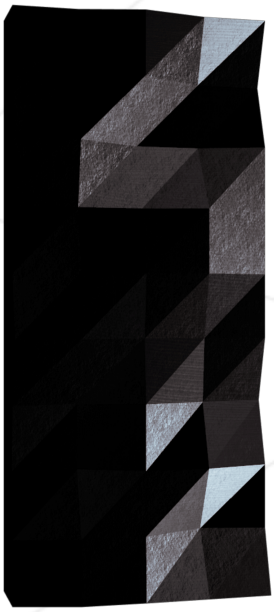

Libftasm

Assembleur toi-même !

42 staff staff@42.fr

Résumé: Ce projet a pour but de vous familiariser avec le langage ASM.

Table des matières

Chapitre I

Préambule

Un langage d'assemblage ou langage assembleur est, en programmation informatique, un langage bas niveau qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des symboles dits « mnémoniques » (du grec mnêmonikos, relatif à la mémoire), c'est-à-dire faciles à retenir. Le programme assembleur convertit ces mnémoniques en langage machine en vue de créer par exemple un fichier objet ou un fichier exécutable.

Wikipedia

Chapitre II

Sujet

II.1 Consignes

- Ce projet ne sera évalué que par des humains.
- La bibliothèque doit s'appeller `libfts.a`.
- Vous devez compiler votre code assembleur avec `nasm`.
- Vous devez écrire de l'ASM 64 bits. Prenez garde à la "calling convention".
- Vous ne devez pas faire de l'ASM inline, vous devez faire des '.s'.
- Vous devez utiliser la syntaxe `Intel`, pas `AT&T`.
- Vous devez rendre un Makefile qui compile votre bibliothèque, et qui doit contenir les règles habituelles.
- Vous devez rendre un `main` qui testera vos fonctions et qui pourra compiler avec votre bibliothèque pour prouver son bon fonctionnement.
- Une fois le projet validé, vous pourrez utiliser vos fonctions en assembleur à la place de vos fonctions `C` dans votre `libft` si vous le souhaitez.

II.2 Introduction à libftasm

Le projet `libasm` a pour but de vous faire coder une minilib en ASM. Vous allez devoir recoder quelques fonctions basiques de la libc afin d'en générer une bibliothèque. A la fin de ce projet vous serez familiarisés avec la syntaxe du langage, le fonctionnement de la stack, mais aussi le comportement du compilateur.

II.2.1 Part 1 - Fonctions simples de la libc

Dans cette première partie, vous devez recoder un ensemble de fonctions de la `libc` telles que décrites dans leur `man` respectif sur votre système. Vos fonctions devront avoir exactement le même prototype et le même comportement que les originales. Leur nom devra être préfixé par `"ft_"`. Par exemple `strlen` devient `ft_strlen`.

Vous devez recoder les fonctions suivantes :

- `bzero`
- `strcat`
- `isalpha`
- `isdigit`
- `isalnum`
- `isascii`
- `isprint`
- `toupper`
- `tolower`
- `puts` (bien entendu, vous pouvez appeller le syscall `write`)

II.2.2 Part 2 - Fonctions simples mais un peu moins de la libc

Dans cette deuxième partie, vous devez recoder encore des fonctions de la libc, mais avec les Instruction Repeat String Operations.

Un peu de documentation

Vous devez recoder les fonctions suivantes :

- `strlen`
- `memset`
- `memcpy`
- `strdup` (bien entendu, vous pouvez appeller `malloc`)

II.2.3 Part 3 - Cat

Pour finir, vous devez coder une fonction `ft_cat` qui prendra un `file descriptor` (par exemple `0`...) en paramètre et qui auras le même comportement que la commande cat, elle retournera void.

`Attention le changement de contexte entre l'user-space et le kernel-space coûte cher en terme de performances donc vous serez pénalisés si vous en abusez.`

II.3 Partie bonus

Pour la partie bonus vous êtes libres d'ajouter d'autres fonctions de votre choix (de la `libc` ou non) à votre `libftasm`. L'intégralité de la partie obligatoire de ce projet doit être parfait pour que vos bonus soient pris en compte.