Λειτουργικά Συστήματα Υπολογιστών

Κακούρης Δημήτριος (03119019)
Μαρδίκης Κωνσταντίνος (03119867)

1η Εργαστηριακή Άσκηση

1.1 Σύνδεση με αρχείο αντικειμένων

Αρχικά αντιγράφουμε από το directory */home/oslab/code/zing* τα αρχεία zing.h και zing.o:

Με την εντολή *cp zing.h /home/oslab/oslab122* και cp *zing.o/home/oslab/oslab122* αντίστοιχα.

Χωρίς το makefile που ζητειται θα έπρεπε να κανουμε μια-μια σε terminal τις "μωβ" εντολές που απεικονίζονται στην εικόνα του makefile, που αντιστοιχούν αρχικά σε μεταγλώττιση των .c αρχειων και στη συνέχεια linking μεταξύ των object files .o ώστε να καταλήξουμε στα εκτελέσιμα zing και zing2.

- *Ερωτήσεις 1.1 (Η αρίθμηση δεν αντιστοιχεί στην αρίθμηση της εκφώνησης):*

1. Τα header files χρησιμοποιούνται απο τον preprocessor και βοηθούν στο να ορίζουμε συναρτήσεις και να τις καλούμε σε οποιοδήποτε .c αρχείο θέλουμε χωρίς να χρειάζεται να τις ορίζουμε σε αυτό ξανά. Βοηθούν στη μείωση του χρόνου μεταγλώττισης εφόσον δε χρειάζεται να γίνονται κάθε φορά compile μαζί με το υπόλοιπο πρόγραμμα και οι συναρτήσεις των header files, απλά τι καλούμε. Επίσης προσφέρουν και μια “προγραμματιστική ασφάλεια” εφόσον είμαστε σίγουροι πως εκθέτουμε στο κώδικα του .c αρχείο μόνο τις απολύτως απαραίτητες παραμέτρους και δε μπορούν να γίνουν προσβάσιμες από λάθος μη επιθυμητές.

2. Δημιουργούμε Makefile ωστε να παραχθούν τα zing και zing2 executable files. Τα ορίζουμε πριν την άνω-κάτω τελεία και στην ίδια σειρά

ακολουθούν τα dependencies ενώ από κάτω με μώβ γραμματοσειρά βλέπουμε τις εντολές μεταγλώττισης και linking μέσω gcc.

```
all: zing zing2

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

zing2: main.o zing2.o
	gcc -o zing2 main.o zing2.o

zing2.o:
	gcc -c zing2.c

main.o: main.c
	gcc -c main.c
clean:
	rm -f main.o zing2.o zing zing2
```

3. Ο κώδικας του zing2

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

void zing(void){
	printf("Auth einai h: %s\n",getlogin());
}
```

Έξοδος της zing:

```
oslab122@orion:~/ex1/1.1/ex1.1_dimitris$ ./zing
Hello, oslab122
oslab122@orion:~/ex1/1.1/ex1.1_dimitris$
```

Εξοδος της zing2:

```
oslab122@orion:~/ex1/1.1/ex1.1_dimitris$ ./zing2
Auth einai h: oslab122
```

4. Ο λόγος που αργεί τόσο η μεταγλώττιση είναι (όπως αναφέρθηκε και πιο πάνω) ότι πρέπει κάθε φορά που γίνονται αλλαγές έστω και σε μια συνάρτηση να μεταγλωτίζονται και όλες οι υπόλοιπες συναρτήσεις μαζί. Για την επιτάχυνση της μεταγλώττισης πρέπουσα θα ήταν η χρήση πολλών διαφορετικών αρχείων .c, ώστε κάθε φορά που θα τροποποιούμε μια συνάντηση να μεταγλωτίζουμε μόνο ένα αρχείο.

5. Με αυτήν την εντολή ουσιαστικά αποθηκεύουμε το εκτελέσιμο αρχείο (γλώσσας μηχανής) σε ένα αρχείο .c. Επομένως, το παραπάνω αρχείο το καθιστά μη αναγνώσιμο.

1.2 Συνένωση δύο αρχείων σε τρίτο

Το πρόγραμμα fconc:

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

void doWrite(int fd_final,char *buff,int len) {

        size_t idx=0; //index in buffer
        ssize_t wcnt;
        do {
        wcnt = write(fd_final,buff + idx, len -idx);

        if (wcnt == - 1){ /* error */
                perror("write");
                close(fd_final);
                exit(1);
        }
        idx += wcnt;
        } while (idx < len);
```

```
}
void write_file(const char *infile_in,int fd_final) {

	char buff[1024];



	int fd = open(infile_in, O_RDONLY);

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

	ssize_t rcnt;
	for (;;) {
	rcnt = read(fd, buff, sizeof(buff));
	/*return the bytes read as an integer, we asked it for sizeof(buff) bytes but it
will not always read this size*/

	if (rcnt == 0){/* end-of-file */
		break;
	}

	if (rcnt == -1) { /* error */
		perror("read");
		exit(1);



	}
	buff[rcnt] = '\0';



	doWrite(fd_final,buff,rcnt);

	}
	close(fd);

}

int main(int argc, char **argv) {
```

```
	if(argc<=2||argc>4){
	printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");return 1;}
	else {
	char *file1 = argv[1];
	char *file2 = argv[2];
	char *file3 = argv[3];

	int fd_final;
	if (argc == 3) {fd_final = open("fconc.out", O_WRONLY | O_CREAT, S_IRUSR |
S_IWUSR);}
	else{fd_final = open(file3, O_WRONLY | O_CREAT,S_IRUSR | S_IWUSR );}

	if(fd_final==-1){
		perror("Destination file error\n");
		exit(1);}

	if(open(argv[1], O_RDONLY)== -1){
		printf("A: ");
		perror("No such file or directory\n");
		exit(1);
	}
	else if(open(argv[2], O_RDONLY) ==-1){
		printf("B: ");
		perror("No such file or directory\n");

		exit(1);
	}


	write_file(file1,fd_final);
	write_file(file2,fd_final);
	}

return 0;
}
```

Ενδεικτικά η έξοδος της συνάρτησης εάν A='I am' και B=', You are':

```
oslab122@orion:~/ex1/1.2/ex1.2_dimitris$ ./fconc A B C
oslab122@orion:~/ex1/1.2/ex1.2_dimitris$ cat C
I am
, you are
```

- Ερωτήσεις 1.2 :

1. strace ./fconc A B

```
oslab122@orion:~/ex1/1.2/ex1.2_dimitris$ strace ./fconc A B
execve("./fconc", ["./fconc", "A", "B"], [/* 29 vars */]) = 0
brk(0)                                  = 0x1123000
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) = 0x7f4d8bf30000
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) = 0x7f4d8bf28000
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) = 0x7f4d8b967000
mprotect(0x7f4d8bb08000, 2097152, PROT_NONE) = 0
mmap(0x7f4d8bd08000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) =
 0x7f4d8bd08000
mmap(0x7f4d8bd0e000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f4
d8bd0e000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4d8bf27000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4d8bf26000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4d8bf25000
arch_prctl(ARCH_SET_FS, 0x7f4d8bf26700) = 0
mprotect(0x7f4d8bd08000, 16384, PROT_READ) = 0
mprotect(0x7f4d8bf32000, 4096, PROT_READ) = 0
munmap(0x7f4d8bf28000, 29766)           = 0
```

Στη παρακάτω συνέχεια του μηνύματος της strace παρατηρούμε τις κλήσεις της fconc σε open(),read(),write() με τα μηνύματα του A και B να εμφανίζονται στο terminal. Παρατηρούμε πως πρώτα ανοίγει και τα δύο αρχεία και μετά για καθένα από αυτά εκτελεί read(), write().

```
open("fconc.out", O_WRONLY|O_CREAT, 0600) = 3
open("A", O_RDONLY)                     = 4
open("B", O_RDONLY)                     = 5
open("A", O_RDONLY)                     = 6
read(6, "I am\n", 1024)                 = 5
write(3, "I am\n", 5)                   = 5
read(6, "", 1024)                       = 0
close(6)                                = 0
open("B", O_RDONLY)                     = 6
read(6, ", you are\n", 1024)            = 10
write(3, ", you are\n", 10)            = 10
read(6, "", 1024)                       = 0
close(6)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
oslab122@orion:~/ex1/1.2/ex1.2_dimitris$
```

Έξτρα υποερωτήματα:

1. Με την εντολή strace -c strace pwd τυπώνουμε τις κλήσεις που κάνει η strace pwd και μια συνολική λίστα των system calls που πραγματοποιήθηκαν. Παρατηρούμε πως καλείται περισσοτερο η ptrace με 226 κλήσεις.

```
+++ exited with 0 +++
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.000025           0       106           write
  0.00    0.000000           0         1           read
  0.00    0.000000           0         2           open
  0.00    0.000000           0         2           close
  0.00    0.000000           0         4         2 stat
  0.00    0.000000           0         2           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         8           rt_sigaction
  0.00    0.000000           0       224           rt_sigprocmask
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           getpid
  0.00    0.000000           0         3           clone
  0.00    0.000000           0         1           execve
  0.00    0.000000           0       116         1 wait4
  0.00    0.000000           0         2           kill
  0.00    0.000000           0         1           uname
  0.00    0.000000           0       226           ptrace
  0.00    0.000000           0         1           getuid
  0.00    0.000000           0         1           getgid
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0        56           process_vm_readv
------ ----------- ----------- --------- --------- ----------------
100.00    0.000025                   777         6 total
oslab122@orion:~$
```

Παρατηρούμε η strace υλοποιείται από την ptrace όπως φαίνεται από το https://man7.org/linux/man-pages/man2/ptrace.2.html

> Usually, the tracer (for example, strace(1))

Το system call ptrace ουσιαστικά συμπεριφέρεται σαν παρατηρητής διαδικασιών μιας εντολής.

2.

Καλούμε τον debugger τόσο για main.o όσο και για zing.

gbd -q main.o

```
oslab122@os-node2:~/ex1/1.1/ex1.1_dimitris$ gdb -q main.o
Reading symbols from main.o...
(No debugging symbols found in main.o)
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000000000 <+0>:     push   %rbp
   0x0000000000000001 <+1>:     mov    %rsp,%rbp
   0x0000000000000004 <+4>:     sub    $0x10,%rsp
   0x0000000000000008 <+8>:     mov    %edi,-0x4(%rbp)
   0x000000000000000b <+11>:    mov    %rsi,-0x10(%rbp)
   0x000000000000000f <+15>:    call   0x14 <main+20>
   0x0000000000000014 <+20>:    mov    $0x0,%eax
   0x0000000000000019 <+25>:    leave
   0x000000000000001a <+26>:    ret
End of assembler dump.
```

gbd -q zing

```
oslab122@os-node2:~/ex1/1.1/ex1.1_dimitris$ gdb -q zing
Reading symbols from zing...
(No debugging symbols found in zing)
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000400596 <+0>:     push   %rbp
   0x0000000000400597 <+1>:     mov    %rsp,%rbp
   0x000000000040059a <+4>:     sub    $0x10,%rsp
   0x000000000040059e <+8>:     mov    %edi,-0x4(%rbp)
   0x00000000004005a1 <+11>:    mov    %rsi,-0x10(%rbp)
   0x00000000004005a5 <+15>:    call   0x4005b1 <zing>
   0x00000000004005aa <+20>:    mov    $0x0,%eax
   0x00000000004005af <+25>:    leave
   0x00000000004005b0 <+26>:    ret
End of assembler dump.
(gdb)
```

Παρατηρούμε πως η διαφορά είναι στο στάδιο call, στο gdb -q main.o καλείται η διεύθυνση 20 bytes “μέσα” στη main(offset δηλαδή από τη διεύθυνση 0x14) ενώ στη zing καλείται απευθείας η υλοποίηση της zing.

3. *Γενίκευση της fconc*

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

void doWrite(int fd_final,char *buff,int len) {

   size_t idx=0; /*index in buffer*/
   ssize_t wcnt;
   do {
         wcnt = write(fd_final,buff + idx, len -idx);

         if (wcnt == - 1){ /* error */
               perror("write");
               close(fd_final);
               exit(1);
         }
         idx += wcnt;
   } while (idx < len);

}

void write_file(const char *infile_in,int fd_final) {
   char buff[1024];



   int fd = open(infile_in, O_RDONLY);

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

   ssize_t rcnt;
   for (;;) {
         rcnt = read(fd, buff, sizeof(buff));
```

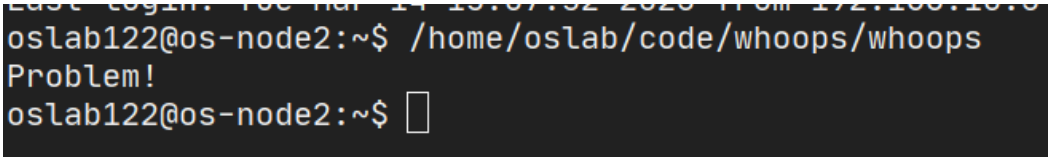
```
        if (rcnt == 0){/* end-of-file */
                break;
        }

        if (rcnt == -1) { /* error */
                perror("read");
                exit(1);


        }
        buff[rcnt] = '\0';


        doWrite(fd_final,buff,rcnt);

    }
    close(fd);



}

int main(int argc, char **argv) {


    int i=1;

    while(access(argv[i], F_OK) == 0){
        i++;
    }


    int fd_final;
    if (argv[i]==NULL) {

        fd_final = open("fconc.out", O_WRONLY | O_CREAT| O_TRUNC, S_IRUSR |
S_IWUSR);}

    else{
        char *file3 = argv[i];
```

```
        fd_final = open(file3, O_WRONLY | O_CREAT| O_TRUNC,S_IRUSR | S_IWUSR );}

    int j=1;
    for(j=1; j<i; j++){
        write_file(argv[j],fd_final);
    }



    return 0;
}
```

Ουσιαστικά στη γραμμή 72 τσεκάρουμε μέσω access(argv[i], F_OK), αν τα ορίσματα της *fconc_extra* που βάζουμε υπάρχουν όλα στο directory, αν π.χ έχουμε βάλει κάποιο αρχείο στο τέλος στο οποίο θέλουμε να γράψουμε τα περιεχόμενα των άλλων μέσα και δεν υπάρχει στο directory τότε μπαίνουμε στο *else* και γράφουμε στο όνομα αυτο τα περιεχόμενα, αν δεν έχουμε ορίσει αρχείο προορισμού τοτε το *argv[i]==NULL* και δημιουργούμε νέο *fconc.out* .

4.

Τρέχοντας την εντολή /home/oslab/code/whoops/whoops εκτελείται το εκτελέσιμο αρχείο whoops και τυπώνει “Problem!” το οποίο παρατηρούμε μέσω της εντολής STRACE ότι προκύπτει από κλήση της write() η οποία καλέσθηκε όταν η open() με όρισμα το /etc/shadow επέστρεψε αρνητικό ακέραιο -1 (EACCES Permission Denied).

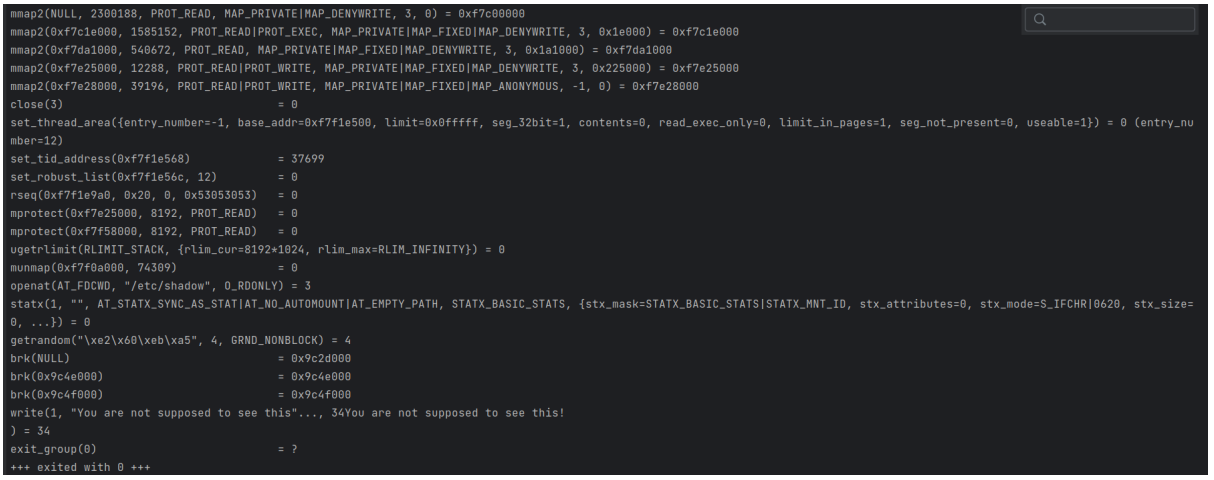


Καλώντας την strace για whoops:

```
oslab122@os-node2:~$ strace /home/oslab/code/whoops/whoops
execve("/home/oslab/code/whoops/whoops", ["/home/oslab/code/whoops/whoops"], 0x7ffe2b87d800 /* 30 v
ars */) = 0
[ Process PID=4439 runs in 32 bit mode. ]
brk(NULL)                               = 0x88b5000
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) = 0xf7f5f000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=19315, ...}) = 0
mmap2(NULL, 19315, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7f5a000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib32/libc.so.6", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360\257\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1991764, ...}) = 0
mmap2(NULL, 2000744, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xf7d71000
mprotect(0xf7d8a000, 1875968, PROT_NONE) = 0
mmap2(0xf7d8a000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19000) =
0xf7d8a000
mmap2(0xf7edf000, 475136, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16e000) = 0xf7edf000
mmap2(0xf7f54000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) =
0xf7f54000
mmap2(0xf7f57000, 10088, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xf7f5
7000
close(3)                                = 0
set_thread_area({entry_number=-1, base_addr=0xf7f600c0, limit=0x0fffff, seg_32bit=1, contents=0, re
ad_exec_only=0, limit_in_pages=1, seg_not_present=0, useable=1}) = 0 (entry_number=12)
mprotect(0xf7f54000, 8192, PROT_READ)   = 0
mprotect(0xf7f91000, 4096, PROT_READ)   = 0
munmap(0xf7f5a000, 19315)               = 0
openat(AT_FDCWD, "/etc/shadow", O_RDONLY) = -1 EACCES (Permission denied)
write(2, "Problem!\n", 9Problem!
)               = 9
exit_group(1)                           = ?
+++ exited with 1 +++
oslab122@os-node2:~$
```

Τρέχουμε locally το whoops έχοντας αντιγράψει το εκτελέσιμο απο το orion και παρατηρούμε οτι τυπώνει ένα διαφορετικό μήνυμα “You are not supposed to see this!”.
Ο λόγος που συμβαίνει αυτό είναι διότι τοπικά έχουμε root privileges και επομένως μπορεί να γίνει open() το /etc/shadow που βλέπουμε με strace στη παρακάτω εικόνα:

```
brewed@brewed-laptop:~/CLionProjects/ex1/whoops_program$ sudo strace ./whoops
execve("./whoops", ["./whoops"], 0x7ffdacdcf300 /* 25 vars */) = 0
[ Process PID=37699 runs in 32 bit mode. ]
brk(NULL)                               = 0x9c2d000
arch_prctl(0x3001 /* ARCH_??? */, 0xffaaa0b8) = -1 EINVAL (Invalid argument)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7f1d000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = 0
openat(AT_FDCWD, "/etc/ld.so.preload", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
statx(3, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASIC_STATS|STATX_MNT_ID, stx_attributes=0, stx_mode=S_IFREG|0644, stx_size=
1, ...}) = 0
mmap2(NULL, 1, PROT_READ|PROT_WRITE, MAP_PRIVATE, 3, 0) = 0xf7f1c000
close(3)                                = 0
munmap(0xf7f1c000, 1)                   = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
statx(3, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASIC_STATS|STATX_MNT_ID, stx_attributes=0, stx_mode=S_IFREG|0644, stx_size=
74309, ...}) = 0
mmap2(NULL, 74309, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7f0a000
close(3)                                = 0
```



```
mmap2(NULL, 2300188, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xf7c00000
mmap2(0xf7c1e000, 1585152, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e000) = 0xf7c1e000
mmap2(0xf7da1000, 540672, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0xf7da1000
mmap2(0xf7e25000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x225000) = 0xf7e25000
mmap2(0xf7e28000, 39196, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xf7e28000
close(3)                                = 0
set_thread_area({entry_number=-1, base_addr=0xf7f1e500, limit=0x0fffff, seg_32bit=1, contents=0, read_exec_only=0, limit_in_pages=1, seg_not_present=0, useable=1}) = 0 (entry_nu
mber=12)
set_tid_address(0xf7f1e568)             = 37699
set_robust_list(0xf7f1e56c, 12)         = 0
rseq(0xf7f1e9a0, 0x20, 0, 0x53053053)   = 0
mprotect(0xf7e25000, 8192, PROT_READ)   = 0
mprotect(0xf7f58000, 8192, PROT_READ)   = 0
ugetrlimit(RLIMIT_STACK, {rlim_cur=8192*1024, rlim_max=RLIM_INFINITY}) = 0
munmap(0xf7f0a000, 74309)               = 0
openat(AT_FDCWD, "/etc/shadow", O_RDONLY) = 3
statx(1, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASIC_STATS|STATX_MNT_ID, stx_attributes=0, stx_mode=S_IFCHR|0620, stx_size=
0, ...}) = 0
getrandom("\xe2\x60\xeb\xa5", 4, GRND_NONBLOCK) = 4
brk(NULL)                               = 0x9c2d000
brk(0x9c4e000)                          = 0x9c4e000
brk(0x9c4f000)                          = 0x9c4f000
write(1, "You are not supposed to see this"..., 34You are not supposed to see this!
) = 34
exit_group(0)                           = ?
+++ exited with 0 +++
```

Η τελευταία write() τυπώνει το κρυφό μήνυμα εφόσον το open(“/etc/shadow”) επέστρεψε με θετικό ακέραιο.

Σημείωση για etc/shadow:

The **/etc/shadow** is a text-based password file. The shadow file stores the hashed passphrase (or “hash”) format for Linux user account with additional properties related to the user password. This shadow file is directly accessible only to the root user. Howe