

# Лекция 4

## Взаимодействие с окружением программы на Си/Си++

# Реализации Си

- Freestanding реализация – поддерживается ограниченный набор заголовочных файлов и стандартных функций (например, `memset`)
  - Для ядер операционных систем
  - Для встроенных систем (embedded) без управления ОС
- Hosted реализация – полный набор (возможно, кроме опциональных) заголовочных файлов и библиотечных функций
  - Программирование на уровне пользовательских программ ОС

# Стандартная библиотека Си (hosted в Linux)

- В Unix системах традиционно называется libc, является частью ОС
- Заголовочные файлы размещаются в /usr/include
- Бинарный динамически загружаемый файл: /lib/libc.so.6 (Linux)
- Помимо функций библиотеки Си содержит и функции POSIX и расширения
- Библиотека математических функций отдельно – libm – требуется опция -lm при компиляции

# Взаимодействие программы на Си с окружением

- Стандартные потоки ввода и вывода `stdin`, `stdout`, `stderr`
- Аргументы командной строки
- Переменные окружения
- Код завершения программы

# Обработка ошибок

- Библиотечные функции и системные вызовы в случае ошибки возвращают специальное значение (например, `fork` возвращает `NULL`, часто возвращается `-1`)
- В этом случае переменная `errno` содержит код ошибки, например, `EPERM`, `EAGAIN`
- Переменная `errno` и коды ошибок определены в `<errno.h>`
- `strerror` из `<string.h>` возвращает строку, соответствующую ошибке
- Сообщения об ошибках должны выводиться на `stderr`

# Взаимодействие со средой

- Процесс завершается системным вызовом `_exit(exitcode)` или `exit` или `_Exit`
- Или возвращаемое значение `return` из `main`
- Значение в диапазоне `[0;255]` — код завершения процесса, он доступен процессу-родителю
- Код `0` — успешное завершение (`/bin/true`)
- Ненулевой код — ошибка (`/bin/false`)

# Аргументы командной строки

- Функция `main` получает аргументы командной строки:

```
int main(int argc, char *argv[])
```

- `argv` — массив указателей на строки Си

```
./prog foo 1 bar
```

```
argv[0] → "./prog";    путь к программе
```

```
argv[1] → "foo";
```

```
argv[2] → "1";
```

```
argv[3] → "bar";
```

```
argv[4] → NULL;
```

- Передаются на стеке процесса

# argv[0]

- Обычно argv[0] – путь, использованный для запуска программы
- Некоторые программы анализируют argv[0] и модифицируют свое поведение (например, busybox)



# Переменные окружения

- Именованные значения доступные процессу
- По умолчанию передаются неизменными порождаемым процессам

```
char *getenv(const char *name);
```

- В процесс передаются на стеке
- Глобальная переменная `environ` содержит указатель на массив переменных

# Строки в Си

- Null-terminated strings – в конце строки находится байт 0 (или '\0') – признак конца строки
- Строковые литералы “abcd” содержат “невидимый” \0 в конце
  - `char s[] = “abcd”; // sizeof(s) == 5`
- Если под строковый литерал память явно не выделяется, он размещаются в read-only памяти
  - `char *s = “abcd”; // sizeof(s) = sizeof(void*)`  
`s[2] = 'd'; // undefined behavior`

# Pros & contras

- (+) для работы со строкой достаточно одного указателя
- (+) сдвигая указатель по строке вперед все равно получаем строку
- (-) получение длины строки (strlen) выполняется за линейное время
  - **НИКОГДА!**  
for (int i = 0; i < strlen(s); ++i) {...}
- (-) нельзя использовать \0 в строке

# Альтернативы

- Хранить пару <указатель, длина> (std::string)
  - (+) нет проблемы байта \0
  - (-) размер такой структуры в два раза больше (а размер самой строки на один байт меньше)
- Хранить длину в начале строки (pascal style)
  - Либо ограниченный размер (если длина – 1 байт), либо неэффективное использование памяти (4 байта длины для коротких строк - много)

# Управление памятью

- В Си практически все управление памятью возложено на программиста
- При работе с указателями важно понимать, как и где выделена память, на которую он указывает:
  - Глобальная/статическая память
  - Thread-local storage
  - Автоматическая память
  - Динамическая память (куча)

# Буфер строки

- Буфер – область памяти, отведенная для хранения строки
- Буфер имеет ограниченный размер, но размер может изменяться
- При обработке строки “на чтение” достаточно только указателя на строку
- При формировании строки в памяти важен и адрес буфера, и размер буфера

# Переполнение буфера

- Если не контролируется размер данных, записываемых в буфер, возможно **переполнение буфера**
- Может иметь катастрофические последствия для безопасности системы (arbitrary code execution)

# Переполнение буфера

- Если не контролируется размер данных, записываемых в буфер, возможно **переполнение буфера**
- Может иметь катастрофические последствия для безопасности системы (arbitrary code execution)
- CVE-2015-2712 (Firefox)
- CVE-2010-1117 (IE)
- CVE-2016-5157 (Chrome)



# Good vs evil

- “Плохие” функции: записывают строку, но не принимают параметр размера буфера: `gets`, `scanf(“%s”, ...)`, `strcpy`, `sprintf`
  - `gets`, `scanf` – запрещены; `strcpy`, `sprintf` – крайне осторожно
- “Хорошие” функции: записывают строку и принимают размер буфера строки: `fgets`, `snprintf`, `scanf(“%100s”, ...)`