ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο Εξάμηνο ΗΜΜΥ

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Εργασία σε assembly του επεξεργαστή ARM

Ερώτημα 1ο: Μετατροπή εισόδου από τερματικό

Να γραφεί πρόγραμμα **σε assembly του επεξεργαστή ARM** το οποίο θα λαμβάνει ως είσοδο από τον χρήστη μέσω τερματικού, μια συμβολοσειρά μεγέθους έως 32 χαρακτήρων. Αν η είσοδος είναι μεγαλύτερη, οι χαρακτήρες που περισσεύουν, να αγνοούνται. Στην συμβολοσειρά αυτή να γίνεται η παρακάτω μετατροπή:

- Αν ο χαρακτήρας είναι κεφαλαίο γράμμα της αγγλικής αλφαβήτου τότε να μετατρέπεται σε πεζό και αντίστροφα.

- Αν ο χαρακτήρας βρίσκεται στο εύρος [‘0’, ‘9’], θα πραγματοποιείται η ακόλουθη μετατροπή:

‘0’ → ‘5’
‘1’ → ‘6’
‘2’ → ‘7’
‘3’ → ‘8’
‘4’ → ‘9’
‘5’ → ‘0’
‘6’ → ‘1’
‘7’ → ‘2’
‘8’ → ‘3’
‘9’ → ‘4’

- Οι υπόλοιποι χαρακτήρες να παραμένουν αμετάβλητοι

Το πρόγραμμα να είναι συνεχούς λειτουργίας και να τερματίζει όταν λάβει ως είσοδο μια συμβολοσειρά μήκους ένα που θα αποτελείται από τον χαρακτήρα ‘Q’ ή ‘q’.

Ο κώδικας που αναφέρεται στην μετατροπή να γραφεί ως ξεχωριστή συνάρτηση. Στόχος είναι η ελαχιστοποίηση των γραμμών κώδικα που απαιτούνται για το σώμα της εν λόγω συνάρτησης.

Η υλοποίηση του προγράμματος μπορεί να γίνει είτε με χρήση συναρτήσεων της βιβλιοθήκης της C (scanf, printf, ….) είτε με system calls του Linux (read, write, ….) είτε με συνδυασμούς και των δύο.

Ένα πιθανό παράδειγμα εκτέλεσης του προγράμματος θα μπορούσε να είναι το παρακάτω:

```
$ ./ask1.out
Input a string of up to 32 chars long: ##asd123$X8B5mnjL9!
Result is: ##ASD678$x3b0MNJl4!

Input a string of up to 32 chars long: Q
Exiting….

$
```

Σημαντική παρατήρηση:

Το ανανεωμένο περιβάλλον qemu, επιτρέπει την χρήση του GNU Debugger (gdb) τόσο για την γλώσσα C όσο και για την assembly. Η χρήση του θα διευκολύνει πάρα πολύ την αποσφαλμάτωση όλων των ασκήσεων. Προσοχή στα flags του gcc που χρειάζονται για την σωστή χρήση του gdb. Πέρα από τον gdb διαθέσιμα είναι και άλλα βοηθητικά εργαλεία για προγραμματισμό συστήματος όπως π.χ το strace.

Ερώτημα 2ο: Επικοινωνία των guest και host μηχανημάτων μέσω σειριακής θύρας.

Το qemu μας παρέχει αρκετές δυνατότητες για την υποστήριξη σειριακής θύρας στο εικονικό μηχάνημα. Η προφανής επιλογή είναι να χρησιμοποιήσει το guest μηχάνημα την σειριακή θύρα του host μηχανήματος. Κατά πάσα πιθανότητα όμως το pc σας δεν διαθέτει σειριακή θύρα. Επομένως πρέπει να δημιουργήσουμε μια virtual, κάτι όμως που δεν είναι τόσο προφανές σε περιβάλλον linux.

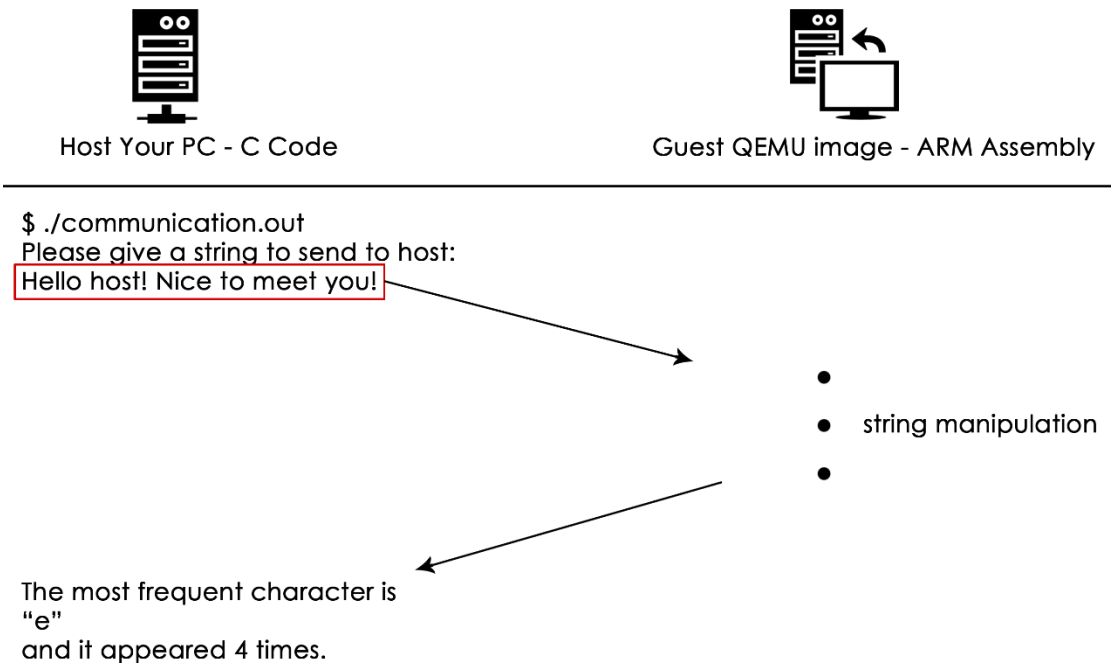
Η λύση είναι μια παροχή του linux που ονομάζεται pseudoterminal. Όλες τις ακριβείς πληροφορίες μπορείτε να τις βρείτε με man pts. Η κεντρική ιδέα είναι ότι κάνοντας open την συσκευή /dev/ptmx αυτόματα δημιουργείται και ένα αρχείο στο /dev/pts/ και τα 2 αυτά αρχεία μπορούν να επικοινωνούν γράφοντας και διαβάζοντας το ένα από το άλλο. Οι επιλογές αναφορικά με το qemu είναι 2:

i. Να τρέξετε το qemu-system-arm με όρισμα –serial pty. Το qemu θα κάνει τις απαραίτητες ενέργειες για την δημιουργία του αρχείου στο /dev/pts/ και θα σας γράψει στο command line ποιο ακριβώς αρχείο είναι αυτό (πχ /dev/pts/7). Αυτό το αρχείο θα χρησιμοποιήσετε στο host μηχάνημα ως το ένα άκρο της σειριακής. Το guest μηχάνημα θεωρεί ότι έχει μια σειριακή θύρα. Για να δείτε ποιες σειριακές θύρες έχουν φορτωθεί κατά την έναρξη του μηχανήματος πληκτρολογήστε dmesg | grep tty. Λογικά θα έχετε μερικές σειριακές με όνομα αρχείου ttyAMA* όπου * ένας αριθμός με την πρώτη από αυτές να είναι αυτή που θα χρησιμοποιήσετε.
ii. Να ανοίξετε με δικό σας πρόγραμμα το /dev/ptmx στο host μηχάνημα και να δώσετε σαν όρισμα στο qemu το αρχείο που δημιουργήθηκε. Και σε αυτή την περίπτωση με το –serial option. Πρέπει όμως να ακολουθήσετε με προσοχή τις οδηγίες που δίνονται στο man pts.

Κάθε μια από τις δύο λύσεις μπορεί να έχει θετικά και αρνητικά, επομένως είναι στην δική σας κρίση ποια θα χρησιμοποιήσετε. Σημειώνεται ότι υπάρχει περίπτωση ανάλογα με την έκδοση του qemu που τρέχετε, να υπάρχουν κάποια δεδομένα ήδη στην σειριακή, την πρώτη φόρα πού την ανοίγετε. Χρησιμοποιείστε όποιον τρόπο επιθυμείτε για να διαβάσετε και να αγνοήσετε τα δεδομένα αυτά, εφόσον υπάρχουν.

Σκοπός της άσκησης είναι να δημιουργηθούν 2 προγράμματα, ένα σε C στο host μηχάνημα και ένα σε assembly του ARM στο guest μηχάνημα τα οποία θα επικοινωνούν μέσω εικονικής σειριακής θύρας. Το πρόγραμμα στο host μηχάνημα θα δέχεται ως είσοδο έναν string μεγέθους έως 64 χαρακτήρων. Το string αυτό θα αποστέλλεται μέσω σειριακής θύρας στο guest μηχάνημα και αυτό με την σειρά του θα απαντάει ποιος είναι ο χαρακτήρας του string με την μεγαλύτερη συχνότητα εμφάνισης και πόσες φορές εμφανίστηκε. **Ο κενός χαρακτήρας (Ascii no 32) εξαιρείται από την καταμέτρηση.** Στην περίπτωση που δύο ή παραπάνω χαρακτήρες έχουν μέγιστη συχνότητα εμφάνισης, το πρόγραμμα να επιστρέφει στον host, τον χαρακτήρα με τον μικρότερο ascii κωδικό.

Στο παρακάτω σχήμα φαίνεται πως θα λειτουργούσαν τα δύο μηχανήματα σε μία πιθανή εκτέλεση:



Παραδοτέα της άσκησης είναι και τα δύο προγράμματα.

<u>Υποδείξεις:</u>

Για την ορθή λειτουργίας της εικονικής σειριακής θύρας τα βήματα που έχουν δοκιμαστεί και λειτουργούν στο εργαστήριο είναι τα παρακάτω:

1. Απενεργοποιείτε την εκτέλεση του προγράμματος getty που έχει σαν όρισμα την σειριακή θύρα ttyAMA0. Για να το πετύχετε. κάντε comment out την γραμμή

   T0:23:respawn:/sbin/getty -L ttyAMA0 9600 vt100

   στο αρχείο /etc/inittab (πιθανότατα γραμμή 63). Το comment out πραγματοποιείται προσθέτοντας το σύμβολο # στην αρχή της γραμμής. Υποχρεωτικά έπειτα πρέπει να κάνετε επανεκκίνηση του virtual machine.

2. Χρήση των βιβλιοθηκών την C για παραμετροποίηση της σειριακής θύρας. Οι συναρτήσεις αυτές περιέχονται στην βιβλιοθήκη termios.h και βασικός στόχος της παραμετροποίησης είναι να έχουν host και guest κοινό configuration σειριακής θύρας.

Στο παρακάτω κομμάτι κώδικα περιέχεται ο απλούστερος τρόπος να δηλώσετε στην assembly μια δομή struct termios που είναι απαραίτητη για την χρήση της παραπάνω βιβλιοθήκης. Έστω ότι η δομή termios έχει το όνομα options και όλα της τα στοιχεία γίνονται initialize σε τιμή 0:

```
.data
        options:        .word 0x00000000 /* c_iflag */
                        .word 0x00000000 /* c_oflag */
                        .word 0x00000000 /* c_cflag */
                        .word 0x00000000 /* c_lflag */
                        .byte 0x00                  /* c_line */
                        .word 0x00000000 /* c_cc[0-3] */
                        .word 0x00000000 /* c_cc[4-7] */
                        .word 0x00000000 /* c_cc[8-11] */
                        .word 0x00000000 /* c_cc[12-15] */
                        .word 0x00000000 /* c_cc[16-19] */
                        .word 0x00000000 /* c_cc[20-23] */
                        .word 0x00000000 /* c_cc[24-27] */
                        .word 0x00000000 /* c_cc[28-31] */
                        .byte 0x00                  /* padding */
                        .hword 0x0000   /* padding */
                        .word 0x00000000 /* c_ispeed */
                        .word 0x00000000 /* c_ospeed */
```

ειδικά για τον πίνακα c_cc όταν θέλετε να πειράξετε μία τιμή δώστε προσοχή στο endianess του εκτελέσιμου το οποίο γενικά είναι little endian και μπορείτε να το βρείτε και με readelf -h exec_name.

Οι τιμές των σταθερών βρίσκονται στο αρχείο

/usr/include/arm-linux-gnueabihf/bits/termios.h

του guest συστήματος σας. Προσοχή στην ανάγνωση του αρχείου καθώς οι σταθερές που ξεκινούν με 0 είναι δηλωμένες σε **οκταδικό** σύστημα αρίθμησης.

Ένας επιπρόσθετος τρόπος για τον προσδιορισμό της τιμής μιας σταθεράς για την χρήση της σε κώδικα assembly είναι με χρήση του προγράμματος strace σε ένα εκτελέσιμο αρχείο. Για παράδειγμα αν έχετε στον κώδικα της C την εντολή

```
fdSP = open(argv[1], O_RDWR | O_NOCTTY | O_NDELAY);
```

και θέλετε να βρείτε την δεκαεξαδική τιμή της παράστασης O_RDWR | O_NOCTTY | O_NDELAY που έχει γίνει compile στο εκτελέσιμο tst_exec.out τότε αν εκτελέσετε της εντολή

```
strace -e trace=open ./tst_exec.out  /dev/pts/4
```

θα πάρετε την έξοδο

```
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/dev/pts/4", O_RDWR|O_NOCTTY|O_NONBLOCK) = -1 EACCES (Permission denied)
```

η εκτέλεση της εντολής

```
strace -e trace=open -e raw=open ./tst_exec.out /dev/pts/4
```

δίνει την έξοδο

```
open(0x1e50a1, 0x80000, 0)            = 0x3
open(0xb774606d, 0x80000, 0x1977e)    = 0x3
open(0xbfab24e5, 0x902, 0x1)          = -1 (errno 13)
```

και βλέπουμε ότι η παράσταση O_RDWR|O_NOCTTY|O_NONBLOCK έχει τιμή 0x902. Το έξτρα argument που βλέπετε στο open είναι λόγω του default mode_t value που δίνει η glibc στο open με 3 ορίσματα. Η συμπεριφορά αυτή δεν συναντάται σε όλα τα system calls.

Γενικότερα για την άσκηση, δεν είναι στόχος η γρηγορότερη επικοινωνία και ταχύτερη απόκριση των προγραμμάτων αλλά η σωστή ανταλλαγή δεδομένων μεταξύ τους. Χρησιμοποιήστε όποιον συνδυασμό blocking / non-blocking επικοινωνίας, baudrate, καθυστέρησης μεταξύ διαδοχικών αποστολών κλπ.

Ερώτημα 3ο: Σύνδεση κώδικα C με κώδικα assembly του επεξεργαστή ARM.

Σκοπός της παρούσας άσκησης είναι να συνδυαστεί κώδικας γραμμένος σε C με συναρτήσεις γραμμένες σε assembly του επεξεργαστή ARM. Στον σύνδεσμο Έγγραφα→Ασκηση_2→ Άσκηση βρίσκεται ένα αρχείο που ονομάζεται string_manipulation.c. Το πρόγραμμα αυτό, ανοίγει ένα αρχείο με 512 γραμμές, κάθε γραμμή του οποίου περιέχει μια τυχαία κατασκευασμένη συμβολοσειρά, μεγέθους από 8 έως 64 χαρακτήρες. Κατά την εκτέλεση του προγράμματος κατασκευάζονται 3 αρχεία εξόδου:

1. Το πρώτο περιέχει το μήκος της κάθε γραμμής του αρχείου εισόδου.
2. Το δεύτερο περιέχει τις συμβολοσειρές του αρχείου εισόδου ενωμένες (concatenated) ανά 2.
3. Το τρίτο περιέχει τις συμβολοσειρές του αρχείου εισόδου ταξινομημένες σε αύξουσα αλφαβητική σειρά.

Για να επιτευχθούν οι παραπάνω στόχοι γίνεται χρήση των συναρτήσεων **strlen**, **strcpy**, **strcat** και **strcmp** από την βιβλιοθήκη string.h. Στόχος σας είναι να αντικαταστήσετε τις παραπάνω συναρτήσεις με δικές σας γραμμένες σε ARM assembly.

Για την σύνδεση assembly και γλώσσας C υπάρχουν δυο διαθέσιμοι τρόποι:

i. Να γραφούν συναρτήσεις τις C, που θα περιέχουν inline assembly εντολές κάνοντας χρήση της εντολής asm. Για παράδειγμα:

```
asm("MOV r0, #5");
__asm__("MOV r1, #10");
```

Όπως βλέπουμε μπορούμε να χρησιμοποιήσουμε τόσο την συνάρτηση asm() όσο και την __asm__(), σε περίπτωση που έχουμε δηλώσει κάποια μεταβλητή ως asm.

ii. Οι συναρτήσεις που θέλουμε να υλοποιήσουμε, δηλώνονται ως extern στον κώδικα της C. Έστω ότι έχουμε ένα αρχείο με πηγαίο κώδικα σε C το οποίο ονομάζεται my_prog.c και περιέχει μια συνάρτηση foo δηλωμένη ως extern. Ο πηγαίος κώδικας της συνάρτησης foo γράφεται σε ένα άλλο αρχείο που περιέχει κώδικα assembly και έστω ότι ονομάζεται my_fun.s. Απαραίτητο είναι η foo να έχει δηλωθεί στο κώδικα assembly με το directive .global για να μπορεί να είναι ορατή από τον linker κατά την σύνδεση των δυο αρχείων. Για να παραχθεί το τελικό εκτελέσιμο αρχείο, ακολουθούμε τα παρακάτω βήματα:

- Κάνουμε μόνο compile (-c flag του gcc) το αρχείο my_prog.c.

```
$ gcc –Wall –g my_prog.c –c my_prog
```

- Κάνουμε μόνο compile το αρχείο my_fun.s

```
$ gcc –Wall –g my_fun.s –c my_fun
```

- Συνδέουμε (link) τα object αρχεία που έχουν παραχθεί από τα παραπάνω βήματα, για την παραγωγή του τελικού εκτελέσιμου αρχείου.

Στα πλαίσια της άσκησης θα χρησιμοποιήσουμε αυστηρά τον τρόπο 2. Συνοψίζοντας, δεν χρειάζεται να αλλάξετε κάτι στον κώδικα C που συνοδεύει την άσκηση, πέρα από το να δηλώσετε τις συναρτήσεις ως extern και να σβήσετε την εντολή #include <string.h>. Οι δηλώσεις των συναρτήσεων **strlen**, **strcpy**, **strcat** και **strcmp** πρέπει να είναι σε μορφή ακριβώς ίδια με αυτή που προδιαγράφεται όταν εκτελέσετε στο τερματικό σας την εντολή man string.

Παραδοτέος κώδικας της άσκησης θα είναι το αρχείο ή αρχεία που περιέχουν τον πηγαίο κώδικα των συναρτήσεων σας σε assembly και ένα makefile για την σωστή μεταγλώττιση και σύνδεση των επιμέρους αρχείων. Το παραγόμενο εκτελέσιμο πρέπει να έχει όνομα **string_manipulation.out**. Στον φάκελο της άσκησης υπάρχουν επίσης δυο example input αρχεία με ονόματα rand_str_input_first και rand_str_input_sec.

Υποσημείωση: Ο gcc εξ ορισμού παράγει όταν μπορεί κώδικα για το Thumb instruction set ώστε να μειώσει το μέγεθος του παραγόμενου εκτελέσιμου αρχείου. Για τον λόγο αυτό προτείνεται να ξεκινάτε την παραγόμενη συνάρτηση σας με τουλάχιστον τα παρακάτω directives:

```
.text
.align  4          /* allignment του κώδικα
.global foo        /* όνομα συνάρτησης */
.type   foo, %function
```

Φυσικά αν θέλετε και εσείς να χρησιμοποιήσετε το Thumb instruction set, είστε ελεύθεροι .Προσοχή όμως στα directives προς τον assembler που θα χρησιμοποιήσετε.

Γενικές παρατηρήσεις:

1. Οι κώδικες σας, να συνοδεύονται από έναν στοιχειώδη σχολιασμό στα βασικά τους σημεία.
2. Θα ληφθεί υπόψη η σωστή χρήση του instruction set του ARM, για παραγωγή κώδικα μειωμένου μεγέθους. Η μη χρήση αυτών των χαρακτηριστικών θεωρείται μη αποδοτική επίλυση και δεν θα βαθμολογηθεί με τον μέγιστο βαθμό.
3. Οι κώδικες πρέπει να συνοδεύονται από μια αναφορά που θα συνοψίζει σε λίγες γραμμές το σκεπτικό πίσω από την υλοποίηση του κάθε προγράμματος σας.
4. Η παράδοση της άσκησης θα γίνει είτε ατομικά είτε σε ομάδες των δύο ατόμων.