Введение в архитектуру ЭВМ

# Лекция 3. Коды символов. Системные вызовы. Файлы stdin, stdout. Разбор работы программы task3

NB. Глубокое понимание этой программы является ключом к успешному выполнению всех лабораторных задач нашей дисциплины.

## План лекции

- Общая схема работы программы.
- Особенности кодировки символов в текущей версии ОС Linux.
- Системные вызовы. Общие положения
  - Системный вызов read
  - Системный вызов write
- Файлы stdin и stdout терминала shell
- Взаимодействие клавиатуры терминала shell и файла stdin
- Макроопределения Getchar и Puts в файле my-macro
  - Макровызов и макрорасширение Getchar
  - Макровызов и макрорасширение Puts
- Разбор исходного файла task3.S
- Таблица символов из файла task3.lst

## Общая схема работы программы

Программа получает байты кодов символов с клавиатуры терминала и после нажатия клавиши Enter помещает их в файл стандартного ввода stdin, откуда эти байты системным вызовом

read читаются в промежуточный буфер c для анализа.

Если прочитан код цифровой клавиши, то он помещается в очередной байт буфера buf в оперативной памяти. Коды остальных клавиш игнорируются. Программа завершает работу по нажатию комбинации ctrl/D.

Особенности кодировки символов в текущей версии ОС Linux.

Используется версия Unicode – utf-8 (RFC - Request For Comments 3629), код символа занимает -1, 2, 3 или 4 8-битовых байта.

NB. Символы с кодами от 0 до 127 (7F) кода ASCII (стр.184 книги) совпадают с однобайтовыми кодами utf-8. Т.о. в нашей версии ОС однобайтовыми являются коды служебных символов (0A – LF), цифр, прописных и строчных букв английского алфавита.

Одно байтовые коды цифровых символов — от кода «0» - 48 (0x30) до кода «9» - 57 (0x39).

NB. Значение одноцифрового числа = Значение кода – 48 (0x30)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 | ^@ NUL | 16 | 10 | ^P DLE | 32 | 20 | ␣ | 48 | 30 | 0 | 64 | 40 | @ |
| 01 | 01 | ^A SOH | 17 | 11 | ^Q DC1 | 33 | 21 | ! | 49 | 31 | 1 | 65 | 41 | A |
| 02 | 02 | ^B STX | 18 | 12 | ^R DC2 | 34 | 22 | " | 50 | 32 | 2 | 66 | 42 | B |
| 03 | 03 | ^C ETX | 19 | 13 | ^S DC3 | 35 | 23 | # | 51 | 33 | 3 | 67 | 43 | C |
| 04 | 04 | ^D EOT | 20 | 14 | ^T DC4 | 36 | 24 | $ | 52 | 34 | 4 | 68 | 44 | D |
| 05 | 05 | ^E ENQ | 21 | 15 | ^U NAK | 37 | 25 | % | 53 | 35 | 5 | 69 | 45 | E |
| 06 | 06 | ^F ACK | 22 | 16 | ^V SYN | 38 | 26 | & | 54 | 36 | 6 | 70 | 46 | F |
| 07 | 07 | ^G BEL | 23 | 17 | ^W ETB | 39 | 27 | ' | 55 | 37 | 7 | 71 | 47 | G |
| 08 | 08 | ^H BS | 24 | 18 | ^X CAN | 40 | 28 | ( | 56 | 38 | 8 | 72 | 48 | H |
| 09 | 09 | ^I HT | 25 | 19 | ^Y EM | 41 | 29 | ) | 57 | 39 | 9 | 73 | 49 | I |
| 10 | 0A | ^J LF | 26 | 1A | ^Z SUB | 42 | 2A | * | 58 | 3A | : | 74 | 4A | J |
| 11 | 0B | ^K VT | 27 | 1B | ^[ ESC | 43 | 2B | + | 59 | 3B | ; | 75 | 4B | K |
| 12 | 0C | ^L FF | 28 | 1C | ^\ FS | 44 | 2C | , | 60 | 3C | < | 76 | 4C | L |
| 13 | 0D | ^M CR | 29 | 1D | ^] GS | 45 | 2D | - | 61 | 3D | = | 77 | 4D | M |
| 14 | 0E | ^N SO | 30 | 1E | ^^ RS | 46 | 2E | . | 62 | 3E | > | 78 | 4E | N |
| 15 | 0F | ^O SI | 31 | 1F | ^_ US | 47 | 2F | / | 63 | 3F | ? | 79 | 4F | O |

NB. Численные значения кодов цифровых символов удовлетворяют неравенству:

## 0x30 <= Код цифрового символа <= 0x39

**Значения кодов `ascii` легко получить командой `man ascii`.**

```
ASCII(7)                  Linux Programmer's Manual                 ASCII(7)

NAME
       ascii - ASCII character set encoded in octal, decimal, and
hexadecimal

DESCRIPTION
       ASCII  is the American Standard Code for Information Interchange. It
is a 7-bit code. Many 8-bit codes (e.g., ISO 8859-1) contain ASCII as their
lower half.  The international counterpart of ASCII is known as ISO 646-
IRV.

       The following table contains the 128 ASCII characters.

       C program '\X' escapes are noted.

       Oct   Dec   Hex   Char                        Oct   Dec   Hex   Char

       000   0     00    NUL '\0' (null character)   100   64    40    @
       001   1     01    SOH (start of heading)      101   65    41    A
       002   2     02    STX (start of text)         102   66    42    B
       003   3     03    ETX (end of text)           103   67    43    C
       004   4     04    EOT (end of transmission)   104   68    44    D
       005   5     05    ENQ (enquiry)               105   69    45    E
       006   6     06    ACK (acknowledge)           106   70    46    F
       007   7     07    BEL '\a' (bell)             107   71    47    G
       010   8     08    BS  '\b' (backspace)        110   72    48    H
       011   9     09    HT  '\t' (horizontal tab)   111   73    49    I
       012   10    0A    LF  '\n' (new line)         112   74    4A    J
       013   11    0B    VT  '\v' (vertical tab)     113   75    4B    K
       014   12    0C    FF  '\f' (form feed)        114   76    4C    L
       015   13    0D    CR  '\r' (carriage ret)     115   77    4D    M

. . .

       040   32    20    SPACE                       140   96    60
       041   33    21    !                           141   97    61    a
       042   34    22    "                           142   98    62    b
       043   35    23    #                           143   99    63    c
       044   36    24    $                           144   100   64    d
       045   37    25    %                           145   101   65    e
       046   38    26    &                           146   102   66    f
       047   39    27    '                           147   103   67    g
       050   40    28    (                           150   104   68    h
       051   41    29    )                           151   105   69    i
       052   42    2A    *                           152   106   6A    j
       053   43    2B    +                           153   107   6B    k
       054   44    2C    ,                           154   108   6C    l
       055   45    2D    -                           155   109   6D    m
       056   46    2E    .                           156   110   6E    n
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 057 | 47 | 2F | / | 157 | 111 | 6F | o |
| 060 | 48 | 30 | 0 | 160 | 112 | 70 | p |
| 061 | 49 | 31 | 1 | 161 | 113 | 71 | q |
| 062 | 50 | 32 | 2 | 162 | 114 | 72 | r |
| 063 | 51 | 33 | 3 | 163 | 115 | 73 | s |
| 064 | 52 | 34 | 4 | 164 | 116 | 74 | t |
| 065 | 53 | 35 | 5 | 165 | 117 | 75 | u |
| 066 | 54 | 36 | 6 | 166 | 118 | 76 | v |
| 067 | 55 | 37 | 7 | 167 | 119 | 77 | w |
| 070 | 56 | 38 | 8 | 170 | 120 | 78 | x |
| 071 | 57 | 39 | 9 | 171 | 121 | 79 | y |
| 072 | 58 | 3A | : | 172 | 122 | 7A | z |
| 073 | 59 | 3B | ; | 173 | 123 | 7B | { |
| 074 | 60 | 3C | < | 174 | 124 | 7C | \| |
| 075 | 61 | 3D | = | 175 | 125 | 7D | } |
| 076 | 62 | 3E | > | 176 | 126 | 7E | ~ |
| 077 | 63 | 3F | ? | 177 | 127 | 7F | DEL |

NOTES

History

An ascii manual page appeared in Version 7 of AT&T UNIX.

On older terminals, the underscore code is displayed as a left arrow, called backarrow, the caret is displayed as an up-arrow and the vertical bar has a hole in the middle.

Uppercase and lowercase characters differ by just one bit and the ASCII character 2 differs from the double quote by just one bit, too. That made it much easier to encode characters mechanically or with a non-microcontroller-based electronic keyboard and that pairing was found on old teletypes.

The ASCII standard was published by the United States of America Standards Institute (USASI) in 1968.

SEE ALSO

charsets(7), iso_8859-1(7), iso_8859-10(7), iso_8859-11(7), iso_8859-13(7), iso_8859-14(7), iso_8859-15(7), iso_8859-16(7), iso_8859-2(7), iso_8859-3(7), iso_8859-4(7), iso_8859-5(7), iso_8859-6(7), iso_8859-7(7), iso_8859-8(7), iso_8859-9(7), utf-8(7)

COLOPHON

This page is part of release 4.16 of the Linux man-pages project. A descrip-ion of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

Linux 2016-10-08

ASCII(7)

## Системные вызовы. Общие положения

Для обеспечения безопасности в ОС оперативная память разделена на пространство пользователя и пространство ядра. Права обычного пользователя не позволяют выполнять операции в пространстве ядра. В ядре выполняются такие важные функции ОС как управление:

- памятью;
- устройствами;
- файловыми системами;
- процессами.

Т.о прикладному программисту НЕ надо самому разрабатывать программы этих функций.

Прикладной программист может выполнять необходимые ему функции ядра, запуская выполнение в пространстве ядра специальных функций, т. н. *системных вызовов* (`syscalls`) с помощью приводимого ниже порядка запуска.

Системный вызов выполняется, когда пользовательский процесс (например `emacs`) требует некоторой службы, реализуемой ядром (например открытие файла), и вызывает специальную функцию (например, системный вызов `open`). В этот момент пользовательский процесс переводится в режим ожидания. Ядро анализирует запрос, пытается его выполнить и передает результаты пользовательскому процессу, который затем возобновляет свою работу.

Системные вызовы в общем случае защищают доступ к ресурсам, которыми управляет ядро, при этом самые большие категории системных вызовов имеют дело с вводом/выводом (`open, close, read, write, poll`) и многие другие, процессами (`fork, execve, kill` и т.д.), временем (`time, settimeofday` и т.п.) и памятью (`mmap, brk` и пр.) Под эти категории подпадают практически все

**системные вызовы.**

**Многие команды языка** `shell` **просто обращаются к системным вызовам, например** `mkdir, kill`**.**

**Список всех доступных в конкретной** `unix` **системе вызовов находится в файле** `/usr/include/asm/unistd_32.h` **для архитектуры IA-32 и в файле** `/usr/include/asm/unistd_64.h` **для архитектуры Intel®64.**

```
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
```

```
#define __NR_lchown 16
. . .
```

Имя системного вызова (но без префиксов) нужно использовать для получения его описания: `man read.2`. Функции системных вызовов описываются прототипами в стиле языка C.

Порядок запуска системного вызова

Пусть системный вызов с номером N описан прототипом

```
syscall (p1, p2, p3, p4, p5)
```

Передача параметров системному вызову осуществляется через 32-х битовые регистры, в которые следует записать значения параметров, указанные в прототипе, в порядке слева направо. Для запуска системного вызова необходимо выполнить следующую последовательность команд:

```
mov N,%eax      # задать номер системного вызова.
mov p1, %ebx        # задать значение p1
mov p2, %ecx        # ... p2
mov p3, %edx        # ... p3
mov p4, %esi        # ... p4
mov p5, %edi        # ... p5
int $0x80           # прочитать из %eax номер системного
                    # вызова и выполнить его
```

По команде `int $0x80` процессор приостанавливает текущий выполняемый процесс, переходит в защищенный режим, читает из `%eax` номер системного вызова и загружает для выполнения соответствующую функцию, реализующую системный вызов. Функция читает из регистров параметры и выполняется.

**Системный вызов `read`**

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

**Номер системного вызова — 3. `read()` пытается прочитать не более `count` байтов из файла, имеющего файловый дескриптор `fd` в буфер в ОП, заданный указателем `*buf`.**

**Дескриптор файла это небольшое целое число, которое используется в системных вызовах работы с файлами для ссылки на конкретный файл.**

**Целое `fd` вычисляет и привязывает к файлу системный вызов `open` при открытии файла.**

**Если байты прочитались в буфер, в регистре `%eax` возвращается их количество. Если достигнут конец файла (`EOF`), то в регистре `%eax` возвращается значение ноль.**

**Системный вызов `write`**

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

**Номер системного вызова — 4. `write()` записывает `count` байт из буфера в ОП, заданного указателем `*buf`, в файл, имеющий файловый дескриптор `fd`.**

**Если байты записались в файл, в регистре `%eax` возвращается их**

количество. При ошибке в регистре %eax возвращается значение -1.

### Файлы stdin и stdout терминала shell

NBNB. В задаче task3 организован диалог с текущим терминалом shell (клавиатура/экран) для которого в ОС назначены файлы и значения их дескрипторов:

- клавиатура — файл stdin (только для чтения), fd = 0;
- экран — файл stdout, (только для записи), fd = 1.

NBNB при вводе с клавиатуры системное событие «конец файла» (EOF) stdin формируется нажатием комбинации ctrl/d (не символ, не имеет кода). Это событие, которое обнаруживают системные вызовы при чтении файла.

### Взаимодействие клавиатуры терминала shell и файла stdin

При нажатии клавиш байты кодов соответствующих символов сначала попадают не в файл stdin а в буфер терминала. Только после нажатия клавиши Enter байты кодов символов всех клавиш, нажатых после предыдущего нажатия клавиши Enter, передаются в файл stdin. Системный вызов read прочтет из файла заданное параметром count количество байтов и разместит их в ОП по адресу *buf.

Следующий запуск read будет читать еще не прочитанные байты, если они остались в файле stdin. Если байтов нет read будет ожидать их поступления в stdin после очередного нажатия клавиши Enter.

**Нажатие `Enter, ctrl/d, Enter` приводит к наступлению события EOF и закрытию файла `stdin` (он становится недоступным для операций).**

### Макроопределения `Getchar` и `Puts` в файле `my-macro`

**Рассмотрим файл `my-macro`**

```
/*
 * Макроопределение завершения работы.
 * Аргументы:
 *   - код завершения программы
 *
 * После выполнения макровызова изменяются регистры: %eax, %ebx
 * См. также 'man 2 exit'
*/
.macro Exit ret_val
    movl $1, %eax          # номер сист. вызова exit
    movl \ret_val, %ebx    # код выхода
    int $0x80              # выполнить системный вызов
.endm


/*
 * Макроопределение для считывания одного байта кода символа из
стандартного ввода
 * Аргументы:
 *   - адрес буфера для считывания байта
 * Результат:
 *   - в %eax количество считанных байтов
 *   - по адресу buf_addr - считанный байт
 *
 * После выполнения макровызова изменяются регистры: %eax, %ebx, %ecx, %edx
```

```
 * См. также 'man 2 read'
*/
.macro Getchar buf_addr
    movl $3, %eax      # номер сист. вызова read
    movl $0, %ebx      # параметр 1: дескриптор стандартного ввода
    movl \buf_addr, %ecx   # параметр 2: адрес буфера (он же - фактический
                           # параметр макровызова)
    movl $1, %edx      # параметр 3: количество байтов для чтения
    int $0x80          # выполнить системный вызов
.endm
```

**Макровызов и макрорасширение `Getchar` (фрагмент файла `task3.lst`)**

***Лекционный комментарий.***

**Появление в исходном файле макровызова `Getchar $c` приводит к включению в этот файл макрорасширения (команд тела макроопределения), в котором формальный параметр макроопределения `buf_addr` заменен на указанный в макровызове фактический параметр `$c`, - адрес однобайтового буфера, определенного в секции `.bss`.**

**Затем для этого буфера формируются правильные команды запуска системного вызова `read`.**

```
  22                    Getchar $c        # макровызов ввода байта из stdin в
  22 0018 B8030000      >  movl $3,%eax
  22      00
  22 001d BB000000      >  movl $0,%ebx
  22      00
  22 0022 B9640000      >  movl $c,%ecx
  22      00
  22                 >
  22 0027 BA010000      >  movl $1,%edx
  22      00
```

```
GAS LISTING task3.S                page 3

  22 002c CD80        >  int $0x80
  23                          # промежуточный буфер с
  24


/*
 * Макроопределение для  вывода строки в файл стандартного вывода
 * Аргументы:
 *   - Строка для вывода.
 *
 * Приметр макровызова:
 *   Puts "Текст выводимой строки"
 *
 * Результат:
 *   - выводит в стандартный вывод символы заданной строки
 *     и вслед за ними символ перевода строки \n
 *
 * После выполнения макровызова изменяются регистры: %eax, %ebx, %ecx, %edx
 * См. также 'man puts', 'man 2 write'
*/
.macro Puts string
.data
    str\@:    .ascii "\string\n"  # формирование фактической строки для
вывода
    strlen\@ =     . - str\@     # получение значения длины строки
.text
     movl $4, %eax      # номер сист. вызова write
     movl $1, %ebx      # параметр 1: дескриптор стандартного вывода
     movl $str\@, %ecx      # параметр 2: адрес памяти с выводимыми
символами
     movl $strlen\@, %edx   # параметр 3: количество байтов для вывода
     int $0x80          # выполнить системный вызов
.endm
```

**Макровызов и макрорасширение `Puts` (фрагмент файла**

task3.lst)

*Лекционный комментарий.*

**Появление в исходном файле макровызова** `Puts "Цифра! Хорошо."` **приводит к включению в этот файл макрорасширения (команд тела макроопределения), в котором формальный параметр макроопределения** `string` **заменен на указанный в макровызове фактический параметр** `"Цифра! Хорошо."`**.При этом:**

1) **В секцию** `.data` **добавляется директива размещения в объектном файле кодов символов подлежащей выводу строки:**
   `str2: .ascii "Цифра! Хорошо.\n"`
   **с присоединением к ней условного кода символа перевода строки** `\n`.
2) **Метка этой строки — `str2:` формируется из конструкции** `str\@`**, где псевдопеременная** `\@` **представляют собой счетчик выполненных макровызовов (т. е. до выполнения рассматриваемого макровызова были выполнены макровызовы с номерами** `0` **и** `1`**).**
3) **Абсолютное символьное имя** `strlen2`**, также формируемое с помощью псевдопеременной** `\@`**, получает значение длины строки. Это символьное имя используется затем при формировании команд запуска системного вызова.**
4) **В секции** `.text` **формируются правильные команды запуска системного вызова write.**

```
41                      Puts "Цифра! Хорошо." # сообщения об успехе вводе
41                  > .data
41 002c D0A6D0B8        >  str2:.ascii "Цифра! Хорошо.\n"
41      D184D180
```

```
  41      D0B02120
  41      D0A5D0BE
  41      D180D0BE
  41              >
  41              >  strlen2 =. - str2
  41              >
  41              > .text
  41 005a B8040000    >  movl $4,%eax
  41      00
  41 005f BB010000    >  movl $1,%ebx
  41      00
  41 0064 B92C0000    >  movl $str2,%ecx
  41      00
  41 0069 BA1A0000    >  movl $strlen2,%edx
  41      00
  41 006e CD80        >  int $0x80
```

### Разбор исходного файла task3.S

```
/*
 * Программа ввода кодов цифровых символов в буфер в ОП
 */

.include "my-macro"

.bss
   .lcomm buf, 100        # 100 байтовый буфер для кодов прочитанных символов
   .lcomm c, 1            # однобайтовый буфер для чтения байта из файла stdin
```

***Лекционный комментарий.***

**Секция `.bss` используется для хранения неинициализированных данных, байты которых при запуске программы имеют нулевые значения. Данные этой секция не хранится в объектном и исполняемом файлах, память для нее распределяет загрузчик исполняемого файла. Символьные имена в секции `bss` определяются директивой `.lcomm`, задающей имя и длину выделяемой ему области ОП в байтах.**

```
.text
.global _start

_start:
```

```
   sub %esi, %esi          # указатель адреса байта в буфере buf (индексный регистр)

show_prompt:
   Puts "Вводите цифру, друг мой!"  # макровызов вывода строки в
                                    # файл stdout (подсказка ввода)

kbd_input:
   Getchar $c        # макровызов ввода байта из stdin в
                     # промежуточный буфер c

   cmpl $0, %eax     # наступило событие EOF (конец файла stdin) ?
   je stop           # Да - на завершение программы

   cmpb $'\n', c             # это символ перевода строки ?
   je kbd_input              # ДА - на ввод следующего символа
   cmpb $'9', c              # код больше кода символа '9' ?
   ja print_err_msg# ДА - на вывод сообщения об ошибке
   cmpb $'0', c              # код меньше кода символа '0' ?
   jb print_err_msg# ДА - на вывод сообщения об ошибке

   movb c, %al               # передать код символа цифры из c в al
   movb %al, buf(%esi)    # передать код символа цифры из al в байт
                     # буфера по адресу &buf + esi
   incl %esi         # указать на следующий байт буфера для
                     # следующего кода
```

***Лекционный комментарий.***

**1. Т.к. оба буфера `c` и `buf` находятся в ОП, нельзя передать значение из `c` в `buf` одной командой, сперва передаем его в `%al` командой `movb c, %al`.**

**2. Каждый введенный в цикле код цифрового символа надо записывать в последовательные байты `buf`, увеличивая в каждом цикле адрес записи на один.**

| **Адрес байта** | `buf+0` | `buf+1` | `buf+2` | `..` | `buf+n` |
|---|---|---|---|---|---|
| **Значение `%esi`** | `0` | `1` | `2` | | `n` |
| **№ веденного байта кода** | `1` | `2` | `3` | | `n+1` |

**Для решения таких задач существует механизм *режимов адресации*, частным случаем которого является команда `movb %al, buf(%esi)`.**

**При такой записи операнда приемника говорят, что `%esi` является базовым регистром, а адрес этого операнда приемника — `A` вычисляется перед каждым выполнением этой команды по формуле `A = *buf + %esi`, как это показано в таблице выше. Увеличение значения `%esi` в каждом цикле обеспечивает команда `incl %esi`.**

```
  Puts "Цифра! Хорошо." # сообщение об успехе вводе

    jmp show_prompt    # на ввод следующего символа

print_err_msg:
    Puts "Не цифровая клавиша. Повторите ввод"    # вывод сообщения об
ошибке
    jmp show_prompt    # на ввод следующего байта

stop:
    Exit $0

.end
```

**Таблица символов из файла task3.lst**

```
DEFINED SYMBOLS
             task3.S:8      .bss:0000000000000000 buf
             task3.S:8      .bss:0000000000000064 c
             task3.S:14     .text:0000000000000000 _start
             task3.S:17     .text:0000000000000002 show_prompt
             task3.S:18     .data:0000000000000000 str0
             task3.S:18     *ABS*:000000000000002c strlen0
             task3.S:21     .text:0000000000000018 kbd_input
             task3.S:49     .text:000000000000008d stop
             task3.S:45     .text:0000000000000072 print_err_msg
             task3.S:41     .data:000000000000002c str2
             task3.S:41     *ABS*:000000000000001a strlen2
             task3.S:46     .data:0000000000000046 str3
             task3.S:46     *ABS*:0000000000000042 strlen3

  NO UNDEFINED SYMBOLS
```

**NBNB. Видно, что определяемые в макрорасширениях `Puts` строки последовательно располагаются в единственной секции `.data`.**