



**Министерство науки и высшего образования Российской
Федерации
Федеральное государственное бюджетное образовательное
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ

Информатика и системы управления

КАФЕДРА

Информационная безопасность (ИУ8)

Отчёт
по лабораторной работе №2
по дисциплине «Безопасность систем баз данных»

Выполнил: Песоцкий А. А.,
студент группы ИУ8-61

Проверил: Зенькович С. А.,
ассистент каф. ИУ8

Москва 2020 г.

ОГЛАВЛЕНИЕ

Table of Contents

ОГЛАВЛЕНИЕ.....	2
ВСТУПЛЕНИЕ.....	3
ЧТО ТАКОЕ MAN-PAGES	4
1. ДОСТУП К MAN-СТРАНИЦАМ.....	4
2. ФОРМАТ СТРАНИЦ	5
ОПИСАНИЕ КОМАНД.....	6
1. СИСТЕМНЫЕ	6
2. ПРОГРАММИРОВАНИЕ.....	13

ВСТУПЛЕНИЕ

Цель работы – ознакомиться с map-pages, изучить представленные в задании страницы и получить общее понимание устройства справочника.

ЧТО ТАКОЕ MAN-PAGES

man-страницы (от англ. *manual* — руководство) — справочные страницы, которые предоставляются почти всеми *nix-дистрибутивами, включая Arch Linux. Для их отображения служит команда `man`.

man-страницы изначально подразумевались как самостоятельные документы. Они ограничены в возможностях ссылаться друг на друга, в отличие от поддерживающих гиперссылки info-файлов — попытки GNU создать замену традиционному формату man-страниц.

1. ДОСТУП К MAN-СТРАНИЦАМ

Чтобы отобразить man-страницу, требуется набрать:

```
$ man имя_страницы
```

Все страницы разделены на несколько категорий:

1. Основные команды
2. Системные вызовы (функции, предоставляемые ядром linux)
3. Библиотечные вызовы (функции стандартной библиотеки языка Си)
4. Специальные файлы (обычно расположенные в `/dev`) и драйверы
5. Форматы файлов и соглашения
6. Игры
7. Прочие страницы (также включая соглашения)
8. Команды для системного администрирования (для которых обычно требуются права суперпользователя) и демоны

2. ФОРМАТ СТРАНИЦ

Для удобства навигации, все man-страницы соответствуют единому стандартному формату. Вот список некоторых разделов, которые часто используются на страницах:

- **NAME** — имя команды и краткое однострочное описание ее назначения.
- **SYNOPSIS** — список опций и аргументов командной строки, которые принимает команда, либо параметры функции и ее заголовочный файл.
- **DESCRIPTION** — более подробное описание назначения и принципов работы команды или функции.
- **EXAMPLES** — типовые примеры использования, обычно от самых простых к более сложным.
- **OPTIONS** — описания для каждой из опций, которые принимает команда.
- **EXIT STATUS** — коды возврата и их значения.
- **FILES** — связанные с командой или функцией файлы.
- **BUGS** — вероятные проблемы, связанные с работой команды или функции и ожидающие решения. Также известны как **KNOWN BUGS**.
- **SEE ALSO** — список связанных команд и функций.
- **AUTHOR, HISTORY, COPYRIGHT, LICENSE, WARRANTY** — информация о программе: ее история, условия использования, создатели программы.

ОПИСАНИЕ КОМАНД

1. СИСТЕМНЫЕ

```
$ man man
```

man (от англ. manual — руководство) — команда Unix, предназначенная для форматирования и вывода справочных страниц. Поставляется почти со всеми UNIX-подобными дистрибутивами. Каждая страница справки является самостоятельным документом и пишется разработчиками соответствующего программного обеспечения.

Если вы укажете раздел, то **man** будет искать только в этом разделе руководства. Имя страницы руководства обычно является именем команды, функции, или файла. Однако, если имя содержит наклонную черту (/), то **man** интерпретирует это как обращение к конкретному файлу.

Чтобы вывести справочное руководство по какой-либо команде (или программе, предусматривающей возможность запуска из терминала), можно в консоли ввести:

```
man <command_name>
```

Например, чтобы посмотреть справку по команде [ls](#), нужно ввести `man ls`.

```
$ man chown
```

Команда **chown** позволяет использовать соответствующую утилиту для изменения владельца файла или директории.

В Linux и других UNIX-подобных операционных системах каждый пользователь имеет свои собственные файлы, причем он может регламентировать возможность доступа других пользователей к ним. Применение концепции владения файлами имеет ряд последствий, причем в некоторых случаях бывает полезно изменять владельца некоторых файлов.

Базовый синтаксис команды выглядит следующим образом:

```
$ chown [параметры] <имя владельца:имя группы владельцев> <имя файла или директории>
```

Примеры использования

Смена владельца файла

Например, если вы хотите предоставить пользователю с именем **john** возможность распоряжаться файлом **picture.jpg** по своему усмотрению, вы можете воспользоваться следующей командой:

```
$ chown john picture.jpg
```

Помимо изменения владельца файла, может изменяться группа его владельцев или его владелец и группа его владельцев одновременно. Следует использовать символ двоеточия для разделения имени пользователя и имени группы пользователей (без символа пробела):

```
$ chown john:family picture.jpg
```

В результате владельцем файла **picture.jpg** станет пользователь с именем **john**, а группой его владельцев — **family**.

```
$ man chmod
```

Команда **chmod** предназначена для изменения прав доступа файлов и директорий в Linux. Название команды произошло от словосочетания «change mode».

Синтаксис команды chmod следующий:

```
chmod разрешения имя_файла
```

Разрешения можно задавать двумя способами:

- Числом
- Символами

Изменение прав доступа командой chmod

Запись прав доступа числом

Пример:

```
chmod 764 myfile
```

В данном формате права доступа задаются не символами `gwx`, как описано выше, а трехзначным числом. Каждая цифра числа означает определенный набор прав доступа.

- Первая цифра используется для указания прав доступа для пользователя.
- Вторая цифра для группы.
- Третья для всех остальных.

В таблице ниже приводятся все возможные комбинации разрешений `gwx` и соответствующие им числа (которые используются в команде `chmod`):

В качестве действий могут использоваться знаки "+" - включить или "-" - отключить. Рассмотрим несколько примеров:

- **u+x** - разрешить выполнение для владельца;
- **ugo+x** - разрешить выполнение для всех;
- **ug+w** - разрешить запись для владельца и группы;
- **o-x** - запретить выполнение для остальных пользователей;
- **ugo+rwX** - разрешить все для всех;

```
$ man fstab
```

`fstab` (сокр. от англ. file systems table) — один из конфигурационных файлов в UNIX-подобных системах, который содержит информацию о различных файловых системах и устройствах хранения информации компьютера; описывает, как диск (раздел) будет использоваться или как будет интегрирован в систему.

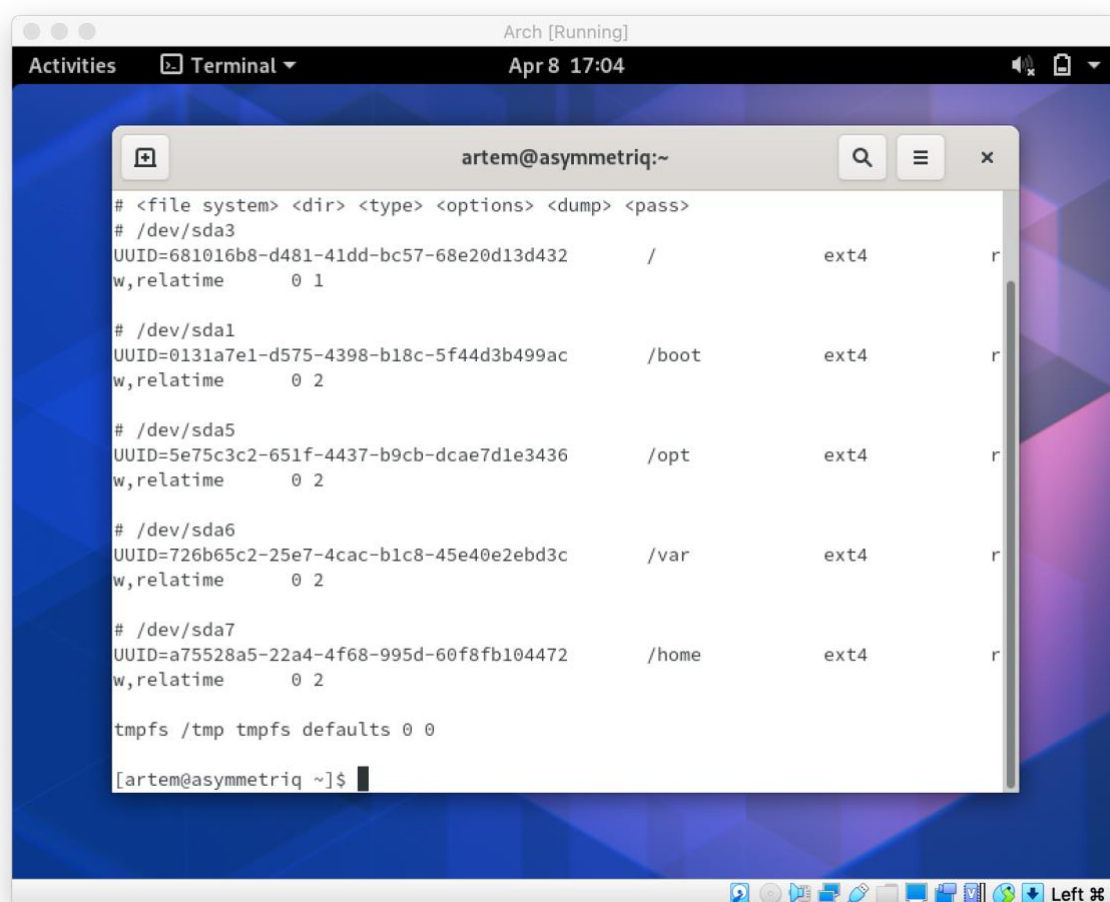
Полный путь к файлу — `/etc/fstab`.

Файл **`fstab`** содержит описательную информацию о различных файловых системах. **`fstab`** только читается программами, но не пишется; создание и изменение этого файла - задача системного администратора. Каждая файловая система описывается отдельной строкой; строки разделяются пробелами или символами табуляции. Начинающиеся на '#' строки считаются комментариями.

Каждая запись имеет следующие поля (которые разделяются пробелом или табуляцией):

<device-spec> <mount-point> <fs-type> <options> <dump> <pass>

- Поле, <device-spec> (*устройство*) сообщает [демону](#) монтирования файловых систем mount, что монтировать, имя монтируемого устройства или его метку.
- Второе поле, <mount-point> (*точка монтирования*), определяет путь, по которому будет смонтировано устройство <device-spec>.
- Поле <fs-type> (*тип файловой системы*) содержит тип файловой системы монтируемого устройства. Полный список поддерживаемых систем можно просмотреть выполнив команду:



```
# <file system> <dir> <type> <options> <dump> <pass>
# /dev/sda3
UUID=681016b8-d481-41dd-bc57-68e20d13d432      /      ext4      r
w,relatime      0 1

# /dev/sda1
UUID=0131a7e1-d575-4398-b18c-5f44d3b499ac      /boot   ext4      r
w,relatime      0 2

# /dev/sda5
UUID=5e75c3c2-651f-4437-b9cb-dcae7d1e3436      /opt    ext4      r
w,relatime      0 2

# /dev/sda6
UUID=726b65c2-25e7-4cac-b1c8-45e40e2ebd3c      /var    ext4      r
w,relatime      0 2

# /dev/sda7
UUID=a75528a5-22a4-4f68-995d-60f8fb104472      /home   ext4      r
w,relatime      0 2

tmpfs /tmp tmpfs defaults 0 0

[artem@asymmetriq ~]$
```

Рисунок 1. Пример таблицы на установленном Arch

\$ man proc

`/proc` — это псевдо-файловая система, которая используется в качестве интерфейса к структурам данных в ядре, чтобы избежать чтения и записи `/dev/kmem`. Большинство расположенных в ней файлов доступны только для чтения, но некоторые файлы позволяют изменять переменные ядра.

Информация о процессах хранится в директориях `/proc/N`, где `N` — числовой идентификатор процесса. В этой директории содержатся различные псевдо-файлы, которые содержат информацию о самом процессе и связанном с ним окружении.

`/proc/N/cmdline` — Содержимое командной строки, которой был запущен процесс.

`/proc/N/envIRON` — Описание окружения, в котором работает процесс. Оно может быть полезно для просмотра содержимого окружения, если вам надо, например, посмотреть, была ли установлена переменная окружения перед запуском программы.

`/proc/N/exe` — Символическая ссылка на выполнимый файл запущенной программы.

`/proc/N/limits` — Лимиты на использование системных ресурсов, актуальные для работающего процесса.

`/proc/N/mounts` — Список смонтированных ресурсов, которые доступны процессу

`/proc/N/status` — Статус работающей программы. Он включает в себя такую информацию как идентификатор родительского процесса, статус самого процесса, его название, его идентификатор, идентификатор пользователя и группы, группы, в которые входит владелец процесса, сколько потоков использует процесс, сколько памяти он использует и так далее.

В этой же директории содержится несколько псевдо-директорий:

`/proc/N/cwd` — Текущая директория для процесса. Представлена символической ссылкой на директорию. Если рабочая директория для процесса изменится, изменится и ссылка.

`/proc/N/fd` — Файловые дескрипторы, которые используются процессом. Для программы `bash`, например, там по умолчанию будут дескрипторы 0,

1, 2 и 255, указывающие на виртуальный терминал, в котором запущен процесс, например, /dev/pts/6.

/proc/N/fdinfo — Информация о файловых дескрипторах. Каждый файл в этой директории содержит поля pos (позиция курсора), flags (флаги, с которыми этот дескриптор был открыт) и mnt_id (идентификатор точки монтирования из списка, содержащегося в файле /proc/N/mountinfo)

/proc/N/root — Символическая ссылка на директорию, которая для данного процесса является корневой

/proc/N/net — Сетевые системные ресурсы и их параметры, действующие для конкретного процесса.

```
$ man signal
```

signal – асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия между процессами. Когда сигнал послан процессу, операционная система прерывает выполнение процесса, при этом, если процесс установил собственный *обработчик сигнала*, операционная система запускает этот обработчик, передав ему информацию о сигнале, если процесс не установил обработчик, то выполняется обработчик по умолчанию.

Linux поддерживает как точные сигналы POSIX (здесь и далее "стандартные сигналы"), так и сигналы POSIX для режима реального времени.

```
$ man sh
```

sh – командное название модернизированной вариации оболочки Bourne shell (Bash). Bash является предустановленной командной оболочкой в ОС Linux.

Утилита sh – стандартный интерпретатор команд.

Вы всегда можете запустить новый экземпляр оболочки `bash`, дав команду `bash` или `sh`. При этом можно заставить новый экземпляр оболочки выполнить какой-то скрипт, если передать имя скрипта в виде аргумента команды `bash`. Так, для выполнения скрипта `myscript` надо дать команду `"sh myscript.sh"`.

Если вы заглянете в какой-нибудь файл, задающий скрипт (таких файлов в системе очень много), вы увидите, что первая строка в нем имеет вид: `#!/bin/sh`. Это означает, что когда мы запускаем скрипт на выполнение как обычную команду, `/bin/sh` будет выполнять ее для нас. Можно заменить эту строку ссылкой на любую программу, которая будет читать файл и исполнять соответствующие команды.

2. ПРОГРАММИРОВАНИЕ

```
$ man stdio
```

`stdio` — библиотека функций стандартного ввода-вывода. Библиотека обеспечивает буферизируемый поток простым и эффективным интерфейсом. Ввод и вывод представляют собой поток логических данных, а физические характеристики ввода-вывода скрываются. Поток соотносится с внешним файлом (который может быть физическим устройством) посредством его открытия. Файл отвязывается от потока посредством закрытия файла.

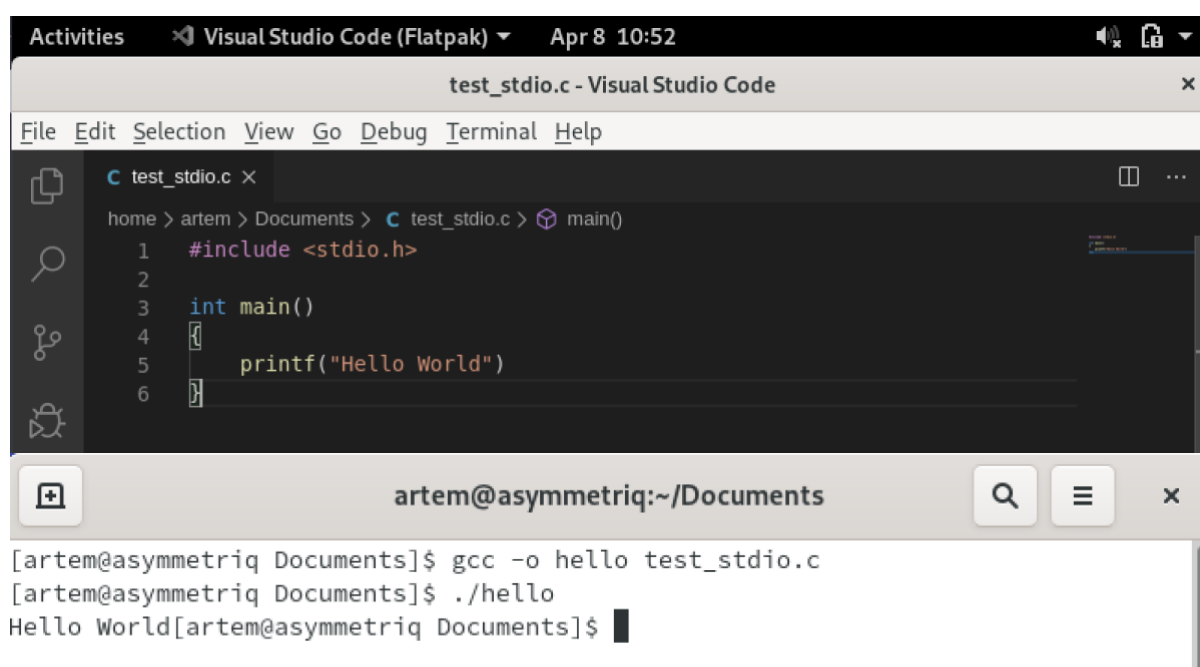


Рисунок 2. Самый очевидный пример – функция `printf()`

```
$ man stdin (stdout / stderr)
```

`stdin` `stdout` `stderr` — стандартные потоки ввода-вывода. В обычных условиях любая программа в Unix имеет три потока, открытых для нее при запуске: один для вывода, другой для ввода и один для вывода диагностики или сообщений об ошибках. Обычно они прикреплены к пользовательскому терминалу, но могут ссылаться и на другие файлы или

устройства в зависимости от того, что установлено родительским процессом. Поток ввода называется ``стандартным вводом (standard input)"; поток вывода называется ``стандартным выводом (standard output)"; а поток сообщений об ошибках называется ``стандартными ошибками (standard error)". Эти термины были сокращены для названий файлов, на которые указывают ссылки, а именно: *stdin* *stdout* и *stderr*.

Дескрипторы:

- 0: *stdin*
- 1: *stdout*
- 2: *stderr*

stdin:

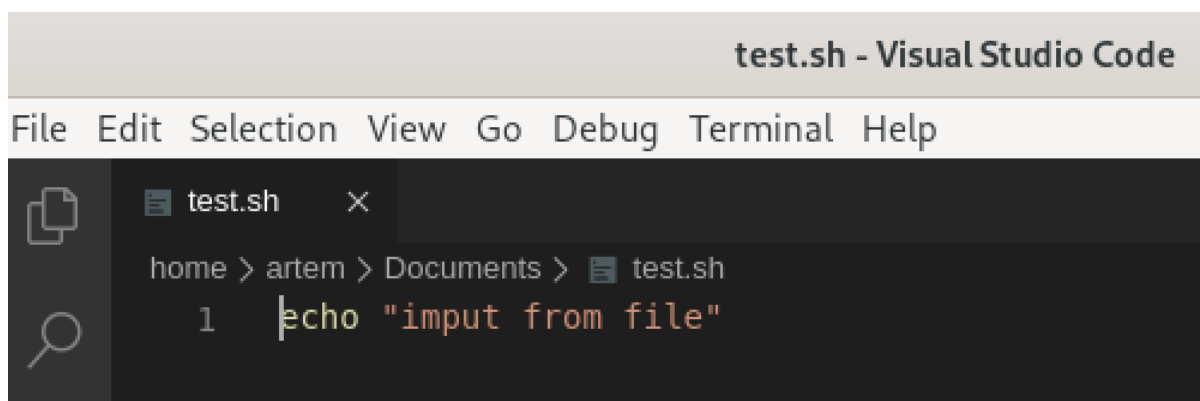
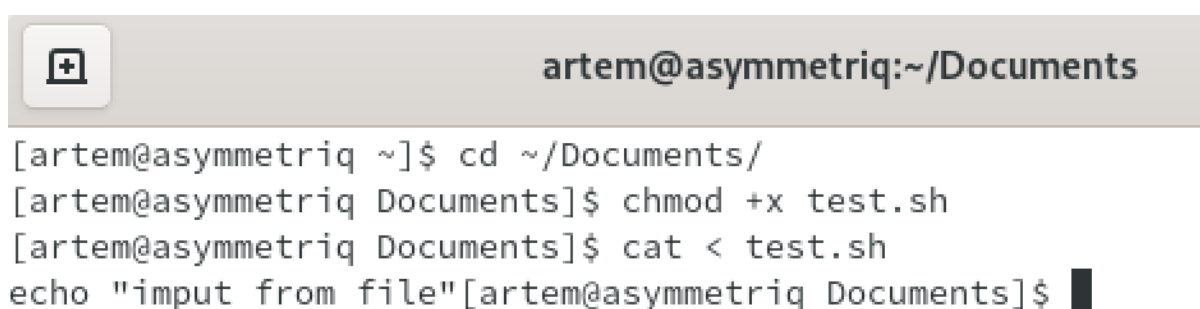
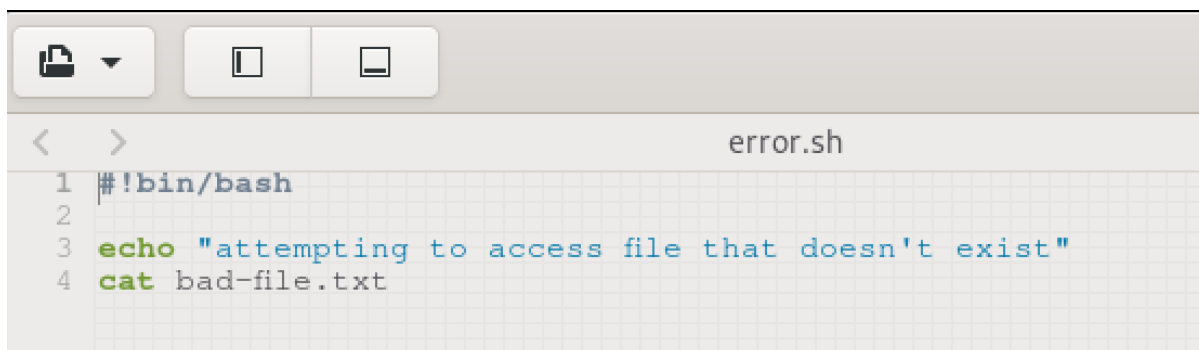


Рисунок 3. Создаём файл, из которого передадим данные в stdin вместо клавиатуры

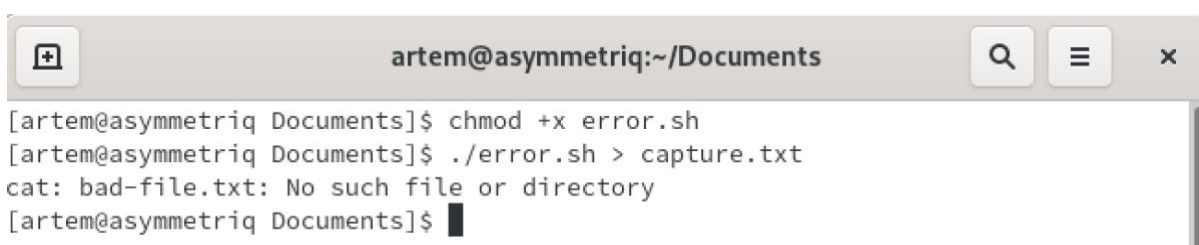


stdout u stderr:



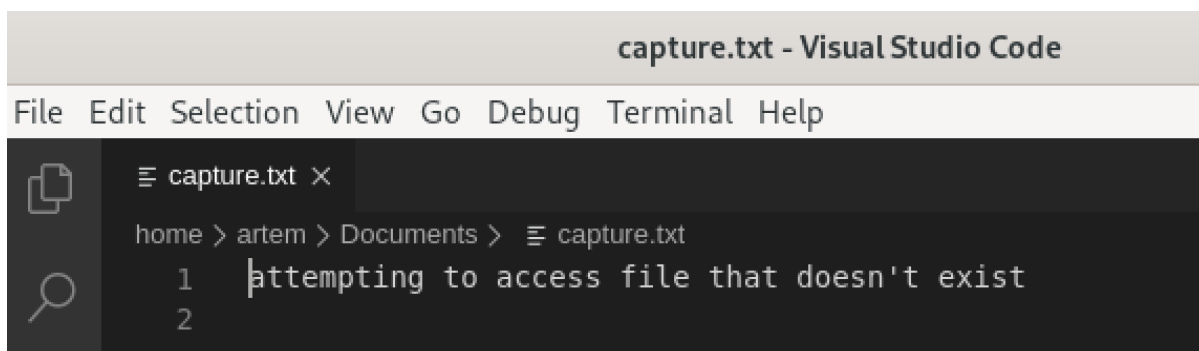
```
error.sh
1 #!/bin/bash
2
3 echo "attempting to access file that doesn't exist"
4 cat bad-file.txt
```

Рисунок 4. Создаём скрипт error.sh для демонстрации разных потоков stdout u stderr:



```
artem@asymmetriq:~/Documents
[artem@asymmetriq Documents]$ chmod +x error.sh
[artem@asymmetriq Documents]$ ./error.sh > capture.txt
cat: bad-file.txt: No such file or directory
[artem@asymmetriq Documents]$
```

Рисунок 5. Ошибка вывелась через отдельный поток stderr



```
capture.txt - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
home > artem > Documents > capture.txt
1 attempting to access file that doesn't exist
2
```

Рисунок 6. При этом результат работы echo (поток stdout) был перенаправлен в capture.txt

`$ man pipe`

Pipe (конвейер) – это однонаправленный канал межпроцессного взаимодействия.

Конвейеры чаще всего используются в shell-скриптах для связи нескольких команд путем перенаправления вывода одной команды (stdout) на вход (stdin) последующей, используя символ конвейера '|':

```
cmd1 | cmd2 | .... | cmdN
```

Например:

```
$ grep -i "error" ./log | wc -l
```

43

`pipe()` создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами. Массив *pipefd* используется для возврата двух файловых дескрипторов, указывающих на концы канала. *pipefd[0]* указывает на конец канала для чтения. *pipefd[1]* указывает на конец канала для записи. Данные, записанные в конец канала, буферизируются ядром до тех пор, пока не будут прочитаны из конца канала для чтения.

```
$ man fork
```

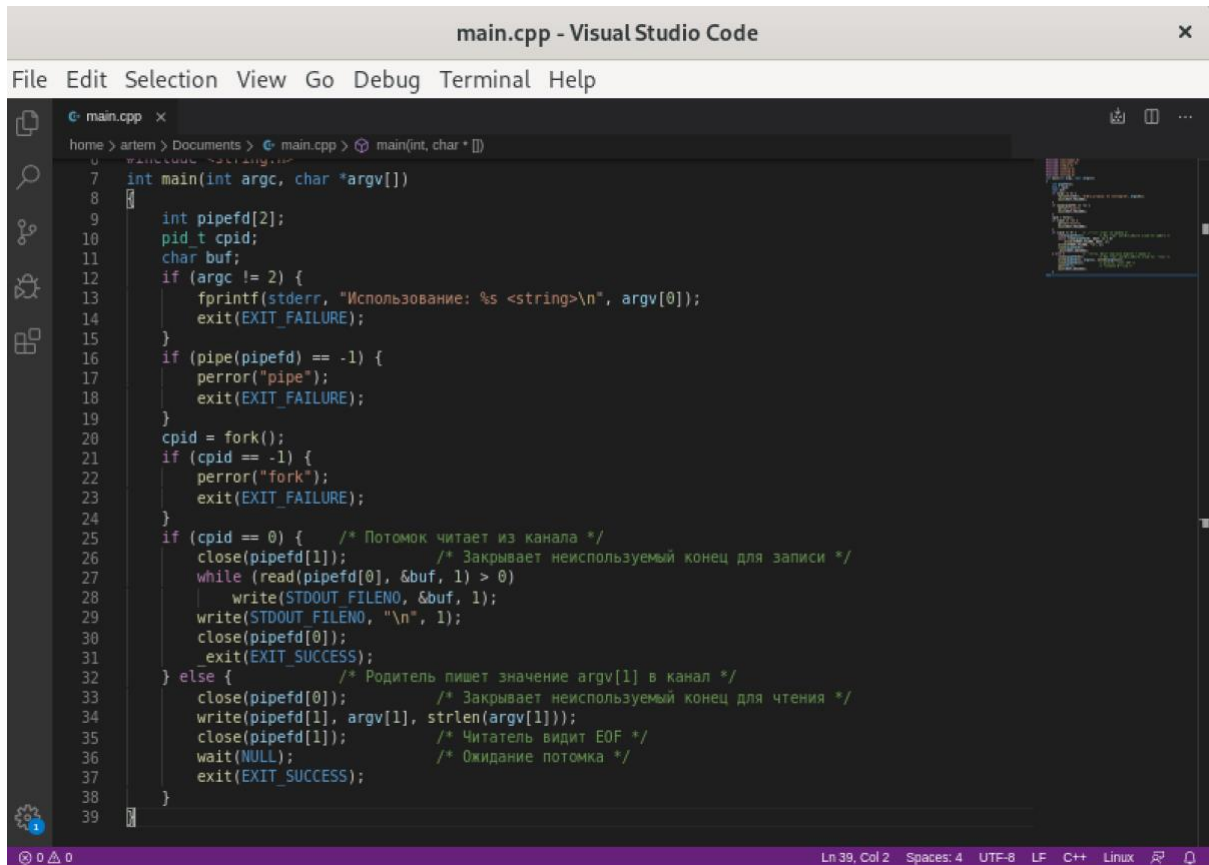
`fork()` — системный вызов в Unix-подобных операционных системах, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.

Вызов **fork()** создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается *дочерним* процессом. Вызывающий процесс считается *родительским* процессом. Дочерний и родительский процессы находятся в отдельных пространствах памяти. Сразу после **fork()** эти пространства имеют одинаковое содержимое. Запись в память, отображение и снятие отображения, выполненных в одном процессе, ничего не изменяет в другом.

Пример (pipe, fork):

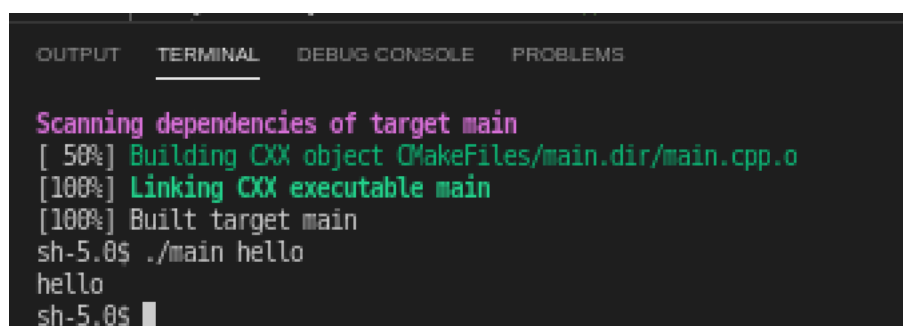
Следующая программа создаёт канал, и затем выполняет `fork` для создания потомка; потомок наследует скопированный набор файловых дескрипторов, которые указывают на тот же канал. После `fork` каждый

процесс закрывает файловые дескрипторы, которые не нужны каналу. Затем родитель записывает строку, переданную в качестве аргумента командной строки, в канал, а потомок читает эту строку из канала по байту за раз, и выводит её на стандартный вывод.



```
main.cpp - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

main.cpp
home > artem > Documents > main.cpp > main(int, char *[])
7 int main(int argc, char *argv[])
8 {
9     int pipefd[2];
10    pid_t cpid;
11    char buf;
12    if (argc != 2) {
13        fprintf(stderr, "Использование: %s <string>\n", argv[0]);
14        exit(EXIT_FAILURE);
15    }
16    if (pipe(pipefd) == -1) {
17        perror("pipe");
18        exit(EXIT_FAILURE);
19    }
20    cpid = fork();
21    if (cpid == -1) {
22        perror("fork");
23        exit(EXIT_FAILURE);
24    }
25    if (cpid == 0) { /* Потомок читает из канала */
26        close(pipefd[1]); /* Закрывает неиспользуемый конец для записи */
27        while (read(pipefd[0], &buf, 1) > 0)
28            write(STDOUT_FILENO, &buf, 1);
29        write(STDOUT_FILENO, "\n", 1);
30        close(pipefd[0]);
31        exit(EXIT_SUCCESS);
32    } else { /* Родитель пишет значение argv[1] в канал */
33        close(pipefd[0]); /* Закрывает неиспользуемый конец для чтения */
34        write(pipefd[1], argv[1], strlen(argv[1]));
35        close(pipefd[1]); /* Читатель видит EOF */
36        wait(NULL); /* Ожидание потомка */
37        exit(EXIT_SUCCESS);
38    }
39 }
```



```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

Scanning dependencies of target main
[ 50%] Building CXX object CMakeFiles/main.dir/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
sh-5.0$ ./main hello
hello
sh-5.0$
```

Рисунок 7. Пример использования

\$ man *dup*

Системный вызов `dup()` создаёт копию файлового дескриптора *oldfd*, используя для нового дескриптора самый маленький свободный номер файлового дескриптора.

После успешного выполнения старый и новый файловые дескрипторы являются взаимозаменяемыми. Они указывают на одно и то же открытое файловое описание и поэтому имеют общее файловое смещение и флаги состояния файла; например, если файловое смещение изменить с помощью `lseek` через один из файловых дескрипторов, то смещение изменится и для другого.

Эти два файловых дескриптора имеют различные флаги дескриптора файла (флаг `close-on-exec`). Флаг `close-on-exec` у копии дескриптора сбрасывается.

\$ man *exec*

Семейство функций `exec` заменяет текущий образ процесса новым образом процесса.

Начальным параметром этих функций будет являться полное имя файла, который необходимо исполнить.

Функции **`execv`** и **`execvp`** предоставляют процессу массив указателей на строки, заканчивающиеся `null`. Эти строки являются списком параметров, доступных новой программе. Первый аргумент, по соглашению, должен указать на имя, ассоциированное с файлом, который необходимо исполнить. Массив указателей *должен* заканчиваться **`NULL`**.

Функция **`execle`** также определяет окружение исполняемого процесса, помещая после указателя **`NULL`**, заканчивающего список параметров (или после указателя на массив), `argv` дополнительного параметра. Этот дополнительный параметр является массивом указателей на строки, завершаемые `null`, и *должен* заканчиваться указателем **`NULL`**. Другие функции извлекают окружение нового образа процесса из внешней переменной *environ* текущего процесса.

ЗАКЛЮЧЕНИЕ

Вывод:

В ходе выполнения лабораторной работы ознакомились с map-pages, изучили представленные в задании страницы и получили общее понимание устройства справочника.