Libasm

Assembly yourself!

Summary: The aim of this project is to get familiar with assembly language.

Version: 5

Contents

Chapter I

Introduction

An assembly (or assembler) language, often abbreviated asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture. In contrast, most high-level programming languages are generally portable across multiple architectures but require interpreting or compiling. Assembly language may also be called symbolic machine code.

Chapter II

Common Instructions

- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and you will receive a `0` during the evaluation.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`. And must recompile/relink only necessary files.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, libraries or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.
- You must write 64 bits ASM. Beware of the "calling convention".
- You can't do inline ASM, you must do '.s' files.
- You must compile your assembly code with nasm.
- You must use the Intel syntax, not the AT&T.

`It is forbidden to use the compilation flag: -no-pie.`

Chapter III

Mandatory part

- The library must be called libasm.a.
- You must submit a main that will test your functions and that will compile with your library to show that it's functional.
- You must rewrite the following functions in asm:
    - ft_strlen (man 3 strlen)
    - ft_strcpy (man 3 strcpy)
    - ft_strcmp (man 3 strcmp)
    - ft_write (man 2 write)
    - ft_read (man 2 read)
    - ft_strdup (man 3 strdup, you can call to malloc)
- You must check for errors during syscalls and properly set them when needed
- Your code must set the variable errno properly.
- For that, you are allowed to call the `extern ___error or errno_location`.

Chapter IV

Bonus part

You can rewrite these functions in asm. The linked list function will use the following structure:

```
typedef struct                  s_list
{
  void                          *data;
  struct s_list         *next;
}                               t_list;
```

- ft_atoi_base (like the one in the piscine)
- ft_list_push_front (like the one in the piscine)
- ft_list_size (like the one in the piscine)
- ft_list_sort (like the one in the piscine)
- ft_list_remove_if (like the one in the piscine)

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.