ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ

ΣΧΟΛΉ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΏΝ ΚΑΙ ΜΗΧΑΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΉΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

(2019-2020)

1η ΕΡΓΑΣΤΗΡΙΑΚΉ ΑΝΑΦΟΡΆ

Ομάδα: oslaba36

Ονοματεπώνυμο: Χρήστος Τσούφης – 03117176

Νίκος Χαράλαμπος Χαιρόπουλος – 03119507

Εκτέλεση Εργαστηρίου: 17/03/2020

Σημείωση: Η άσκηση αυτή επιλύθηκε ατομικά από τον καθένα μας για την καλύτερη κατανόηση των εννοιών οπότε κρίνεται σκόπιμο να παρουσιαστούν και οι δύο προσπάθειες. Συγκεκριμένα για την 1.1 οι επιλύσεις είναι παρόμοιες οπότε παρατίθενται μια φορά, σε αντίθεση με την 1.2 όπου παρατίθενται και οι δύο υλοποιήσεις.

Άσκηση 1: Εισαγωγή στο περιβάλλον προγραμματισμού

1.1 Σύνδεση με αρχείο αντικειμένων (Κοινή)

Βήματα:

1. Αντιγραφή αρχείων zing.h και zing.o στο κατάλογο εργασίας:
   ```
   mkdir ask11
   cd ask11/
   cp home/oslab/code/zing/zing.h .
   cp home/oslab/code/zing/zing.o .
   ```
2. Δημιουργία αρχείου αντικειμένων main.o για τη συνάρτηση main():
   ```
   vi main.txt
   mv main.txt main.c
   gcc -Wall -c main.c
   gcc main.o zing.o -o zing1
   ./zing1
   ```
   όπου main.c :

```c
#include <stdio.h>
#include "zing.h"
int main(void) {
    zing();
    return 0;
}
```

Εκτελώντας την εντολή `./zing` προκύπτει ως output:

```
Hello, oslaba36
```

Ερωτήσεις:

1. *Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;*

   Οι επικεφαλίδες αποτελούν την διεπαφή προς άλλα κομμάτια κώδικα (API). Περιέχουν πρότυπα και δηλώσεις (είτε συναρτήσεις είτε καθολικές μεταβλητές). Έχουν κατάληξη .h και στην προκειμένη περίπτωση είναι το zing.h. Η επικεφαλίδα εκτός του prototype της συνάρτησης zing.h, είναι σημαντική για την σωστή κλήση της συνάρτησης από τον compiler (ορίσματα και τύπος που επιστρέφει).

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

   ```
   vi Makefile
   make
   ```
   και τυπώνει:
   ```
   gcc -Wall -c main.c
   gcc -o zing1 zing.o main.o
   ```

   όπου το Makefile είναι:
   ```
   zing1: zing.o main.o
      gcc -o zing1 zing.o main.o
   main.o: main.c
      gcc -Wall -c main.c
   ```

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.*

```
vi zing2.txt
mv zing2.txt zing2.c
gcc -Wall -c zing2.c
```

όπου main.c :

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

void zing(){
    char *team;
    team = getlogin();
    printf("Hello and welcome team %s!\n",team);
}
```

Έπειτα,

```
vi Makefile
make
```

και τυπώνει:

```
gcc -Wall main.o zing2.o -o zing2
```

(Δεν εμφανίζεται κάτι για το zing1 διότι δεν έγινε αλλαγή σε σχέση με πριν.)

Και το τροποποιημένο Makefile είναι:

```
CC = gcc
CFLAGS = -Wall
all: zing1 zing2

zing1: main.o zing.o
        $(CC) $(CFLAGS) main.o zing.o -o zing1

zing2: main.o zing2.o
        $(CC) $(CFLAGS) main.o zing2.o -o zing2

main.o:
        $(CC) $(CFLAGS) -c main.c

zing2.o:
        $(CC) $(CFLAGS) -c zing2.c

clean:
                rm zing.o main.o zing1 zing2.o zing2
```

Το output είναι: `Hello and welcome team oslaba36!`

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;*

   Με την δημιουργία ενός Makefile γιατί εκεί μπορούν να γίνουν μεμονωμένες αλλαγές. Κάθε φορά που μεταβάλλεται ένα αρχείο, το Make μπορεί να δει πότε είχε παραχθεί, πότε τροποποιήθηκε κλπ. οπότε θα δημιουργήσει τα dependencies εκείνα τα οποία χρειάζεται να γίνουν updated και αυτό θα προκαλέσει παραγωγή καινούριων object αρχείων και του τελικού εκτελέσιμου. Οπότε, οποιαδήποτε αλλαγή θα οδηγήσει σε compile μόνο το αρχείο που έχει τροποποιηθεί ενώ τα υπόλοιπα δεν θα εκτελεστούν εφόσον παραμένουν ίδια. Με άλλα λόγια, μια σωστή υλοποίηση θα ήταν να υπάρχει ένα ξεχωριστό αρχείο για τη συνάρτηση και να γίνεται μεταγλώττιση μόνο του δικού της κώδικα και έπειτα, ο νέος object code της συνάρτησης να γίνεται link με τον προϋπάρχοντα object code των υπόλοιπων συναρτήσεων.

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 συνέβη 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;

   hint #1: Επειδή τα λάθη είναι ανθρώπινα, συνιστούμε να κρατάτε backup των αρχείων σας και να χρησιμοποιείτε Makefile.

   Αυτό που συνέβη είναι ότι δεν διαγράφηκε αλλά αντικαταστάθηκε με τον εκτελέσιμο κώδικα του προγράμματος. Η διαφορά είναι η εξής: στην πρώτη περίπτωση το αρχείο foo.c δεν υπάρχει πλέον ενώ στην δεύτερη το αρχείο υπάρχει αλλά ο πηγαίος κώδικας όχι. Με την εκτέλεση αυτής της εντολής έγινε compile του input foo.c και γράφτηκε το πρόγραμμα στο αρχείο foo.c . Με άλλα λόγια, ο compiler δημιουργεί το τελικό αρχείο για να κρατήσει το τελικό εκτελέσιμο όταν ξεκινάει η ένωση, αλλά όταν αυτή αποτυγχάνει το αρχείο διαγράφεται. Έτσι, το αρχείο εξαφανίζεται επειδή αρχίζει να γίνεται overwrite από το εκτελέσιμο αρχείο αλλά και εκείνο το αρχείο έχει επίσης διαγραφεί επειδή η ένωση απέτυχε και το εκτελέσιμο δεν δημιουργείται επιτυχώς. Εν κατακλείδι, η εντολή αυτή λέει στον compiler να μεταγλωττίσει το αρχείο foo.c και να παράξει ως έξοδο object code στο αρχείο foo.c. Έτσι, με την εκτέλεση αυτής της εντολής το αρχικό αρχείο foo.c γίνεται overwrite με τον object code και χάνεται ο πηγαίος κώδικας που γράφθηκε.

1.2 Συνένωση δύο αρχείων σε τρίτο (2 Υλοποιήσεις)

(1η Υλοποίηση – Χρήστος)
Ζητείται πρόγραμμα που θα δημιουργεί ένα αρχείο εξόδου τα περιεχόμενα του οποίου θα προκύπτουν συνενώνοντας τα περιεχόμενα δύο αρχείων εισόδου. Το πρόγραμμα (fconc) θα δέχεται δύο ή τρία ορίσματα. Πιο συγκεκριμένα:

- Αν το πρόγραμμα κληθεί χωρίς τα κατάλληλα ορίσματα θα εμφανίζεται μήνυμα βοήθειας.
- Το πρώτο και το δεύτερο όρισμα είναι τα αρχεία εισόδου.
- Το τρίτο όρισμα είναι προαιρετικό και είναι το αρχείο εξόδου.
- Η προεπιλεγμένη (default) τιμή για το αρχείο εξόδου είναι fconc.out
- Σε περίπτωση που ένα από τα αρχείο εισόδου δεν υπάρχει, το πρόγραμμα θα πρέπει να εμφανίζει κατάλληλο μήνυμα λάθους.

Προτεινόμενος σκελετός υλοποίησης (συναρτήσεις):

- void doWrite(int fd, const char *buff, int len)
  Συνάρτηση που αναλαμβάνει την εγγραφή στον περιγραφητή αρχείου fd.
- void write_file(int fd, const char *infile)
  Συνάρτηση που γράφει τα περιεχόμενα του αρχείου με όνομα infile στον περιγραφητή αρχείου fd. Χρησιμοποιεί την doWrite().

Ερωτήσεις:

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

   Διαδικασία:
   ```
   mkdir ask12          cd ask12/
   vi doWrite.c
   gcc -Wall -c doWrite.c
   ```
   όπου doWrite.c :

```c
#include <stdio.h>
#include <unistd.h>
//Writes the buffer from index 0 till length to
//the output file descriptor fd.
int doWrite(int fd, char buff[], int len){
        ssize_t wcnt;
        ssize_t idx = 0;
        do {
                wcnt = write(fd, buff + idx, len - idx);
                if (wcnt == -1) { //error
                        perror("write");
                        return 1;
                }
                idx += wcnt;
        } while (idx < len);
        return 0;
}
```

Και η write_file.c :

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "func.h"

// Tries to write the contents of in_fd to out_fd.
// Return 0 on success, 1 on failure.
int write_file(int out_fd, int in_fd) {
    char buff[BUFF_SIZE];
    ssize_t rcnt;
    for (;;){
        rcnt = read(in_fd, buff, BUFF_SIZE - 1);
        if (rcnt == 0) /* End-of-file */
            break;
        if (rcnt == -1){ /* error */
            perror("Error while reading input file");
            return 1;
        }
        doWrite(out_fd, buff, rcnt);
    }
    return 0;
}
```

Έπειτα, η main.c :

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include "func.h"

int main(int argc, char **argv) {
    int i;
    if (argc < 3 || argc > 4) {
        printf("Usage: .fconc inFile1 inFile2 [outFile (default:fconc.out)]\n");
        return 1;
    }

    //Open Input File 1
    int fd1 = open(argv[1], O_RDONLY);
    if (fd1 == -1) {
        perror("Error while opening inFile1");
        return 2;
    }
```

```
    //Open Input File 2
    int fd2 = open(argv[2], O_RDONLY);
    if (fd2 == -1) {
        perror("Error while opening inFile2");
        return 2;
    }

    int fd3;
    // Determine output file name
    if (argc == 4) {
        if (strcmp(argv[3], argv[1]) == 0
            || strcmp(argv[3], argv[2]) == 0) {
            printf("ERROR: OutFile name must be different than input file name.\n
");
            return 2;
        }
        else {
            fd3 = open(argv[3], O_WRONLY | O_CREAT | O_TRUNC, 0666);
        }
    }
    else {
        if (strcmp("fconc.out", argv[1]) == 0
            || strcmp("fconc.out", argv[2]) == 0) {
            printf("ERROR: File fconc.out must not be used as input.\n");
            return 2;
        }
        else {
            fd3 = open("fconc.out", O_WRONLY | O_CREAT | O_TRUNC, 0666);
        }
    }
    if (fd3 == -1){
            perror("Error opening/creating the outFile");
            return 3;
    }
    //Write inFile1 to OutFile
    i = write_file(fd3, fd1);
    if (i == 1) {
            perror("Write inFile1 to outFile Failed.");
            return 4;
    }
    //Write inFile1 to OutFile
    i = write_file(fd3, fd2);
    if (i == 1){
            perror("Write inFile2 to outFile Failed.");
            return 4;
    }
    return 0;
}
```

Και το Makefile:

```
vi Makefile
make
```

όπου Makefile:

```
fconc: main.o doWrite.o write_file.o
    gcc -Wall -o fconc main.o doWrite.o write_file.o

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

doWrite.o: doWrite.c
    gcc -Wall -c doWrite.c

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

Και το output είναι:

```
gcc -Wall -o fconc main.o doWrite.o write_file.o
```

Και η func.h:

```c
#ifndef FUNC_H__
#define FUNC_H__

#define BUFF_SIZE 1024

void doWrite(int fd, char buff[], int len);

int write_file(int out_fd, int in_fd);

#endif
```

Και f1.in:

```
This is file 1
```

Και f2.in:

```
This is file 2
```

Και με την εντολή : strace ./fconc f1.in f2.in

```
execve("./fconc", ["./fconc", "A", "B"], [/* 18 vars */]) = 0
brk(0)                                  = 0x2334000
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) = 0x7f8
b781c1000
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=30952, ...}) = 0
mmap(NULL, 30952, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8b781b9000
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) = 0x7f
8b77bf8000
mprotect(0x7f8b77d99000, 2097152, PROT_NONE) = 0
mmap(0x7f8b77f99000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENY
WRITE, 3, 0x1a1000) = 0x7f8b77f99000
mmap(0x7f8b77f9f000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANON
YMOUS, -1, 0) = 0x7f8b77f9f000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8
b781b8000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8
b781b7000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8
b781b6000
arch_prctl(ARCH_SET_FS, 0x7f8b781b7700) = 0
mprotect(0x7f8b77f99000, 16384, PROT_READ) = 0
mprotect(0x7f8b781c3000, 4096, PROT_READ) = 0
munmap(0x7f8b781b9000, 30952)           = 0
open("A", O_RDONLY)                     = -1 ENOENT (No such file or directory)
dup(2)                                  = 3
fcntl(3, F_GETFL)                       = 0x8002 (flags O_RDWR|O_LARGEFILE)
brk(0)                                  = 0x2334000
brk(0x2355000)                          = 0x2355000
fstat(3, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8
b781c0000
lseek(3, 0, SEEK_CUR)                   = -1 ESPIPE (Illegal seek)
write(3, "Error while opening inFile1: No "..., 55Error while opening inFile1: N
o such file or directory
) = 55
close(3)                                = 0
munmap(0x7f8b781c0000, 4096)            = 0
exit_group(2)                           = ?
+++ exited with 2 +++
oslaba36@os-node1:~/ask12$
```

1.3 Προαιρετικές Ερωτήσεις

1. Εκτελώντας `man strace`:

```
STRACE(1)                    General Commands Manual                    STRACE(1)

NAME
       strace - trace system calls and signals

SYNOPSIS
       strace   [-CdffhikqrtttTvVxxy]  [-In]  [-bexecve]  [-eexpr]...   [-acolumn]
       [-ofile] [-sstrsize] [-Ppath]... -ppid... / [-D] [-Evar[=val]]...  [-uuserβ
       name] command [args]

       strace   -c[df]   [-In]  [-bexecve]  [-eexpr]...   [-Ooverhead]  [-Ssortby]
       -ppid... / [-D] [-Evar[=val]]... [-uusername] command [args]

DESCRIPTION
       In the simplest case strace runs the specified command until it exits.   It
       intercepts  and  records the system calls which are called by a process and
       the signals which are received by a process.  The name of each system call,
       its  arguments and its return value are printed on standard error or to the
       file specified with the -o option.

       strace is a useful diagnostic, instructional, and debugging  tool.   System
       administrators, diagnosticians and trouble-shooters will find it invaluable
       for solving problems with programs for which  the  source  is  not  readily
       available  since  they do not need to be recompiled in order to trace them.
       Students, hackers and the overly-curious will find that a great deal can be
       learned  about  a system and its system calls by tracing even ordinary proβ
       grams.  And programmers will find that since system calls and  signals  are
       events  that  happen  at  the user/kernel interface, a close examination of
       this boundary is  very  useful  for  bug  isolation,  sanity  checking  and
       attempting to capture race conditions.

       Each line in the trace contains the system call name, followed by its arguβ
       ments in parentheses and its return value.  An example  from  stracing  the
       command "cat /dev/null" is:
```

```
-o filename Write  the trace output to the file filename rather than
            to stderr.  Use filename.pid if -ff  is  used.   If  the
            argument  begins  with  '|' or with '!' then the rest of
            the argument is treated as a command and all  output  is
            piped  to  it.  This is convenient for piping the debugβ
            ging output to a program without affecting the  redirecβ
            tions of executed programs.
```

Οπότε, εκτελώντας την εντολή `strace -o main` , η έξοδος της εντολής θα είναι:

```
oslaba36@os-node1:~/askl1$ strace -o main
usage: strace [-CdffhiqrtttTvVxxy] [-I n] [-e expr]...
              [-a column] [-o file] [-s strsize] [-P path]...
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]
   or: strace -c[df] [-I n] [-e expr]... [-O overhead] [-S sortby]
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]
-c -- count time, calls, and errors for each syscall and report summary
-C -- like -c but also print regular output
-w -- summarise syscall latency (default is system time)
-d -- enable debug output to stderr
-D -- run tracer process as a detached grandchild, not as parent
-f -- follow forks, -ff -- with output into separate files
-i -- print instruction pointer at time of syscall
-q -- suppress messages about attaching, detaching, etc.
-r -- print relative timestamp, -t -- absolute timestamp, -tt -- with usecs
-T -- print time spent in each syscall
-v -- verbose mode: print unabbreviated argv, stat, termios, etc. args
-x -- print non-ascii strings in hex, -xx -- print all strings in hex
-y -- print paths associated with file descriptor arguments
-h -- print help message, -V -- print version
-a column -- alignment COLUMN for printing syscall results (default 40)
-b execve -- detach on this syscall
-e expr -- a qualifying expression: option=[!]all or option=[!]val1[,val2]...
   options: trace, abbrev, verbose, raw, signal, read, write
-I interruptible --
   1: no signals are blocked
   2: fatal signals are blocked while decoding syscall (default)
   3: fatal signals are always blocked (default if '-o FILE PROG')
   4: fatal signals and SIGTSTP (^Z) are always blocked
      (useful to make 'strace -o FILE PROG' not stop on ^Z)
-o file -- send trace output to FILE instead of stderr
-O overhead -- set overhead for tracing syscalls to OVERHEAD usecs
-p pid -- trace process with process id PID, may be repeated
-s strsize -- limit length of print strings to STRSIZE chars (default 32)
-S sortby -- sort syscall counts by: time, calls, name, nothing (default time)
-u username -- run command as username handling setuid and/or setgid
-E var=val -- put var=val in the environment for command
-E var -- remove var from the environment for command
-P path -- trace accesses to path
```

2. Χρησιμοποιώντας το πρόγραμμα gdb (GNU debugger), εμφανίζεται η assembly συναρτήσεων.

```
oslaba36@os-node1:~/askl1$ gdb -q main.o
Reading symbols from main.o...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000000000 <+0>:     push   %rbp
   0x0000000000000001 <+1>:     mov    %rsp,%rbp
   0x0000000000000004 <+4>:     callq  0x9 <main+9>
   0x0000000000000009 <+9>:     mov    $0x0,%eax
   0x000000000000000e <+14>:    pop    %rbp
   0x000000000000000f <+15>:    retq
End of assembler dump.
(gdb)
```

Αντίστοιχα για το εκτελέσιμο zing1 η έξοδος θα είναι η εξής:

```
oslaba36@os-node1:~/ask11$ gdb -q zing1
Reading symbols from zing1...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x00000000004005d2 <+0>:     push   %rbp
   0x00000000004005d3 <+1>:     mov    %rsp,%rbp
   0x00000000004005d6 <+4>:     callq  0x400596 <zing>
   0x00000000004005db <+9>:     mov    $0x0,%eax
   0x00000000004005e0 <+14>:    pop    %rbp
   0x00000000004005e1 <+15>:    retq
End of assembler dump.
(gdb)
```

Η αλλαγή που εντοπίζεται είναι, στην assembly, το όρισμα της εντολής call. Στην πρώτη περίπτωση εκτελείται απευθείας η main ενώ στην δεύτερη εκτελείται μέσω της zing1.

3. Το νέο πρόγραμμα της άσκησης 1.2, είναι το fconc2.c και υποστηρίζει αόριστο αριθμό αρχείων εισόδου.

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

int main(int argc, char **argv){
    /*------------declarations------------------*/
    char buff[1024]; //for files up to 1Kbyte
    int fd[argc - 1], i;

    /*------------open input files------------------*/
    for(i=0; i<(argc - 2); i++)
        fd[i] = open(argv[i + 1], O_RDONLY);

    /*------------open output file------------------*/
    fd[argc - 2] = open(argv[argc - 1], O_WRONLY|O_CREAT|O_TRUNC,
S_IRUSR|S_IWUSR);

    /*----------------copy-paste------------------*/
    for(i=0; i<(argc - 2); i++){
        read(fd[i], buff, sizeof(buff) - 1);
        write(fd[argc - 2], buff, strlen(buff));
    }

    for(int i=0; i<(argc - 2); i++){
        close(fd[i]);
    }

    return 0;
}
```

4. Εκτελώντας την εντολή `/home/oslab/code/whoops/whoops`

```
oslaba36@os-node1:~$ /home/oslab/code/whoops/whoops
Problem!
oslaba36@os-node1:~$
```