Задание

По пути */home/<your_username>/src/simple_example/hello.c* находится код элементарной программы на языке C. Необходимо, находясь в этой же директории, скомпилировать её командой

```
gcc -o hello hello.c
```

Далее выполнить трассировку полученной программы с выводом статистики по системным вызовам

```
strace -c ./hello
```

После чего описать кратко каждый системный вызов из полученной табицы.

Решение

Прикрепим сначала код программы *hello.c*

```
#include <stdio.h>

int main(void) {
    puts("Hello, Fintech World!");
    return 0;
}
```

Системные вызовы (*различные* вызовы: некоторые вызывались по несколько раз)

- mmap — системный вызов Unix, позволяющий выполнить отображение файла или устройства на память: отображает *length* байтов, начиная со смещения *offset* файла, определенного файловым описателем *fd*, в память, начиная с адреса *start*.

  Параметр *prot* описывает желаемый режим защиты памяти (не должен конфликтовать с режимом открытия файла)

  Параметр flags задает тип отражаемого объекта, опции отражения и указывает, принадлежат ли отраженные данные только этому процессу или их могут читать и другие.

  При удачном выполнении *mmap()* возвращает указатель на область с отраженными данными. При ошибке возвращается значение -1, а переменная *errno* приобретает соответствующее значение.

```
void * mmap(void *start, size_t length, int prot,
int flags, int fd, off_t offset)
```

- mprotect — контролирует доступ к области памяти (некоторого размера по некоторому адресу):
  - запретить доступ
  - разрешить чтение
  - разрешить запись
  - разрешить содержание исполняемого кода

  Если программой производится запрещенный этой функцией доступ к памяти, то такая программа получает сигнал *SIGSEGV*.

При удачном завершении вызова возвращаемое значение равно нулю. При ошибке оно равно -1, а переменной *errno* присваивается номер ошибки.

```
int mprotect(const void *addr, size_t len, int prot)
```

- open — открывает файл или устройство, используется, чтобы преобразовать путь к файлу в описатель файла. Если системный вызов завершается успешно, возвращенный файловый описатель является наименьшим описателем, который еще не открыт процессом. В результате этого вызова появляется новый открытый файл, не разделяемый никакими процессами. Указатель устанавливается в начале файла. Можно открывать файлы "только для чтения", "только для записи" и "для чтения и записи".

  Параметр *mode* задает права доступа, которые используются в случае создания нового файла.

```
int open(const char *pathname, int flags)
int open(const char *pathname, int flags, mode_t
mode)
```

- munmap — снимает отражение файла или устройства в памяти: удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти" (invalid memory reference). Отражение удаляется автоматически при завершении процесса. Но закрытие файла не приводит к снятию отражения.

При удачном выполнении возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная *errno* приобретает соответствующее значение.

```
int munmap(void *start, size_t length)
```

- write — записывает до *count* байтов из буфера *buf* в файл, на который ссылается файловый описатель *fd*. В случае успешного завершения возвращается количество байтов, которые были записаны.

  В случае ошибки возвращается -1, а переменной *errno* присваивается соответствующее значение.

```
ssize_t write(int fd, const void *buf, size_t count)
```

- access — проверяет, имеет ли процесс права на чтение или запись, или же просто проверяет, существует ли файл (или другой объект файловой системы), с именем *pathname*.

  Если *pathname* является символьной ссылкой, то проверяются права доступа к файлу, на который она ссылается. Параметр *mode* — маска, состоящая из одного или более флагов-проверок:

  - существования файла + возможности его чтения
  - существования файла + возможности его записи
  - существования файла + возможности его выполнения
  - существования файла

  В случае успеха (есть все запрошенные права) возвращается нуль. При ошибке (по крайней мере один запрос прав из *mode* был не удовлетворен, или случилась другая ошибка), возвращается -1, а *errno*

устанавливается должным образом.

```
int access(const char *pathname, int mode)
```

- fstat — возвращает информацию об открытом файле, на который указывает filedes (возвращаемый *open()*) и заполняет буфер *buf*.

  Для вызова не требуется иметь права доступа к файлу, но требуются права поиска во всех каталогах, указанных в полном имени файла.

  В случае успеха возвращается ноль. При ошибке возвращается -1, а переменной *errno* присваивается номер ошибки.

```
int fstat(int filedes, struct stat *buf)
```

- read — пытается записать *count* байтов файлового описателя *fd* в буфер, адрес которого начинается с *buf*. Если количество *count* равно нулю, то *read()* возвращает это нулевое значение и завершает свою работу. Если count больше, чем *SSIZE_MAX*, то результат не может быть определен.

  При успешном завершении возвращается количество байтов, которые были считаны (нулевое значение означает конец файла), а позиция файла увеличивается на это значение. Если количество прочитанных байтов меньше, чем количество запрошенных, то это не считается ошибкой. В случае ошибки возвращаемое значение равно -1, а переменной *errno* присваивается номер ошибки. В этом случае позиция файла не определена.

```
ssize_t read(int fd, void *buf, size_t count)
```

- execve — выполняет программу, заданную параметром *filename*. Программа должна быть или двоичным исполняемым файлом, или скриптом, начинающимся со строки вида "#! интерпретатор [аргументы]". Файл будет выполнен как `интерпретатор [arg] filename`. Параметр *argv* — это массив строк, аргументов новой программы.

  Параметр *envp* — это массив строк в формате *key=value*, которые передаются новой программе в качестве окружения (environment).

  При успешном завершении *execve()* не возвращает управление, при ошибке возвращается -1, а значение *errno* устанавливается должным образом.

```
int execve(const char *filename, char *const argv
[], char *const envp[])
```

- arch_prctl — устанавливает специфичное для данной архитектуры состояние процесса или треда. Параметр *code* выбирает подфункцию и передаёт ей аргумент *addr*.

  В случае успеха возвращается ноль, иначе — минус единица, и устанавливается *errno*.

```
int arch_prctl(int code, unsigned long addr)
```

- close — закрывает файловый дескриптор *fd*, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются.

Если *fd* является последней копией какого-либо файлового дескриптора, то ресурсы, связанные с ним, освобождаются.

Возвращает ноль при успешном завершении или -1, если произошла ошибка.

```
int close(int fd)
```

- brk — устанавливает конец сегмента данных вызывающего процесса в значение, указанное в аргументе *end_data_segment*: точнее, происходит переустановка break value процесса (адреса первого байта за пределами сегмента данных процесса).

  При успехе возвращает ноль, иначе — минус один.

```
int brk(void *end_data_segment)
```