Лабораторна робота №3. Асемблер

Мета

Метою виконання цього комп'ютерного практикуму є знайомство з мовою Асемблера і використання її для вирішення задач управління ОС.

В результаті його виконання будуть отримані базові знання з мови Асемблера і засвоєні навички написання програм з використанням цієї мови.

Завдання

Необхідно написати програму на мові Асемблера для Linux (з використанням GAS або NASM), яка виконує те ж завдання, що і в роботі №2.

Приклади програм на Асемблері

Програма "Hello world", яка використовує для роботи тільки системні виклики

```
.data
    hello_str:
        .string "Hello, world!\n"

.text
    .globl  main

    main:
        // системний виклик write
        movl    $4, %eax // номер виклику — 4
        movl    $1, %ebx // запис до STDOUT (fd 1)
        movl    $hello_str, %ecx // що пишемо?
        movl    $14, %edx        // довжина рядку
        int     $0x80 // переривання 0x80

        // системний виклик exit
        movl    $1, %eax // номер виклику — 1
        movl    $0, %ebx // код повернення — 0
        int     $0x80
```

Для того щоб запустити цю програму, її треба зкомпілювати і зібрати:

* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting

```
as hello.s -o hello.o
ld hello.o -o hello
```

Аналог цієї програми мовою C:

```
#include <unistd.h>

int main() {
    char hello_str[] = "Hello, world!\n";
    write(1, hello_str, sizeof(hello_str) - 1);
    _exit(0);
}
```

Програма, що друкує свої аргументи командного рядка. Використовує бібліотечну функцію puts

Програма мовою C:

```
#include<stdio.h>

int main(int argc, char** argv) {
    int i = 0;
    int j = argc;
    do {
        puts(argv[i]);
    } while (--j > 0);
}
```

Програма мовою Асемблера:

```
.section .text
.globl _start

_start:
    // запам'ятати поточне положення верхівки стеку
    movl  %esp, %ebp
    // на горі стеку — argc
    // записати його до лічильника
    movl  (%ebp), %esi
    // edi — номер аргументу
    movl  $1, %edi

print_loop:
    // зчитування пам'яті за адресою ebp + edi*4
```


* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting

```
    // за цією адресою аргумент argv[edi]
    movl  (%ebp, %edi, 4), %eax

    // виклик puts: аргумент передається через стек
    pushl %eax
    call  puts

    // змінюємо лічильники
    // перевіряємо, чи не пора завершувати цикл
    incl  %edi
    decl  %esi
    test  %esi, %esi
    jnz   print_loop

    // вихід
    movl  $1, %eax
    movl  $0, %ebx
    int   $0x80
```

Для того щоб запустити цю програму, її треба зкомпілювати і зібрати разом зі стандартною бібліотекою C. Це можна зробити за допомогою `gcc`:

```
gcc -m32 -nostartfiles args.s -o args
```

Флаг `-m32` використовується, щоб примусити програму компілюватись як 32-бітну у 64-бітному середовищі. У 32-бітному середовищі він не потрібен.

Програма, що використовує системні виклики

Це програма-аналог виклику shell `cat log.txt | wc -l`:

```
.data
    # масив для виклику cat log.txt
    cmd_cat: .string "/bin/cat"
    arg_cat: .string "log.txt"
    args_cat: .long cmd_cat, arg_cat, 0

    # масив для виклику wc -l
    cmd_wc: .string "/usr/bin/wc"
    arg_wc: .string "-l"
    args_wc: .long cmd_wc, arg_wc, 0

    # масив файлових дескрипторів для pipe
    fds: .int 0, 0
```


* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting

```
.text
.globl _start
_start:
        # виклик pipe(fds)
        pushl $fds
        call pipe

        # виклик fork()
        call fork

        # перехід до коду дочірнього процесу для cat,
        # якщо fork повернув 0
        cmpl $0, %eax
        je child_cat

        # виклик fork() у батьківському процесі
        call fork

        # перехід до коду дочірнього процесу для wc,
        # якщо fork повернув 0
        cmpl $0, %eax
        je child_wc

        # close(fd[0]) у батьківському процесі
        movl $fds, %eax
        pushl 0(%eax)
        call close

        # close(fd[1]) у батьківському процесі
        movl $fds, %eax
        pushl 4(%eax)
        call close

        # виклик wait(NULL) - для cat
        pushl $0
        call wait

        # ще один виклик wait(NULL) - для wc
        pushl $0
        call wait

finish:
        # виклик exit(0)
        movl $1, %eax
```


* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting

```
        movl $0, %ebx
        int $0x80

# код дочірнього процесу для cat
child_cat:
        # виклик dup2(fds[1],1)
        pushl $1
        movl $fds, %eax
        pushl 4(%eax)
        call dup2

        # виклик close(fds[0]), close(fds[1])
        movl $fds, %eax
        pushl 0(%eax)
        call close
        movl $fds, %eax
        pushl 4(%eax)
        call close

        # виклик execve(cmd_cat, args_cat)
        pushl $args_cat
        pushl $cmd_cat
        call execve

        call finish

# код дочірнього процесу для wc
child_wc:
        # виклик dup2(fds[0],0)
        pushl $0
        movl $fds, %eax
        pushl (%eax)
        call dup2

        # виклик close(fds[0]), close(fds[1])
        movl $fds, %eax
        pushl 0(%eax)
        call close
        movl $fds, %eax
        pushl 4(%eax)
        call close

        # виклик execve(cmd_wc, args_wc)
        pushl $args_wc
```


* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting

```
        pushl $cmd_wc
        call execve

        call finish
```

Література

- Асемблер в Linux для C программистов
- Ассемблеры для Linux: Сравнение GAS и NASM
- An Introduction to x86_64 Assembly Language
- Nasm Tutorial
- Say hello in x64 assembly
- Краткое введение в reverse engineering для начинающих
- https://www.hackerschool.com/blog/7-understanding-c-by-learning-assembly
- The Art of Assembly Programming

* pdf generated on 2016-01-03 from markdown sources with SBCL 1.2.16 on Linux 3.16.0-38-generic with 3bmd, cl-pdf & cl-typesetting