μεταγλωττίσουμε μόνο το .c file στο οποίο έχει υλοποιηθεί το σώμα της συνάρτησης. Επίσης, χρήσιμος είναι και ο ορισμός των κατάλληλων Makefile έτσι ώστε με μία εντολή να προκύπτουν όλες οι απαραίτητες μεταγλωττίσεις και συνδέσεις αρχείων μειώνοντας δραστικά τον χρόνο εργασίας.

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 συνέβει 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;*

   Με την χρήση της παραπάνω εντολής ο συνεργάτης μας ήθελε να μεταγλωτίσει το αρχείο foo.c (2ο όρισμα), το οποίο θα παρήγαγε το foo.o. Ωστόσο, με το πρώτο όρισμα της εντολής, δηλώνουμε το όνομα που θα θέλαμε αυτό το object file να έχει. Συνεπώς, εδώ πρακτικά αντικαθιστούμε το αρχείο κώδικα foo.c με το object file του, σε γλώσσα μηχανής, χάνοντας έτσι τον πηγαίο κώδικα για πάντα!

Άσκηση 1.2

Παρατίθεται ο πηγαίος κώδικας της άσκησης:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
//writes up to len bytes from the buffer starting at buff
//to the file referred to by the file descriptor fd.
void doWrite (int fd, const char *buff, int len) {
    size_t idx = 0; //buffer index
    ssize_t wcnt; //number of bytes written
    do {
        wcnt = write (fd, buff+idx, len-idx);
        if (wcnt == -1) {
            perror ("write");
            close (fd);
            exit(1);
        } idx += wcnt;
    } while (idx < len);
}
//writes the infile's content into a buffer
//and then it calls the doWrite function in order to transfer the buffer's content
//into the file referred to by the file descriptor fd.
void write_file (int fd, const char *infile) {
    int fd_infile = open (infile, O_RDONLY);  //file descriptor of the invoked file
    char buff[1024];  //initialization of the buffer at 1kB
    ssize_t rcnt; //number of bytes read
    for (;;) {
```

```
        rcnt = read (fd_infile, buff, sizeof(buff)-1);
        if (rcnt == 0) return; //rcnt=0 means end-of-file
        if (rcnt == -1) {
            perror ("read");
            exit(1);
        }
        buff[rcnt] = '\0';
        doWrite (fd, buff, rcnt);//writes from the buffer to the file referred by fd
    } close (fd_infile);
}

int main (int argc, char **argv) {
    if (argc !=  3 && argc !=  4) {  //checks for the right amount of arguments
        printf ("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");
        exit(1);
    }
    int fd_A=open(argv[1],O_RDONLY);  //file descriptor of the first argument

    if(fd_A==-1){  //checks if the first argument exists
        perror(argv[1]);
        close(fd_A);
        exit(1);
    }
    int fd_B=open(argv[2],O_RDONLY);  //file descriptor of the second argument

    if(fd_B==-1){  //checks if the second argument exists
        perror(argv[2]);
        close(fd_B);
        exit(1);
    }
    //now that we have checked that the 2 files exist
    //we write them together to a third file
    int oflags, mode;
    oflags = O_CREAT | O_WRONLY | O_TRUNC;
    mode = S_IRUSR | S_IWUSR;
    int fd_C; //fd of the final file

    if(argc==3){
        fd_C= open("fconc.out", oflags, mode); //if the final file doesn't exist
    }
    else{
        fd_C= open(argv[3],oflags,mode);//fd of the given final file
    }
    if(fd_C==-1){
        perror("open");
        exit(1);
    }
    //we write the two given files to the final
    write_file(fd_C, argv[1]);
    write_file(fd_C, argv[2]);

    close(fd_A);
    close(fd_B);
    close(fd_C);  //close all file descriptors

    return 0;
}
```

Αφού μεταγλωττίσουμε τον κώδικα μέσω της εντολής ‘gcc -Wall fconc.c -o fconc’ τρέχουμε διάφορα παραδείγματα έτσι ώστε να ελέγξουμε ότι λειτουργεί καθώς πρέπει:

```
oslab20@orion:~$ gcc -Wall fconc.c -o fconc
oslab20@orion:~$ ls
a.out  fconc  fconc.c  hello  hello.c  hello.s  main.c  main.o  Makefile  mips.c  mips.s  test1.txt  zing  zing2  zing2.c  zing2.o  zing.h  zing.o
oslab20@orion:~$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]
oslab20@orion:~$ ./fconc A B
A: No such file or directory
oslab20@orion:~$ echo 'Good morning' > A
oslab20@orion:~$ echo 'to all the beautiful people!' > B
oslab20@orion:~$ ./fconc A B
oslab20@orion:~$ cat fconc.out
Good morning
to all the beautiful people!
oslab20@orion:~$ ./fconc A B C
oslab20@orion:~$ cat C
Good morning
to all the beautiful people!
```

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

   Στο terminal γράφουμε την εντολή 'strace cat fconc.out', όπου fconc.out είναι το προκύπτον αρχείο απ' την συνένωση των αρχείων A και B από την προηγούμενη εικόνα. Παρακάτω φαίνονται τα αποτελέσματα:

```
oslab20@orion:~$ strace cat fconc.out
execve("/bin/cat", ["cat", "fconc.out"], [/* 19 vars */]) = 0
brk(0)                                  = 0x1fa1000
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) = 0x7fb2f14ea000
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) = 0x7fb2f14e2000
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) = 0x7fb2f0f21000
mprotect(0x7fb2f10c2000, 2097152, PROT_NONE) = 0
mmap(0x7fb2f12c2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7fb2f12c2000
mmap(0x7fb2f12c8000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb2f12c8000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2f14e1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2f14e0000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2f14df000
arch_prctl(ARCH_SET_FS, 0x7fb2f14e0700) = 0
mprotect(0x7fb2f12c2000, 16384, PROT_READ) = 0
mprotect(0x60b000, 4096, PROT_READ)     = 0
mprotect(0x7fb2f14ec000, 4096, PROT_READ) = 0
munmap(0x7fb2f14e2000, 29766)           = 0
brk(0)                                  = 0x1fa1000
brk(0x1fc2000)                          = 0x1fc2000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=1607712, ...}) = 0
mmap(NULL, 1607712, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb2f1356000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 5), ...}) = 0
open("fconc.out", O_RDONLY)             = 3
fstat(3, {st_mode=S_IFREG|0600, st_size=42, ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2f1334000
read(3, "Good morning\nto all the beautifu"..., 131072) = 42
write(1, "Good morning\nto all the beautifu"..., 42Good morning
to all the beautiful people!
) = 42
read(3, "", 131072)                     = 0
munmap(0x7fb2f1334000, 139264)          = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Οι εντολές από το 'open ("fconc.out", O_RDONLY) =3' έως την γραμμή 'close(2) = 0' είναι τα system calls που καλούνται στο πρόγραμμά μας.