ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2016-2017

Άσκηση 1: Εισαγωγή στο περιβάλλον προγραμματισμού

1 Ασκήσεις

1.1 Σύνδεση με αρχείο αντικειμένων

Στην άσκηση αυτή ζητείται η δημιουργία ενός εκτελέσιμου αρχείου (με όνομα `zing`) που θα καλεί τη συνάρτηση `zing()`. Η συνάρτηση αυτή δηλώνεται στο αρχείο `zing.h`. Επιπρόσθετα, είναι διαθέσιμο το αρχείο αντικειμένων (object file) `zing.o`. Τα αρχεία αυτά βρίσκονται στον κατάλογο `/home/oslab/code/zing` και θα πρέπει να αντιγραφούν στον κατάλογο εργασίας σας.

Βήματα:

1. Αντιγραφή αρχείων `zing.h` και `zing.o` στον κατάλογο εργασίας σας.
2. Δημιουργία αρχείου αντικειμένων `main.o` για τη συνάρτηση `main()`.
3. Σύνδεση (linking) των δύο αρχείων αντικειμένων.

Ερωτήσεις:

1. Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;
2. Ζητείται κατάλληλο `Makefile` για τη δημιουργία του εκτελέσιμου της άσκησης.
3. Γράψτε 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`.

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;
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 συνέβει 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`.

Σημείωση: Το αρχείο `zing.o` είναι για 32-bit x86 αρχιτεκτονική και, συνεπώς, ασύμβατο με αρχεία διαφορετικών αρχιτεκτονικών. Σε ένα τυπικό περιβάλλον 64-bit x86 μπορεί να χρησιμοποιηθεί ο διακόπτης `-m32` στον gcc, ώστε να παράγει κώδικα 32-bit.

1.2 Συνένωση δύο αρχείων σε τρίτο

Στην άσκηση αυτή ζητείται πρόγραμμα που θα δημιουργεί ένα αρχείο (αρχείο εξόδου). Τα περιεχόμενα του αρχείου εξόδου θα προκύπτουν συνενώνοντας τα περιεχόμενα δύο αρχείων εισόδου. Το πρόγραμμα (`fconc`) θα δέχεται δύο ή τρία ορίσματα.

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.

Παραδείγματα εκτέλεσης:

```
$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]
$ ./fconc A B
A: No such file or directory
$ echo 'Goodbye,' > A
$ echo 'and thanks for all the fish!' > B
$ ./fconc A B
$ cat fconc.out
Goodbye,
and thanks for all the fish!
$ ./fconc A B C
$ cat C
Goodbye,
and thanks for all the fish!
```

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.

2 Εξέταση άσκησης και αναφορά

Η προθεσμία για την εξέταση της άσκησης στο εργαστήριο θα ανακοινωθεί στη λίστα του μαθήματος. Μετά την εξέταση η κάθε ομάδα θα πρέπει να συντάξει μια (σύντομη) αναφορά και να τη στείλει μέσω e-mail στους υπευθύνους της ομάδας εργαστηρίου. Αναλυτικές οδηγίες για τη διεύθυνση αποστολής υπάρχουν στη σελίδα του μαθήματος. Η προθεσμία για την αναφορά είναι μια εβδομάδα μετά την προθεσμία εξέτασης της άσκησης.
Η αναφορά αυτή θα περιέχει:

- Για την άσκηση 1.1:
  - Τον πηγαίο κώδικα (source code) της άσκησης
  - Τη διαδικασία μεταγλώττισης και σύνδεσης
  - Την έξοδο εκτέλεσης του προγράμματος
  - Σύντομες απαντήσεις στις ερωτήσεις
- Για την άσκηση 1.2:
  - Τον πηγαίο κώδικα (source code) της άσκησης
  - Σύντομες απαντήσεις στις ερωτήσεις

3 Προαιρετικές ερωτήσεις (επιπλέον βαθμοί: 0)

1. Χρησιμοποιήστε την εντολή `strace` για να εντοπίσετε με ποια κλήση συστήματος υλοποιείται η εντολή `strace`. Υπόδειξη: με `strace -o file` η έξοδος της εντολής τοποθετείται στο αρχείο `file`.

2. Χρησιμοποιώντας το πρόγραμμα gdb (GNU debugger), μπορείτε να δείτε την assembly συναρτήσεων. Για παράδειγμα αν το χρησιμοποιήσετε για τη συνάρτηση `main()` του αρχείου `main.o` της άσκησης 1.1 θα δείτε κάτι σαν το παρακάτω:

```
$ gdb -q main.o
(no debugging symbols found)
(gdb) disassemble main
Dump of assembler code for function main:
0x00000000 <main+0>:    lea    0x4(%esp),%ecx
0x00000004 <main+4>:    and    $0xfffffff0,%esp
0x00000007 <main+7>:    pushl  -0x4(%ecx)
0x0000000a <main+10>:   push   %ebp
0x0000000b <main+11>:   mov    %esp,%ebp
0x0000000d <main+13>:   push   %ecx
0x0000000e <main+14>:   sub    $0x4,%esp
0x00000011 <main+17>:   call   0x12 <main+18>
0x00000016 <main+22>:   mov    $0x0,%eax
0x0000001b <main+27>:   add    $0x4,%esp
0x0000001e <main+30>:   pop    %ecx
0x0000001f <main+31>:   pop    %ebp
0x00000020 <main+32>:   lea    -0x4(%ecx),%esp
0x00000023 <main+35>:   ret
End of assembler dump.
```

   Η πρώτη στήλη είναι η διεύθυνση, η δεύτερη είναι η απόσταση από την αρχή της συνάρτησης, και οι επόμενες στήλες είναι η εντολή assembly με τα ορίσματά της.

   Αντίστοιχα για το εκτελέσιμο `zing` η έξοδος θα μοιάζει κάπως έτσι:

```
$ gdb -q zing
(no debugging symbols found)
(gdb) disassemble main
Dump of assembler code for function main:
0x080483e0 <main+0>:    lea    0x4(%esp),%ecx
0x080483e4 <main+4>:    and    $0xfffffff0,%esp
0x080483e7 <main+7>:    pushl  -0x4(%ecx)
0x080483ea <main+10>:   push   %ebp
0x080483eb <main+11>:   mov    %esp,%ebp
0x080483ed <main+13>:   push   %ecx
0x080483ee <main+14>:   sub    $0x4,%esp
0x080483f1 <main+17>:   call   0x8048404 <zing>
0x080483f6 <main+22>:   add    $0x4,%esp
0x080483f9 <main+25>:   xor    %eax,%eax
0x080483fb <main+27>:   pop    %ecx
0x080483fc <main+28>:   pop    %ebp
0x080483fd <main+29>:   lea    -0x4(%ecx),%esp
0x08048400 <main+32>:   ret
End of assembler dump.
```

Παρατηρήστε ότι εκτός των απολύτων διευθύνσεων, η μοναδική αλλαγή στην assembly είναι το όρισμα της εντολής `call` (κλήση συνάρτησης). Πού οφείλεται η αλλαγή; ποιος την έκανε;

3. Γράψτε το πρόγραμμα της άσκησης 1.2, ώστε να υποστηρίζει αόριστο αριθμό αρχείων εισόδου (π.χ. 1, 2, 3, 4, ...). Θεωρήστε ότι το τελευταίο όρισμα είναι πάντα το αρχείο εξόδου.

4. Εάν τρέξετε το εκτελέσιμο `/home/oslab/code/whoops/whoops` θα δώσει:

   ```
   $ /home/oslab/code/whoops/whoops
   Problem!
   ```

   Θεωρήστε ότι **πραγματικά** υπάρχει πρόβλημα. Εντοπίστε το.

   hint #2: Χρησιμοποιήστε επεξηγηματικά μηνύματα λάθους στα προγράμματά σας.