Σχεδιασμός Ενσωματωμένων Συστημάτων

Φιλιππόπουλος Ορφέας el18082
Παπαρρηγόπουλος Θοδωρής el18040

3η Εργαστηριακή Άσκηση

1. Μετατροπή εισόδου από τερματικό

Σκοπός αυτής της άσκησης είναι η ανάπτυξη μιας εφαρμογής σε ARM Assembly το οποίο δέχεται ως είσοδο μια ακολουθία το πολύ 32 χαρακτήρων, τη μετασχηματίζει και την τυπώνει πίσω στο terminal.

- Μετατρέπουμε τους πεζά γράμματα σε κεφαλαία
- Μετατρέπουμε τα κεφαλαία σε πεζά
- Αν υπάρχει χαρακτήρας αριθμού μεταξύ ‘0’ και ‘9’ προσθέτουμε 5 και παίρνουμε τον αριθμό mod 10
- Οι υπόλοιποι χαρακτήρες παραμένουν ίδιοι

Το πρόγραμμα είναι σε συνεχή λειτουργία και όταν πατηθεί μια συμβολοσειρά “q” ή “Q” τότε τερματίζει.

Αρχικά για την υλοποίηση της μετατροπής. Έστω χαρακτήρας c με ascii τιμή k

- Αν k <= 47 (το 48 είναι το ‘0’) τότε κρατάμε τον χαρακτήρα ώς έχει
- Αν k <= 57 (‘9’) τότε κάνουμε την μετατροπή των αριθμών. Δηλαδή προσθέτουμε 5 στην τιμή, και έπειτα, άμα ο αριθμός είναι μεγαλύτερος του 57 (‘5’ + 5 = 58) τότε αφαιρούμε 10.
- Αν k <= 64 (65 = ‘A’) τότε κρατάμε τον χαρακτήρα ώς έχει
- Αν k <= 90 (= ‘Z’) τότε απλά προσθέτουμε 32 στον ascii αριθμό
- Αν k <= 96 (97 = ‘a’) τότε κρατάμε τον χαρακτήρα ώς έχει
- Αν k <= 122 (= ‘z’) τότε απλά αφαιρούμε 32 στον ascii αριθμό
- Τέλος αν k > 122 τότε απλά τυπώνουμε το ίδιο.

Ορίζουμε έναν πίνακα για input και output 32 θέσεων και αποθηκεύουμε τις διευθύνσεις τους στους registers r5 & r7 αντίστοιχα (με ονόματα input_array και output_array αντίστοιχα).
Κρατάμε έναν counter για να διαβάσουμε το input στον register r4.
Έχουμε τον r10 για global counter, δηλαδή για το συνολικό μέγεθος του input.

Διαβάζουμε λοιπόν όλο το input μας, και κοιτάμε άμα το μέγεθος είναι ένας χαρακτήρας + ‘\n’ και τότε ελέγχουμε άμα αυτός ο χαρακτήρας είναι το ‘Q’ ή ‘q’. Τότε, απλά κάνουμε exit.

Σε κάθε άλλη περίπτωση πηγαίνουμε στο result_is label και όπου ξεκινάει η επεξεργασία των δεδομένων.

Εκεί τυπώνουμε το string “Result is: ” και προχωράμε στην convert όπου κοιτάμε τον πάνω πάνω χαρακτήρα τον μετατρέπουμε με βάση τα παραπάνω και εν τέλη μαζεύουμε τις μετατροπές στον πίνακα output.

Τέλος μόλις ολοκληρώσουμε την μετατροπή καλούμε την printf και τυπώνουμε έναν έναν του χαρακτήρες που έχουμε αποθηκεύσει στον πίνακα output_array.

Τον κώδικα του παραπάνω μπορεί κανείς να τον βρει στο αρχείο ./1st_ex/ex1_new.s

Παράδειγμα εκτέλεσης:

```
root@debian-armel:~/ex1_# ./ex1_new
Input a string of up to 32 chars long:hallothere12345!!!qq

Result is:HALLOTHERE67890!!!QQ
Input a string of up to 32 chars long:^[[A^X^C
root@debian-armel:~/ex1_# ./ex1_new
Input a string of up to 32 chars long:hAlOtHeRe12345!!qqQQ

Result is:HaLoThErE67890!!QQqq
Input a string of up to 32 chars long:iwillexitnow

Result is:IWILLEXITNOW
Input a string of up to 32 chars long:q
```

2. Επικοινωνία guest – host μηχανημάτω μέσω σειριακής θύρας

2.1 Εισαγωγή

Σκοπός της άσκησης είναι να γίνει ένα bind μεταξύ του host και του Qemu-guest. Το αντίστοιχο πρόγραμμα που τρέχει στον host είναι γραμμένο σε C, ενώ αυτό στο Qemu-guest είναι σε arm assembly. Το πρόγραμμα λοιπόν του host δέχεται ως είσοδο ένα string μεγέθους το πολύ 64 χαρακτήρων. Το string αποστέλλεται μέσω σειριακής θύρας στο guest μηχάνημα και αυτό υπολογίζει και απαντάει ποιος είναι ο χαρακτήρας του string με την μεγαλύτερη συχνότητα εμφάνισης και με το πλήθος των εμφανίσεων που είχε. Όταν δύο ή παραπάνω χαρακτήρες έχουν την ίδια συχνότητα τότε επιστρέφεται αυτός που έχει τον μικρότερο ascii αριθμό.

2.2 Υλοποίηση

2.2.1 Host

Το πρώτο πράγμα που υλοποιήθηκε είναι στο host να διαβαστεί η είσοδος του χρήστη με χρήση της **fscanf**. Στη συνέχεια ανοίξαμε τη κατάλληλη σειριακή (**πχ /dev/pts/7**, ότι έβγαζε όταν τρέχαμε το QEMU) και κάναμε **load τα default settings** σε ένα **struct** τύπου **termios**. Στη συνέχεια, μέσω της βιβλιοθήκης **termios.h**, **βάλαμε τα settings στη σειριακή** (φαίνονται στο κώδικα με πλήρη σχόλια) όπως πχ baudrate = 9600. Τέλος γράψαμε το input string στη παραπάνω σειριακή και **περιμένουμε** για την **απάντηση** από το **QEMU**.

2.2.2 Guest

Από την πλευρά του **QEMU**, και πάλι το πρώτο πράγμα που υλοποιήθηκε είναι να **ανοίξουμε** τη κατάλληλη **σειριακή** (**/dev/ttyAMA0**) και να τη βάλουμε τα **ίδια settings** με αυτά που βάλαμε και από τον **host**. Για να το πετύχουμε αυτό στο host υλοποιήσαμε το f**ind_flags.c** το οποίο απλά **τυπώνει** (σε decimal) τα **settings** της αντίστοιχης **σειριακής** (μπορείτε και εσείς να το τρέξετε και

να τα δείτε). Στη συνέχεια μέσω του πίνακα **options** και της **συνάρτησης tcsetattr περάσαμε** τα **configurations** που **βρήκαμε** μέσω του **find_flags.c** αρχείου. Ως εδώ έχουμε **configurar-αρει σωστά** τη **σειριακή**. Στη συνέχεια, **διαβάσαμε** τα τα **δεδομένα** του χρήστη (δηλαδή απλά διαβάσαμε τα περιεχόμενα της σειριακής). Μέσω **"hashing"** (δηλαδή ένας πίνακας 256 θέσεων που κάθε θέση-index είναι ο αντίστοιχος ascii και το value το frequency του) **βρήκαμε** τη **συχνότητα** του **κάθε χαρακτήρα** και τέλος **γράψαμε** τα **αποτελέσματα** στη **σειριακή** για να τα πάρει ο **host**.

Τους κώδικες των παραπάνω μπορεί κανείς να τους βρει στα αρχεία:

./2on_ex/host/host.c
./2on_ex/host/find_flags.c
./2on_ex/qutest-qemu/quest.s

Παράδειγμα εκτέλεσης:

Τρέχουμε το QEMU και παίρνουμε τη κατάλληλη σειριακή.

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/3rd_Lab/code_material/qemu$ ./serial_qemuu.sh
[sudo] password for hdron:
char device redirected to /dev/pts/3 (label serial0)
```

Τρέχουμε το host.c και γράφουμε μία συμβολοσειρά.

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/3rd_Lab/2nd_ex/host$ sudo ./host /dev/pts/3
Please give a string to send to host:
hi guest i am host!
```

Ο guest περιμένει να πατηθεί enter από τον host.

```
root@debian-armel:~/ex2_# ./guest
```

Πατάμε το enter στον host και παίρνουμε (στον host) το αποτέλεσμα:

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/3rd_Lab/2nd_ex/host$ sudo ./host /dev/pts/3
Please give a string to send to host:
hi guest i am host!
50
The most frequent character is
 h
and it appeared 2 times.
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/3rd_Lab/2nd_ex/host$
```

Μπορεί κανείς να επιβεβαιώσει πως τα **αποτελέσματα** είναι **σωστά**!

3 Σύνδεση κώδικα C με κώδικα assembly του επεξεργαστή ARM

3.1 Εισαγωγή

Σκοπός είναι η σύνδεση κώδικα C με συναρτήσεις γραμμένες σε assembly του ARM. Ειδικότερα θέλουμε να υλοποιήσουμε μερικές συναρτήσεις της string.h της C οι οποίες χρησιμοποιούνται στο πρόγραμμα string_manipulation.c

Συγκεκριμένα θέλουμε να υλοποιήσουμε:

- **size_t strlen(const char *s);**
  Επιστρέφει τον αριθμό των bytes του s χωρίς 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 του s1 και ταυτόχρονα επιστρέφει και έναν pointer στην διεύθυνση αυτή.
- **int strcmp(const char *s1, const char *s2);**
  Συγκρίνει της 2 συμβολοσειρές, αν η πρώτη λέξη είνια μεγαλύτερη επιστρέφει 1, -1 αν είναι μιρκότερη και 0 αν είναι ίσες.
- **char *strcat(char *s1, const char *s2);**
  Κάνει concatenate στο s1 τα περιεχόμενα του s2 κάνοντας overwrite τον τερματικό χαρακτήρα του s1 και επιστρέφει επίσης την διεύθυνση του πίνακα.

3.2 Υλοποιήσεις

Αρχικά δημιουργήσαμε ένα αρχείο “.s” για κάθε συνάρτηση, και αντίστοιχα υλοποιήσαμε 4 test files στον φάκελο tests τα οποία χρησιμοποιήθηκαν για να τεστάρουμε τις 4 αυτές συναρτήσεις.

<u>Συγκεκριμένα:</u>

- **strlen.s**
  Η συνάρτηση δέχετε στον r0 τη διεύθυνση μιας συμβολοσειράς και επιστρέφει το μέγεθος της συμβολοσειράς που δείχνει. Για την υλοποίηση ξεκινάμε τον counter (r2 register) από το 0, φορτώνει το πρώτο element του πίνακα, κοιτάει αν είναι το τερματικό, άμα δεν είναι αυξάνει τον counter και αυξάνει την θέση του πίνακα, αλλιώς τερματίζει και επιστρέφει το μέγεθος.

```
1   .text
2   .align 4
3   .global strlen
4   .type strlen, %function
5
6   @ r0: string address
7   @ r0 is the const char *s
8   strlen:
9       PUSH {ip, lr}
10      mov r2, #0      @ initialise the counter to 0
11
12  loop:
13      ldrb r1, [r0], #1 @ r1 = s[0] ++s;
14      cmp r1, #0
15      beq exit @ if s ended jump to exit
16
17      add r2, #1 @ increase counter (counter++)
18      b loop
19  exit:
20      mov r0, r2 @ return the counter
21      POP {lr, ip}
22      bx lr
23
```

- **strcpy.s**
  Η συνάρτηση δέχεται ως ορίσματα 2 πίνακες στους καταχωρήτες r0 & r1 και θα αντιγράψει τα δεδομένα του r1 στον r0. Οπότε έχουμε μια λούπα όπου φορτώνουμε το πρώτο στοιχείο του r1, και το αντιγράφουμε στον r0, και έπειτα αυξάνουμε τις διευθύνσεις των πινάκων.

```
1   .text
2   .align 4
3   .global strcpy
4   .type strcpy, %function
5
6   @ r0: dest, r1: src
7   strcpy:
8       PUSH {ip, lr}
9       mov r3, r0  @ copy the dest address
10
11  loop:
12      ldrb r2, [r1], #1   @ r2 = src[0] & ++src;
13      strb r2, [r0], #1    @ dest[0] = r2 & ++dest;
14      cmp r2, #0      @ If the src hasn't ended yet
15      bne loop
16
17  exit:
18      mov r0, r3  @ restore and return destination's initial address
19
20      POP {lr, ip}
21      bx lr
22
```

- **strcmp.s**
  Στην cmp λοιπόν περνάμε πάλι 2 πίνακες όπου θέλουμε να τους συγκρίνουμε. Άμα όλα τα στοιχεία είναι ακριβώς ίδια, μόνο τότε θα επιστρέψει 0. Σε κάθε άλλη περίπτωση με το που ανιχνεύσει 2 διαφορετικούς χαρακτήρες για τις αντίστοιχες συμβολοσειρές τότε κάνουμε jump στο notEqual όπου και επιστρέφουμε τον αντίστοιχο αριθμό.

```
1   .text
2   .align 4
3   .global strcmp
4   .type strcmp, %function
5
6   @ r0: str1, r1: str2
7   @ if str1 == str2 return 0
8   strcmp:
9       PUSH {ip, lr}
10      mov r4, #0x0 @ r4 keeps the return value, we will mov it to r0 in the end
11  loop:
12      ldrb r2, [r0], #1   @ r2 = s1[0] and ++s1
13      ldrb r3, [r1], #1   @ r3 = s2[0] and ++s2
14      cmp r2, r3 @ if r2 != r3 jump to notEqual
15      bne notEqual
16
17      cmp r2, #0      @ if we parsed s1 jump to exit
18      beq exit
19
20      b loop          @ else continue looping
21
22  notEqual:
23      movlt r4, #0xffffffff   @ if s1 < s2 -> -1
24      movgt r4, #0x1      @ if s1 > s2 -> 1
25
26  exit:
27      mov r0, r4  @ return the result(r4) into r0
28      POP {lr, ip}
29      bx lr
```

- **strcat.s**
  Κάνει concatenate στο s1 τα περιεχόμενα του s2 κάνοντας overwrite τον τερματικό χαρακτήρα του s1 και επιστρέφει επίσης την διεύθυνση του πίνακα. Είναι ακριβώς ίδιας λογικής με τα προηγούμενα.

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

@ r0: dest, r1: src
strcat:
    PUSH {ip, lr}

    mov r6, r0  @ r6 is the address of our destination string
    mov r7, r1  @ r7 is the address of our source string

loop:
    ldrb r2, [r0], #1 @ r2 = string1[0] & ++string1
    cmp r2, #0      @ If the first string ended continue to sub
    bne loop

    @ we have passes string1
    @ r2 = '\0' and r0 is len + 1 so we need to subtract 1 to override '\0'
    sub r0, #1

concat:
    ldrb r3, [r1], #1   @ r3 = string2[0] & ++string2
    strb r3, [r0], #1   @ string1[0] = r3 & ++string1
    cmp r3, #0     @ If string2 ended continue to exit
    bne concat

exit:
    mov r0, r6  @ restore and return string1
    mov r1, r7  @ restore string2
    POP {lr, ip}
    bx lr
```

Παραδείγματα εκτέλεσης:

```
#include <stdio.h>

extern char *strcat(char *dest, const char *src);

int main() {
   char str1[100] = "This is ", str2[] = "our test";
   char *s;
   // concatenates str1 and str2
   // the resultant string is stored in str1.
   s = strcat(str1, str2);

   printf("%s\n%s\n%s\n", str1, str2, s);

   return 0;
}
```

```
root@debian-armel:~# gcc -g test_strcat.c -c
root@debian-armel:~# gcc -g strcat.s -c
strcat.s: Assembler messages:
strcat.s:31: Warning: register range not in ascending order
root@debian-armel:~# gcc test_strcat.o strcat.o -o main
root@debian-armel:~# ./main
This is our test
our test
This is our test
```

```
#include <stdio.h>

extern size_t strlen(const char *s);

int main(int argc, char *argv[])
{
    printf("Length of the string is: %d\n", strlen(argv[1]));
    return 0;
}
```

```
root@debian-armel:~# gcc -g test_strlen.c -c
root@debian-armel:~# gcc -g strlen.s -c
strlen.s: Assembler messages:
strlen.s:21: Warning: register range not in ascending order
root@debian-armel:~# gcc -g strlen.o test_strlen.o  -o main
root@debian-armel:~# ./main "This is a large test"
test
Length of the string is: 20
root@debian-armel:~#
```

```c
#include <stdio.h>

extern char *strcpy(char *dest, const char *src);

int main(int argc, char *argv[])
{
    char dest[100];
    char *test;

    test = strcpy(dest, argv[1]);

    printf("Initial string was %s\nCopy is: %s\ntest: %s\n", argv[1], dest, test);
    return 0;
}
```

```
root@debian-armel:~# gcc -g test_strcpy.c -c
root@debian-armel:~# gcc -g strcpy.s -c
strcpy.s: Assembler messages:
strcpy.s:20: Warning: register range not in ascending order
root@debian-armel:~# gcc strcpy.o test_strcpy.o -o main
root@debian-armel:~# ./main "This is the initial text"
Initial string was This is the initial text
Copy is: This is the initial text
test: This is the initial text
root@debian-armel:~#
```

```c
#include <stdio.h>

extern int strcmp(const char *s1, const char *s2);

int main(int argc, char *argv[])
{
    int result;

    result = strcmp(argv[1], argv[2]);
    printf("String 1 - String 2: %d\n", result);

    result = strcmp(argv[2], argv[1]);
    printf("String 2 - String 1: %d\n", result);

    result = strcmp(argv[1], argv[3]);
    printf("String 1 - String 3: %d\n", result);

    result = strcmp(argv[2], argv[3]);
    printf("String 1 - String 3: %d\n", result);

    result = strcmp(argv[3], argv[3]);
    printf("String 3 - String 3: %d\n", result);

    return 0;
}
```

```
root@debian-armel:~# gcc -g test_strcmp.c -c
root@debian-armel:~# gcc -g strcmp.s -c
strcmp.s: Assembler messages:
strcmp.s:28: Warning: register range not in ascending order
root@debian-armel:~# gcc -g test_strcmp.o strcmp.o -o main
root@debian-armel:~# ./main String1 String12 String1
String 1 - String 2: -1
String 2 - String 1: 1
String 1 - String 3: 0
root@debian-armel:~#
```