Λειτουργικά Συστήματα, 6^{ο} Εξάμηνο ΗΜΜΥ

Ακαδημαϊκή Περίοδος 2019-2020

Άσκηση 1: Εισαγωγή στο Περιβάλλον Προγραμματισμού

Ομάδα: oslabb33

Γιαννούλης Παναγιώτης (03117812)

Κανελλόπουλος Σωτήρης (03117101)

1.1. Για να αντιγράψουμε τα δύο αρχεία στον κατάλογο εργασίας μας χρησιμοποιήσαμε την εντολή:

cp /home/oslab/code/zing/ .*

Στη συνέχεια γράψαμε το πρόγραμμα main.c, που θα καλεί την zing(), όπως φαίνεται παρακάτω.

```c
#include "zing.h"

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

Με σκοπό τη μεταγλώττιση του παραπάνω προγράμματος και τη σύνδεσή του με το zing.o, φτιάξαμε το παρακάτω Makefile:

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

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

Χρησιμοποιώντας την εντολή *make*, είχαμε το παρακάτω αποτέλεσμα:

```
oslabb33@os-node1:~$ make
gcc -Wall -c main.c
gcc -o zing zing.o main.o
```

Τέλος, τρέξαμε το πρόγραμμά μας με την εντολή *./zing* και είχαμε την παρακάτω έξοδο:

```
oslabb33@os-node1:~$ ./zing
Hello, oslabb33
```

Απαντήσεις στις ερωτήσεις:

1. Η επικεφαλίδα μάς δίνει τη δυνατότητα να κάνουμε compile τη συνάρτηση zing() πριν γράψουμε την main. Πράγματι, η εκφώνηση της άσκησης μας δίνει εξ αρχής το αρχείο zing.o (μεταγλωττισμένο), το οποίο χρησιμοποιούμε χωρίς να ασχοληθούμε με τον πηγαίο κώδικά του. Αυτό θα μπορούσε να είναι χρήσιμο στην περίπτωση που η συνάρτηση zing() ήταν περίπλοκη, ώστε να μη χρειάζεται να την μεταγλωττίζουμε κάθε φορά που μεταγλωττίζουμε είτε την main μας είτε κάποιο άλλο πρόγραμμα που την χρησιμοποιεί και επομένως θα γλιτώναμε χρόνο.

2. Το ζητούμενο Makefile είναι το παρακάτω, όπως είδαμε και πρωτύτερα:

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

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

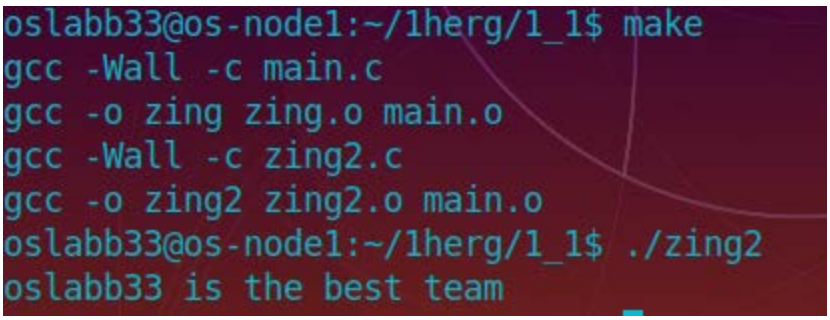
3. Γράψαμε το παρακάτω πρόγραμμα (zing2.c):

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

void zing ()
{
    char *name;
    name = getlogin();
    printf("%s is the best team\n", name);
}
```

Τροποποιήσαμε το Makefile ως εξής:

Παρακάτω φαίνεται το αποτέλεσμα της εντολής *make*, καθώς και η έξοδος του προγράμματός μας:



4. Θα μεταγλωττίσουμε τις 499 συναρτήσεις τις οποίες δεν μεταβάλλουμε, σε κοινό αρχείο, έστω το zing.o. Στη συνέχεια θα φτιάξουμε ένα header file, έστω zing.h, που θα περιέχει τις επικεφαλίδες όλων αυτών των συναρτήσεων. Η τελευταία συνάρτηση (αυτή που μεταβάλλουμε) θα βρίσκεται σε ένα αρχείο, έστω file.c, στο οποίο θα κάνουμε include το zing.h. Έτσι, κάθε φορά θα κάνουμε compile μόνο το file.c (δηλαδή τη μία συνάρτηση) και link το zing.o με το file.o (και μετά θα εκτελούμε το πρόγραμμα που προκύπτει).

5. Το λάθος είναι ότι το μεταγλωττισμένο πρόγραμμα δεν αποθηκεύεται σε object file (π.χ. foo.o), αλλά στο ίδιο το foo.c, με αποτέλεσμα να διαγράφεται ο κώδικας του προγράμματος. Για την αποφυγή τέτοιων λαθών χρησιμοποιούμε Makefile.

1.2. Παρακάτω φαίνεται ο κώδικάς μας:

```c
#include <sys/types.h>
#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){
  size_t idx;
  ssize_t wcnt;
  idx = 0;
  do {
    wcnt = write(fd, buff + idx, len - idx);
    if (wcnt == -1) {
       perror("write");
       exit(1);
    }
    idx += wcnt;
  } while(idx < len);
}

void write_file(int fd, const char *infile){
  char buff[1024];
  ssize_t rcnt;
  int fdin = open(infile, O_RDONLY);
  if (fdin == -1){
    perror(infile);
    exit(1);
  }
  while ((rcnt = read(fdin, buff, sizeof(buff) - 1)) > 0) {
        if (rcnt == -1) { /* error */
            perror("read");
            exit(1);
        }
        buff[rcnt] = '\0';
        doWrite(fd, buff, strlen(buff));
  }
  close(fdin);
}
```

```c
int main(int argc, char **argv)
{
  int fd, oflags = O_CREAT | O_WRONLY | O_TRUNC, mode = S_IRUSR | S_IWUSR;
  if (argc < 3 || argc > 4) {
    fprintf(stderr, "Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");
    exit(1);
  }
  if (argc == 3) {
      fd = open("fconc.out", oflags, mode);
      if (fd == -1) {
          perror("fconc.out");
          exit(1);
      }
  }
  else {
    fd = open(argv[3], oflags, mode);
    if (fd  == -1) {
        perror(argv[3]);
        exit(1);
    }
  }
  if (argv[1] == argv[3] || argv[2] == argv[3]) {
        perror("You shouldn't write where you read\n");
        exit(1);
  }
  write_file(fd, argv[1]);
  write_file(fd, argv[2]);
  close(fd);

  return 0;
}
```

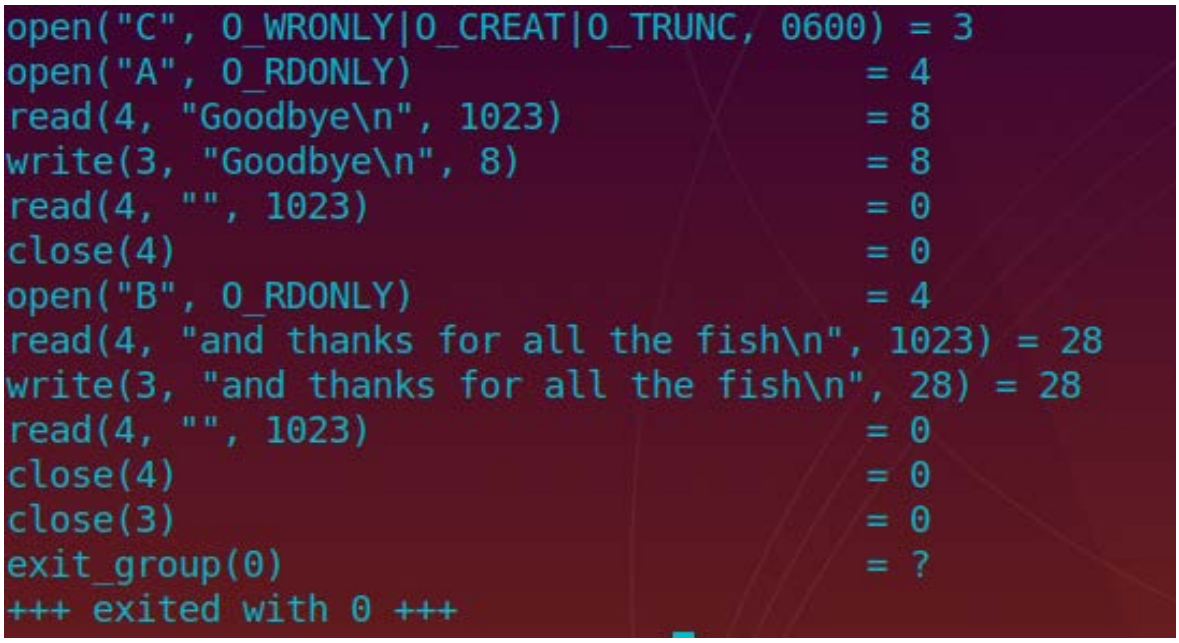
Απαντήσεις στις ερωτήσεις:

1. Εκτελούμε την εντολή:

strace ./fconf A B C

και προκύπτει η παρακάτω έξοδος:

```
oslabb33@os-node1:~/1herg/1_2$ strace ./fconc A B C
execve("./fconc", ["./fconc", "A", "B", "C"], [/* 27 vars */]) = 0
```



Στην αρχή βλέπουμε την κλήση του executable αρχείου μας (fconc) το οποίο θέλουμε να τρέξουμε. Στη συνέχεια υπάρχουν κάποιες κλήσεις που κάνει το ΛΣ οι οποίες δεν σχετίζονται με τον κώδικα που έχουμε εμείς γράψει. Τέλος στη δεύτερη εικόνα βλέπουμε το άνοιγμα του αρχείου C με την επιλογή να δημιουργηθεί αν δεν υπάρχει ήδη. Σε αυτό δίνεται ο file descriptor 3. Έπειτα ανοίγει το αρχείο A (fd = 4) και καλείται η read. Παρατηρούμε ότι διαβάζεται το Goodbye\n και η read επιστρέφει την τιμή 8 που είναι ίση με τα bytes που διαβάστηκαν. Μετά γίνονται write στο C τα 8 αυτά bytes. Τέλος κλείνει το αρχείο αυτό και επαναλαμβάνεται η διαδικασία για το B.

<u>Προαιρετικές ερωτήσεις:</u>

1.

Όπως αναμενόταν η εντολή strace καλείται μέσω της execve με 3 παραμέτρους.

Η πρώτη είναι το pathname του executable (“/usr/bin/strace”) και το δεύτερο είναι το argument που χρησιμοποιούμε για την κλήση της strace.

Παρατίθεται και η σύνοψη του man page της execve().

```
EXECVE(2)                  Linux Programmer's Manual                  EXECVE(2)

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *pathname, char *const argv[],
                  char *const envp[]);
```

2. Η αλλαγή αυτή οφείλεται στον linker (και άρα στο ΛΣ) αφού πριν την ένωση η main.o καλεί την συνάρτηση με symbol zing() χωρίς να υπάρχει στο πρόγραμμά της. Γι’ αυτό το λόγο στην πρώτη περίπτωση δεν γνωρίζει ο επεξεργαστής που να μεταπηδήσει όταν κάνει call αυτή τη συνάρτηση. Με το linking των δύο αρχείων ο επεξεργαστής αναγνωρίζει πια την συνάρτηση zing και γνωρίζει τουλάχιστον το offset στο οποίο θα μεταφερθεί. Ο loader φορτώνει το πρόγραμμα στη μνήμη οπότε οι απόλυτες διευθύνσεις (offsets) αλλάζουν και αυτές.

3.

```c
#include <sys/types.h>
#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) {
  size_t idx;
  ssize_t wcnt;
  idx = 0;
  do {
    wcnt = write(fd, buff + idx, len - idx);
    if (wcnt == -1) {
       perror("write");
       exit(1);
    }
    idx += wcnt;
  } while(idx < len);
}

void write_file(int fd, const char *infile){
  char buff[1024];
  ssize_t rcnt;
  int fdin = open(infile, O_RDONLY);
  if (fdin == -1){
    perror(infile);
    exit(1);
  }
  while ((rcnt = read(fdin, buff, sizeof(buff) - 1)) > 0) {
        if (rcnt == -1) { /* error */
            perror("read");
            exit(1);
        }
        buff[rcnt] = '\0';
        doWrite(fd, buff, strlen(buff));
  }
  close(fdin);
}
```

```c
int main(int argc, char **argv)
{
  int fd, oflags = O_CREAT | O_WRONLY | O_TRUNC, mode = S_IRUSR | S_IWUSR;
  if (argc < 3) {
    fprintf(stderr, "Usage: ./fconc infile1 ... infile(n) [outfile (default:fconc.out)]\n");
    exit(1);
  }
  fd = open(argv[argc-1], oflags, mode);
  if (fd  == -1) {
      perror(argv[argc-1]);
      exit(1);
  }
  int i;
  for (i = 1; i < argc-1; i++) {
      if (argv[i] == argv[argc-1]) {
        fprintf(stderr, "[-] Please believe me you don't want to write to file you read from (: %s)\n", argv[argc-1]);
        exit(1);
        }
  }
  for (i = 1; i < argc - 1; i++) {
      write_file(fd, argv[i]);
  }
  close(fd);
  return 0;
}
```

4. Με την εντολή disas main του gdb βλέπουμε τον assembly κώδικα του executable whoops. Για αρχή

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048470 <+0>:     lea    ecx,[esp+0x4]
   0x08048474 <+4>:     and    esp,0xfffffff0
   0x08048477 <+7>:     push   DWORD PTR [ecx-0x4]
   0x0804847a <+10>:    push   ebp
   0x0804847b <+11>:    mov    ebp,esp
   0x0804847d <+13>:    push   ecx
   0x0804847e <+14>:    sub    esp,0x14
   0x08048481 <+17>:    mov    DWORD PTR [esp+0x4],0x0
   0x08048489 <+25>:    mov    DWORD PTR [esp],0x80485b0
   0x08048490 <+32>:    call   0x8048348 <open@plt>
   0x08048495 <+37>:    add    eax,0x1
   0x08048498 <+40>:    je     0x80484b1 <main+65>
   0x0804849a <+42>:    mov    DWORD PTR [esp],0x80485c8
   0x080484a1 <+49>:    call   0x8048388 <puts@plt>
   0x080484a6 <+54>:    add    esp,0x14
   0x080484a9 <+57>:    xor    eax,eax
   0x080484ab <+59>:    pop    ecx
   0x080484ac <+60>:    pop    ebp
   0x080484ad <+61>:    lea    esp,[ecx-0x4]
   0x080484b0 <+64>:    ret
   0x080484b1 <+65>:    mov    eax,ds:0x8049704
   0x080484b6 <+70>:    mov    DWORD PTR [esp+0x8],0x9
   0x080484be <+78>:    mov    DWORD PTR [esp+0x4],0x1
   0x080484c6 <+86>:    mov    DWORD PTR [esp],0x80485bc
   0x080484cd <+93>:    mov    DWORD PTR [esp+0xc],eax
   0x080484d1 <+97>:    call   0x8048378 <fwrite@plt>
   0x080484d6 <+102>:   mov    DWORD PTR [esp],0x1
   0x080484dd <+109>:   call   0x8048398 <exit@plt>
End of assembler dump.
(gdb) x/s 0x80485b0
0x80485b0:       "/etc/shadow"
(gdb) x/s 0x8049704
0x8049704 <stderr@@GLIBC_2.0>:  ""
(gdb) x/s 0x80485bc
0x80485bc:       "Problem!\n"
(gdb) x/s 0x80485c8
0x80485c8:       "You are not supposed to see this!"
(gdb)
```

κοιτάμε τις κλήσεις που γίνονται. Γρήγορα παρατηρούμε μια κλήση στην fopen με παραμέτρους μια διεύθυνση (μάλλον τη διεύθυνση του ονόματος του αρχείου που θέλουμε να ανοίξουμε και κάποιο mode (0)). Με την εντολή x/s βλέπουμε κάποια strings που υπάρχουν σε διάφορες διευθύνσεις που εμφανίζονται στον κώδικα. Π.χ. η 0x80485b0 που περνιέται ως παράμετρος στην fopen μας μαρτυρά αμέσως ότι θέλουμε να ανοίξουμε το αρχείο "*/etc/shadow"* . Αμέσως καταλαβαίνουμε ότι κάτι τέτοιο είναι απίθανο αφού θα θέλαμε root privileges. Και όντως στην παρακάτω εικόνα κάνοντας break μια εντολή παρακάτω από την fopen βλέπουμε ότι γυρνάει -1(στον register eax). Που σημαίνει ότι απέτυχε το άνοιγμα του αρχείου. Μετά βλέπουμε ένα if στην περίπτωση τέτοιας αποτυχίας που μεταπηδά το πρόγραμμά μας στην main+65. Εκεί βλέπουμε ότι καλείται η fwrite που γράφει στο stderr το string "Problem\n" το οποίο και βλέπουμε στην οθόνη μας. Τέλος καλείται η exit(1). Αν είχαμε τα δικαιώματα για το άνοιγμα του αρχείου θα καλούταν η puts("You are not supposed to see this!").