

Лабораторная работа №3.

Ассемблер

Цель

Целью выполнения этого компьютерного практикума является знакомство с языком Ассемблера и использование его для решения задач управления ОС.

В результате его выполнения будут получены базовые знания по языку Ассемблера и усвоены навыки написания программ с использованием этого языка.

Задание

Необходимо написать программу на языке ассемблера для Linux (с использованием GAS или NASM), которая выполняет ту же задачу, что и в работе № 2.

Примеры программ на ассемблере

Программа "Hello world" , которая использует для работы только системные вызовы

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

.text
    .globl main

main:
    // системный вызов write
    movl    $4, %eax // номер вызова – 4
    movl    $1, %ebx // запись в STDOUT (fd 1)
    movl    $hello_str, %ecx // что пишем?
    movl    $14, %edx // длина строки
    int     $0x80 // прерывание 0x80

    // системный вызов exit
    movl    $1, %eax // номер вызова – 1
    movl    $0, %ebx // код возврата – 0
```

```
int    $0x80
```

Для того чтобы запустить эту программу, ее нужно скомпилировать и собрать:

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

Аналог этой программы на языке C:

```
#include <unistd.h>

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

Программа, которая печатает свои аргументы командной строки. Использует библиотечную функцию puts

Программа на языке C:

```
#include<stdio.h>

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

На языке Ассемблера:

```
.section .text
.globl _start

_start:
    // запомнить текущее положение верхушки стека
    movl %esp, %ebp
    // на вершине стека – argc
    // записать его в счетчик
    movl (%ebp), %esi
```

```

// edi – номер аргумента
movl $1, %edi

print_loop:
// считывание памяти по адресу ebp + edi*4
// по этому адресу находится аргумент argv[edi]
movl (%ebp, %edi, 4), %eax

// вызов puts: аргумент передается через стек
pushl %eax
call puts

// меняем счетчики
// проверяем , не пора завершать цикл
incl %edi
decl %esi
test %esi, %esi
jnz print_loop

// выход
movl $1, %eax
movl $0, %ebx
int $0x80

```

Для того чтобы запустить эту программу, ее нужно скомпилировать и собрать вместе со стандартной библиотекой C. Это можно сделать с помощью gcc:

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

Флаг -m32 используется, чтобы заставить программу компилироваться как 32-битную в 64-битной среде. В 32-битной среде он не нужен.

Программа, использующая системные вызовы

Это программа-аналог вызова shell `cat log.txt | wc -l`:

```

.data
# массив для вызова cat log.txt
cmd_cat: .string "/bin/cat"
arg_cat: .string "log.txt"
args_cat: .long cmd_cat, arg_cat, 0

# массив для вызова wc -l
cmd_wc: .string "/usr/bin/wc"
arg_wc: .string "-l"

```

```
args_wc: .long cmd_wc, arg_wc, 0

# массив файловых дескрипторов для pipe
fds: .int 0, 0

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

    # вызов fork()
    call fork

    # переход к коду дочернего процесса для cat,
    # если fork вернул 0
    cmpl $0, %eax
    je child_cat

    # вызов fork() в родительском процессе
    call fork

    # переход к коду дочернего процесса для wc,
    # если fork вернул 0
    cmpl $0, %eax
    je child_wc

    # close(fd[0]) в родительском процессе
    movl $fds, %eax
    pushl 0(%eax)
    call close

    # close(fd[1]) в родительском процессе
    movl $fds, %eax
    pushl 4(%eax)
    call close

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

    # еще один вызов wait(NULL) - для wc
    pushl $0
```

```

        call wait

finish:
    # вызов exit(0)
    movl $1, %eax
    movl $0, %ebx
    int $0x80

# код дочернего процесса для cat
child_cat:
    # вызов dup2(fds[1],1)
    pushl $1
    movl $fds, %eax
    pushl 4(%eax)
    call dup2

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

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

    call finish

# код дочернего процесса для wc
child_wc:
    # вызов dup2(fds[0],0)
    pushl $0
    movl $fds, %eax
    pushl (%eax)
    call dup2

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

```

```
    pushl 4(%eax)
    call close

    # вызов execve(cmd_wc, args_wc)
    pushl $args_wc
    pushl $cmd_wc
    call execve

    call finish
```

Литература

- [Асемблер в Linux для C программистов](#)
- [Асемблеры 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](#)