

Computer Architecture - Project I

Professor Billoo

Shyam Paidipati & Kevin Kerliu

Design Rationale

The following sections include an explanation and discussion of the fulfilled requirements of the project, the program architecture, and the challenges faced in completing the project.

Fulfillment of Requirements

Once the user compiles the source code using the Makefile, as explained in the README, the user can run the executable. Upon running the executable, the user will be prompted to input a string. Once the user enters the first string, the program will check whether the number of characters exceeds the 10-character limit. If the string does exceed this limit, then the program will exit with a 21 error-code. If the number of characters is less than 10 characters, then the user will be prompted to input a second string. The program repeats the same check, but if the second string exceeds the character limit, then the program will instead exit with a 22 error-code. If the user successfully inputs both the first and second string, then the program will output the concatenated string to stdout and return an error-code that consists of the total number of characters in the concatenated string.

Program Architecture

The source code begins with the data section, which is dedicated to 4-byte aligning all the prompts, input, and output values. Specifically, the prompt for the user to input the first string, the prompt for the user to input the second string, the prompt that indicates that the string is too

long, and the prompt that indicates the output, that is, the concatenated strings are all declared. Furthermore, the return values for each scenario and each of the strings are also all declared.

Next, in the text section of the source code, main stores the original address of the link register so that we can later return to main while still having its original contents intact. The following part of main is dedicated to calling printf to prompt the user to input a string and calling scanf to store the input of the user. The method then branches to loop1.

The purpose of loop1 is to iterate through each of the characters of the string that the user has inputted and find the total number of characters in the string. Moreover, it also copies each character into another string that will later serve as the concatenated string. The loop does this by looping back to itself until it branches to err1 or prompt2. The loop will branch to err1 if the inputted string is greater than 10 characters long and exit with a 21 error-code. Otherwise, it will branch to prompt2 once the null character is reached.

Prompt2 essentially follows the same logic as the second half of main, which prompts the user to input a string, and collects the response and stores it in memory. Prompt2 then branches to loop2 unconditionally.

The purpose of loop2 is exactly that of loop1, with one small caveat, that is, it branches to err2, will return a 22 error-code in the case that the inputted string is greater than 10 characters long. Otherwise, it will branch to end once the null character is reached.

End returns the final concatenated string and exits with the total number of characters in this string. It exits the program by returning the original link register contents back into the link register.

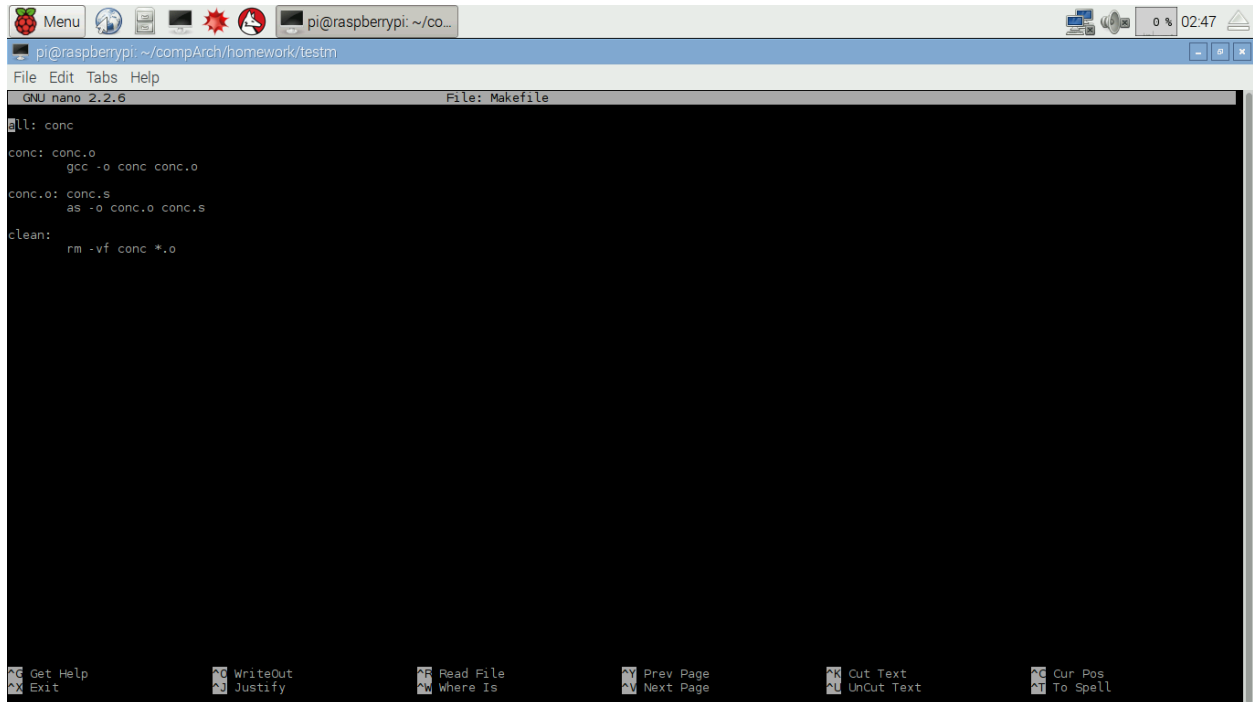
Challenges

One of the challenges that was encountered was when there was a space in between characters in one of inputted string from the user. The program would interpret this as two strings and proceed to exit out of the program by concatenating the two fragments together. This problem was remedied by using different format pattern for our scanf function. Instead of using “%s” as the format, “ %s[^\n]” was used, allowing scanf to ignore any spaces between the fragments.

A few functions used from the C standard library were printf and scanf. The reasoning behind this choice is because the write to standard output, which is the terminal. This allows the user to see the prompts. Additionally, these functions abstract out some of the complexities of reading to and writing from a device.

Makefile

Below please find a copy of the Makefile for the project.



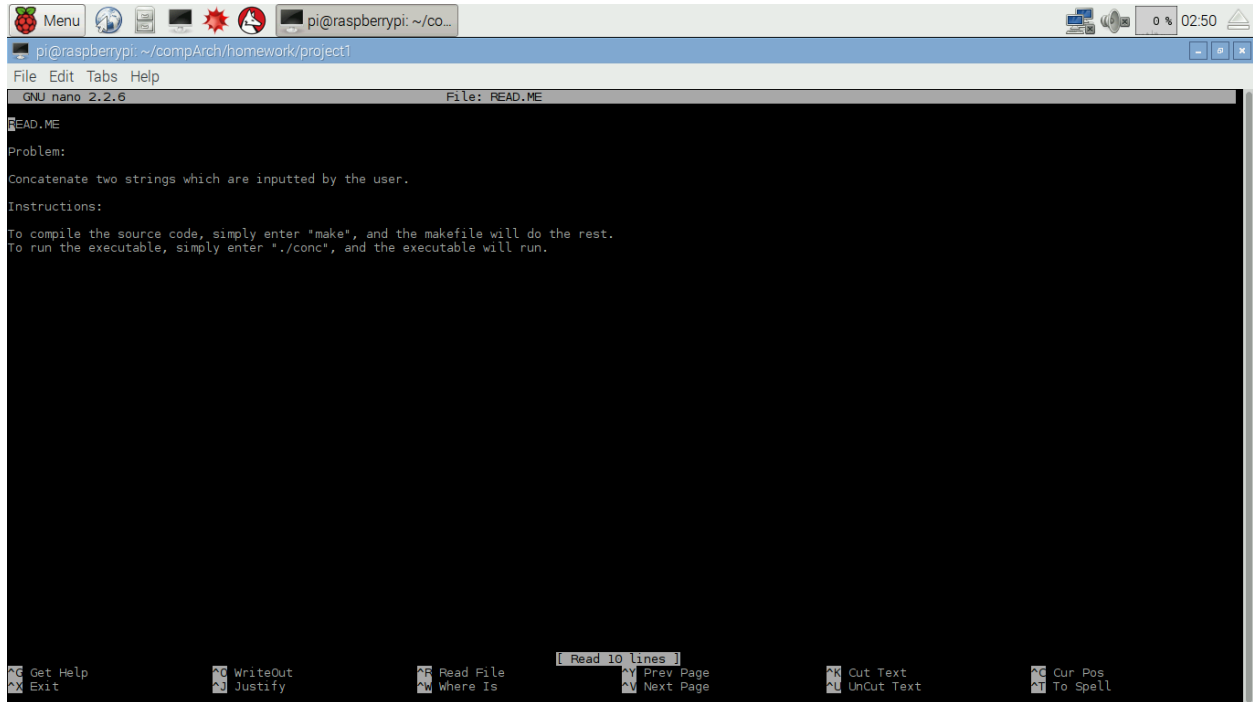
The screenshot shows a terminal window on a Raspberry Pi. The window title is "pi@raspberrypi: ~/compArch/homework/testm". The terminal displays the contents of a Makefile being edited in the nano 2.2.6 editor. The Makefile content is as follows:

```
all: conc
conc: conc.o
    gcc -o conc conc.o
conc.o: conc.s
    as -o conc.o conc.s
clean:
    rm -vf conc *.o
```

The nano editor interface includes a menu bar with "File", "Edit", "Tabs", and "Help". The status bar at the bottom shows the GNU nano version (2.2.6) and the file name (File: Makefile). The bottom status bar also contains various keyboard shortcuts for navigation and editing, such as ^G Get Help, ^X Exit, ^O WriteOut Justify, ^R Read File Where Is, ^Y Prev Page Next Page, ^K Cut Text UnCut Text, and ^C Cur Pos To Spell.

README

Below please find a copy of the README for the project.



The screenshot shows a terminal window on a Raspberry Pi. The top bar indicates the user is 'pi' at 'raspberrypi' in the directory '~/co...'. The terminal title is 'pi@raspberrypi: ~/compArch/homework/project1'. The nano text editor is open, editing 'File: README.ME'. The editor's status bar shows 'GNU nano 2.2.6'. The content of the file is as follows:

```
README.ME
Problem:
Concatenate two strings which are inputted by the user.
Instructions:
To compile the source code, simply enter "make", and the makefile will do the rest.
To run the executable, simply enter "./conc", and the executable will run.
```

The bottom of the terminal shows the nano editor's command shortcuts:

^G Get Help	^O WriteOut	^R Read File	^Y Prev Page	^K Cut Text	^C Cur Pos
^X Exit	^J Justify	^W where Is	^V Next Page	^U UnCut Text	^T To Spell