Σχεδιασμός Ενσωματωμένων Συστημάτων

3η Εργαστηριακή Άσκηση

Εργασία σε assembly του επεξεργαστή ARM

Νικολέτα-Μαρκέλα Ηλιακοπούλου
03116111

Φοίβος-Ευστράτιος Καλεμκερής
03116010

1 Μετατροπή εισόδου από τερματικό

1.1 Εισαγωγή

Στο πρώτο μέρος της άσκησης στόχος υπήρξε η ανάπτυξη προγράμματος σε assembly του επεξεργαστή ARM, το οποίο λαμβάνει ως είσοδο από το χρήστη μια συμβολοσειρά 32 χαρακτήρων και τη μετασχηματίζει σύμφωνα με τα παρακάτω:

- Αν η είσοδος είναι γράμμα του λατινικού αλφαβήτου:
  - Μετατρέπει τα πεζά γράμματα σε κεφαλαία.
  - Μετατρέπει τα κεφαλαία σε πεζά.
- Αν ο χαρακτήρας βρίσκεται στο εύρος [‘0’, ‘9’]:
  - Αν η είσοδος είναι μικρότερη του 5, προσθέτει 5.
  - Αν η είσοδος είναι μεγαλύτερη ή ίση του 5, αφαιρεί 5.
- Οι υπόλοιποι χαρακτήρες παραμένουν αμετάβλητοι.

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

1.2 Υλοποίηση

Η υλοποίηση της ζητούμενης συνάρτησης αποτελείται από 4 μέρη. Το πρώτο αφορά στη λήψη της εισόδου από το χρήστη. Αρχικά, εμφανίζουμε με χρήση της write system call (r7=4) προτροπή προς το χρήστη, ώστε να δώσει την προς μετασχηματισμό συμβολοσειρά. Στη συνέχεια, χρησιμοποιώντας τη read system call (r7=3), διαβάζουμε 33 χαρακτήρες (33 bytes) απ’ το stdin, δηλαδή τους 32 που επιτρέπουμε στο χρήστη να δώσει, συν το χαρακτήρα αλλαγής γραμμής. Αν η συμβολοσειρά που δόθηκε είναι μήκους δύο, τότε αυτή μπορεί να είναι το μήνυμα τερματισμού του προγράμματος (’q’ + ’endl’). Ελέγχουμε τον πρώτο χαρακτήρα και αν είναι ίσος με ’Q’ ή ’q’ τερματίζουμε. Διαφορετικά προχωράμε στο κύριο μέρος του προγράμματος που είναι η μετατροπή.
Το δεύτερο μέρος αποτελείται από τον βρόχο converter. Σε κάθε επανάληψη αυτού, διαβάζουμε έναν χαρακτήρα από τη διεύθυνση στην οποία αποθηκεύτηκε η συμβολοσειρά εισόδου (βρίσκεται στον r1) και ελέγχουμε αν χρειάζεται να πραγματοποιήσουμε οποιονδήποτε μετασχηματισμό. Έτσι:

- Αν ο χαρακτήρας βρίσκεται στο διάστημα [’a’, ’z’], τότε αφαιρούμε από τον ascii κωδικό του 32, ώστε να τον μετατρέψουμε στο αντίστοιχο κεφαλαίο και αποθηκεύουμε το αποτέλεσμα στη θέση μνήμης που βρισκόταν ο αρχικός χαρακτήρας.
- Αν ο χαρακτήρας βρίσκεται στο διάστημα [’A’, ’Z’], τότε προσθέτουμε στον ascii κωδικό του 32, ώστε να τον μετατρέψουμε στο αντίστοιχο πεζό και αποθηκεύουμε το αποτέλεσμα στη θέση μνήμης που βρισκόταν ο αρχικός χαρακτήρας.
- Αν ο χαρακτήρας βρίσκεται στο διάστημα [’0’, ’9’], τον συγκρίνουμε με το 5 και, για τους μεν μικρότερους προσθέτουμε στον ascii κωδικό το 5, για τους υπόλοιπους, δε, αφαιρούμε το 5 και αποθηκεύουμε το αποτέλεσμα στη θέση μνήμης που βρισκόταν ο αρχικός χαρακτήρας.

Η επανάληψη τερματίζει όταν εντοπιστεί ο χαρακτήρας αλλαγής γραμμής (κωδικός ascii 10) για την περίπτωση που η συμβολοσειρά εισόδου περιείχε λιγότερους από 32 χαρακτήρες ή όταν μετρήσουμε 33 χαρακτήρες. Μπορούμε τότε να προχωρήσουμε στο επόμενο μέρος της συνάρτησης που πραγματοποιεί την εκτύπωση της μετασχηματισμένης συμβολοσειράς.
Στο σημείο αυτό, αν ο χρήστης είχε δώσει λιγότερους από 32 χαρακτήρες, μπορούμε απλώς με μια write system call να τυπώσουμε στο stdout αριθμό χαρακτήρων ίσο με r4 (ο μετρητής που χρησιμοποιήσαμε διατρέχοντας τη συμβολοσειρά εισόδου) που ξεκινούν στη διεύθυνση της αρχικής συμβολοσειράς, αφού οι μετασχηματισμοί πραγματοποιήθηκαν με απευθείας αντικαταστάσεις χαρακτήρων σ’ αυτή, χωρίς να απαιτείται οποιαδήποτε επιπλέον ενέργεια.

Σε περίπτωση, ωστόσο, που ο μετρητής έχει φτάσει στο 33, θα πρέπει να βεβαιωθούμε, αφενός, ότι ο τελευταίος χαρακτήρας είναι η αλλαγή γραμμής και, αφετέρου, ότι ο buffer είναι κενός, αφού θα μπορούσε ο χρήστης να έχει δώσει περισσότερους από 32 χαρακτήρες, οι οποίοι αγνοήθηκαν κατά τη μετατροπή, αλλά παραμένουν στον input buffer, δημιουργώντας πρόβλημα στην επόμενη επανάληψη του προγράμματος. Ως εκ τούτου, τοποθετούμε αρχικά στην 33η θέση της συμβολοσειράς τον χαρακτήρα αλλαγής γραμμής και, αφού τυπώσουμε τη μετασχηματισμένη συμβολοσειρά καθαρίζουμε τον buffer (flush), διαβάζοντας (read system call) έναν πολύ μεγάλο αριθμό χαρακτήρων (2^{30}). Ύστερα, είμαστε έτοιμοι να συνεχίσουμε την επόμενη επανάληψη του προγράμματος.

Η συνάρτηση main που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στο αρχείο ex1.s και δίνεται παρακάτω.

```
.text
.syntax unified
.align 4
.global main
.type main, %function

main:
    push {ip,lr}
loop:
    mov r4, #0      @ initialise counter
    mov r5, #10     @ newline character

    mov r0, #1      @ stdout
    ldr r1, =prompt @ location of prompt in memory
    mov r2, #len    @ length of prompt
    mov r7, #4      @ write system call
    swi 0

    mov r0, #0      @ stdin
    ldr r1, =inp_str @ location of inp_str in memory
    mov r2, #33     @ read 32 characters + \n from input
    mov r7, #3      @ read system call
    swi 0

    @ Exit?
    cmp r0, #2      @ Could it be 'q\n' or 'Q\n'?
    bne converter
    ldrb r3, [r1]
    cmp r3, #81     @ Is it Q?
    beq exit
    cmp r3, #113    @ Is it q?
    beq exit

converter:
    ldrb r3, [r1]
    add r4, #1      @ increase counter
    cmp r3, #10     @ Is the character '\n'?
    beq print
    cmp r3, #97     @ Is it >= a?
    blt next1
    cmp r3, #122    @ Is it <= z?
    ble to_capital
next1:
    cmp r3, #65     @ Is it >= A?
    blt next2
    cmp r3, #90     @ Is it <= Z?
    ble to_lower
next2:
    cmp r3, #48     @ Is it >= 0?
    blt next3
    cmp r3, #57     @ Is it <= 9?
    ble conv_number
next3:
    strb r3, [r1], #1
final:
    cmp r4, #33
```

```
    beq print
    b converter

to_capital:
    sub r3, r3, #32
    strb r3, [r1], #1
    b final

to_lower:
    add r3, r3, #32
    strb r3, [r1], #1
    b final

conv_number:
    cmp r3, #53
    addlt r3, #5
    subge r3, #5
    strb r3, [r1], #1
    b final

print:
    cmp r4, #33
    strbeq r5, [r1, #-1]    @if 33rd character != '\n' make it '\n'
    mov r0, #1      @ stdout
    ldr r1, =inp_str    @ location of string in memory
    mov r2, r4      @ length string
    mov r7, #4      @ write system call
    swi 0
    cmp r4, #33
    beq flush
    b loop

@ flush buffer
flush:
    mov r7, #3      @ read system call
    mov r0, #2
    ldr r1,=inp_str
    mov r2, #(1 << 30) @ flush
    swi 0
  b loop

exit:
    pop {ip,pc}
    mov r7, #1      @ exit system call
    swi 0

.data
  prompt: .ascii "Please input a string 32 character long: "
  len = . - prompt @ length of prompt is the current memory indicated by (.) minus the
                        memory location of the first element of string

@ pre-allocate 32 bytes for input string, initialize them with null character '\0'
  inp_str: .ascii "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
  inplen = . - inp_str

```

Το πρόγραμμα μεταγλωττίζεται με χρήση του ακόλουθου εξαιρετικά απλού Makefile.

```
ex1: ex1.s
    gcc -Wall ex1.s -o ex1.out

clean:
    rm -f *.out
```

Στη συνέχεια το τρέχουμε όπως φαίνεται στη συνέχεια:

```
$ ./ex1.out
```

2 Επικοινωνία guest - host μηχανημάτων μέσω σειριακής θύρας

2.1 Εισαγωγή

Σκοπός του δεύτερου μέρους 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 δέχεται ως είσοδο έναν string μεγέθους έως 64 χαρακτήρων. Το string αυτό αποστέλλεται μέσω σειριακής θύρας στο guest μηχάνημα και αυτό υπολογίζει και απαντάει ποιος είναι ο χαρακτήρας του string με την μεγαλύτερη συχνότητα εμφάνισης και πόσες φορές εμφανίστηκε. Στην περίπτωση που δύο ή παραπάνω χαρακτήρες έχουν μέγιστη συχνότητα εμφάνισης, το πρόγραμμα επιστρέφει στον host, τον χαρακτήρα με τον μικρότερο ascii κωδικό (εξαιρείται ο κενός χαρακτήρας).

2.2 Υλοποίηση

Αρχικά, τρέχουμε το qemu με όρισμα -serial pty για τη δημιουργία του αρχείου της εικονικής σειριακής θύρας στο /dev/pts/x. Μέσω αυτού στον host και του /dev/ttyAMA0 ως το άλλο άκρο της σειριακής θύρας, θα γίνει η επικοινωνία, για την πραγματοποίηση της οποίας, χρησιμοποιήθηκε η βιβλιοθήκη termios. Ανοίγουμε τη σειριακή θύρα στον host και τον guest και μέσω της βιβλιοθήκης, ρυθμίζουμε τα configurations ούτως ώστε να πετύχουμε τον επιθυμητό τρόπο επικοινωνίας, προσέχοντας να είναι κοινά και στους δύο.

2.2.1 Host

1. Αρχικά, ελέγχουμε ότι γίνεται σωστή εκτέλεση του αρχείου και διαβάζουμε από το stdin τη συμβολοσειρά που θα στείλουμε στον guest. Πρέπει να έχει μήκος μέχρι 64 bytes (65 bytes αν λάβουμε υπόψη μας και τον ειδικό χαρακτήρα FL).

```
    int fd, len;
    char input[BUFF_SIZE], output[BUFF_SIZE];
    ssize_t rcnt;
    struct termios config;

    if (argc != 2) {
      fprintf(stderr, "Usage: ./host device\n \tdevice: serial port destination (
    example: /dev/pts/7)\n");
      exit(1);
    }

    printf("Please give a string to send to guest:\n");
    rcnt = read(0, input, BUFF_SIZE);
    if (rcnt<0){
      perror("read");
      exit(1);
    }
    len = rcnt;
    if (len == 66) {
      fprintf(stderr, "Host: string is more than 64 bytes!\n");
      fflush(0);
      exit(1);
    }
    else
      printf("Host: string is %d bytes. Valid!\n", len);
```

2. Έπειτα, ανοίγουμε το ειδικό αρχείο για την εικονική σειριακή θύρα. Θέτουμε τα flags O_RDWR και O_NOCTTY, ώστε να μπορούμε να γράψουμε και να διαβάσουμε στη θύρα και αυτό χωρίς να είναι η συσκευή μας το terminal που ελέγχει τη διεργασία γιατί αυτό θα είχε ανεπιθύμητα αποτελέσματα. Θέτουμε τα flags του termios struct με τις τιμές που θέλουμε και, έπειτα, ενημερώνουμε τη συσκευή με τις ρυθμίσεις που επιλέξαμε.

Πιο αναλυτικά, επιλέγουμε να θέσουμε τη σειριακή θύρα σε canonical mode. Αυτό μας εξυπηρετεί αφού δίνουμε την είσοδο ως μια γραμμή με τελικό χαρακτήρα το EOL και το διάβασμα από αυτή τελειώνει εκεί. Χρησιμοποιούμε την 8-N-1 ρύθμιση, κάτι το οποίο συνηθίζεται στη σειριακή επικοινωνία. Υπάρχει 1 start bit, 8 data bits, κανένα parity bit και 1 stop bit. Λόγω της ρύθμισης αυτής θέσαμε και το BAUDRATE σε B9600.

Τέλος με την εντολή tcflush με όρισμα TCIOFLUSH καθαρίζουμε τη θύρα από δεδομένα που έχουν ληφθεί αλλά δεν έχουν διαβαστεί και από δεδομένα που έχουν γραφτεί αλλά δεν έχουν μεταδοθεί.

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

    if (fd == -1) {
        printf("Failed to open port\n");
        return 1;
    }



    if(tcgetattr(fd, &config) < 0) {
        printf("Couln't get the data of the terminal\n");
        return 1;
    }


    config.c_lflag = 0;
    config.c_lflag |= ICANON; /* Make port canonical */
    config.c_cflag |= (CLOCAL | CREAD | CS8);
    config.c_cflag &= ~CSTOPB;
    config.c_cc[VMIN] = 1; /* bogus */
    config.c_cc[VTIME] = 0; /* bogus */


   /* Set baud speed */
    if(cfsetispeed(&config, B9600) < 0 || cfsetospeed(&config, B9600) < 0) {
        printf("Problem with baudrate\n");
        return 1;
    }

    /* Finally, apply the configuration */
    if(tcsetattr(fd, TCSANOW, &config) < 0) {
        printf("Couldn't apply settings\n");
        return 1;
    }

    tcflush(fd, TCIOFLUSH); /* Clear everything that is in the terminal*/
```

3. Μετά την ολοκλήρωση των ρυθμίσεων της σειριακής επικοινωνίας, γράφουμε τη συμβολοσειρά που πήραμε από την είσοδο στη σειριακή θύρα και περιμένουμε να διαβάσουμε από αυτή τα αποτελέσματα. Κάνουμε χρήση blocking read και write προς διευκόλυνση μας. Ο guest επεξεργάζεται τη συμβολοσειρά με τον τρόπο που θα δούμε παρακάτω και μας επιστρέφει τον χαρακτήρα που εμφανίζεται περισσότερες φορές στη συμβολοσειρά, καθώς και τη συχνότητα εμφάνισής του αυξημένη κατα 48. Πρόκειται για τον ascii κωδικό του χαρακτήρα 0 με τον οποίο είχαμε αρχικοποιήσει τον πίνακα συχνοτήτων μας στον guest. Κάνουμε τις απαραίτητες τροποποιήσεις και εκτυπώνουμε τα αποτελέσματα.

   Σημείωση: Την τροποποίηση την κάνουμε στον host και όχι στον guest καθώς η μεταφορά byte με πολύ μικρες τιμές (πχ 3, 4) στη σειριακή θύρα μπορεί να έχει ανεπιθύμητα αποτελέσματα.

```
    printf("Sending to guest...\n");

    write(fd, input, BUFF_SIZE);

    printf("Reading to guest...\n");

    read(fd, output, BUFF_SIZE);
```

```
8
9       int i, final;
10
11      final = output[2];
12
13      final=final-'0';
14
15
16      if(final!=0)
17        printf("The most frequent character is %c and it appeared %d times.\n", output
        [0], final);
18      else
19        printf("No characters were given\n");
20
21      close(fd);
22      return 0;
23  }
```

2.2.2 Guest

Από την πλευρά του guest μηχανήματος, στόχος είναι η ανάγνωση της συμβολοσειράς που μεταφέρθηκε μέσω της σειριακής θύρας και το μέτρημα στη συνέχεια της συχνότητας των χαρακτήρων, ώστε να επιστραφεί στον host ο συχνότερος χαρακτήρας, μαζί με το συνολικό αριθμό εμφανίσεών του. Εξαίρεση αποτελεί ο χαρακτήρας του κενού που δεν προσμετράται.

Αρχικά, ανοίγουμε το αρχείο που αντιστοιχεί στη σειριακή θύρα και με τη χρήση της tcsetattr, όπως και στο πρόγραμμα host, εφαρμόζουμε τις ρυθμίσεις της θύρας, οι οποίες έχουν αποθηκευτεί στη δομή options. Ύστερα, ξεκινάμε την ανάγνωση από το αρχείο της σειριακής θύρας. Ειδικότερα, θέλουμε να διαβάσουμε 64 χαρακτήρες, συν έναν για τον χαρακτήρα τέλους γραμμής, τους οποίους αποθηκεύουμε στη μνήμη στη δομή string. Εκτός από την προηγούμενη δομή, η διεύθυνση της οποίας αποθηκεύεται στον r0 για τη συνέχεια του προγράμματος, θα χρησιμοποιήσουμε και τη δομή arr- η διεύθυνση του οποίου βρίσκεται στον r3- στον οποίο θα αποθηκεύουμε τη συχνότητα εμφάνισης κάθε χαρακτήρα. Ο arr είναι αρχικοποιημένος στο 0 και έχει μέγεθος 255, όσοι και οι extended ascii χαρακτήρες.
Ο κορμός του προγράμματος αποτελείται από δύο μέρη:

1. Την ανάγνωση των χαρακτήρων της συμβολοσειράς και τη μέτρηση της συχνότητας εμφάνισής τους.

2. Την εύρεση του χαρακτήρα με τη μεγαλύτερη συχνότητα εμφάνισης ή αν υπάρχουν περισσότεροι με την ίδια συχνότητα, αυτού που έχει και τον μικρότερο κωδικό ascii.

Για το πρώτο μέρος, διατρέχουμε τη συμβολοσειρά διαβάζοντας έναν έναν τους χαρακτήρες μέχρι να συναντήσουμε το χαρακτήρα αλλαγής γραμμής, η εύρεση του οποίου θα σημάνει το τέλος του πρώτου μέρους. Οι χαρακτήρες που διαβάζονται απ' τη μνήμη αποθηκεύονται στον καταχωρητή r2. Στη συνέχεια, διαβάζουμε από τη θέση r2 του arr το πλήθος των μέχρι στιγμής εμφανίσεων που αντιστοιχεί στο συγκεκριμένο χαρακτήρα, τον αυξάνουμε κατά 1 και τον αποθηκεύουμε στην ίδια θέση στη μνήμη.

Μόλις ολοκληρωθεί το διάβασμα της συμβολοσειράς και, συνεπώς, ο arr έχει ενημερωθεί ώστε να περιλαμβάνει για κάθε χαρακτήρα ascii τον αριθμό εμφανίσεών του, προχωράμε στο δεύτερο μέρος της επεξεργασίας προκειμένου να εντοπίσουμε τη θέση του arr που περιέχει τον μεγαλύτερο αριθμό. Πρώτα, όμως, φροντίζουμε να μηδενίσουμε τη θέση του πίνακα που αντιστοιχεί στον χαρακτήρα του κενού, αφού ο αριθμός των εμφανίσεών του δε μας ενδιαφέρει. Αρχικοποιούμε τον r0 στο 0, ώστε να αποθηκεύουμε σ' αυτόν τη μεγαλύτερη συχνότητα. Για κάθε συχνότητα χαρακτήρα που διαβάζουμε από τον arr ελέγχουμε αν είναι μεγαλύτερη από τη συχνότητα που έχουμε αποθηκεύσει στον r0 και αναλόγως ενημερώνουμε τον r0 με το νέο μέγιστο. Κρατάμε, δε, την τιμή του χαρακτήρα, στον οποίο αυτό αντιστοιχεί, στον r4. Επαναλαμβάνουμε τη διαδικασία μέχρι να φτάσουμε στο τέλος του arr. Όταν συμβεί αυτό αποθηκεύουμε το συχνότερο χαρακτήρα όπως προέκυψε στη θέση 0 της δομής res και το πλήθος εμφανίσεων στη θέση 2.

Τέλος, γράφουμε το αποτέλεσμα (res) πίσω στη σειριακή θύρα, κλείνουμε το αρχείο που της αντιστοιχεί και τερματίζουμε το πρόγραμμα.

```
.text
.align 4 /* code alignment */
.global main
.extern tcsetattr

main:

open:
    ldr r0, =path           /* device to open */
    ldr r1, =#258           /* 0x102 to decimal */
    mov r7, #5              /* open syscall */
    swi 0

    /* now r0 has the fd */
    mov r6, r0              /* save for later */

set_device:
    mov r0, r6              /* call tcsetattr to set the settings for our port */
    ldr r2, =options
    mov r1, #0
    bl tcsetattr

read:
    mov r0, r6              /* read from serial port */
    ldr r1, =string
    mov r2, #64
    mov r7, #3
    swi 0

    ldr r0, =string

    str_manip:
        ldr r3, =arr
        mov r1, #0

    str_manip_loop:
        ldrb r2, [r0], #1   /* load character and move on to the next address  */
        cmp r2, #10         /* since we have canonical input, the last useful char will be
                               EOL */
        beq str_manip_end
        ldrb r4, [r3, r2]   /* loads the value in the array with offset r2 (character that
                               we read) */
        add r4, r4, #1      /* increment the times seen the character */
        strb r4, [r3, r2]   /* save the value */
        b str_manip_loop

    str_manip_end:
        mov r1, #0          /* offset pointer*/
        mov r0, #0          /* max freq in r0 */
        ldr r3, =arr        /* load array of frequencies to r3 */
        strb r0, [r3, #32]  /* we don't care about spaces */

    str_manip_end_loop:
        ldrb r2, [r3, r1]   /* load value of array with offset r1 */
        cmp r0, r2          /* maximum freq in r0 */
        movlt r4, r1        /* if r2>r0 save the char */
        movlt r0, r2        /* change max */
        add r1, r1, #1      /* increase offset for next iteration */
        cmp r1, #256        /* if end of the array exit */
        beq str_manip_exit
        b str_manip_end_loop

    str_manip_exit:

    /* Now r0 holds the number of times char showed up  */
    /* r4 holds the character */
        ldr r3, =res
```

```
        strb r4, [r3]
        strb r0, [r3, #2]

write:
    mov r0, r6
    ldr r1, =res
    ldr r2, =len_res
    mov r7, #4          /* write system call */
    swi 0

close:
    mov r0, r6
    mov r7, #6          /* close system call */
    swi 0

exit:
    mov r0, #0
    mov r7, #1          /* exit system call */
    swi 0

.data
    options: .word 0x00000000 /* c_iflag */
             .word 0x00000000 /* c_oflag */
             .word 0x000008bd /* c_cflag */
             .word 0x00000002 /* c_lflag */
             .byte 0x00       /* c_line */
             .word 0x00000000 /* c_cc[0-3] */
             .word 0x00010000 /* 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 0x0000000d /* c_ispeed */
             .word 0x0000000d /* c_ospeed */

    path: .asciz "/dev/ttyAMA0"

    string: .asciz "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
                    \0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
    len = . - string

    arr: .ascii "00000000000000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000"

    res: .asciz "cccc\n"

    len_res = . - res
```

Για την εκτέλεση των 2 προγραμμάτων θα πρέπει να μεταφέρουμε αρχικά τον κώδικα για τη μετατροπή της συμβολοσειράς στο guest μηχάνημα και να το μεταγλωττίσουμε (*make*). Ύστερα, το εκκινούμε όπως φαίνεται παρακάτω:

```
$ ./guest
```

Αντίστοιχα, στο host μηχάνημα, μεταγλωττίζουμε το πρόγραμμα με make και στη συνέχεια το τρέχουμε παρέχοντας ως είσοδο το αρχείο της εικονικής σειριακής θύρας που δημιουργήθηκε, όπως φαίνεται παρακάτω.

```
$ sudo ./host /dev/pts/x
```

όπου x ο αριθμός της θύρας που εμφανίστηκε στο terminal.

3 Σύνδεση κώδικα C με κώδικα assembly του επεξεργαστή ARM

3.1 Εισαγωγή

Σκοπός του τελευταίου μέρους της άσκησης ήταν η σύνδεση κώδικα C με συναρτήσεις γραμμένες σε assembly του ARM. Ειδικότερα, κληθήκαμε να υλοποιήσουμε σε ARM assembly τις συναρτήσεις strlen, strcpy, strcmp και strcat της βιβλιοθήκης string.h της C, οι οποίες χρησιμοποιήθηκαν στη συνέχεια σε ένα πρόγραμμα μετασχηματισμού συμβολοσειρών (*string_manipulation.c*), το οποίο προσαρμόστηκε, ώστε αντί για τις υλοποιήσεις της string.h να καλεί τις δικές μας συναρτήσεις. Οι συναρτήσεις υλοποιήθηκαν σύμφωνα με τις προδιαγραφές που ορίζονται στο documentation της προαναφερθείσας βιβλιοθήκης. Πιο συγκεκριμένα:

- **size_t strlen(const char *s);**
  Επιστρέφει των αριθμό των bytes (αριθμός χαρακτήρων) που περιέχονται στη συμβολοσειρά στην οποία δείχνει το s, χωρίς να υπολογίζει τον τερματικό χαρακτήρα.
- **char *strcpy(char *s1, const char *s2);**
  Αντιγράφει τους χαρακτήρες της συμβολοσειράς στην οποία δείχνει το s2 (συμπεριλαμβανομένου του τερματικού χαρακτήρα) στη θέση που δείχνει το s1.
- **int strcmp(const char *s1, const char *s2);**
  Συγκρίνει τη συμβολοσειρά στην οποία δείχνει το s1 με αυτή στην οποία δείχνει το s2. Επιστρέφει 1 αν η πρώτη είναι λεξικογραφικά μεγαλύτερη, -1 αν είναι μικρότερη και 0 αν είναι ίσες.
- **char *strcat(char *s1, const char *s2);**
  Αντιγράφει τους χαρακτήρες της συμβολοσειράς στην οποία δείχνει το s2 (συμπεριλαμβανομένου του τερματικού χαρακτήρα) στο τέλος της συμβολοσειράς στην οποία δείχνει το s1.

3.2 Υλοποίηση

Δημιουργήσαμε 4 ξεχωριστά αρχεία, ένα για κάθε μία εκ των συναρτήσεων.

- **strlen.s**
  Η συγκεκριμένη συνάρτηση δέχεται ως είσοδο στον καταχωρητή r0 τη διεύθυνση μιας συμβολοσειράς και επιστρέφει μέσω του ίδιου καταχωρητή το μήκος της. Στην υλοποίηση χρησιμοποιήσαμε τον καταχωρητή r2 ως μετρητή (αρχικοποιημένο στο 0). Βάση αυτής είναι ο βρόχος loop.
  Σε κάθε επανάληψή του, φορτώνουμε από τη θέση μνήμης στην οποία δείχνει ο r0 έναν χαρακτήρα (1 byte) και αυξάνουμε τον r0 κατά 1, προκειμένου να δείχνει στην επόμενη θέση μνήμης. Αν ο χαρακτήρας που διαβάσαμε είναι το τερματικό σύμβολο, η συνάρτηση επιστρέφει. Διαφορετικά, η τιμή του μετρητή αυξάνεται κατά 1 και συνεχίζουμε με την επόμενη επανάληψη, ώστε να διαβάσουμε τον επόμενο χαρακτήρα της συμβολοσειράς.

```
1  .text
2  .align 4
3  .global strlen
4  .type strlen, %function
5
6  @ r0: string address
7  strlen:
8      mov r2, #0      @ initialise counter
9  loop:
10     ldrb r1, [r0], #1   @ load string character and move on to the next address
11     cmp r1, #0      @ was this the end of the string?
12     beq exit
13     add r2, #1      @ if not, increase counter and continue
14     b loop
15 exit:
16     mov r0, r2      @ return length in r0
17     bx lr
```

- **strcpy.s**
  Η συγκεκριμένη συνάρτηση δέχεται ως είσοδο στον καταχωρητή r0 τη διεύθυνση μιας συμβολοσειράς s1 και στον r1 τη διεύθυνση μιας συμβολοσειράς s2, την οποία και θα αντιγράψει στη διεύθυνση που δείχνει το s1. Εδώ θα χρειαστεί να αποθηκεύσουμε τη διεύθυνση προορισμού προτού προχωρήσουμε στον βασικό πυρήνα της εκτέλεσης, ο οποίος είναι και πάλι ένας βρόχος.
  Σε κάθε επανάληψή του, διαβάζουμε ένα byte (έναν χαρακτήρα) από τη διεύθυνση που περιέχει ο r1, την οποία και αυξάνουμε στη συνέχεια κατά 1, για να προετοιμαστούμε για την επόμενη επανάληψη. Ο χαρακτήρας που διαβάστηκε αποθηκεύεται στη θέση μνήμης, η διεύθυνση της οποίας βρίσκεται στον r0. Στη συνέχεια, αυξάνουμε το διεύθυνση που περιέχει ο r0, για να προετοιμαστούμε για την επόμενη επανάληψη. Αν ο χαρακτήρας που διαβάστηκε ήταν ο τερματικός, η επανάληψη σταματάει. Διαφορετικά, συνεχίζουμε με την αντιγραφή του επόμενου χαρακτήρα.
  Είναι πολύ σημαντικό, αφού εντοπιστεί ο τερματικός χαρακτήρας του string s2, γεγονός που σημαίνει ότι αυτό έχει αντιγραφεί ολόκληρο στο s1, να επαναφέρουμε στον r0 τη διεύθυνση που περιείχε αρχικά, αφού σε αυτή περιμένει όποιος κάλεσε τη συνάρτηση να βρει το αντεγραμμένο string.

```
.text
.align 4
.global strcpy
.type strcpy, %function

@ r0: dest, r1: src
strcpy:
    mov r3, r0  @ store the dest address
loop:
    ldrb r2, [r1], #1   @ read character from string and move on to the next
    strb r2, [r0], #1   @ store the above read character
    cmp r2, #0      @ was this the end of the string?
    bne loop
exit:
    mov r0, r3  @ restore the initial address of the destination
    bx lr
```

- **strcmp.s**
  Η συγκεκριμένη συνάρτηση δέχεται ως είσοδο στον καταχωρητή r0 τη διεύθυνση μιας συμβολοσειράς s1 και στον r1 τη διεύθυνση μιας συμβολοσειράς s2, τις οποίες και συγκρίνει λεξικογραφικά, επιστρέφοντας τελικά, μέσω του r0, το αποτέλεσμα της σύγκρισης. Και πάλι ο πυρήνας της εκτέλεσης αποτελείται από έναν βρόχο για το διάβασμα των συμβολοσειρών.
  Σε κάθε επανάληψη αυτού, διαβάζουμε ένα byte (έναν χαρακτήρα) από τη διεύθυνση που περιέχει ο r0 και ένα από τη διεύθυνση που περιέχει ο r1, τις οποίες και αυξάνουμε στη συνέχεια κατά 1, για να προετοιμαστούμε για την επόμενη επανάληψη. Οι δύο χαρακτήρες που διαβάστηκαν συγκρίνονται μεταξύ τους. Σε περίπτωση που δεν είναι ίσοι, η επανάληψη σταματάει. Αν ο χαρακτήρας που διαβάστηκε από το s1 είναι μικρότερος αυτού απ' το s2, τότε έχουμε $s1 < s2$ και επομένως επιστρέφουμε αρνητική τιμή (συγκεκριμένα -1). Αν από την άλλη προκύψει ότι ο χαρακτήρας που διαβάστηκε από το s1 είναι μεγαλύτερος αυτού απ' το s2, τότε $s1 > s2$ και επομένως επιστρέφουμε θετική τιμή (συγκεκριμένα 1). Σε περίπτωση που φτάσουμε στο τέλος των συμβολοσειρών (εντοπίσουμε τον τερματικό χαρακτήρα) και όλοι οι χαρακτήρες έχουν βρεθεί ίσοι επιστρέφουμε 0 ($s1 == s2$), το οποίο έχουμε αποθηκεύσει εξ αρχής στον καταχωρητή r4.

```
.text
.align 4
.global strcmp
.type strcmp, %function

@ r0: str1, r1: str2
strcmp:
    mov r4, #0x0    @ if str1 == str2 return 0
loop:
    ldrb r2, [r0], #1   @ load str1 character
    ldrb r3, [r1], #1   @ load str2 character
    cmp r2, r3
    bne notEqual    @ if they are not equal we should determine which one comes first
```

```
    cmp r2, #0      @ was this the end of the strings?
    beq exit
    b loop
notEqual:
    movlt r4, #0xffffffff   @ if str1 < str2 return -1
    movgt r4, #0x1      @ if str1 > str2 return 1
exit:
    mov r0, r4  @ result in r0
    bx lr
```

- **strcat.s**
  Η τελευταία συνάρτηση δέχεται ως είσοδο στον καταχωρητή r0 τη διεύθυνση μιας συμβολοσειράς s1 και στον r1 τη διεύθυνση μιας συμβολοσειράς s2 και αντιγράφει τη δεύτερη στο τέλος της πρώτης. Όπως και για την strcpy, αποθηκεύουμε την αρχική διεύθυνση της s1, στον καταχωρητή r4.
  Η υλοποίηση της συνάρτησης χωρίζεται σε δύο μέρη. Αρχικά, διατρέχουμε τη συμβολοσειρά s1 μέχρι να φτάσουμε στον τερματικό χαρακτήρα. Ειδικότερα, όπως κάναμε και για τις υπόλοιπες συναρτήσεις, διαβάζουμε έναν χαρακτήρα τη φορά και προχωράμε στην επόμενη θέση μνήμης, μέχρι να εντοπίσουμε το τέλος της συμβολοσειράς. Όταν συμβεί αυτό, ελαττώνουμε τη διεύθυνση κατά 1, επειδή επιθυμούμε να αντικαταστήσουμε το τερματικό σύμβολο με τον πρώτο χαρακτήρα της s2. Ύστερα, ακολουθούμε διαδικασία όμοια με αυτή της strcpy, διαβάζοντας έναν έναν τους χαρακτήρες της s2 και αποθηκεύοντας τους στη διεύθυνση που δείχνει ο r0, η οποία αυξάνεται κατά 1 σε κάθε επανάληψη.
  Μόλις αντιληφθούμε ότι έχει αντιγραφεί και ο τερματικός χαρακτήρας της δεύτερης συμβολοσειράς βγαίνουμε από το βρόχο. Επαναφέρουμε, τέλος, τη διεύθυνση της συμβολοσειράς που κατασκευάσαμε, η οποία ταυτίζεται με τη διεύθυνση που αποθηκεύσαμε στην αρχή στον r4 και η συνάρτηση μπορεί πλέον να επιστρέψει.

```
.text
.align 4
.global strcat
.type strcat, %function

@ r0: dest, r1: src
strcat:
    mov r4, r0    @ store destination address
loop:
    ldrb r2, [r0], #1   @ load str character and move on to the next address
    cmp r2, #0      @ was this the end of the string?
    bne loop        @ read characters until you reach the end of the string
    sub r0, #1
concat:
    ldrb r3, [r1], #1   @ load str char from source and move on to the next address
    strb r3, [r0], #1   @ store character at the end of the destination string
    cmp r3, #0      @ was this the end of the string?
    bne concat
exit:
    mov r0, r4  @ restore dest address
    bx lr
```

Τέλος, τροποποιήσαμε το αρχείο *string_manipulation.c* προκειμένου να χρησιμοποιεί τις συναρτήσεις που υλοποιήσαμε αντί για αυτές της string.h, αντικαθιστώντας τη δήλωση της συγκεκριμένης βιβλιοθήκης με τις παρακάτω:

```
extern size_t strlen(const char *s);
extern char *strcpy(char *dest, const char *src);
extern char *strcat(char *dest, const char *src);
extern int strcmp(const char *s1, const char *s2);
```

Για τη μεταγλώττιση και σύνδεση των παραπάνω χρησιμοποιήσαμε το Makefile που παρατίθεται στη συνέχεια. Ειδικότερα, μεταγλωττίζουμε πρώτα τους κώδικες των τεσσάρων συναρτήσεων με την οδηγία -c προς τον compiler, προκειμένου να μην επιχειρήσει σύνδεση. Ύστερα μεταγλωττίζουμε με τον ίδιο τρόπο και το *string_manipulation.c* και, τελικά, τα συνδέουμε παράγωντας το εκτελέσιμο *string_manipulation.out*.

```
CC = gcc
CFLAGS = -Wall -g
OBJ = string_manipulation.o strcmp.o strcpy.o strcat.o strlen.o
ASS = strcmp.s strcpy.s strcat.s strlen.s

all: string_manipulation.out

string_manipulation.o: string_manipulation.c
  $(CC) -c -o $@ $^ $(CFLAGS)

%.o: $(ASS).s
  $(CC) -c -o $@ $< $(CFLAGS)

string_manipulation.out: $(OBJ)
  $(CC) -o $@ $^

clean:
  rm -f *.o *.out
```

Στη συνέχεια, μπορούμε να εκτελέσουμε το αρχείο που προκύπτει, όπως φαίνεται παρακάτω.

```
$ ./string_manipulation.out rand_str_input.txt
```

Η ορθότητα των υλοποιήσεων μας, ελέγχθηκε με σύγκριση των τριών αρχείων που παράγει η παραπάνω εκτέλεση, με αυτά που παρήγαγε πριν την εισαγωγή των δικών μας συναρτήσεων.