ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων

9ο Εξάμηνο, 2024-2025

5η Εργαστηριακή Άσκηση

αναφορά της φοιτήτριας:

Λάζου Μαρία-Αργυρώ (el20129)

Ομάδα:**35**

Ερώτημα 1ο

Το παρακάτω πρόγραμμα σε Assembly πραγματοποιεί την ζητούμενη λειτουργία με βάση τις ακόλουθες παρατηρήσεις:

- Για την μετατροπή από πεζά σε κεφαλαία και το αντίστροφο απαιτείται απλώς η πρόσθεση (ή αφαίρεση αντίστοιχα) της σταθεράς 32 από τον ASCII κώδικό του συγκεκριμένου χαρακτήρα που αντιστοιχεί σε 1 εντολή arm assembly (addge/ subge). Η επιλογή χρήσης των ιδιωματικών εντολών arm θα φανεί στην συνέχεια.

- Για την μετατροπή των αριθμών απαιτείται ο υπολογισμός του (x + 5) mod 10, όπου x ε [1,...,9]. Προς αποφυγή διαίρεσης κάνω το εξής τρικ: προσθέτω 5, ελέγχω εαν προκύπτει τιμή > 9 και αφαιρώ 10 σε αυτήν περίπτωση, αντιτοιχεί λοιπόν σε 3 εντολές arm assembly (add, cmp, subgt).

- Για την διάκριση των modes λειτουργίας (arithmetic shift, lowercase -> uppercase, uppercase -> lowercase) συγκρίνω το στοιχείο της συμβολοσειράς εισόδου με τους κωδικούς ASCII των οριακών τιμών ‘0’, ‘9’, ‘Z’, ‘z’ σε hex μορφή και κατά αύξουσα σειρά, που αντιστοιχόυν σε μια εντολή assembly (cmp). Τα άνω modes λειτουργίας αντιστοιχούν στα ενδιάμεσα διαστήματα που σχηματίζουν οι οριακές αυτές τιμές (*_is_upper*, *_is_lower*). Στην περίπτωση των γραμμάτων ελέγχω έπειτα και τον χαρακτήρα ‘A’ ή ‘α’ για επιβεβαίωση ότι βρίσκεται πράγματι στο σωστό εύρος). Εαν δεν ικανοποιηθεί κανένας έλεγχος ο χαρακτήρας μένει αμετάβλητος.

- Η σύγκριση με ‘q’ ή ‘Q’ πραγματοποιείται πρωτού εισέλθουμε στην συνάρτηση *_transform* για έγκαιρο τερματισμό.

- Για την αγνόηση του input > 32 bytes καλούμε συνεχώς στο loop *_discard* το read system call μέχρι να διαβάσουμε τιμή != 32.

- Η ανάγνωση της συμβολοσειράς γίνεται καλώντας απευθείας το read systemcall (offset 3) και η εκτύπωση με jump στην external printf. (Θα μπορούσε να χρησιμοποιηθεί και επευθείας το write systemcall (offset 4)).

ex1.s

```
.text
.global main
.extern printf

main :
        ldr r0, =input_msg
        bl printf

        ldr r0, =0                  //stdout fd
        ldr r1, =buff_in            // input buffer
        ldr r2, =32                 // max number of bytes read
        mov r7, #3                  // syscall number for read
        svc #0                      // trigger the syscall

        cmp r0, #2                  //check number of bytes read
        mov r5, r0                  //save bytes read in r5
        bne _transform              // if > 2 call transform

        ldrb r2, [r1]               //load first charachter
        cmp r2, #0x71               // 'q' is pressed
        beq _quit
        cmp r2, #0x51                // 'Q' is pressed
        beq _quit
```

```
_transform :
               ldr r6, =buff_out

_loop :        ldrb r2, [r1], #1          //postindex
               cmp r2, #0x30              //checks if <='0'
               bllt _store_res
               cmp r2, #0x39              // checks if <= '9'
               blgt _is_upper
               addlt r2,r2, #5            // + 5
               cmp r2, #0x39              // modulo trick
               subgt r2, #10
               bl _store_res

_is_upper:   cmp r2, #0x5A              // checks if <= 'Z'
             blge _is_lower
             cmp r2, #0x41              // checks if >= 'A'
             addge r2, #32              // converts to lowercase
             bl _store_res

_is_lower:    cmp r2, #0x7A              //checks if <= 'z'
              blge _store_res
              cmp r2, #0x61              //check if >= 'a'
              subge r2, #32              //converts to uppercase

_store_res :  strb r2, [r6], #1
              subs r0, r0, #1           // decr bytes & update flags
              beq _output               // r0 = 0
              bl _loop                  // continue


_output  :   mov r2 , #0x0A         // append newline
             strb r2, [r6], #1
             mov r2, #0             //append null terminator
             strb r2, [r6]
             ldr r0, =trans_msg
             bl printf
             ldr r0, =buff_out
             bl printf

            mov r0, r5        //restore bytes number
            subs r0, r0, #32
            bne main           //loop back in main

            ldr r1, =buff_in
            mov r2, #32        //define max number of bytes again
            mov r7, #3
_discard:   svc #0
            subs r0, r0, #32
            bleq _discard       //read again until end of input
            bne main

_quit : ldr r0, =exit_msg
        bl printf
        mov r7, #1                  // syscall for exiting
        svc #0

.data
input_msg: .asciz "Input a string up to 32-bytes long\n"
exit_msg: .asciz "Exiting...\n"
trans_msg: .asciz "Transformig input string...\n"
buff_in:  .space 32
buff_out: .space 32
```

Κάποια παραδείγματα εκτέλεσης μέσα στο περιβάλλον QEMU φαίνονται παρακάτω:

```
root@debian-armel:/home/user# gcc ex1.s -o ex1
root@debian-armel:/home/user# ./ex1
Input a string up to 32-bytes long
Hello 42!
Transformig input string...
hELLO 97!

Input a string up to 32-bytes long
Team 35
Transformig input string...
tEAM 80

Input a string up to 32-bytes long
THIS IS A TEST OF a very long string to check overflow of buffer...
Transformig input string...
this is a test of A VERY LONG ST

Input a string up to 32-bytes long
q
Exiting...
root@debian-armel:/home/user#
```

Ερώτημα 2ο

Για την επικοινωνία μεταξύ host & guest χρησιμοποιήθηκε η 2η μέθοδος (χρήση flag *-serial pty*) και η εικονική σειριακή θύρα που δημιουργήθηκε από το QEMU είναι η */dev/ttyAMA0* στην πλευρά του guest και */dev/ttys004* στην πλευρά του host.

Δοκιμάστηκαν διαφορετικές τιμές baudrate και τελικά χρησιμοποίησα την 115200 ως default. Σημειώνεται ότι αυτή που οριζε το getty, το οποίο απενεργοποιήθηκε με comment out όπως αναφέρει η εκφώνηση, ήταν 9600. Οι υπόλοιπες τιμές του termios struct καθορίστηκαν και στα 2 μηχανήματα με τον ίδιο τρόπο θέτοντας απευθείας τα bits στην arm assembly και χρησιμοποιώντας τα αντίστοιχα macros στην C. Αξιοσημείωτα εδώ είναι μόνο τα flags *c_cflag* που χρειάζεται να προσδιορίσουμε ότι η μεταφορά γίνεται ανά byte με το options *CS8*, καθώς και το *c_lflag* το οποίο με option *ICANON* διαβάζει το input ως ολόκληρη γραμμή μετά από Enter και μας επιτρέπει να ελέγχουμε το end of input string συγκρίνοντας με τν χαρακτήρα '\n' στον assembly κώδικα.

Και στις 2 περιπτώσεις το άνοιγμα της σειριακής θύρας γίνεται σε blocking mode με δικαίωματα ανγάγνωσης και εγγραφής αφού και τα 2 μηχανήματα στέλνουν και λαμβάνουν δεδομένα. Ο host κοιμάται για 1s αφού στείλει τα δεδομένα για να εξασφαλίσω ότι ο guest έχει προλάβει να γράψει το αποτέλεσμα στον buffer του serial port, όμως μπορεί να παραλειφθεί μιας και το read είναι blocking.

Ακολουθεί ο κώδικας σε C που τρέχει στο host μηχάνημα:

host.c

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <stdio.h>

char buff[64];
char results[2];

```

```c
const char *serial_port = "/dev/ttys004";

int main() {
    ssize_t bytes;
    struct termios options;
    int fd;
    char res, freq;

    fd = open(serial_port, O_RDWR | O_NOCTTY);
    if (fd < 0) {
        perror("Error openning serial port");
        exit(1);
    }

    // Get current serial port settings
    if (tcgetattr(fd, &options) < 0) {
        perror("Error getting serial port attributes");
        close(fd);
        exit(1);
    }

    options.c_lflag = 0;
    options.c_lflag |= ICANON;
    options.c_cflag |= (CLOCAL | CREAD | CS8);
    options.c_cflag &= ~CSTOPB;
    options.c_cc[VMIN] = 1;
    options.c_cc[VTIME] = 0;

    if(cfsetispeed(&options, B115200) < 0 || cfsetospeed(&options, B115200) < 0) {
        perror("Error setting baudrate\n");
        exit(1);
    }

    if (tcsetattr(fd, TCSANOW, &options) < 0){
        perror("Error applying termios configurations");
        exit(1);
    }
    //clear terminal buffer
    tcflush(fd, TCIOFLUSH);

    printf("Input a string up to 64-bytes long:\n");
    bytes = read(STDIN_FILENO, buff, sizeof(buff));
    if (bytes < 0) {
        perror("Error reading input string");
        exit(1);
    }
    if (bytes == 64) {
        buff[63] = '\0'; //discard rest bytes
        tcflush(STDIN_FILENO, TCIOFLUSH);
        printf("Input string exceeded 64-bytes! Rest will be ignored\n", STDERR_FILENO);
    }

    // transfer input over serial port
    if (write(fd, buff, bytes) < 0) {
        perror("Error writing to serial port");
        exit(1);
    }
    printf("Bytes sent to guest successfully\n");

    // get result from serial port
    printf("Waiting for results...\n");
    sleep(1);
    if (read(fd, results, 4) < 0) {
        perror("Error reading results");
        exit(1);
    }
    res = results[0];
    freq = results[1];

    printf("The most frequent charachter is: %c and it appeared %c times.\n", res, freq);

    close(fd);
    return 0;
}
```

Στην συνέχεια παρατίθεται ο κώδικας σε arm assembly που τρέχει στο QEMU vm:

guest.s

```
.text
.align 4
.global main
.extern tcsetattr
.extern printf

main:
        ldr r0, =serial_port                    // device path
        ldr r1, =0x102                          // (O_RDWR | O_NOCTTY)
        mov r7, #5                              // open syscall
        svc #0

        mov r6, r0                              // save fd

        mov r0, r6
        ldr r2, =options
        mov r1, #0
        bl tcsetattr                            //apply configuration settings

        mov r0, r6                              //set fd
        ldr r1, =input
        mov r2, #64                             //max bytes read
        mov r7, #3                              //read syscall
        svc #0


            ldr r2, =freqs                      //holds base address

_count :    ldrb r3, [r1], #1                   //postindex increment
            cmp r3, #10                         //checks for EOF
            beq _find_max
            cmp r3, #32                         //check if space
            beq _count                          //skip
            ldrb r4, [r2, r3]
            add r4, r4, #1
            strb r4, [r2, r3]
            b _count


_find_max:  mov r0, #0                           //init counter
            mov r1, #0                          //init offset
            mov r2, #0                          //init max value
            ldr r3, =freqs

_loop :      ldrb r4, [r3], #1                  //postindex
             cmp r2, r4
             movlt r1, r0                        //new offset
             movlt r2, r4                        //new max value

            add r0, r0, #1
            cmp r0, #255
            bne _loop

            ldr r0, =output
            strb r1, [r0]                       //save offset
            strb r2, [r0, #1]                   //save max value

            mov r0, r6                          // fd
            ldr r1, =output
            ldr r2, =len_out
            mov r7, #4                          // write syscall
            svc #0

            mov r0, r6                          // fd
            mov r7, #6                          // close syscall
            svc #0

            mov r0, #0
            mov r7, #1                          // exit syscall
```

```
            svc #0


.data
    options: .word 0x00000000 /* c_iflag */
             .word 0x00000000 /* c_oflag */
             .word 0x000008BD /* c_cflag: CLOCAL | CS8 | CREAD */
             .word 0x00000002 /* c_lflag : ICANON */
             .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 0x00001002 /* c_ispeed 115200*/
             .word 0x00001002 /* c_ospeed 115200*/

serial_port: .asciz "/dev/ttyAMA0"
input: .space 64
/*holds a counter for every ascii charachter*/
freqs: .ascii "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
output:  .asciz "cc\n"
len_out = . - output
```

Για την εύρεση του στοιχείο με υψηλότερη συχνότητα, διατηρείται στο .data section ένα αρχικοποιημένο σε 0 διάστημα *_freqs* και το κύριο μέρος του κώδικα φορτώνει πριν το σημείο *_count* έναν δείκτη στο base address αυτού. Σε κάθε επανάληψη, φορτώνει την τιμή του στο offset ίσο με τον ASCII κωδικό του χαρακτήρα εισόδου (η arm assembly το επιτρέπει σε μια εντολή *ldrb [r<num> ,<offset>]*) και την αυξάνει κατά 1(το space ελέγχεται ξεχωριστά και δεν προσμετράται). Επειδή τα στοιχεία είναι το πολύ 64 ένα byte αρκεί για την καταμέτρηση του κάθε χαρακτήρα. Για την εύρεση του μεγίστου διατρέχει μια φορά τα στοιχεία του *_freqs* με postindex τρόπο και ανανεώνει το offset του μέγιστου όταν χρειαστεί πάλι με τις ειδικές εντολές υπό συνθήκη που προσφέρει ο arm επεξεργαστής (*movlt*). Τέλος προτιμάται η εντολή *svc* έναντι της *swi* για την πρόκληση των interrupts από τα read / write system calls.

Ακολουθούν κάποια παραδείγματα εκτέλεσης:

```
marialazou@mlazoy ex2 % sudo chmod 666 /dev/ttys004
marialazou@mlazoy ex2 % ./host
Input a string up to 64-bytes long:
Hello from host!
Bytes sent to guest successfully
Waiting for results...
The most frequent charachter is: o and it appeared 3 times.
marialazou@mlazoy ex2 % ./host
Input a string up to 64-bytes long:
11111111hmmmm
Bytes sent to guest successfully
Waiting for results...
The most frequent charachter is: 1 and it appeared 8 times.
marialazou@mlazoy ex2 %
```

Ερώτημα 3ο

Δημιουργήθηκαν 4 ξεχωριστά αρχεία για την κάθε συνάρτηση γραμμένα σε arm assembly όπως φαίνονται παρακάτω. Σε κάθε ένα από αυτά δηλώνεται ως .global το όνομα της συνάρτησης και ακολουθεί ο προσδιορισμός του τύπου τους ως function. Το αποτέλεσμα αποθηκεύεται στον register r0. Για την επιστροφή στο σωστό σημείο όλα τερματίζουν με την εντολή *bx lr* η οποία κάνει jump στο return address που βρίσκεται αποθηκευμένο στον lr. Ο ορισμός του .size στο τέλος βοηθάει τον linker να καθορίσει τα αρκιβή όρια του ορισμού της συνάρτησης στο symbol table.

strlen.s

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

strlen:
    mov r2, r0                  @ push base address
_cnt:
    ldrb r1, [r0], #1           @ string on r0
    cmp r1, #0                  @ checks if '\0'
    bne _cnt
_end:
    sub r0, r0, r2              @ len = (final offset - base offset)
    sub r0, r0, #1
    bx lr                       @return from func

.size strlen, .-strlen
```

strcmp.s

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

strcmp:
    mov r4, #0x0                @ initialize ret value
_match:
    ldrb r2, [r0], #1           @ load s1 character from r0
    ldrb r3, [r1], #1           @ load s2 character from r1
    cmp r2, r3
    bne _mismatch
    cmp r2, #0                  @ checks if '\0'
    beq _end                  @ if s1 == s2 return 0
    b _match

_mismatch:
    movlt r4, #0xffffffff       @ if r2 < r3 return -1
    movgt r4, #0x1              @ if r2 > r3 return 1

_end: mov r0, r4                @ pass result
      bx lr

.size strcmp, .-strcmp
```

strcpy.s

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

```

```
strcpy:
    mov r3, r0                  @ push base address of dest
_cpy:
    ldrb r2, [r1], #1           @ source on r1
    strb r2, [r0], #1           @ dest on r0
    cmp r2, #0                  @ checks if '\0'
    bne _cpy
_end:
    mov r0, r3                  @ restore dest's base address
    bx lr

.size strcpy, .-strcpy
```

strcat.s

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


strcat:
    mov r4, r0              @ push dest's base address
_consume:
    ldrb r2, [r0], #1       @ dest on r0
    cmp r2, #0              @ checks for '\0' in dest
    bne _consume            @ consume bytes
    sub r0, #1              @ overwrite '\0'
_concat:
    ldrb r3, [r1], #1       @ src on r1
    strb r3, [r0], #1       @ append
    cmp r3, #0              @ checks for '\0' in src
    bne _concat
_end:
    mov r0, r4              @ restore dest's base address
    bx lr

.size strcat, .-strcat
```

Για την μεταγλώττιση και σύνδεση των object files χρησιμοποιήθηκε το ακόλουθο Makefile που παράγει τα .o αρχεία μέσω pattern matching με τα κατάλληλα .s. Ακόμη για συντομία χρησιμοποιούνται τα $^ $< $@ arguement directives.

Makefile

```
CC = gcc
CFLAGS = -g -Wall -O2
OBJ = strlen.o strcmp.o strcpy.o strcat.o
ARM = strlen.s strcmp.s strcpy.s strcat.s

all : string_manipulation.out

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

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

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

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

Το αποτέλεσμα της make είναι το ακόλουθο:

```
root@debian-armel:/home/user/ex3# make
gcc -c string_manipulation.c -o string_manipulation.o -g -Wall -O2
gcc -c strlen.s -o strlen.o -g -Wall -O2
gcc -c strcmp.s -o strcmp.o -g -Wall -O2
gcc -c strcpy.s -o strcpy.o -g -Wall -O2
gcc -c strcat.s -o strcat.o -g -Wall -O2
gcc string_manipulation.o strlen.o strcmp.o strcpy.o strcat.o -o string_manipulation.out
```

Για την επιβεβαίωση της ορθής λειτουργίας του τελικού εκτελέσιμου χρησιμοποιήθηκε το ακόλουθο script που συγκρίνει τα αρχεία εξόδου του *string_manipulation.out* που παρήχθη από την παραπάνω διαδικασία με τις συναρτήσεις δηλωμένες ως extern, με εκείνα που δημιουργούνται από το ίδιο sourcefile *string_manipulation.c* κάνοντας include την <string.h> και χρησιμοποιώντας τις built-in συναρτήσεις με το ίδιο όνομα.

```
#!/bin/bash
declare -a files1=(
    "rand_str_input_first.txt_concat_out"
    "rand_str_input_first.txt_len_out"
    "rand_str_input_first.txt_sorted_out"
    "rand_str_input_sec.txt_concat_out"
    "rand_str_input_sec.txt_len_out"
    "rand_str_input_sec.txt_sorted_out"
)
declare -a files2=(
    "rand_str_input_first.txt_concat_out_libc"
    "rand_str_input_first.txt_len_out_libc"
    "rand_str_input_first.txt_sorted_out_libc"
    "rand_str_input_sec.txt_concat_out_libc"
    "rand_str_input_sec.txt_len_out_libc"
    "rand_str_input_sec.txt_sorted_out_libc"
)
for i in "${!files1[@]}"; do
    file1=${files1[$i]}
    file2=${files2[$i]}

    if cmp -s "$file1" "$file2"; then
        echo "Files $file1 and $file2 are identical."
    else
        echo "Files $file1 and $file2 are not identical."
    fi
done
```

Τρέχοντας το επιβεβαιώνω τα αποτελέσματα :

```
root@debian-armel:/home/user/outputs# ./cmp_results.sh
Files rand_str_input_first.txt_concat_out and rand_str_input_first.txt_concat_out_libc are identical.
Files rand_str_input_first.txt_len_out and rand_str_input_first.txt_len_out_libc are identical.
Files rand_str_input_first.txt_sorted_out and rand_str_input_first.txt_sorted_out_libc are identical.
Files rand_str_input_sec.txt_concat_out and rand_str_input_sec.txt_concat_out_libc are identical.
Files rand_str_input_sec.txt_len_out and rand_str_input_sec.txt_len_out_libc are identical.
Files rand_str_input_sec.txt_sorted_out and rand_str_input_sec.txt_sorted_out_libc are identical.
```