Введение в архитектуру ЭВМ

Лекция 12. Библиотека `glibc`.

План лекции

Общие сведения

Язык C спроектирован как лаконичный язык и не предоставляет встроенных средств на уровне его ключевых слов для выполнения таких распространенных операций, как:

- **низкоуровневой или форматированный ввод/вывод;**
- **управление строками;**
- **вычисление арифметических и математических функций;**
- **управление памятью, процессами, файловыми системами, се`тевыми` соединениями;**
- **и т. п.**

Для решения таких задач применяется стандартная библиотека языка C (также известная как `libc`). Сама по себе она не является частью языка, однако предусмотренный в ней набор функций, а также определений типов и макроопределений составляет системную среду, поддерживающую разработку ПО на C.

В 1978 году **Брайан Керниган** и **Деннис Ритчи** опубликовали первую редакцию книги «Язык программирования C», содержащую систематическое описание такого подхода. Эта книга, известная среди программистов как «**K&R**», служила многие годы неформальной спецификацией языка и выдержала в США около сорока изданий. Подход «K&R» и его реализации оказались очень плодотворными и активно использовались при разработке большого числа проектов.

Язык C был и остаётся одним из самых распространённых языков программирования в течение пятидесяти лет и, по сути является - языка общения (`lingua franca`) между разработчиками. Большая группа языков унаследовали базовый синтаксис C (использование фигурных скобок в качестве ограничителей блоков кода, описание переменных, характерные формы операторов `for`, `while`, `if`, `switch` с параметрами в скобках, комбинированные операции `++`, `--`, `+=`, `-=` и другие), из-за чего программы на этих языках имеют характерный внешний вид, ассоциирующийся именно с `C` . Это такие языки как `C++`, `Objective C`, `Java`, `JavaScript`, `PHP`, `Perl`, `AWK`, `C#`, `Go` и др.

C — язык системного программирования:

- Эффективный машинный код. Используется вместо ассемблера.
- Применяется при разработке системного ПО для платформ от супер ЭВМ до встроенных.
- Неполный список продкутов, реализованных на C.
  - Ядра ОС Unix (и всех ее производных), MS Windows, Linux, OS X, iOS, Android, Windows Phone.
  - СУБД Oracle Database, MySQL, MS SQL Server, PostgreSQL.

- Первые реализации языков Java, Python, Perl и PHP.

- Cpython - эталонная реализация языка Python.
- Промежуточный язык. В Первых реализациях языков C++, Objective-C, Go — код, написанный на этих языках, транслировался на язык C.
- Реализован для большинства аппаратных платформ и ОС.}

C – отличный язык для выражения общих идей в программировании способом, который удобен большинству программистов. Более того, многие принципы, используемые в C – например, `argc` и `argv` для параметров командной строки, будут отображаться во многих других языках, так что вы сможете общаться с людьми, даже если они не знают C, способом, который является общим для вас обоих. Эти сопоставления говорят о существенной «C-орентации» мышления современных системных программистов.

C - фундаментальная основа квалификации программиста.

Библиотека `libc`

Библиотека `libc` была стандартизована в 1989 г. как часть стандарта `ANSI C`, который в 1990 г. был признан `ISO` (текущая актуальная версия стандарта, т. н. `C18 — ISO/IEC 9899:2018 Information technology — Programming languages — C`).

Библиотека состоит из нескольких десятков заголовочных файлов, подключаемых к программному проекту директивами. Эти файлы содержит описания одной или более функций, определения типов данных и макроопределения.

NBNB. В стандарте библиотека представлена как описание интерфейса прикладного программирования — API (Application Programming Interface) и для практического применения она должна быть реализована для конкретного компилятора и/или операционной системы.

Таким образом libc является неотъемлемой частью каждой реализации языка C.

Существует несколько десятков реализаций libc, в том числе такие широко применяемые как:

- **GNU C Library — glibc — самая распространенная реализация, используемая в ОС Linux;**
- **Microsoft C Run-time Library, используемая в ОС Windows;**
- **libSystem, используемая в ОС iOS компании Apple;**
- **Bionic, разработанная компанией Google для ОС Android.**

Библиотека libc оказала основополагающее влияние на реализации большинства современные системы программирования, среди которых C++, Python, Rust, GO и др. Например каждый заголовочный файл из стандартной библиотеки языка C включен в стандартную библиотеку языка C++ под именами, созданными путём отсечения в имени файла расширения .h и добавлением символа 'c' в начале имени, например имени 'time.h' соответствует 'ctime'.

Единственное отличие между этими заголовочными файлами и традиционными заголовочными файлами стандартной библиотеки языка C заключается в том, что функции должны быть помещены в пространство имен std:: (хотя некоторые компиляторы сами делают это).

Библиотека glibc

Библиотека glibc обновляется два раза в год. Ее функции, требующие работы в адресном пространстве ядра, используют системные вызовы. В руководстве к актуальной версии glibc функции разбиты на 36 групп различного назначения. Информацию о текущей версии glibc, установленной в конкретной версии ОС Linux можно получит командой:

```
/lib/libc.so.6
```

Ссылки на описание glibc:

https://www.gnu.org/software/libc/manual/

Общая характеристика glibc на 25.04.2021 *{25.07.23}* со стр.

https://www.openhub.net/p/glibc

Code Locations: `git://sourceware.org/git/glibc.git`

In a Nutshell, GNU C Library...

...

has had 37,147 *{40307}* commits made by 604 *{720}* contributors

representing 1,302,579 *{1419588}* lines of code

...

is mostly written in C

with an average number of source code comments:

Across all C projects on Open Hub, 19% of all source code lines are comments. This holds true for GNU C Library as well.

...

has a well established, mature codebase maintained by a very large development team with increasing Y-O-Y commits

...

took an estimated 371 *{406}* years of effort (COCOMO model)

starting with its first commit in February, 1995

ending with its most recent commit 7 months ago

COCOMO - Constructive Cost Model

Данные на 25.04.2021 о распределении языков программирования в коде со стр.:

https://www.openhub.net/p/glibc/analyses/latest/languages_summary

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	971,675	238,172	19.7%	152,531	1,362,378	74.0%
Assembly	257,940	75,262	22.6%	40,667	373,869	20.3%
C++	17,017	6,662	28.1%	4,131	27,810	1.5%
Autoconf	15,117	153	1.0%	1,719	16,989	0.9%
Make	12,581	2,802	18.2%	2,759	18,142	1.0%
shell script	9,694	2,026	17.3%	1,177	12,897	0.7%
Python	8,863	3,897	30.5%	1,413	14,173	0.8%
TeX/LaTeX	7,163	3,695	34.0%	816	11,674	0.6%
AWK	1,898	373	16.4%	266	2,537	0.1%
Perl	631	143	18.5%	87	861	0.0%
Totals	1,302,579	333,185		205,566	1,841,330	

Освоив соглашения стандарта ABI о связях функций разработчик получает доступ из языка ассемблера и из всех других языков и систем программирования, поддерживающих ABI к примерно 1500 (количество зависит от версии и реализации) функций библиотеки языка glibc, представляющей *золотой фонд программирования*.

Краткое содержание руководства по библиотеке glibc (1270 стр.)

Table of Contents

Полное содержание руководства по библиотеке `glibc`

Table of Contents

Текст подключаемый препроцессором C к файлу `call-as.c`

Для того чтобы продемонстрировать масштаб `glibc` и объем проделанной работы рассмотрим какой текст подключает препроцессор C к файлу `call-as.c` из предыдущей лекции.

```
#include <stdio.h>
/* прототип внешней функции.
 Параметр 1 - количество символов, кот. надо обработать
 Параметр 2 - адрес первого элемента массива
*/
extern void *Read_Sym(int, char*);
/* массив результатов*/
```

```
int Numbers[10];
int main()
{
  char Symbols[14] = "91A23B456C789"; /* исходный массив*/
    int i;
    printf ("%s\n",Symbols);
    Read_Sym(8, Symbols);          /* вызов as функции */
    /* печать результата */
    for (i = 0; i < 8; i++)
        printf("%d\t", Numbers[i]);
    return 0;
}
```

Код, формируемый препроцессором, можно получить командой:

```
gcc -m32 -gstabs+ -E -P call-as.c > call-as.1pp
```

Приведенный ниже файл `call-as.1pp` объемом 306 строк содержит код, полученный препроцессорром для фазы компиляции из файла `call-as.c` объемом 46 строк.

```
typedef unsigned int size_t;
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;
typedef signed char __int8_t;
typedef unsigned char __uint8_t;
typedef signed short int __int16_t;
typedef unsigned short int __uint16_t;
typedef signed int __int32_t;
typedef unsigned int __uint32_t;
```

```
__extension__ typedef signed long long int __int64_t;
__extension__ typedef unsigned long long int __uint64_t;
__extension__ typedef long long int __quad_t;
__extension__ typedef unsigned long long int __u_quad_t;
__extension__ typedef long long int __intmax_t;
__extension__ typedef unsigned long long int __uintmax_t;
__extension__ typedef __u_quad_t __dev_t;
__extension__ typedef unsigned int __uid_t;
__extension__ typedef unsigned int __gid_t;
__extension__ typedef unsigned long int __ino_t;
__extension__ typedef __u_quad_t __ino64_t;
__extension__ typedef unsigned int __mode_t;
__extension__ typedef unsigned int __nlink_t;
__extension__ typedef long int __off_t;
__extension__ typedef __quad_t __off64_t;
__extension__ typedef int __pid_t;
__extension__ typedef struct { int __val[2]; } __fsid_t;
__extension__ typedef long int __clock_t;
__extension__ typedef unsigned long int __rlim_t;
__extension__ typedef __u_quad_t __rlim64_t;
__extension__ typedef unsigned int __id_t;
__extension__ typedef long int __time_t;
__extension__ typedef unsigned int __useconds_t;
__extension__ typedef long int __suseconds_t;
__extension__ typedef int __daddr_t;
__extension__ typedef int __key_t;
__extension__ typedef int __clockid_t;
__extension__ typedef void * __timer_t;
__extension__ typedef long int __blksize_t;
__extension__ typedef long int __blkcnt_t;
__extension__ typedef __quad_t __blkcnt64_t;
__extension__ typedef unsigned long int __fsblkcnt_t;
__extension__ typedef __u_quad_t __fsblkcnt64_t;
```

```
__extension__ typedef unsigned long int __fsfilcnt_t;
__extension__ typedef __u_quad_t __fsfilcnt64_t;
__extension__ typedef int __fsword_t;
__extension__ typedef int __ssize_t;
__extension__ typedef long int __syscall_slong_t;
__extension__ typedef unsigned long int __syscall_ulong_t;
typedef __off64_t __loff_t;
typedef __quad_t *__qaddr_t;
typedef char *__caddr_t;
__extension__ typedef int __intptr_t;
__extension__ typedef unsigned int __socklen_t;
typedef int __sig_atomic_t;
struct _IO_FILE;
typedef struct _IO_FILE __FILE;
struct _IO_FILE;
typedef struct _IO_FILE FILE;
typedef struct
{
  int __count;
  union
  {
    unsigned int __wch;
    char __wchb[4];
  } __value;
} __mbstate_t;
typedef struct
{
  __off_t __pos;
  __mbstate_t __state;
} _G_fpos_t;
typedef struct
{
  __off64_t __pos;
```

```
  __mbstate_t __state;
} _G_fpos64_t;
typedef __builtin_va_list __gnuc_va_list;
struct _IO_jump_t; struct _IO_FILE;
typedef void _IO_lock_t;
struct _IO_marker {
  struct _IO_marker *_next;
  struct _IO_FILE *_sbuf;
  int _pos;
};
enum __codecvt_result
{
  __codecvt_ok,
  __codecvt_partial,
  __codecvt_error,
  __codecvt_noconv
};
struct _IO_FILE {
  int _flags;
  char* _IO_read_ptr;
  char* _IO_read_end;
  char* _IO_read_base;
  char* _IO_write_base;
  char* _IO_write_ptr;
  char* _IO_write_end;
  char* _IO_buf_base;
  char* _IO_buf_end;
  char *_IO_save_base;
  char *_IO_backup_base;
  char *_IO_save_end;
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
```

```
    int _flags2;
    __off_t _old_offset;
    unsigned short _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];
    _IO_lock_t *_lock;
    __off64_t _offset;
    void *__pad1;
    void *__pad2;
    void *__pad3;
    void *__pad4;
    size_t __pad5;
    int _mode;
    char _unused2[15 * sizeof (int) - 4 * sizeof (void *) -
sizeof (size_t)];
  };
  typedef struct _IO_FILE _IO_FILE;
  struct _IO_FILE_plus;
  extern struct _IO_FILE_plus _IO_2_1_stdin_;
  extern struct _IO_FILE_plus _IO_2_1_stdout_;
  extern struct _IO_FILE_plus _IO_2_1_stderr_;
  typedef __ssize_t __io_read_fn (void *__cookie, char
*__buf, size_t __nbytes);
  typedef __ssize_t __io_write_fn (void *__cookie, const
char *__buf,
       size_t __n);
  typedef int __io_seek_fn (void *__cookie, __off64_t
*__pos, int __w);
  typedef int __io_close_fn (void *__cookie);
  extern int __underflow (_IO_FILE *);
  extern int __uflow (_IO_FILE *);
  extern int __overflow (_IO_FILE *, int);
  extern int _IO_getc (_IO_FILE *__fp);
```

```
  extern int _IO_putc (int __c, _IO_FILE *__fp);
  extern int _IO_feof (_IO_FILE *__fp) __attribute__
((__nothrow__ , __leaf__));
  extern int _IO_ferror (_IO_FILE *__fp) __attribute__
((__nothrow__ , __leaf__));
  extern int _IO_peekc_locked (_IO_FILE *__fp);
  extern void _IO_flockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
  extern void _IO_funlockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
  extern int _IO_ftrylockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
```

```
  extern int _IO_vfscanf (_IO_FILE * __restrict, const char
* __restrict, __gnuc_va_list, int *__restrict);
  extern int _IO_vfprintf (_IO_FILE *__restrict, const char
*__restrict, __gnuc_va_list);
```

```
  extern __ssize_t _IO_padn (_IO_FILE *, int, __ssize_t);
  extern size_t _IO_sgetn (_IO_FILE *, void *, size_t);
  extern __off64_t _IO_seekoff (_IO_FILE *, __off64_t, int,
int);
  extern __off64_t _IO_seekpos (_IO_FILE *, __off64_t, int);
  extern void _IO_free_backup_area (_IO_FILE *)
__attribute__ ((__nothrow__ , __leaf__));
  typedef __gnuc_va_list va_list;
  typedef __off_t off_t;
  typedef __ssize_t ssize_t;
  typedef _G_fpos_t fpos_t;
  extern struct _IO_FILE *stdin;
  extern struct _IO_FILE *stdout;
  extern struct _IO_FILE *stderr;
```

```
  extern int remove (const char *__filename) __attribute__
((__nothrow__ , __leaf__));
  extern int rename (const char *__old, const char *__new)
__attribute__ ((__nothrow__ , __leaf__));
  extern int renameat (int __oldfd, const char *__old, int
__newfd,
         const char *__new) __attribute__ ((__nothrow__ ,
__leaf__));
  extern FILE *tmpfile (void) ;
  extern char *tmpnam (char *__s) __attribute__
((__nothrow__ , __leaf__)) ;
  extern char *tmpnam_r (char *__s) __attribute__
((__nothrow__ , __leaf__)) ;
  extern char *tempnam (const char *__dir, const char
*__pfx)
       __attribute__ ((__nothrow__ , __leaf__))
__attribute__ ((__malloc__)) ;
  extern int fclose (FILE *__stream);
  extern int fflush (FILE *__stream);
  extern int fflush_unlocked (FILE *__stream);
  extern FILE *fopen (const char *__restrict __filename,
       const char *__restrict __modes) ;
  extern FILE *freopen (const char *__restrict __filename,
         const char *__restrict __modes,
         FILE *__restrict __stream) ;
  extern FILE *fdopen (int __fd, const char *__modes)
__attribute__ ((__nothrow__ , __leaf__)) ;
  extern FILE *fmemopen (void *__s, size_t __len, const char
*__modes)
    __attribute__ ((__nothrow__ , __leaf__)) ;
  extern FILE *open_memstream (char **__bufloc, size_t
*__sizeloc) __attribute__ ((__nothrow__ , __leaf__)) ;
```

```
  extern void setbuf (FILE *__restrict __stream, char
*__restrict __buf) __attribute__ ((__nothrow__ , __leaf__));
  extern int setvbuf (FILE *__restrict __stream, char
*__restrict __buf,
        int __modes, size_t __n) __attribute__ ((__nothrow__
, __leaf__));
  extern void setbuffer (FILE *__restrict __stream, char
*__restrict __buf,
          size_t __size) __attribute__ ((__nothrow__ ,
__leaf__));
  extern void setlinebuf (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
```

```
  extern int fprintf (FILE *__restrict __stream,
        const char *__restrict __format, ...);
  extern int printf (const char *__restrict __format, ...);
  extern int sprintf (char *__restrict __s,
        const char *__restrict __format, ...) __attribute__
((__nothrow__));
  extern int vfprintf (FILE *__restrict __s, const char
*__restrict __format,
         __gnuc_va_list __arg);
  extern int vprintf (const char *__restrict __format,
__gnuc_va_list __arg);
  extern int vsprintf (char *__restrict __s, const char
*__restrict __format,
         __gnuc_va_list __arg) __attribute__
((__nothrow__));
  extern int snprintf (char *__restrict __s, size_t
__maxlen,
         const char *__restrict __format, ...)
```

```
      __attribute__ ((__nothrow__)) __attribute__
((__format__ (__printf__, 3, 4)));
  extern int vsnprintf (char *__restrict __s, size_t
__maxlen,
         const char *__restrict __format, __gnuc_va_list
__arg)
      __attribute__ ((__nothrow__)) __attribute__
((__format__ (__printf__, 3, 0)));
  extern int vdprintf (int __fd, const char *__restrict
__fmt,
        __gnuc_va_list __arg)
      __attribute__ ((__format__ (__printf__, 2, 0)));
  extern int dprintf (int __fd, const char *__restrict
__fmt, ...)
      __attribute__ ((__format__ (__printf__, 2, 3)));
```

```
  extern int fscanf (FILE *__restrict __stream,
      const char *__restrict __format, ...) ;
  extern int scanf (const char *__restrict __format, ...) ;
  extern int sscanf (const char *__restrict __s,
      const char *__restrict __format, ...) __attribute__
((__nothrow__ , __leaf__));
  extern int fscanf (FILE *__restrict __stream, const char
*__restrict __format, ...) __asm__ ("" "__isoc99_fscanf") ;
  extern int scanf (const char *__restrict __format, ...)
__asm__ ("" "__isoc99_scanf") ;
  extern int sscanf (const char *__restrict __s, const char
*__restrict __format, ...) __asm__ ("" "__isoc99_sscanf")
__attribute__ ((__nothrow__ , __leaf__));
```

```
  extern int vfscanf (FILE *__restrict __s, const char
*__restrict __format,
       __gnuc_va_list __arg)
       __attribute__ ((__format__ (__scanf__, 2, 0))) ;
  extern int vscanf (const char *__restrict __format,
__gnuc_va_list __arg)
       __attribute__ ((__format__ (__scanf__, 1, 0))) ;
  extern int vsscanf (const char *__restrict __s,
       const char *__restrict __format, __gnuc_va_list
__arg)
       __attribute__ ((__nothrow__ , __leaf__))
__attribute__ ((__format__ (__scanf__, 2, 0)));
  extern int vfscanf (FILE *__restrict __s, const char
*__restrict __format, __gnuc_va_list __arg) __asm__ (""
"__isoc99_vfscanf")
       __attribute__ ((__format__ (__scanf__, 2, 0))) ;
  extern int vscanf (const char *__restrict __format,
__gnuc_va_list __arg) __asm__ ("" "__isoc99_vscanf")
       __attribute__ ((__format__ (__scanf__, 1, 0))) ;
  extern int vsscanf (const char *__restrict __s, const char
*__restrict __format, __gnuc_va_list __arg) __asm__ (""
"__isoc99_vsscanf") __attribute__ ((__nothrow__ , __leaf__))
       __attribute__ ((__format__ (__scanf__, 2, 0)));
```

```
  extern int fgetc (FILE *__stream);
  extern int getc (FILE *__stream);
  extern int getchar (void);
  extern int getc_unlocked (FILE *__stream);
  extern int getchar_unlocked (void);
  extern int fgetc_unlocked (FILE *__stream);
  extern int fputc (int __c, FILE *__stream);
  extern int putc (int __c, FILE *__stream);
  extern int putchar (int __c);
```

```
  extern int fputc_unlocked (int __c, FILE *__stream);
  extern int putc_unlocked (int __c, FILE *__stream);
  extern int putchar_unlocked (int __c);
  extern int getw (FILE *__stream);
  extern int putw (int __w, FILE *__stream);
  extern char *fgets (char *__restrict __s, int __n, FILE
*__restrict __stream)
       ;
  extern __ssize_t __getdelim (char **__restrict __lineptr,
           size_t *__restrict __n, int __delimiter,
           FILE *__restrict __stream) ;
  extern __ssize_t getdelim (char **__restrict __lineptr,
          size_t *__restrict __n, int __delimiter,
          FILE *__restrict __stream) ;
  extern __ssize_t getline (char **__restrict __lineptr,
         size_t *__restrict __n,
         FILE *__restrict __stream) ;
  extern int fputs (const char *__restrict __s, FILE
*__restrict __stream);
  extern int puts (const char *__s);
  extern int ungetc (int __c, FILE *__stream);
  extern size_t fread (void *__restrict __ptr, size_t
__size,
         size_t __n, FILE *__restrict __stream) ;
  extern size_t fwrite (const void *__restrict __ptr, size_t
__size,
          size_t __n, FILE *__restrict __s);
  extern size_t fread_unlocked (void *__restrict __ptr,
size_t __size,
           size_t __n, FILE *__restrict __stream) ;
  extern size_t fwrite_unlocked (const void *__restrict
__ptr, size_t __size,
            size_t __n, FILE *__restrict __stream);
```

```
  extern int fseek (FILE *__stream, long int __off, int
__whence);
  extern long int ftell (FILE *__stream) ;
  extern void rewind (FILE *__stream);
  extern int fseeko (FILE *__stream, __off_t __off, int
__whence);
  extern __off_t ftello (FILE *__stream) ;
  extern int fgetpos (FILE *__restrict __stream, fpos_t
*__restrict __pos);
  extern int fsetpos (FILE *__stream, const fpos_t *__pos);
  extern void clearerr (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
  extern int feof (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern int ferror (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern void clearerr_unlocked (FILE *__stream)
__attribute__ ((__nothrow__ , __leaf__));
  extern int feof_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern int ferror_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern void perror (const char *__s);
  extern int sys_nerr;
  extern const char *const sys_errlist[];
  extern int fileno (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern int fileno_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern FILE *popen (const char *__command, const char
*__modes) ;
  extern int pclose (FILE *__stream);
```

```
  extern char *ctermid (char *__s) __attribute__
((__nothrow__ , __leaf__));
  extern void flockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
  extern int ftrylockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
  extern void funlockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));

  extern void *Read_Sym(int, char*);
  int Numbers[10];
  int main()
  {
      char Symbols[14] = "91A23B456C789";
      int i;
      printf ("%s\n",Symbols);
      Read_Sym(8, Symbols);
      for (i = 0; i < 8; i++)
          printf("%d\t", Numbers[i]);
      return 0;
  }
```

«Впечатляющий» список разнообразного ПО, разработанного на C **с использование** glibc **можно найти по ссылке:**

https://github.com/kozross/awesome-c