Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο εξάμηνο ΣΗΜΜΥ
3η εργαστηριακή άσκηση

Ομάδα 6:
Δούκας Θωμάς Α.Μ.: 03116081
Ψαρράς Ιωάννης Α.Μ.:03116033

Άσκηση 1

Για την πρώτη άσκηση υλοποιήθηκε πρόγραμμα σε assembly του arm επεξεργαστή το οποίο πραγματοποιεί μετασχηματισμούς σε συμβολοσειρά που εισάγεται από τον χρήστη. Αναφέρεται ότι η συμβολοσειρά εισάγεται από τον χρήστη μέσω του τερματικού και απαιτείται να έχει μήκος 32 χαρακτήρων. Κατά την συγγραφή του κώδικα ***str_man.s*** προνοήσαμε ώστε επιπλέον χαρακτήρες να αγνοούνται στη λειτουργία του προγράμματος, η οποία θα είναι συνεχής. Οι τεχνικές που χρησιμοποιήθηκαν θα παρουσιαστούν στη συνέχεια μαζί με τον αντίστοιχο κώδικα.

Αναλυτικότερα, οι μετατροπές που υλοποιούνται παρακάτω είναι:

- Μετατροπή χαρακτήρα από πεζό σε κεφαλαίο και αντίστροφα.
- Αύξηση αριθμών 0-4 κατά 5.
- Μείωση αριθμών 5-9 κατά 5.
- Δεν πραγματοποιούνται μετατροπές στους υπόλοιπους ASCII χαρακτήρες.
- Εισαγωγή μεμονωμένων χαρακτήρων *q* η *Q* οδηγούν σε τερματισμό προγράμματος.

Για την μεταγλώττιση του προγράμματος υλοποιήσαμε το *Makefile*:

```
1  all: str_man
2
3  str_man: str_man.s
4      @gcc -Wall str_man.s -o str_man
5
6  clean:
7      @rm str_man
```

Σχετικά με την υλοποίηση, για την ανάγνωση και εγγραφή των συμβολοσειρών στα std_in και std_out χρησιμοποιήσαμε *system calls* του λειτουργικού συστήματος. Ταυτόχρονα, η συνάρτηση μετατροπής *transformation* υλοποιήθηκε ξεχωριστά από τον κώδικα με έμφαση στην αποδοτική χρήση του arm instruction set και μείωση του πλήθους των απαιτούμενων εντολών. Ακολουθεί παρουσίαση της λογικής του προγράμματος μας.

Υλοποίηση

Αρχικά, διαβάζουμε την συμβολοσειρά εισόδου (*standard input*) και την αποθηκεύουμε στις θέσεις μνήμης που ορίζονται από το buffer. Διαβάζουμε 33 χαρακτήρες ώστε να δίνεται η δυνατότητα στον χρήστη να εισάγει 32 χαρακτήρες μαζί με τον χαρακτήρα \n, οποίος σηματοδοτεί και την λήξη της συμβολοσειράς.

Προκειμένου να ικανοποιηθεί ο περιορισμός των 32 χαρακτήρων και να αγνοήσουμε τυχόν μεγαλύτερη είσοδο είναι απαραίτητο να ελέγξουμε τον τελευταίο από αυτούς, ώστε να είναι "\n". Σε περίπτωση που έχει αναγνωριστεί χαρακτήρας διαφορετικός του "\n". Ταυτόχρονα χρησιμοποιώντας την τιμή επιστροφής της κλήσης συστήματος *write* (to buffer) γνωρίζουμε το ακριβές πλήθος χαρακτήρων και στην περίπτωση που αυτό είναι μεγαλύτερο από 2 (σίγουρα διαφορετικό από "q\n" ή "Q\n"), αποφεύγουμε τον έλεγχο *check* για τερματισμό του προγράμματος.

Στο σημείο αυτό πρέπει να τονιστεί ότι το κομμάτι κώδικα *clean_buf* εκτελείται στην περίπτωση που η συμβολοσειρά αποτελείται από λιγότερους των 33 χαρακτήρων (32 + "\n"). Η λειτουργία του παραπάνω απευθύνεται στον καθαρισμό των χαρακτήρων που τυχόν υπάρχουν μέσα στον buffer από την προηγούμενη εκτέλεση. Συγκεκριμένα, συνεχίζει την αρχικοποίηση του buffer τοποθετώντας τον χαρακτήρα null στις επόμενες διευθύνσεις. Το παρόν βήμα είναι ιδιαίτερα σημαντικό στάδιο καθώς μειώνει τις απαιτούμενες επαναλήψεις την συνάρτησης μετασχηματισμού.

Μετά την κλήση της συνάρτησης *transform,* οφείλουμε να καθαρίσουμε την standard input καθώς, όπως ήδη αναφέρθηκε, στην αρχή της υλοποίησής μας διαβάζουμε συγκεκριμένο πλήθος χαρακτήρων. Το βήμα αυτό συμβάλλει στην λειτουργία αγνόησης των επιπλέον χαρακτήρων καθώς εξασφαλίζει ότι η είσοδος δεν χρησιμοποιείται σε διαδοχικές επαναλήψεις του προγράμματος.

Transformation

Η συνάρτηση μετασχηματισμού αποτελείται από επαναλήψεις ίσες με το πλήθος των χαρακτήρων που εισήγαγε ο χρήστης. Σε κάθε επανάληψη ελέγχεται το περιεχόμενο μιας συγκεκριμένης θέσης μνήμης του buffer, ο οποίος περιέχει αποκλειστικά τα δεδομένα εισόδου που μας ενδιαφέρουν εκείνη τη στιγμή, πραγματοποιείται η κατάλληλη επεξεργασία και αποθηκεύεται στην ίδια θέση . Ειδικότερα πραγματοποιούνται έλεγχοι στον ASCII αριθμό του χαρακτήρα ώστε να προσδιοριστεί ότι πρόκειται για ψηφίο 0-9 ή λατινικό χαρακτήρα. Σε αντίθετη περίπτωση δεν επηρεάζεται από τον πυρήνα της συνάρτησης.

Εξασφαλίζοντας ότι το στοιχείο του buffer πρέπει να αλλάξει καταλήγουμε σε δύο βασικές προσεγγίσεις. Για τον μετασχηματισμού αριθμητικών ψηφίων αρκεί να ελέγξουμε το ψηφίο με τον αριθμό 5, ώστε να πραγματοποιηθεί η σωστή πράξη σε κάθε περίπτωση (*subge r3, r3, #5 και addlo r3, r3, #5*). Αντίστοιχη διαδικασία επιδιώξαμε να ακολουθήσουμε στους μετασχηματισμούς λατινικων χαρακτήρων. Η ανάγκη να αγνοήσουμε τους ενδιάμεσους ASCII χαρακτήρες μας οδήγησε στη χρήση συγκρίσεων και branches όπως φαίνεται και στον κώδικα. Για την αντιστοίχιση πεζών χαρακτήρων σε κεφαλαίους και αντίστροφα αρκεί η αύξηση ή μείωση του ακεραίου αριθμού που τους αντιστοιχεί κατά 32.

```
.text
.global main

main:
    @Print start message
    mov r0, #1 @std_out
    ldr r1, =msg_start
    ldr r2, =msg_start_len
    mov r7, #4 @write syscall
    swi 0

    @Read data from user
    mov r0, #0 @std_in
    ldr r1, =buffer
    mov r2, #33
    mov r7, #3 @read syscall
    swi 0

    @Read syscall returns number of bytes read in r0
    mov r4, r0
    mov r6, r0 @r4,r6 = chars user entered
    mov r8, #0 @flag (1 = last user char is \n)

    cmp r4, #33 @If user input is 32 chars make sure that pos 33 is \n
    bne clean_buf

    mov r5, #10@End line
    ldrb r3, [r1, #32]
    cmp r5, r3
    moveq r8, #1 @Last char was already \n
    beq cont
    strb r5, [r1, #32] @Otherwise store \n to last buffer position
    b cont @No need to check for q or Q bc input=33chars

clean_buf:
    @Add \0 to erase older trash stored in buffer
    mov r5, #0
    cmp r4, #33
    beq check
    strb r5, [r1, r4]
    add r4, r4, #1
    b clean_buf

check:
    @Check for quiting
    cmp r0, #2
    bne cont   @If 2 chars are read => check for q, Q
    ldrb r0, [r1] @Read 1st character from buffer
    @Quit case: q
    cmp r0, #113
    beq end
    @Quit case: Q
    cmp r0, #81
    beq end

cont:
    @String transformation
    bl transformation
```

```
cont:
    @String transformation
    bl transformation

    @Print result message
    mov r0, #1 @std_out
    ldr r1, =msg_result
    ldr r2, =msg_result_len
    mov r7, #4 @write syscall
    swi 0

    @Print result
    mov r0, #1 @std_out
    ldr r1, =buffer
    mov r2, #33
    mov r7, #4 @write syscall
    swi 0

clean_input:
    cmp r6, #33 @No need to clean input if user entered 32 chars
    blo main

    cmp r8, #1 @If last char was already a \n stdin is empty
    beq main

read_stdin:
    @Repeatedly read until std_in is empty
    mov r0, #0 @std_in
    ldr r1, =buffer
    mov r2, #33
    mov r7, #3 @read syscall
    swi 0

    cmp r0, #33 @If read_chars < 33 stdin is empty
    blo main
    b read_stdin

end:
    @Print result
    mov r0, #1 @std_out
    ldr r1, =msg_quit
    ldr r2, =msg_quit_len
    mov r7, #4 @write syscall
    swi 0

    @Exit with status code 0
    mov r0, #0
    mov r7, #1
    swi 0

```

```
transformation:
    push {r1-r6}
    mov r2, #0 @counter/byte offset
start:
    cmp r2, r6 @Change #input_chars
    beq exit
    ldrb r3,[r1,r2]

    cmp r3, #48 @ <0
    blo next_iteration

number:
    cmp r3, #57
    bgt check_letter
    @Number Transform
    cmp r3, #53 @If number is 5
    subge r3, r3, #5 @ 56789 -> 01234
    addlo r3, r3, #5 @ 01234 -> 56789
    strb r3, [r1, r2]
    add r2,r2, #1 @next iteration
    b start

check_letter:
    cmp r3, #65
    blo next_iteration

uppercase_letter:
    cmp r3, #90
    bgt lowercase_letter
    @Uppercase tsanformation
    add r3, r3, #32
    strb r3, [r1, r2]
    add r2,r2, #1
    b start

lowercase_letter:
    cmp r3, #97
    blo next_iteration
    cmp r3, #122
    bgt next_iteration
    @Lowercase transformation
    sub r3, r3, #32
    strb r3, [r1, r2]
    add r2,r2, #1
    b start

next_iteration:
    add r2, r2, #1
    b start

exit:
    pop {r1-r6}
    bx lr

```

Στην προσπάθειας μας να ελαχιστοποιήσουμε τις εντολές που απαιτούνται οδηγηθήκαμε ακολουθία τον εντολών που φαίνεται σε επόμενη σελίδα. Ιδιαίτερα χρήσιμο φάνηκε το πεδίο *condition* του instruction set του επεξεργαστή arm οι οποίες μας συνέβαλαν στην αποφυγή χρήσης περιττών branches. Μάλιστα παρόμοιες εντολές χρησιμοποιήθηκαν και στις επόμενες ασκήσεις της αναφοράς, όπως θα δούμε στη συνέχεια.

Αποτελέσματα εκτέλεσης:

```
root@debian-armel:/home/user/lab3/1# ./str_man
Please enter string up to 32 chars long: Initializing test sequence
Result is: iNITIALIZING TEST SEQUENCE
Please enter string up to 32 chars long: I am... inevitable.
Result is: i AM... INEVITABLE.
Please enter string up to 32 chars long: Snap! O_O
Result is: sNAP! o_o
Please enter string up to 32 chars long: And I... am... Iron Man!
Result is: aND i... AM... iRON mAN!
Please enter string up to 32 chars long: !@#$%^&*()
Result is: !@#$%^&*()
Please enter string up to 32 chars long: 12345678901234567890123456789abc
Result is: 67890123456789012345678901234AB
Please enter string up to 32 chars long: !@#$%^&*()abcdefghij1234567890WORSTCASESCENARIO...
Result is: !@#$%^&*()ABCDEFGHIJ6789012345wo
Please enter string up to 32 chars long: QUIT
Result is: quit
Please enter string up to 32 chars long: QuIt
Result is: qUiT
Please enter string up to 32 chars long: q
Quiting...
root@debian-armel:/home/user/lab3/1#
```

Άσκηση 2

Στην επόμενη εργαστηριακή άσκηση θα παρουσιαστούν 2 προγράμματα που επικοινωνούν μεταξύ τους μέσω εικονικής σειριακής θύρας. Το πρώτο πρόγραμμα λειτουργεί στο host μηχάνημα σε γλώσσα C και επικοινωνεί με το δεύτερο του guest μηχανήματος σε arm assembly. Σκοπός της επικοινωνίας είναι αποστολή ενός string από το host μηχάνημα προς τον guest για τον προσδιορισμό του χαρακτήρα που εμφανίζεται πιο συχνά αλλά και το πλήθος των εμφανίσεων αυτών.

Ως περιορισμός της συγκεκριμένης υλοποίησης θεωρείται το μέγεθος της συμβολοσειράς το οποίο δεν πρέπει να ξεπερνά τους 64 χαρακτήρες, ενώ ταυτόχρονα ο κενός χαρακτήρας εξαιρείται από την καταμέτρηση. Ταυτόχρονα σε περίπτωση που 2 οι περισσότεροι χαρακτήρες παρουσιάζουν το ίδιο πλήθος εμφανίσεων στη συμβολοσειρά εισόδου το assembly πρόγραμμα θα επιστρέφει τον χαρακτήρα με το μικρότερο κωδικό ASCII.

Για την επικοινωνία των προγραμματων εκτελούμε το qemu-system-arm με όρισμα –serial pty και πραγματοποιούνται οι κατάλληλες ενέργειες για την αρχικοποίηση του pseudoterminal. Ενημερωνόμαστε ότι ο host θα χρησιμοποιεί το αρχείο /dev/pts/1 ως ένα άκρο της επικοινωνίας. Αντίστοιχα με την εντολή dmesg | grep tty βλέπουμε ότι το guest virtual machine χρησιμοποιεί το αρχείο ttyAMA0. Οι πληροφορίες αυτές θα χρησιμοποιηθούν στη συνέχεια για την παραμετροποίηση της σειριακής θύρας μέσω της βιβλιοθήκης *termios.h*.

Για την μεταγλώττιση των προγραμμάτων συντάσσουμε τα παρακάτω Makefiles για host και guest αντίστοιχα.

```
all: host

host: host.c
    @gcc -Wall host.c -o host

clean:
    @rm host
```

```
all: guest

str_man: guest.s
    @gcc -Wall guest.s -o guest

clean:
    @rm guest
```

Αναφέρουμε ότι λόγω δικαιωμάτων απαιτείται να εκτελέσουμε το πρόγραμμα σε περιβάλλον root καθώς σε αντίθετη περίπτωση δεν επιτρέπεται το άνοιγμα του αρχείου */dev/tty/1*. Δοκιμάσαμε και την εκτέλεση της εντολής *$sudo adduser $USER dialout* ωστόσο δεν παρατηρήσαμε διαφορά.

Υλοποίηση

Κώδικας C (*host.c*)

Η διαδικασία εκκινείται εκτελώντας την συνάρτηση open για το άνοιγμα του άκρου επικοινωνίας από την πλευρά του host. Με την επιτυχή εκτέλεση της εντολής είναι απαραίτητο να πραγματοποιήσουμε το configuration του ***termios*** που θα επιτρέψει την επικοινωνία με το guest σύστημα ARM. Για τις επιλογές αυτές ακολουθήσαμε τις υποδείξεις της εκφώνησης αλλά και συμβουλευόμενοι το διαδίκτυο[1].

Ακολούθως μεταφερόμαστε στο κύριο κομμάτι της υλοποίησης μας όπου προτρέπουμε τον χρήστη να εισάγει την συμβολοσειρά εως και 64 χαρακτήρων, την οποία και θα αποθηκεύσουμε σε buffer συγκεκριμένων διαστάσεων. Διαβάζοντας από τον χρήστη συγκεκριμένα 64 στο πλήθος χαρακτήρες διασφαλίζουμε ότι οι υπόλοιποι θα αγνοηθούν κατά την διαδικασία προσδιορισμού του συχνότερου.

΄Υστερα, σειρά έχει η εγγραφή του περιεχομένου του buffer στο host ακρο της επικοινωνίας για την αποστολή προς το πρόγραμμα guest. Στο σημείο αυτό είναι ιδιαίτερα σημαντικό να αναμείνουμε τον υπολογισμό από την πλευρά του guest αλλά και την λήψη αποτελέσματος. Για το λόγο αυτό χρησιμοποιούμε blocking read, ώστε να εξασφαλίσουμε ότι θα συνεχίσουμε να διαβάζουμε, μέχρις ότου να είναι διαθέσιμο ολόκληρο το αποτέλεσμα. Διαβάζουμε συγκεκριμένο πλήθος ψηφίων καθώς η εκτέλεση του guest.s πρόκειται να επιστρέψει συμβολοσειρά συγκεκριμένης μορφής “<χαρακτήρας><space><συχνότητα>”.

```c
int main(void){

    //Thank you embedded ninja: https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/
    //Initilize
    int fd;
    const char serial_port[] = "/dev/pts/1"; //Used by host in serial communication
    char buffer[BUF_LEN], result[BUF_LEN];
    struct termios termios_conf;

    //Open serial port device
    fd = open(serial_port, O_RDWR | O_NOCTTY );
    //Check for errors
    if (fd == -1){
        printf("Failed to open serial port\n");
        return 1;
    }

    // Read in existing settings, and handle any error
    tcgetattr(fd, &termios_conf);
    //Flag Initialization - Set Control Mode
    termios_conf.c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
    termios_conf.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
    termios_conf.c_cflag &= ~CSIZE; // Clear all bits that set the data size
    termios_conf.c_cflag |= CS8; // 8 bits per byte (most common)
    termios_conf.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
    termios_conf.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

    termios_conf.c_lflag &= ~ICANON; // Disable Canonical Mode
    termios_conf.c_lflag &= ~ECHO; // Disable echo
    termios_conf.c_lflag &= ~ECHOE; // Disable erasure
    termios_conf.c_lflag &= ~ECHONL; // Disable new-line echo
    termios_conf.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
    termios_conf.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off software flow ctrl
    termios_conf.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

    termios_conf.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
    termios_conf.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed

    termios_conf.c_cc[VTIME] = 10;    // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.
    termios_conf.c_cc[VMIN] = 0;
    //This is a blocking read of any number chars with a maximum timeout (given by VTIME).
    //read() will block until either any amount of data is available, or the timeout occurs.

    // Set in/out baud rate
    if(cfsetispeed(&termios_conf, B9600) < 0 || cfsetospeed(&termios_conf, B9600) < 0) {
        printf("Problem with baudrate\n");
        return 1;
    }
    //Saving termios
    if (tcsetattr(fd, TCSANOW, &termios_conf) != 0) {
        printf("Couldn't apply settings\n");
        return 1;
    }

    //Starting
    printf("Please give a string of up to 64chars to send to host:\n");
    fgets(buffer, BUF_LEN, stdin);

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

    //Sent to guest
    write(fd, buffer, BUF_LEN);

    //Read from guest
    while (read(fd, result, 3) <= 0);
    printf("The most frequent character is\n%c\nand it appeared %d times.\n", result[0], result[2]);

    //Done with serial port
    close(fd);
    return 0;
}
```

Κώδικας ARM - assembly (guest.s)

Σκοπός της άσκησης από την πλευρά του guest είναι να διαβάζει την συμβολοσειρά που γράφει στο άκρο του ο host, να μετράει τη συχνότητα του κάθε χαρακτήρα που εμφανίζεται (εκτός του space) και να επιστρέφει το αποτέλεσμα μέσω της serial port.

Αρχικά, ανοίγουμε με χρήση του ***open sys_call*** την guest πλευρά της σειριακής. Σε αυτό το σημείο, και προτού προχωρήσουμε παρακάτω, ελέγχουμε για ενδεχόμενα errors κατά του ανοίγματος του serial port. Στη συνέχεια, κάνουμε configure το termio σύμφωνα με τις παραμέτρους που περιγράφονται στο options. Για την διαδικασία αυτή συμβουλευόμαστε τις αντίστοιχες αρχικοποιήσεις στον κώδικα *host.c*.

Αφού ολοκληρωθεί η διαδικασία αυτή της αρχικοποίησης, εκτελείται **read sys_call** για να διαβάσουμε την συμβολοσειρά (το πολύ 64ων χαρακτήρων) που έδωσε ο host ως input στην serial port. Τώρα μπορούμε μέσω της συνάρτησης **freq_calculate** να υπολογίσουμε τον χαρακτήρα με την μέγιστη συχνότητα εμφάνισης, αλλά και τη συχνότητα αυτή.

Ξεκινώντας, η **freq_calculate** αρχικοποιεί έναν πίνακα **freq_arr** γιά αποθήκευση της συχνότητας του κάθε χαρακτήρα που συναντάει, αλλά και έναν μετρητή για την διάσχιση του input. Σε κάθε επανάληψη της διαδικασίας αυτής, εξετάζεται ο χαρακτήρας που υποδεικνύει ο counter. Αν ο χαρακτήρας αυτός είναι “\n”, τότε έχουμε φτάσει στο τέλος του input και πρέπει να προχωρήσουμε στον υπολογισμό της μέγιστης συχνότητας. Στην περίπτωση, που ο χαρακτήρας είναι το κενό (ASCII #32), αυξάνουμε κατά 1 τον μετρητή και συνεχίζουμε στην επόμενη επανάληψη, προνοώντας έτσι ώστε να μην προσδιορίζεται η συχνότητα εμφάνισης του.Για τους υπόλοιπους χαρακτήρες εύρους ASCII #33, μέχρι και τον ASCII #126, αφαιρούμε 33 (για να πάρουμε το offset αποθήκευσης του στον πίνακα **freq_arr**), και αυξάνουμε το περιεχόμενο του πίνακα στο offset αυτό κατά 1. Όταν πλέον έχουμε διασχίσει όλο το input string, μπορούμε να προχωρήσουμε στον υπολογισμό της μέγιστης συχνότητας.

Ξεκινώντας από την αρχή του πίνακα θέτουμε ως μέγιστη τιμή συχνότητας το 0. Διασχίζοντας λοιπόν τον πίνακα εξετάζουμε κάθε φορά αν ο τρέχων χαρακτήρας εμφανίζεται περισσότερες φορές από όσες υποδεικνύονται από την τιμή του μεγίστου μέχρι τώρα. Αν πράγματι ο εξεταζόμενος χαρακτήρας υπερβαίνει σε συχνότητα εμφάνισης το προηγούμενο τότε η τιμή του μεγίστου ανανεώνεται ανάλογα, αλλιώς προχωράμε στον επόμενο χαρακτήρα. Αξίζει να σημειωθεί, πως αυτή η πρακτική μας εξασφαλίζει και το ότι σε περίπτωση που έχουμε παραπάνω από δύο χαρακτήρες με την ίδια μέγιστη συχνότητα εμφάνισης, αυτός που θα επιλεγεί είναι τελικά εκείνος με τον μικρότερο αριθμό ASCII. Αυτό ισχύει, καθώς θα βρίσκεται σε χαμηλότερο offset στον πίνακα και επομένως θα έχει ήδη επιλεχθεί.Αφού βρούμε την μέγιστη συχνότητα, υπολογίζουμε τον χαρακτήρα στον οποίο αντιστοιχεί προσθέτοντας 33 στο offset που την εντοπίσαμε. Έπειτα αποθηκεύουμε τα δύο αυτά return values στον πίνακα result και επιστρέφουμε στην συνέχεια του κύριου προγράμματος μας.

Τέλος, εκτελούμε **write sys_call**, γράφοντας τα αποτελέσματα στο υπεύθυνο για την επικοινωνία άκρο του guest. Έτσι γίνονται διαθέσιμα στον host και μπορούμε να αποδεσμεύσουμε τον file descriptor.

```
.text
.global main
.extern tcsetattr
.extern printf

main:
    @Open serial port
    ldr r0, =serial_port
    mov r1, #2 @ReadWrite
    ldr r2, =0666 @permissions
    mov r7, #5 @open sys call
    swi 0

    @Check open for errors
    cmp r0, #0
    bmi end
    @Save fd from r0
    mov r5, r0

    @Setting up termio
    @r0 has file descriptor
    mov r1, #0 @TCSANOW
    ldr r2, =options
    bl tcsetattr

    @Read all chars from serial port - 64Worst case
    mov r0, r5 @Fd
    ldr r1, =buffer
    mov r2, #64
    mov r7, #3 @read syscall
    swi 0
    @Number of chars read on r0

    bl freq_calculate

    @Write results to the serial port
    mov r0, r5 @Fd
    ldr r1, =result
    ldr r2, =len_result
    mov r7, #4 @write syscall
    swi 0

    @Close the serial port
    mov r0, r5 @Fd
    mov r7, #6 @close syscall
    swi 0

end:
    @Exit with status code 0
    mov r0, #0
    mov r7, #1
    swi 0

```

```
54  freq_calculate:
55      push {r1-r6}
56      @r1 = buffer
57      mov r2, #0 @Counter
58      @r3 = current load-store
59      ldr r4, =freq_arr
60
61  start:
62      cmp r2, #64 @When counter reaches the end of freq_arr stop iterating
63      beq cont    @This is for the special case where the user enters 64 characters + \n
64
65      ldrb r3, [r1, r2]
66      cmp r3, #10 @If we find \n frequency culculator is ended
67      beq cont
68
69      cmp r3, #32 @Do not count space character's frequency
70      addeq r2, r2, #1 @ Increase coutner
71      beq start
72
73      sub r3, r3, #33 @Calculate offset in frequency array
74      ldrb r5, [r4, r3] @Load frequency of input char from frequrency array
75      add r5, r5, #1 @Increment frequency by 1
76      strb r5, [r4, r3] @Offset is already calculated
77      add r2, r2, #1 @ Increase coutner
78      b start
79
80  cont:
81      mov r2, #0 @Initialize counter
82      mov r5, #0 @Use r5 for max_freq
83      mov r6, #0 @Position of max frequency in freq_arr
84
85  max_freq_loop:
86      cmp r2, #94 @Last possition of freq_arr (freq_arr length = 94Bytes - 1 For each possible input)
87      beq finalize
88      ldrb r3, [r4, r2] @Get frequency of r2 cell in freq_arr.
89      cmp r3, r5  @If new_freq(r3) > max_freq (r5)
90      movgt r5, r3
91      movgt r6, r2 @Save new_max_freq, character's offset in freq_arr
92      add r2, r2, #1 @Increase counter
93      b max_freq_loop
94
95  finalize:
96
97      @Save results in variable results
98      @r4 has result array address
99      ldr r4, =result
100     add r6, r6, #33 @Most frequent character
101     strb r6, [r4]
102     sub r5, r5, #48
103     strb r5, [r4, #2] @Frequency of most frequent character
104
105 exit:
106     pop {r1-r6}
107     bx lr
108
```

Αποτελέσματα εκτέλεσης:

```
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
"Obi-Wan never told you what happened to your father."
The most frequent character is
e
and it appeared 5 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
"He told me enough! He told me you killed him!"
The most frequent character is
e
and it appeared 6 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
"No. I am your father."
The most frequent character is
"
and it appeared 2 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
"Noooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo"
The most frequent character is
o
and it appeared 62 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
!!                !@
The most frequent character is
!
and it appeared 3 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2# ./host
Please give a string of up to 64chars to send to host:
999911110000
The most frequent character is
0
and it appeared 4 times.
root@thomas-VirtualBox:/home/thomas/Documents/Ece/EmbeddedSystems/Lab_3/2#
```

Άσκηση 3

Για την τρίτη άσκηση ζητούμενο ήταν η αντικατάσταση των συναρτήσεων ***strlen, strcpy, strcat*** και ***strcmp*** της βιβλιοθήκης ***string.h*** της C, με αντίστοιχες συναρτήσεις υλοποιημένες σε assembly του ARM. Οι συναρτήσεις αυτές χρησιμοποιούνται στο αρχείο string_manipulation.c το οποίο ανοίγει ένα αρχείο με 512 γραμμές, κάθε γραμμή του οποίου περιέχει μια τυχαία κατασκευασμένη συμβολοσειρά, μεγέθους από 8 έως 64 χαρακτήρες, και κατά την εκτέλεση του παράγονται τα εξής τρία αρχεία εξόδου:

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

Για να υπάρχει σαφής διαχωρισμός σε σχέση με τις ήδη υπάρχουσες συναρτήσεις της C, οι ARM assembly συναρτήσεις που υλοποιήσαμε ονομάστηκαν ***_strlen, _strcpy, _strcat*** και ***_strcmp***.

Για σωστή μεταγλώττιση και εκτέλεση των τροποποιημένων προγραμμάτων δημιουργήσαμε το Makefile:

```
all: string_manipulation

string_manipulation: string_manipulation.c functions.s
	@gcc -Wall -g -c string_manipulation.c -o string_manipulation
	@gcc -Wall -g -c functions.s -o functions
	@gcc -Wall string_manipulation functions -o string_manipulation.out
	@rm string_manipulation functions

clean:
	@rm string_manipulation functions
```

ακολουθώντας τις οδηγίες της εκφώνησης.

Στη συνέχεια θα εξετάσουμε τις λειτουργίες των συναρτήσεων που υλοποιήθηκαν. Για όλες τις περιπτώσεις προσέξαμε να ακολουθείται η πρακτική arm assembly και το αποτέλεσμα να επιστρέφει στον καταχωρητή r0 μετά το πέρας της εκτέλεσης. Επιπλέον, χρησιμοποιούνται οι εντολές *push* και *pop* για να διασφαλίσουμε την διατήρηση τιμών που χρησιμοποιούν προγράμματα εκτός των υλοποιήσεων των συναρτήσεων μας. Ακολουθεί η υλοποίηση σε κώδικα assembly και αναλυτική επεξήγηση του τρόπου σκέψης μας.

_strlen:

```
4   .global _strlen
5   .type _strlen, %function
6
7   _strlen:
8       @Input string at r0
9       push {r1-r3}
10      mov r2, #0 @r2: Counter - String length
11
12  start__strlen:
13      ldrb r3, [r0, r2] @Load string character
14      cmp r3, #0 @Check if NULL (ASCII 0) - end of string
15      addne r2, r2, #1 @Add 1 to the string lenght
16      bne start__strlen @Continue to the next offset
17
18  end__strlen:
19      mov r0, r2 @Result to r0
20      pop {r1-r3}
21      bx lr
```

Στη συγκεκριμένη συνάρτηση λαμβάνουμε ως όρισμα ένα string χαρακτήρων η διεύθυνση του οποίου είναι αποθηκευμένη στον καταχωρητή r0. Σκοπός είναι να μετρήσουμε το μήκος της συμβολοσειράς και να επιστρέψουμε την τιμή αυτή.

Ξεκινάμε αρχικοποιώντας τον καταχωρητή r2, ο οποίος λειτουργεί ως μετρητής - offset για την προσπέλαση της συμβολοσειράς. Η τιμή του byte της συμβολοσειράς που υποδεικνύεται από το offset r2, φορτώνεται στον καταχωρητή r3 και ελέγχεται αν πρόκειται για τον χαρακτήρα NULL. Αν ο χαρακτήρας είναι διάφορος του NULL τότε ο μετρητής αυξάνεται και συνεχίζουμε στην εξέταση του επόμενου χαρακτήρα. Όταν συναντήσουμε τον χαρακτήρα NULL (ASCII Code #0), δεν αυξάνουμε εκ νέου τον μετρητή, αλλά αποθηκεύουμε το περιεχόμενό του στον καταχωρητή r0, και επιστρέφουμε.

_strcpy:

```
24  .global _strcpy
25  .type _strcpy, %function
26
27  _strcpy:
28      @Copying string from r1 to r0 (MUST ALSO COPY NULL AT THE END), returns copied string
29      push {r2-r4}
30      mov r2, #0 @r2: character offset
31
32  start__strcpy:
33      ldrb r3, [r1, r2] @Take byte of source string r1 (at offset r2)
34      cmp r3, #0  @Check if it is NULL character - end of string
35      beq end__strcpy
36      strb r3, [r0, r2] @Copy the selected byte to destination r0
37      add r2, r2, #1 @Increment character offset
38      b start__strcpy @Continue to the next offset
39
40  end__strcpy:
41      strb r3, [r0, r2] @Copy NULL at the end of string, now r0 contains copied string
42      pop {r2-r4}
43      bx lr
44
```

Η συνάρτηση **_strcpy** παίρνει ως ορίσματα δύο συμβολοσειρές str1 και str2 οι διευθύνσεις των οποίων βρίσκονται στους καταχωρητές r0 και r1 αντίστοιχα, και αντιγράφει τη

συμβολοσειρά str2 στην θέση της συμβολοσειράς str1 επιστρέφοντας στον χρήστη την συμβολοσειρά που αντιγράφηκε.

Όπως και προηγουμένως, ο καταχωρητής r2 χρησιμοποιείται ως offset για προσπέλαση των συμβολοσειρών. Έτσι φορτώνουμε τον χαρακτήρα της συμβολοσειράς str2 που υποδεικνύει το offset r2 στον καταχωρητή r3 και αφού ελέγξουμε πως δεν πρόκειται για τον χαρακτήρα NULL, αποθηκεύουμε το περιεχόμενο του r3 στην κατάλληλη θέση του str1. Όταν συναντήσουμε NULL, πραγματοποιούμε branch στο τελικό στάδιο της συνάρτησης, όπου το NULL αποθηκεύεται στο τέλος της συμβολοσειράς. Ύστερα, επιστρέφουμε με την διεύθυνση του επιθυμητού return value να είναι αποθηκευμένη στον καταχωρητή r0.

_strcat:

```
46  .global _strcat
47  .type _strcat, %function
48
49  _strcat:
50      @Destination string at r0, source string at r1
51      push {r2-r4}
52      mov r2, #0 @r2: 1st counter to find destination length
53  start_strcat:
54      @Find end of destination
55      ldrb r3, [r0, r2] @Load destination character
56      cmp r3, #0 @And check if its NULL (ASCII 0) - end of string
57      addne r2, r2, #1
58      bne start_strcat @And continue to the next offset
59
60      @Copy part
61      mov r4, #0 @Second counter to use on source offset string
62  cont_strcat:
63      ldrb r3, [r1, r4] @Load character of source string
64      cmp r3, #0 @Check if it is NULL character - end of string
65      beq end_strcat
66      strb r3, [r0, r2] @r2 represents the current end of destination string
67      add r4, r4, #1 @Increment both counters
68      add r2, r2, #1
69      b cont_strcat
70
71  end_strcat:
72      strb r3, [r0, r2] @Copy NULL at the end of the destination string
73      pop {r2-r4} @Result to r0
74      bx lr
```

Στη συνάρτηση **_strcat** πρέπει να τοποθετήσουμε την συμβολοσειρά str2 (r1) στο τέλος της συμβολοσειράς str1 (r0) επιστρέφοντας στον χρήστη την νέα συμβολοσειρά που δημιουργείται και η διεύθυνσή της βρίσκεται στον καταχωρητή r0.

Παρόμοια με την συνάρτηση **_strlen**, απαιτείται να υπολογίσουμε το μήκος της συμβολοσειράς str_dst για να τοποθετούμε τους χαρακτήρες της str_src στο κατάλληλο offset. Έτσι, χρησιμοποιούμε ξανά τον καταχωρητή r2 για να υπολογίσουμε το μήκος της str_dst και για να εξετάσουμε έναν έναν τους χαρακτήρες της, ενώ ο τρέχων χαρακτήρας αποθηκεύεται στον καταχωρητή r3. Μόλις συναντήσουμε τον χαρακτήρα NULL, έχουμε προσδιορίσει τη θέση στην οποία πρέπει να αντιγραφεί το πρώτο στοιχείο της str_src. Πανομοιότυπα, ο καταχωρητής r4 χρησιμοποιείται για να προσπελάσουμε την δεύτερη συμβολοσειρά, τα στοιχεία της οποίας αποθηκεύονται στον r3. Ακολούθως, εξετάζεται η περίπτωση του NULL και στην περίπτωση που ο χαρακτήρας είναι διάφορος αυτής, αποθηκεύεται στο offset που

υποδεικνύεται από τον καταχωρητή r2. Για την ομαλή εκτέλεση της επόμενης επανάληψης αυξάνονται οι δύο μετρητές κατά 1. Όταν φτάσουμε στο τέλος της δεύτερης συμβολοσειράς, τοποθετείται NULL στο τέλος της συμβολοσειράς προορισμού. Με αυτόν τον τρόπο η διαδικασία ολοκληρώνεται με την διεύθυνση προορισμού (str1) να βρίσκεται στο r0.

_strcmp:

```
77  .global _strcmp
78  .type _strcmp, %function
79
80  _strcmp:
81      @r0: str_1
82      @r1: str_2
83      @return value is placed on r0 and its referenced to str1
84      @(calculates the difference between the 1st unmatched character of the 2 strings)
85      push {r2-r5}
86      mov r2, #0 @r2: counter
87
88  start__strcmp:
89      ldrb r3, [r0, r2] @Load character at offset r2 from both strings
90      ldrb r4, [r1, r2]
91      cmp r3, r4
92      bne calc_diff
93      cmp r4, #0
94      addne r2, r2, #1  @ beq calc_diff @ add r2, r2, #1
95      bne start__strcmp  @ b start__strcmp <-- A better implementation
96
97  calc_diff:
98      sub r0, r3, r4 @Return value is calculated
99
100 end__strcmp:
101     pop {r2-r5} @Result to r0 (r0 < 0 => str1<str2, r0>0 => str1>str2, r0=0 => str1=str2)
102     bx lr
103
```

Τέλος, στη συνάρτηση **_strcmp** συγκρίνονται οι συμβολοσειρές str1 (r0) και str2 (r1) με βάση το εξής κριτήριο σύγκρισης. Εξετάζονται ένας ένας οι χαρακτήρες κάθε συμβολοσειράς, μέχρι να βρεθεί το πρώτο ζεύγος διαφορετικών χαρακτήρων. Τότε οι δύο χαρακτήρες αυτοί συγκρίνονται με βάση τον κωδικό ASCII τους, ενώ το αποτέλεσμα της σύγκρισης (και return value της συνάρτησης) είναι η διαφορά των κωδικών ASCII αυτών. Η σύγκριση γίνεται σε σχέση με την συμβολοσειρά str1 που σημαίνει πως για θετικό αποτέλεσμα , ισχύει str1>str2, για αρνητικό, str1<str2, ενώ στην περίπτωση του μηδενικού αποτελέσματος, οι συμβολοσειρές ταυτίζονται.

Ξεκινάμε λοιπόν με χρήση του καταχωρητή r2 ως counter για προσπέλαση των στοιχείων των συμβολοσειρών. Σε κάθε επανάληψη, φορτώνονται στους r3 και r4 οι χαρακτήρες των συμβολοσειρών str1 και str2 αντίστοιχα, που βρίσκονται στο offset r2. Όταν οι χαρακτήρες διαφέρουν αρκεί να υπολογίσουμε την διαφορά τους, και έπειτα να την αποθηκευσουμε στο καταχώρητη r0 ο οποίος και πάλι θα επιστρέψει το τελικό αποτέλεσμα.Αντιθέτως, αν είναι ίσοι διακρίνουμε 2 υποπεριπτώσεις. Είτε πρόκειται για ισότητα χαρακτήρων διάφορων του NULL, οπότε και πρέπει να αυξήσουμε τον μετρητή και να συνεχίσουμε, είτε οι και οι 2 καταχωρητές (r3,r4) περιέχουν τον χαρακτήρα NULL. Στην πρώτη περίπτωση (*(r3 == r4) != NULL*) πρέπει να να αυξήσουμε τον μετρητή και να επαναλάβουμε την διαδικασία δηλαδή τον υπολογισμό της διαφοράς των κωδικών ASCII και την τοποθέτηση αυτού στον καταχωρητή r0. Αντιθέτως στην δεύτερη (*(r3 == r4) == NULL*) οι συμβολοσειρές ταυτίζονται και ο καταχωρητής r0 πρέπει να λάβει την τιμή 0.

Στη συνέχεια, το πρόγραμμα τερματίζει, με τον r0 να περιέχει την κατάλληλη τιμή για τον προσδιορισμό της σχέσης των συμβολοσειρών όπως παρουσιάστηκε προηγουμένως.

Επιπρόσθετες αλλαγές χρειάστηκε να πραγματοποιηθούν και στον κώδικα ***string_manipulation.c.*** Σημαντικότερη αυτών είναι η αντικατάσταση των *#include* με τις οδηγίες extern ώστε να εκτελεστούν οι κατάλληλες υλοποιήσεις.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  extern size_t _strlen(const char * str_src);
5  extern char* _strcpy(char * str_dst, const char * str_src);
6  extern char* _strcat(char * str_dst, const char * str_src);
7  extern int _strcmp(const char * str_1, const char * str_2);
```

Αναφέρουμε ότι για την σύγκριση της ορθότητας των υλοποιήσεων μας εκτελέσαμε τόσο τα αρχικό πρόγραμμα, όσο και το αντίστοιχο που πραγματοποιεί χρήση των συναρτήσεων μας, με είσοδο τα αρχείο δεδομένων που παρέχονται από το εργαστήριο. Τα αποτελέσματα των εκτελέσεων αυτών βρίσκονται στους υποφακέλους *ARM* και *Initial* του *Results*. Για την γρήγορη σύγκριση των αρχείων εξόδου χρησιμοποιήθηκε online πρόγραμμα λογισμικού με το οποίο διαπιστώθηκε ότι ταυτίζονται.

Παραπομπές:

[1]: https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/