CompArch Project 1 Documentation

Frank Zhang, Allister Liu

Prof. Billoo

CU_ECE251

March 18, 2019

Goal&Requirements

The goal of Project 1 is to build a program that will ask the user to input 2 strings, each with at most 10 ASCII characters, and concatenate the two. The program will ask the user to enter the first string after which they will hit enter. If the number of characters exceeds the 10 character limit, it will print an error and exit the program with a 21 error code. If the number of characters is less than 10 characters, the user will be asked to enter another string, and the program will do the same check and error out if necessary. However, if the second string exceeds the character limit, the user will exit with a 22 return code. Otherwise, if both strings are under 10 ASCII characters, the program will output a string that concatenates the first string and the second string together, and it will return an error code equals to the total number of characters in the concatenated string.

Overall-Architecture

The program is generally composed of three parts: getting and validating inputs, concatenation, and printing output. It will receive each one of the two inputs from the users respectively, check the length, and store them in two 16-byte memory spaces (the reason why 16 bytes but not 12 bytes is explained in the challenge section). Then, the program will copy each one of them into a 24-byte memory space using a loop, in which the loop counter will also give the length of the string. At last, the concatenated string will be printed and the length will be returned. In the case when the length of input exceeds 10, a separate function will print the error message and return the error code.

The program will get the first inputs from the user and check if either of the two strings exceeds the 10-character limit. It uses scanf() function to get inputs from the user, and then branch to a function to check the length called boundary_judge. It checks if the 11th element of the string is empty to determine if the string is under 10 characters. If the string length is shorter then or equal to 10, the 11th byte must be all zero (memory are initialized to zero) or the null terminator '\0', which also has the ASCII code 0. If the first string passes the check, the program will keep going and do the same thing for the second string.

Assume both two strings pass the check, to concatenate the two input strings, the program uses the concept of a for loop. By using r3 as an incrementing counter, It will load the r3th element in the input string and load to the r3th position after the output address, and then r3 increment by 1, until the character copied is 0. The first input string will go through the loop, and after it is finished, r3 - 1, which is equal the length of the string, will be added into r5. Then, the address of output string will be added by the length so now it points to the empty position right after the end of the first string. After that, the second string will go through the same loop, and its length will be added to r5 also, so r5 now contains the total length of the concatenated string. Because both two input strings use the same loop, a cat_judge function is used to differentiate them. It compares the address of the current input string with the address of the second sting. If they are the same, that means copying the second string is finished

After the concatenation, printf() function is used to print out the output, and then r5 value, which is the length of the string 1, will be copied to r0 to return the length. During the input, when the program detects one string exceeding the length of 10, the program will branch to a function called error. use printf() to print an error message and exit with corresponding error code--same principle as the normal exit but no concatenation will be done.

Challenge

The most prominent technical challenge in this program is the input/output, as it requires deep knowledge on Linux. However, the standard c library provides accesses to the standard input/output in Linux, such as scanf() and printf() under the stdio library. Therefore, we don’t need to develop the code to receive input and give output in the shell but can link to the library directly. There are other I/O functions available like gets() and puts(), but scanf() and printf() can format the I/O strings and is more commonly used, so we stick to them. Some other technical challenges include allocating memory spaces to hold strings, which we solved by referring to the code presented in class.

The trickiest problem during the debugging is when linking to external functions. We didn’t realize that external functions can alter the value in r0-r3 registers, so we couldn’t reuse the addresses in those registers. For example, when using the scanf() function, the address of the format string is loaded into r0, and after that we wanted to reuse the same address for the second scanf(), which is impossible because the first scanf() had already modified r0. Hence, the program threw a segmentation fault because we referred to an invalid address. After examing the code with breakpoints in GDB and checking the registers’ value, we found and solved this issue.

Another problem is that when the input string has length 9 or 10, the program will throw a segmentation fault. This problem only happened when the string is long, so we guess that something passed the memory boundary. Because each char takes one byte, 12-byte memory should be enough for a string with length 10, even including the null terminator. However, while the array size declared is 12 which is enough, we suspected that there are some special mechanisms in forming a string array, such as some array guard byte, so the array size wasn't enough. By expanding the array to 16 bytes (4-aligned, so the multiple of 4), this problem is solved. However, the exact mechanism is still unknown.

A minor problem is some typos. Without noticing, we carelessly declare the the address of input string 1 to be the address of input string 2, causing the program to repeat the string 2 twice. Thankfully we quickly noticed and corrected the typo, and it didn't consume too much time.