Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2017-2018

Άσκηση 1:

Εισαγωγή στο περιβάλλον προγραμματισμού

oslabc19

Χαράλαμπος Κάρδαρης
A.M. 03114074
bkardaris@hotmail.com

Μιχάλης Παπαδόπουλλος
A.M. 03114702
mechatr0nic.cy@gmail.com

Άσκηση 1:

[source codes]

--- main.c code ---

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

#include "zing.h"

int main(void)
{

	// call zing
	zing();
	return 0;
}
```

Οι κώδικες των υπολοίπων αρχείων βρίσκονται παρακάτω, στα αντίστοιχα ερωτήματα από την εκφώνηση.

[compile and link]
Compile των προγραμμάτων γίνεται με τις εντολές:
$ gcc main.c -c
$ gcc zing2.c -c

Το linking με τις εντολές (παράγεται το εκτελέσιμο πρόγραμμα):
$ gcc main.o zing.o -lc
$ gcc main.o sing2.o -lc
για τα δύο προγράμματα αντίστοιχα.

Το Makefile κάνει την παραπάνω διαδικασία αυτόματα με τις εντολές:
$ make prog
$ make prog2

Με τις εντολές:
$ ./prog
$ ./prog2

παίρνουμε το output των προγραμμάτων που είναι:
Hello, oslabc19
Hello, oslabc19!
αντίστοιχα

Τον χρήστη (oslabc19), όπως φαίνεται και στον κώδικα της zing2.c (παρακάτω) μας τον δίνει η συνάρτηση getLogin().

Από τα manpages πληροφορούμαστε για αυτή τη συνάρτηση:
getlogin() returns a pointer to a string containing the name of the user logged in on the controlling terminal of the process, or a null pointer if this information cannot be determined.

1. Χρησιμοποιούμε τα αρχεία επικεφαλίδας (header files), ωστε να ορίσουμε τις απαραίτητες συναρτήσεις (definitions), δομές δεδομένων, σταθερές και preprocessor directives, που θα χρειαστεί το προγραμμά μας και γενικότερα που μπορούν να χρησιμοποιηθούν απο πληθώρα προγραμμάτων όπως για παράδειγμα η κλασσική βιβλιοθήκη της C – stdio.h.
Κάνοντας include κάποιο αρχείο επικεφαλίδας σε κάποιο αρχείο κώδικα, είναι σαν να αντιγράφουμε τα περιεχόμενα του αρχείου αυτού στη θέση που κάνουμε include. Αυτό βοηθά τον προγραμματιστή να είναι πιο productive, αφου μειώνει το χρόνο που χρειάζεται ώστε να κάνει αλλαγές στον κώδικα (αφου δεν θα χρειάζεται να κανει copy-paste σε όλα τα αρχεία που περιέχουν τον κώδικα που πρέπει να αλλαχτει) καθώς επίσης η διαδικασία αποσφαλμάτωσης γίνεται πιο εύκολα. Έτσι, το προγραμμά μας δεν θα έχει πρόβλημα με inconsistent κωδικα – αφου μόνο μια φορα χρειάζεται να κάνουμε την αλλαγή – στο αναλογο αρχείο υλοποίησης και επικεφαλίδας.

Όσον αφορά την γραμμή στον κώδικα του zing.h “#pragma once” έχουμε:

Αυτη η εντολή εξασφαλίζει ότι κάθε δηλωθείσα συνάρτηση σε ένα header δεν θα γίνει include παραπάνω από μία φορά σε ένα πρόγραμμα.

2. Το Makefile είναι:

```
CC='gcc'
CFLAGS='-Wall'
LIBS='-lc'
ECLEAN='main.o zing2.o prog prog2'

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

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

prog : main.o zing.o
	$(CC) $^ $(CFLAGS) -o $@ $(LIBS)

prog2 : main.o zing2.o
	$(CC) $^ $(CFLAGS) -o $@ $(LIBS)

clean :
	rm -rf $(ECLEAN)
```

3. Το zing2.c είναι:

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

#include "zing.h"

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

4.
Το πρόβλημα αντιμετωπίζεται με τη δημιουργία ξεχωριστών αρχείων για κάθε συνάρτηση (ή ομάδα συναρτήσεων) και τη χρήση Makefile. Με αυτόν τον τρόπο κάθε φορά που κάνουμε αλλαγές σε κάποια συνάρτηση, δεν χρειάζεται να κάνουμε compile όλο τον κώδικα από την αρχή, αλλά μόνο του αρχείου που την περιέχει και απλώς μετά να κάνουμε link ξανά όλα τα παραγόμενα .ο αρχεία. Το Makefile ειναι γραμμένο με τέτοιο τρόπο, ώστε να αυτοματοποείται η διαδικασία επιλογής του αρχείου που πρέπει να γίνει compile και του linking.

5. Everything is a file [Unix]. Το αρχείο `foo.c` περιέχει πλέον εκτελέσιμο κώδικα που εγινε generate απο τον compiler gcc.

Άσκηση 2:

1.

[show help menu code]

```
if ((argc != 3 && argc != 4) || strcmp(argv[1], "--help") == 0) {
        usage();
}

void usage( void )// prints help message to STDOUT
{
        printf ("Usage: ./fconc <infile1> <infile2> [<outfile>]\n");
        exit(0);
}
```

[show error handling code]

```
 //Wrong arguments, or, invokation with –help:
 if ((argc != 3 && argc != 4) || strcmp(argv[1], "--help") == 0) {
         usage();
 }
 char *outf = "fconc.out";
 if (argc == 4) {
         outf = argv[3];
         if ( (strcmp(argv[3], argv[2]) == 0 || strcmp(argv[3], argv[1]) == 0) ) {
                 warning();
                 // Warning: outfile is the same as one of its input file
                 // print Warning, and ask user to give an alternative output filename
         }
 }

 [File descriptors]
         // open()
         fd[0] = open(argv[1], O_RDONLY);
         if (fd[0] == -1) {  // error handling :: fd[0]
         perror("open[infile1]");
         exit(-1);
         }

         fd[1] = open(argv[2], O_RDONLY);
         if (fd[1] == -1) {  // error handling :: fd[1]
         close(fd[0]);
         perror("open[infile2]");
         exit(-1);
         }

         fd[2] = open(outf, opflags, opmode);
         if (fd[2] == -1) {  // error handling :: fd[2]
         close(fd[0]); close(fd[1]);
         perror("open[outfile]");
         exit(-1);
         }
```

[show writep() implementation code + description]

```
ssize_t writep( int fd, const void* buf, size_t size )
{
        /*  EINTR          Interrupted function call
         *  EIO            Input/output error
         *  EAGAIN         Resource temporarily unavailable
         */

    ssize_t numWritten;                 // bytes actually written
    while (size > 0) {                  // while there is something to write

        do {                            // write to file at least once, retry on error
            numWritten = write(fd, buf, size);
        } while ( (numWritten < 0) && (errno == EINTR || errno == EAGAIN) );

        if (numWritten < 0) {     // in case of other error
            return numWritten;    // return the actual value from write()
         }

        size -= numWritten;       // in case of incomplete write, continue again until size = 0
        buf += numWritten;        // move pointer to the right offset
    }
    return 0;
}
```

Από τα manpages της write παίρνουμε (man 2 write):

The number of bytes written may be less than count if, for example, there is insufficient space on the underlying physical medium, or the RLIMIT_FSIZE resource limit is encountered (see setrlimit(2)), or the call was interrupted by a signal handler after having written less than count bytes.

Για να αντιμετωπίσουμε τον παραπάνω κίνδυνο, άρα να είμαστε σίγουροι ότι θα γραφτούν όλοι οι χαρακτήρες που διαβάστηκαν με τη read() και υπάρχουν στον buffer, δημιουργήσαμε τη συνάρτηση writep, η οποία υποχρεώνει να γραφτούν όλοι οι χαρακτήρες του buffer στο αρχείο.
Ξέροντας το συνολικό size που θέλουμε να γράψουμε αν η write γράψει κάτι λιγότερο από αυτό, εφαρμόζεται επαναληπτικά σε loop μέχρι να γραφτούν όλοι οι χαρακτήρες, καθώς επίσης και αν για κάποιο λόγο υπάρξει σφάλμα κατά το write() υποχρεώνουμε να ξαναγίνει προσπάθεια για write.

[import fconc.c]
Με βάση τα παραπανω, ο συνολικός κώδικας της fconc.c έχει ως εξής:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>             /* strcmp */
#include <stdlib.h>             /* exit   */
#include <errno.h>

// ==================
void usage(void);
void warning(void);
ssize_t writep( int fd, const void* buf, size_t size );

// ==================
#ifndef BUFSIZE                 // allow gcc to override size using -DBUFSIZE <SIZE>
#define BUFSIZE 1024
#endif
// ==================

int main( int argc, char *argv[] )
{
        if ((argc != 3 && argc != 4) || strcmp(argv[1], "--help") == 0) {
                usage();
        }
        char *outf = "fconc.out";
        if (argc == 4) {
                outf = argv[3];
                if ( (strcmp(argv[3], argv[2]) == 0 || strcmp(argv[3], argv[1]) == 0) ) {
                        warning();
                        // Warning: outfile is the same as one of its input file
                        // print Warning, and ask user to give an alternative output filename
                }
        }

        int fd[3] = {0, 0, 0};          /* file descriptors */
        int opflags = O_CREAT           /* create */
                        | O_WRONLY      /* write only */
                        | O_APPEND;     /* append: set file offset at end-of-file */
        //              | O_TRUNC;      // TRUNCATE existing file to zero length (we
                                        //might need to use this flag)
        mode_t opmode = 0666;           /* rw-rw-rw */
        ssize_t numRead;                // bytes succesfully read

        char buffer[BUFSIZE];

        // open()
        fd[0] = open(argv[1], O_RDONLY);
        if (fd[0] == -1) {    // error handling :: fd[0]
                perror("open[infile1]");
                exit(-1);
        }
```

```
        fd[1] = open(argv[2], O_RDONLY);
        if (fd[1] == -1) {    // error handling :: fd[1]
                close(fd[0]);
                perror("open[infile2]");
                exit(-1);
        }

        fd[2] = open(outf, opflags, opmode);
        if (fd[2] == -1) {    // error handling :: fd[2]
                close(fd[0]); close(fd[1]);
                perror("open[outfile]");
                exit(-1);
        }
        /*
                man 2 write:
                The number of bytes written may be less than count if, for example,
                there  is  insufficient  space  on the underlying physical
               medium, or the RLIMIT_FSIZE resource limit is  encountered  (see
               setrlimit(2)),  or  the call was interrupted by a signal handler
               after having written less than count bytes.
                Thus we provide a wrapper, which is persistent.
        */

        // read() / write() [infile1]
        while((numRead = read(fd[0], buffer, BUFSIZE)) > 0) {
                // 0 indicates EOF
                writep(fd[2], buffer, (size_t)numRead);
        }

        // read() / write() [infile2]
        while((numRead = read(fd[1], buffer, BUFSIZE)) > 0) {
                // 0 indicates EOF
                writep(fd[2], buffer, (size_t)numRead);
        }

        // close()
        int cnt;
        for (cnt = 0; cnt < 3; cnt++) {
                if (close(fd[cnt]) == -1) {
                  perror("close");
                  exit(-1);
                 }
        }

        return 0;
}
```

```
void usage( void )  // prints help message to STDOUT
{
        printf ("Usage: ./fconc <infile1> <infile2> [<outfile>]\n");
        exit(0);
}

void warning(void){
        printf("Warning: outfile is the same as one of the input file\n");
        printf("Please give alternative output file name\n");
        exit(0);
}

ssize_t writep( int fd, const void* buf, size_t size )
{
        /*  EINTR          Interrupted function call
         *  EIO            Input/output error
         *  EAGAIN         Resource temporarily unavailable
         */

   ssize_t numWritten;                  // bytes actually written
   while (size > 0) {                   // while there is something to write

      do {                              // write to file at least once, retry on error
          numWritten = write(fd, buf, size);
      } while ( (numWritten < 0) && (errno == EINTR || errno == EAGAIN) );

      if (numWritten < 0) {         // in case of other error
         return numWritten;         // return the actual value from write()
        }

      size -= numWritten;           // in case of incomplete write, continue again until size =
0
      buf += numWritten;            // move pointer to the right offset
   }
   return 0;
}
```

[ενδεικτικές εκτελέσεις του fconc]

```
$ ls
fconc    fconc.o    in1.txt  Makefile  out3.out
fconc.c  fconc.out  in2.txt  notes     strace.out

$ cat in1.txt
This is a test!

$ cat in2.txt
This is fun!

$ ./fconc –help
Usage: ./fconc <infile1> <infile2> [<outfile>]

$ ./fconc in1.txt
Usage: ./fconc <infile1> <infile2> [<outfile>]

$ ./fconc in1.txt in2.txt in2.txt
Warning: outfile is the same as one of the input file
Please give alternative output file name

$ ./fconc in1.txt in2.txt

$ cat fconc.out
This is a test!
This is fun!
```

[strace output]

Το output της εκτέλεσης της εντολής ./fconc in1.txt in2.txt είναι:

```
execve("./fconc", ["./fconc", "in1.txt", "in2.txt"], [/* 20 vars */]) = 0 <0.000251>
brk(0)                                  = 0x1ec8000 <0.000011>
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory) <0.000013>
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe8ccdb6000 <0.000010>
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory) <0.000010>
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3 <0.000013>
fstat(3, {st_mode=S_IFREG|0644, st_size=29766, ...}) = 0 <0.000009>
mmap(NULL, 29766, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe8ccdae000 <0.000009>
close(3)                                = 0 <0.000008>
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory) <0.000010>
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3 <0.000014>
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 <0.000009>
fstat(3, {st_mode=S_IFREG|0755, st_size=1738176, ...}) = 0 <0.000008>
mmap(NULL, 3844640, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe8cc7ed000 <0.000012>
mprotect(0x7fe8cc98e000, 2097152, PROT_NONE) = 0 <0.000019>
```

```
mmap(0x7fe8ccb8e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1a1000) = 0x7fe8ccb8e000 <0.000014>
mmap(0x7fe8ccb94000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7fe8ccb94000 <0.000011>
close(3)                                = 0 <0.000008>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7fe8ccdad000 <0.000009>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7fe8ccdac000 <0.000010>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7fe8ccdab000 <0.000009>
arch_prctl(ARCH_SET_FS, 0x7fe8ccdac700) = 0 <0.000008>
mprotect(0x7fe8ccb8e000, 16384, PROT_READ) = 0 <0.000012>
mprotect(0x7fe8ccdb8000, 4096, PROT_READ) = 0 <0.000013>
munmap(0x7fe8ccdae000, 29766)           = 0 <0.000016>
open("in1.txt", O_RDONLY)               = 3 <0.000012>
open("in2.txt", O_RDONLY)               = 4 <0.000011>
open("fconc.out", O_WRONLY|O_CREAT|O_APPEND, 0666) = 5 <0.000013>
read(3, "This is a test!\n", 1024)      = 16 <0.000009>
write(5, "This is a test!\n", 16)       = 16 <0.000020>
read(3, "", 1024)                       = 0 <0.000008>
read(4, "This is fun!\n", 1024)         = 13 <0.000009>
write(5, "This is fun!\n", 13)          = 13 <0.000012>
read(4, "", 1024)                       = 0 <0.000008>
close(3)                                = 0 <0.000009>
close(4)                                = 0 <0.000009>
close(5)                                = 0 <0.000009>
exit_group(0)                           = ?
+++ exited with 0 +++
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         5           read
  0.00    0.000000           0         2           write
  0.00    0.000000           0         5           open
  0.00    0.000000           0         5           close
  0.00    0.000000           0         2           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         3           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         1           brk
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    37         3 total
```

Όπως φαίνεται το κομμάτι που μας ενδιαφέρει άμεσα και αναφέρεται στις κλήσεις συστήματος του κώδικα μας, είναι από την γραμμή --open("in1.txt", O_RDONLY) = 3 <0.000012>-- και κάτω.