03. Обработка строковых данных

Егор Орлов

Курс: UNIX-DEV-SYS. Системное программирование в среде UNIX (Linux/FreeBSD). ВИШ СПбПУ, 2021

Содержание

1. Вводная информация. Проект. Цели-задачи

1.1. Отправная точка

- Пишем Web-сервер, работающий через **xinetd**
- **myweb/myweb.c** из основного репозитория

```
https://github.com/hse-labs/unix-dev-sys
```

1.2. Задачи в проекте

- **Модуль 3**
  - Собрать и запустить
  - Заставить работать за xinetd
  - Неализовать вывод HTTP-страницы
  - Реализовать полноценный разбор HTTP-заголовков запроса
  - Протестировать работу с реальным браузером
  - Описать сборку/развертывание для make/CMake

1.3. Инструменты

- Основное. Функции ISO C по работе со строками
- Дополнительно. Расширения POSIX C
- Дополнительно. Расширения C90 для работы с многосимвольными кодировками
- Дополнительно. Расширения C11 для работы с строками в Unicode
- Дополнительно. Возможности библиотеки **glib**

1.4. Дополнительная информация

- Простым языком об HTTP
  - https://habr.com/ru/post/215117/
- Обзор протокола HTTP
  - https://developer.mozilla.org/ru/docs/Web/HTTP/Overview

1.5. Особенности передачи сообщений в HTTP

- Символы переноса строки в протоколе HTTP

- **Carriage Return (CR)** - возврат каретки
- **Line Feed (LF)** - перевод строки

1.5.1. Пример запроса

- Запрос, передаваемый по TCP

```
GET / HTTP/1.1\r\n
Host: 127.0.0.1\r\n
\r\n
```

- Средствами netcat

```
$ echo "GET / HTTP/1.1\r\n\r\n" | nc 127.0.0.1 80
```

1.5.2. Пример ответа

- Ответ, передаваемый по TCP

```
HTTP/1.1 200 OK\r\n
Content-Type: text/html\r\n
\r\n
<html><body><title>Page title</title><h1>Page Header</h1></doby></html>\r\n
```

- Реализация на Си

```
printf("HTTP/1.1 200 OK\r\n");
printf("Content-Type: text/html\r\n");
printf("\r\n");
printf("<html><body><title>Page title</title><h1>Page Header</h1></doby></html>\r\n");
```

2. Стандартные потоки ввода-вывода и xinetd

2.1. Консольный ввод-вывод

Рис. 1: Консольный ввод-вывод процесса

2.2. Стандартные дискрипторы ввода/вывода

ID	Имя	Описание	Перенаправление
0	STDIN	стандартный поток ввода	<
1	STDOUT	стандартный поток вывода	>(1>) / »

ID	Имя	Описание	Перенаправление
2	STDERR	стандартный поток ошибок	2>

2.2.1. Пример

```
$ ls -l dir 2> err-list 1> corr-list
```

2.3. Получение STDIN из файла

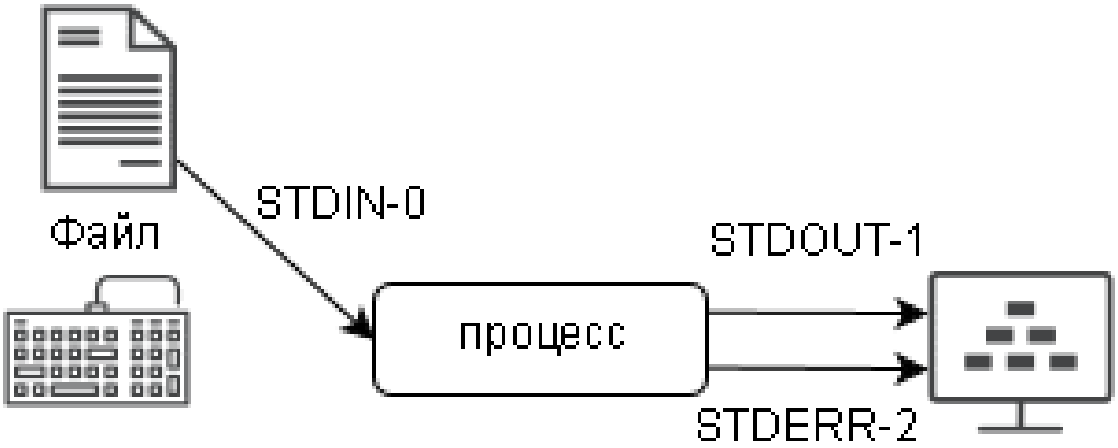

Рис. 2: Перенаправление стандартного потока ввода процесса

2.3.1. Пример

```
$ sort < /etc/passwd
```

2.4. Перенаправление STDOUT в файл

2.4.1. Пример

```
$ echo "Line 2" >> example.txt
$ ls -l /var/log > log.files
$ tar -zc include > ~/include-backup.tar.gz
$ gzip < file.txt > file.txt.gz
```

2.5. Перенаправление STDERR в файл

```
$ ls /fake
ls: cannot access /fake: No such file or directory
$ ls /fake > output.txt
ls: cannot access /fake: No such file or directory
$ ls /fake 2> error.txt
```

- Подавление вывода

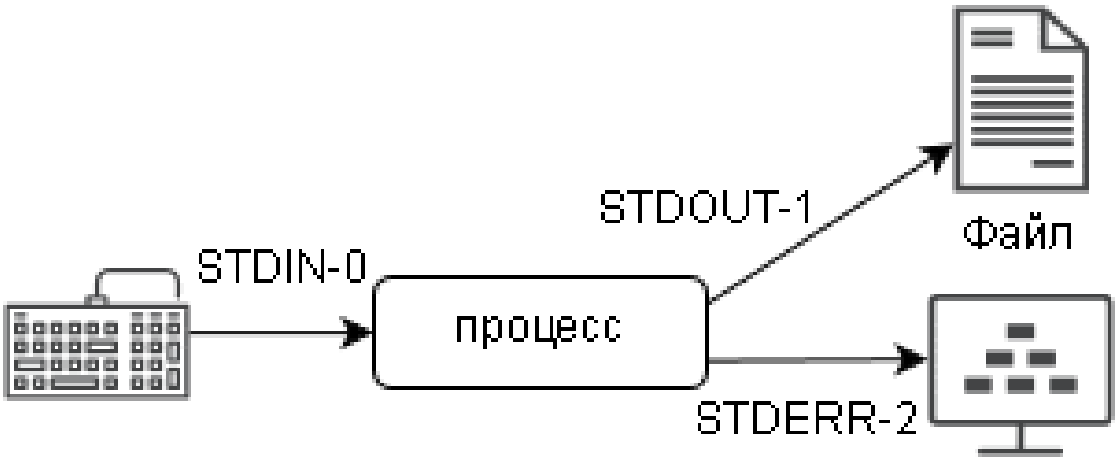

Рис. 3: Перенаправление стандартного потока вывода в файл

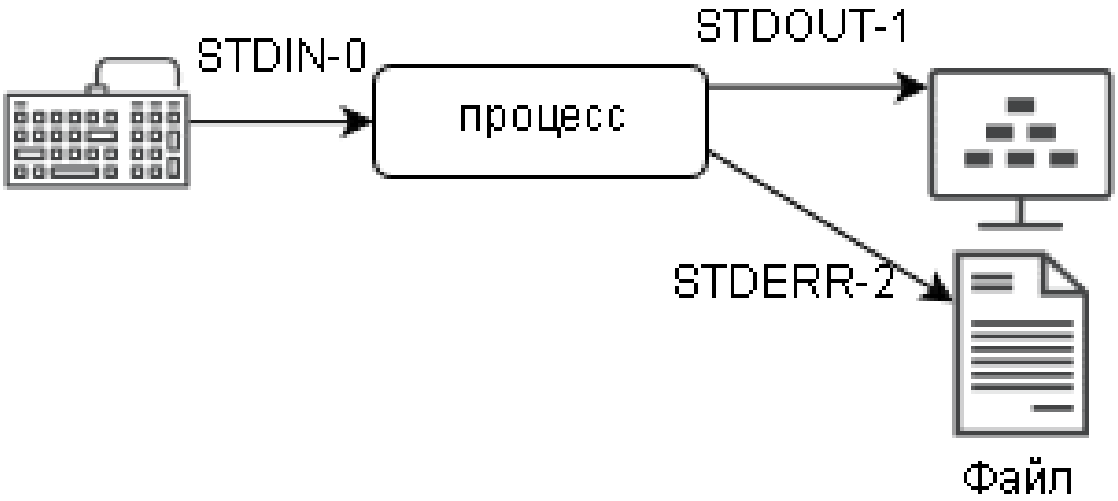

Рис. 4: Перенаправление стандартного потока ошибок в файл

```
$ ls /not-exist 2> /dev/null
```

вывод в никуда - подавление потока ошибок

2.6. (Д) Объединение потоков вывода

```
$ ls /fake /etc/ppp > example.txt 2> error.txt
```

Направляем **STDOUT** и **STDERR** в разные файлы

```
$ ls /fake /etc/ppp &> all.txt
```

Объединяем потоки **STDOUT** и **STDERR**

2.7. Каналы команд (Command Line Pipes)

```
$ ls /etc | head
$ ls /etc/ppp | nl
$ cat /etc/passwd | sort
$ cat /var/log/messages | grep 'error'
$ cat /var/log/syslog | grep 'error'
```

Принцип - **STDOUT** одного процесса объединяется с **STDIN** другого процессом средствами командного интерпретатора

```
$ ls /fake /etc/ppp 2>&1 | nl
```

Передача потока ошибок в канал - требует конструкции объединения каналов

2.8. Стандартные потоки ввода-вывода и xinetd

- **xinetd** - замена ранее используемого в UNIX-подобных системах **inetd**
- Прослушивает сетевые соединения и передает получаемые/передаваемые данные на процессы-серверы, связываясь с ними путем замыкания на себя всех стандартных потоков ввода-вывода.

```
$ cat /etc/xinetd.d/myweb-xinetd
service http
{
       socket_type             = stream
       wait                    = no
       user                    = root
       server                  = /usr/local/bin/myweb
       server_args             = arg1
}
```

- Описание xinetd
  - https://www.opennet.ru/docs/RUS/xinetd/xinetd.html
  - https://www.opennet.ru/docs/RUS/xinetd/xinetd.conf.html
- Пример реализации (на bash)

```
https://github.com/rglaue/xinetd_bash_http_service
```

2.9. Сборка и установка сервиса в систему

```
$ sudo apt-get install xinetd
```

Установка xinetd (суперпользователь)

```
$ git pull origin lesson3
$ cd unix-dev-sys/myweb
$ make myweb
```

Получение ветки репозитория lesson3 и сборка

```
$ sudo cp myweb /usr/local/bin/
$ sudo cp myweb-xinetd /etc/xinetd.d/
$ sudo systemctl restart xinetd
```

Установка в систему, перезапуск службы

3. Функции string.h (ISO C)

3.1. Описание

- Содержит функции для работы со строками, оканчивающимися на 0 (указатели на char), и различными функциями работы с памятью
- Функции гарантированно работают на всех платформах, поддерживающих Си
- Строковые функции работают только с набором символов ASCII или его совместимыми расширениями
- При использовании с многобайтовыми кодировками (UTF-8) будут работать, с отличием, что «длина» строки будет определяться как число байтов, а не число символов Юникода, которым они соответствуют.
- Большинство функций не занимается выделением памяти
- Значительная часть не занимается контролем границ

```
$ man string.h
```

3.2. Вычисление длины строки - strlen

```
int strlen(const char *str);
```

Вычисление длины строки

```
$ man strlen
```

3.3. Копирование строки - strcpy

```
char *strcpy(char *dest, const char *src)
```

Копирование строки **src** в заранее подготовленный массив **dest**, в котором должно быть достаточно места, иначе переполнение буфера. Возвращает **dest**.

```
$ man strcpy
```

3.4. Копирование строки с контролем длины - strncpy

```
char *strncpy(char *dest, const char *src, int size)
```

Копирование **int** байт из строки **src** в заранее подготовленный массив **dest**. Возвращает **dest**.

```
$ man strncpy
```

3.5. Сравнение строки - strcmp

```
int strcmp(const char *s1, const char *s2)
```

Посимвольно сравнивает строки, начиная с адресов **s1** и **s2** до конца одной из строк, или первого расхождения. Возвращает 0, если строки равны, отрицательное число, если **s1** оказалась очерендым символом “меньше” или закончилась раньше и положительное число, если наоборот.

```
$ man strcmp
```

3.6. Сравнение строки с контролем длины - strncmp

```
int strncmp(const char *s1, const char *s2, int n)
```

Посимвольно сравнивает строки, начиная с адресов **s1** и **s2** до символа с номером **n**, конца одной из строк или первого расхождения. Возвращает 0, если строки равны, отрицательное число, если **s1** оказалась очерендым символом “меньше” или закончилась раньше и положительное число, если наоборот.

```
$ man strncmp
```

3.7. Адрес первого вхождения символа - strchr

```
char *strchr(const char *s, int c);
```

Возвращает адрес первого вхождения символа с кодом c в строке s, если символа не нашлось, то NULL

```
$ man strchr
```

3.8. Адрес последнего вхождения символа - strrchr

```
char *strrchr(const char *s, int c);
```

Возвращает адрес последнего вхождения символа с кодом c в строке s, если символа не нашлось, то NULL

```
$ man strrchr
```

3.9. Поиск в строке подстроки - strstr

```
char *strstr(const char *haystack, const char *needle)
```

Возвращает адрес того места в строке haystack, где найдена искомая подстрока needle, либо NULL, если ничего не нашлось

```
$ man strstr
```

3.10. Пример - myweb.c

```
#define HTTP_METHOD_LEN 6
#define HTTP_URI_LEN 100

#define REQ_END 100
#define ERR_NO_URI -100
#define ERR_ENDLESS_URI -101

struct http_req {
        char request[HTTP_REQUEST_LEN];
        char method[HTTP_METHOD_LEN];
        char uri[HTTP_URI_LEN];
        // uri_path
        // uri_params
        // version
        // user_agent
        // server
        // accept
};
int fill_req(char *buf, struct http_req *req) {
        if (strlen(buf) == 2) {
                // пустая строка (\r\n) означает конец запроса
                return REQ_END;
        }
        char *p, *a, *b;
        // Это строка GET-запроса
        p = strstr(buf, "GET");
        if (p == buf) {
                // Строка запроса должна быть вида
                // GET /dir/ HTTP/1.0
                // GET /dir HTTP/1.1
                // GET /test123?r=123 HTTP/1.1
                // и т.п.
                strncpy(req->request, buf, strlen(buf));
                strncpy(req->method, "GET", strlen("GET"));
                a = strchr(buf, '/');
                if ( a != NULL) { // есть запрашиваемый URI
                        b = strchr(a, ' ');
                        if ( b != NULL ) { // конец URI
                                strncpy(req->uri, a, b-a);
                        } else {
                                return ERR_ENDLESS_URI;
                                // тогда это что-то не то
                        }
                } else {
                        return ERR_NO_URI;
```

```
                        // тогда это что-то не то
                }
        }

        return 0;
}
int log_req(struct http_req *req) {
        // fprintf(stderr, "%s %s\n%s\n", req->request, req->method, req->uri);
        return 0;
}

int make_resp(struct http_req *req) {
        printf("HTTP/1.1 200 OK\r\n");
        printf("Content-Type: text/html\r\n");
        printf("\r\n");
        printf("<html><body><title>Page title</title><h1>Page Header</h1></doby></html>\r\n");
        return 0;
}
int main (void) {
        char buf[HTTP_HEADER_LEN];
        struct http_req req;
        while(fgets(buf, sizeof(buf),stdin)) {
                int ret = fill_req(buf, &req);
                if (ret == 0)
                        // строка запроса обработана, переходим к следующей
                        continue;
                if (ret == REQ_END )
                        // конец HTTP запроса, вываливаемся на обработку
                        break;
                else
                        // какая-то ошибка
                        printf("Error: %d\n", ret);

        }
        log_req(&req);
        make_resp(&req);
}
```

4. Дополнительно

4.1. POSIX-расширение string.h

4.2. strdup

```
char *strdup(const char *);
```

Функция сама измеряет длину строки, выделит (malloc) для нее область динамической памяти необходимого размера, создает там копию строки и вернет указатель на неё. После использования необходимо освободить память при помощи free.

4.3. Функции wchar.h (ISO C, расширение C90)

- Содержит функции для работы с многобайтовыми и широкими символами
- Описание в стандарте SUS (The Single UNIX Specification) от The Open Group
  - https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/wchar.h.html
- Описание сопоставление типов и функций стандартной библиотеки и расширения C90
  - https://en.wikipedia.org/wiki/C_string_handling

4.4. Функции uchar.h (ISO C, расширение C11)

4.4.1. Описание

- Функции и типы для работы с символами Юникода (UTF-8, UTF-16, UTF-32)
- Стали важным нововведением C11
- Подробнее по форматам Unicode и их декодированию
  - https://unicodebook.readthedocs.io/unicode_encodings.html
  -

4.4.2. Типы

- **char16_t**

  Целочисленный тип без знака, используемый для представления 16-битных символов.

- **char32_t**

  Целочисленный тип без знака, используемый для представления 32-битных символов.

4.4.3. Функции

```
size_t c16rtomb (char * pmb, char16_t c16, mbstate_t * ps)
```

Преобразует 16-битный символ c16 в его многобайтовый эквивалент и сохраняет его в массиве, на который указывает pmb. Функция возвращает длину в байтах сохраненной многобайтовой последовательности.

```
size_t c32rtomb (char * pmb, char32_t c32, mbstate_t * ps)
```

Преобразует 32-битный символ c32 в его многобайтовый эквивалент и сохраняет его в массиве, на который указывает pmb. Функция возвращает длину в байтах сохраненной многобайтовой последовательности.

```
size_t mbrtoc16 (char16_t * pc16, const char * pmb, size_t max, mbstate_t * ps)
```

Читает не более max байт многобайтовой последовательности pmb и сохраняет ее эквивалент в виде 16-битного символа в переменную, на которую указывает pc16. Функция возвращает количество байт, которые потребовалось считать из последовательности pmb, чтобы получить 16-битный символ.

```
size_t mbrtoc32 (char32_t * pc32, const char * pmb, size_t max, mbstate_t * ps)
```

Читает не более max байт многобайтовой последовательности pmb и сохраняет ее эквивалент в виде 32-битного символа в переменную, на которую указывает pc32. Функция

возвращает количество байт, которые потребовалось считать из последовательности pmb, чтобы получить 32-битный символ.

4.5. Возможности библиотеки glib по работе со строками

- https://wiki.gnome.org/Projects/GLib
- Тип **GString** - структура, с точки зрения использования похожа на стандартные C строки, за исключением того, что они автоматически расширяются, когда текст добавляется или вставляется. Также, он хранит длину строки, так что может быть использован для двоичных данных с нулевыми байтами
  - https://docs.gtk.org/glib/struct.String.html
  - https://www.opennet.ru/docs/RUS/glib_api/glib-Strings.html

```
struct GString {
  gchar* str;
  gsize len;
  gsize allocated_len;
}
```