Computer Architecture - Assembly
DM548

Michelle Dung Hoang
Danny Rene Jensen
Lea Fog-Fredsgaard

Instructor: Richard Röttger
Teaching assistent: Svend Knudsen

8. november 2015

1 Introduktion:

This project is to create a program in Assembly. This program needs to able to execute following requirements. The program needs to be able to search for a given string (the query) in a text file. The program needs to be able to pass the input as command line arguments. The program should be able to, open a file and read it into memory. When done searching for a given string, the program should print the string onto the screen and return its character position in the text file. In case where the string does not exist in the text file, the program should output the value -1.

2 Design

The algorithm we used for this project is the naive string search algorithm, which is a string matching algorithm, that will return where one pattern is found within a text. This algorithm will go through and test all possibilities to match the pattern.

3 Implementation

Short program description:
The program reads the second argument on std input and opens the file with queries and saving it in memory. The length of the file is stored in a register.

```
1  _start:
2    # file 1
3    # Open the file
4    mov $2, %rax
5    mov 16(%rsp), %rdi
6    mov $0, %rsi
7    mov $2, %rdx
8    syscall
9
10   mov %rax, %rdi
11
12   call get_file_size
13   mov %rax, %r10
14   call allocate_memory
15   mov %rax, %r13
16
17   # Read the contents of the file into allocated memory
18   mov $0, %rax
19   mov %r13, %rsi
20   mov %r10, %rdx
21   syscall
```

The program reads the third argument on std input and opens the file with text and saving it in memory. Also saving the length in a register. Same procedure as above, except it is the third argument read:

```
mov 24(%rsp), %rdi
```

The program reads each character one at a time, printing it to the screen (std output) until it reaches newline, while searching the text for the first match. When matching the query string and the text, the program reads and compares one character at a time. If a character in the text does not match the query, the program will loop back at the beginning of the query while proceeding through the text file.

```
# Loop over each letter and compare them
   mov $0, %rdx
   mov %r9, %r12
   cmp $0, %r11
    je end_loop

   movb (%r12), %bl
   movb (%rsi), %al
   inc %r14
   cmp %al, %bl
    je equal_loop

   inc %rsi
   mov $0, %rdx
   sub $1, %r11
   cmp $0, %bh
    jne jmp_back
   jmp_back_done:

   jmp place_loop

  equal_loop:
   inc %rdx
   inc %r12
   inc %rsi
   mov $1, %bh
   sub $1, %r11
   movb (%r12), %bl
   cmp $10, %bl
    je sw_found

   jmp place_loop
```

When finding a match the character position is printed to the screen (see code bellow). The program proceeds with the next query, starting at the beginning of the search text.

```
print_place:
   mov %r15, %rsi
   mov %r8, %r11
   mov $1, %r14
   mov $0, %bh
```

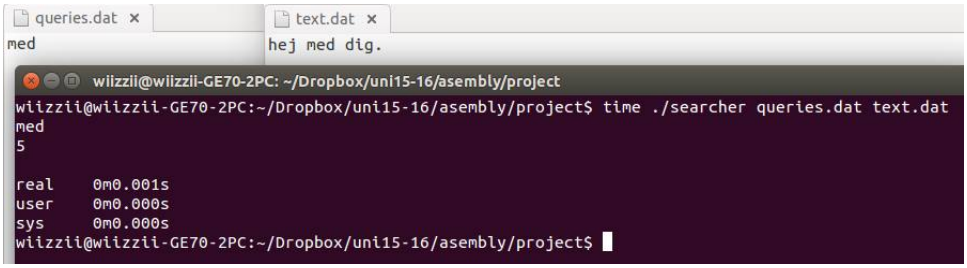
If there are no matching strings the program will print -1 to the screen.

```
1  end_loop:
2    mov $1, %rax
3    mov $1, %rdi
4    mov $notfound, %rsi       #prints -1 f the search word is not in text
5    mov $4, %rdx
6    syscall
7    jmp run_search_words_done
```

4 Test

For testing we used a laptop with these specifications: CPU: 4th generation Intel® Core™ i7-4710HQ Processor Settings: 2.5GHz, 6 MB CHASE.

A test with a short query file and a short text file is shown below. The test shows that the query and the position is printed to the screen, as well as the time it took to find the query.



In the table below, are descriptions of the the test we have run with our program. We tried testing with a search string with different lengths. Searching for the given string, within a string. Searching for short strings and long strings. From testing our program, we can conclude, that it indeed is able to find a given string fast. When searching for longer strings, the run time seems to be longer.

	Query length	Numer of queries	Text length	Real time	User time	Sys time
Test 1	3	1	50000	0.001s	0.000s	0.000s
Test 2	3	1	12	0.001s	0.000s	0.000s
Test 3	30	15	12	0.001s	0.000s	0.000s
Test 4	30	15	50000	0.001s	0.000s	0.000s
Test 5	30000	400	12	0.018s	0.000s	0.016s
Test 6	30000	1	12	0.013s	0.000s	0.000s
Test 7	30000	400	50000	0.080s	0.024s	0.036s
Test 8	30000	1	50000	0.011s	0.000s	0.008s

5 Conclusion

The program is able to pass the input as command line arguments. With the search program as first argument, the file with the queries as second argument and the text file as third argument. The program is able to open and read the file with query strings as well as the text file, into two different buffers. A pointer points to a specific character in the string, comparing it with all the characters in the text file, one character at the time. When searching for a given string the program output the string and its character position in the text file. If the string doesn't appear in the text, the output is the string and -1, as specified.