# Λειτουργικά Συστήματα Υπολογιστών


## Αναφορά - 1η Εργαστηριακή Άσκηση

**Ομάδα: oslab04**

Κατσιάνης Κωνσταντίνος - 03120183

Κουρής Γεώργιος - 03120116

**6ο Εξάμηνο - Μάρτιος 2023**

# Περιεχόμενα Αναφοράς:


# 1 Ασκήσεις

# 2 Εξέταση άσκησης και αναφορά

## 1.1 Σύνδεση με αρχείο αντικειμένων

Α) Ο πηγαίος κώδικας (source code) της άσκησης

Β) Η διαδικασία μεταγλώττισης και σύνδεσης

Γ) Η έξοδος εκτέλεσης του προγράμματος

Δ) Σύντομες απαντήσεις στις ερωτήσεις (1,2,3,4,5)

## 1.2 Συνένωση δύο αρχείων σε τρίτο

Α) Ο πηγαίος κώδικας (source code) της άσκησης

Β) Σύντομες απαντήσεις στις ερωτήσεις (1)

# 3 Προαιρετικές ερωτήσεις (1,2,3,4)

# 2 Εξέταση άσκησης και αναφορά

## 1.1 Σύνδεση με αρχείο αντικειμένων

Α) Ο πηγαίος κώδικας (source code) της άσκησης

### main.c:

```
#include "zing.h"

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

### zing2.c:

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

void zing(void)
{
	printf("Hello humans of %s\n", getlogin());
}
```

## Β) Η διαδικασία μεταγλώττισης και σύνδεσης

Αρχικά δημιουργήσαμε το object-file main.o, με την εντολή:

*gcc -Wall -c main.c*

Στη συνέχεια δημιουργήσαμε το αρχείο zing συνδέοντας τα object-file main.o και zing.o με την ακόλουθη εντολή:

*gcc -o zing zing.o main.o*

Ομοίως για το αρχείο zing2:

*gcc -o zing2 zing2.o main.o*

## Γ) Η έξοδος εκτέλεσης του προγράμματος

Για το zing και το zing2 η έξοδος είναι:

```
oslab04@os-node2:~/ex1/zing$ ./zing
Hello, oslab04
oslab04@os-node2:~/ex1/zing$ ./zing2
Hello humans of oslab04
```

## Δ) Σύντομες απαντήσεις στις ερωτήσεις (1,2,3,4,5)

1. Χρησιμοποιούμε επικεφαλίδες για να δηλώσουμε τις συναρτήσεις που θα χρησιμοποιήθουν χωρίς να χρειάζεται να το κάνουμε αυτό στον κώδικα. Έτσι, είναι πιο εύκολο να ξαναχρησιμοποιήσουμε τις συναρτήσεις σε διαφορετικό κώδικα. Επίσης, χωρίζοντας τη δήλωση συναρτήσεων από την υλοποίησή τους, ο κώδικας γίνεται πιο ευανάγνωστος και κατανοητός, ενώ διευκολύνεται και η οργάνωσή του.

2. Αρχικό **Makefile:**

   make_all: zing

   zing: zing.o main.o

   gcc -o zing zing.o main.o

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

delete_all:
	rm -f zing zing2 zing2.o main.o
```

**3.** Το zing2 το παρουσιάσαμε με τις παραπάνω εντολές. Το τελικό **Makefile** είναι:

```
make_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

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

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

delete_all:
	rm -f zing zing2 zing2.o main.o
```

4. Το πρόγραμμα του αρχείου μας περιέχει 500 συναρτήσεις. Αφού είναι όλες σε ένα αρχείο, ακόμα κι όταν διορθώνουμε μόνο μία από αυτές, στην μεταγλώττιση θα μεταγλωττιστούν όλες οι 500 συναρτήσεις, το οποίο θα καθυστερήσει σημαντικά την όλη διαδικασία. Μια εύκολη λύση είναι η κάθε συνάρτηση να έχει δικό της αρχείο (ή άμα δεν μπορούμε να έχουμε τόσα πολλά διαφορετικά αρχεία, κάθε αρχείο να περιέχει όσο το δυνατόν λιγότερες συναρτήσεις) έτσι ώστε να μεταγλωτίζεται μόνο η ζητούμενη συνάρτηση και όχι καμία έξτρα. Στην συνέχεια μπορούμε απλά να χρησιμοποιήσουμε header files για να το επιτύχουμε αυτό.

5. Η συγκεκριμένη εντολή δημιουργεί ένα εκτελέσιμο αρχείο με το όνομα foo.c, οπότε, καθώς το αρχείο υπάρχει ήδη, το προηγούμενο foo.c αντικαθιστάται από το εκτελέσιμο. Προφανώς αυτό δεν είναι επιθυμητό.

## 1.2 Συνένωση δύο αρχείων σε τρίτο

Α) Ο πηγαίος κώδικας (source code) της άσκησης

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

void doWrite(int fd, const char *buff, int len) {
//similar to write without the for loop
	size_t idx;
	ssize_t wcnt;
	idx = 0;
	do {
		wcnt = write(fd, buff + idx, len - idx);
		if (wcnt == -1){ /* error */
			perror("write");
			return 1;
		}
		idx += wcnt;
	} while (idx < len);
}

void write_file(int fd, const char *infile) {
//similar to open read and then read
	int fdopen;
	char buff[1024];
```

```
    ssize_t rcnt;
    fdopen = open(infile, O_RDONLY);
    if (fdopen == -1){
        perror(infile);
        exit(1);
    }
    // perform read(...)
    for (;;){
        rcnt = read(fdopen, buff, sizeof(buff)-1);
        if (rcnt == 0) /* end.of.file */
            return 0;
        if (rcnt == -1){ /* error */
            perror("read");
            return 1;
        }
          doWrite(fd, buff, rcnt);
    }
    close(fdopen);
}


int main(int argc, char **argv) {
    int fd, oflags, mode;
    oflags = O_CREAT | O_WRONLY | O_TRUNC;
    mode = S_IRUSR | S_IWUSR;
    if (argc != 3 && argc != 4) {
        printf("Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]\n");
        exit(1);
```

```
    }


    if (argc == 3) {
        fd = open("fconc.out", oflags, mode);
    }
    else {
       if ((strcmp(argv[3], argv[1]) == 0) || (strcmp(argv[3], argv[2]) == 0)) {
        printf("ERROR SAME NAME\n");
         exit(1);
        }
        fd = open(argv[3], oflags, mode);
    }
    if (fd == -1){
        perror("open");
        exit(1);
    }
    else {
        write_file(fd, argv[1]);
        write_file(fd, argv[2]);
        close(fd);
    }
    return 0;
}
```

Β) Σύντομες απαντήσεις στις ερωτήσεις (1)

```
oslab04@os-node2:~/ex1/fconc$ strace ./fconc A B
execve("./fconc", ["./fconc", "A", "B"], 0x7fff3b8f43b0 /* 22 vars */) = 0
brk(NULL)                               = 0x55ad66315000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=20405, ...}) = 0
mmap(NULL, 20405, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fadl4999000
close(3)                                = 0
openat(AT_FDCWD, "/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\3\0>\0\1\0\0\0@>\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1905632, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fadl4997000
mmap(NULL, 1918592, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fadl47c2000
mmap(0x7fadl47e4000, 1417216, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fadl47e4000
mmap(0x7fadl493e000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17c000) = 0x7fadl493e000
mmap(0x7fadl498d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ca000) = 0x7fadl498d000
mmap(0x7fadl4993000, 13952, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fadl4993000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fadl4998540) = 0
mprotect(0x7fadl498d000, 16384, PROT_READ) = 0
mprotect(0x55ad658f7000, 4096, PROT_READ) = 0
mprotect(0x7fadl49c8000, 4096, PROT_READ) = 0
munmap(0x7fadl4999000, 20405)           = 0
openat(AT_FDCWD, "fconc.out", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
openat(AT_FDCWD, "A", O_RDONLY)         = 4
read(4, "Goodbye,\n", 1023)             = 9
write(3, "Goodbye,\n", 9)               = 9
read(4, "", 1023)                       = 0
openat(AT_FDCWD, "B", O_RDONLY)         = 5
read(5, " humans\n", 1023)              = 8
write(3, " humans\n", 8)                = 8
read(5, "", 1023)                       = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

## 3 Προαιρετικές ερωτήσεις (1,2,3,4)

1. Χρησιμοποιήσαμε την εντολή strace -c strace pwd για να βρούμε ποια εντολή καλείται πιο συχνά. Αυτή η εντολή θα εκτελέσει την εντολή pwd και θα βρει τις κλήσεις του συστήματος κατά την εκτέλεση της pwd. Η έξοδος θα περιλαμβάνει τις κλήσεις συστήματος που γίνονται από το ίδιο το strace, οπότε βλέποντας ποια εντολή καλείται περισσότερες φορές, μπορούμε να εντοπίσουμε με ποια κλήση συστήματος υλοποιείται η εντολή strace.

```
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 46.32    0.018398          46       392         2 wait4
 25.75    0.010226          26       393           ptrace
 16.66    0.006617          35       187           write
  2.67    0.001061          25        41           process_vm_readv
  2.54    0.001008          27        37           mmap
  1.62    0.000644         214         3           clone
  0.92    0.000365          28        13         2 openat
  0.78    0.000309          18        17           close
  0.57    0.000227          25         9           mprotect
  0.47    0.000187          18        10           read
```

Παρατηρούμε ότι η ptrace καλείται περισσότερες φορές. Ξανατρέχουμε για την εντολή ls (strace -c strace ls) και καταλήγουμε στο ίδιο αποτέλεσμα.

```
------ ----------- ----------- --------- --------- ----------------
 51.92    0.000054           1        76           process_vm_readv
 48.08    0.000050           0       482           ptrace
  0.00    0.000000           0         1           read
  0.00    0.000000           0       234           write
  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
```

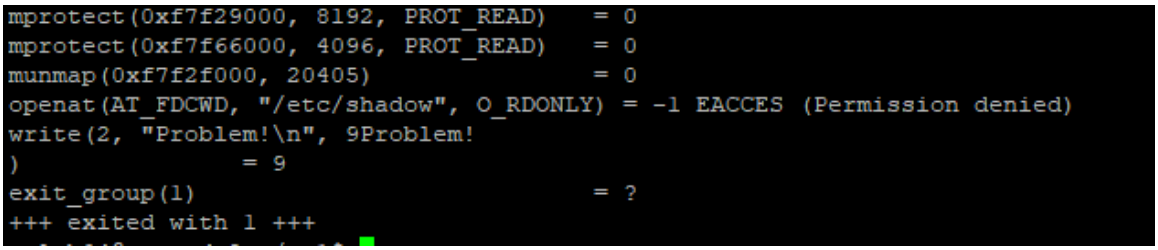
2. Στο object file main.o, δεν έχουμε όλες τις διευθύνσεις από τα υπόλοιπα object files που χρειάζονται για την παραγωγή του τελικού εκτελέσιμου, οπότε η callq, για την κλήση της συνάρτησης zing αναφέρεται στην σχετική της διεύθυνση <main+20>, και η απόλυτη διεύθυνση της zing θα καθοριστεί αφού γίνει το linking στο εκτελέσιμο. Αντίθετα, στο εκτελέσιμο (zing), καθώς έχει προηγηθεί το linking, γνωρίζουμε ήδη όλες τις απόλυτες διευθύνσεις και συνεπώς χρησιμοποιούμε την απόλυτη διεύθυνση της συνάρτησης zing.

3. Το μόνο που άλλαξε είναι το int main, επομένως παραλείψαμε να ξαναγράψουμε τις void συναρτήσεις.

```
int main(int argc, char **argv) {
    int fd, oflags, mode;
    oflags = O_CREAT | O_WRONLY | O_TRUNC;
    mode = S_IRUSR | S_IWUSR;
    if (argc < 3) {
        printf("Usage: WRONG]\n");
        exit(1);
    }
        fd = open(argv[argc-1], oflags, mode);
    if (fd == -1){
        perror("open");
        exit(1);
    }
    else {
for(int i=1; i<argc-1; i++) {
        write_file(fd, argv[i]);
        }
```

```
        close(fd);
    }
    return 0;
}
```

4. Τρέχουμε την εντολή strace ./whoops και παρατηρούμε το πρόβλημα (Permission denied):




Δεν έχουμε άδεια πρόσβασης στο αρχείο /etc/shadow.