ФИО: Бюрчиев Тимур Зольванович
Ожидаемая оценка: 8 баллов

Интерфейс программы

Программа запускается из командной строки и принимает два аргумента:

```
./file_copy_syscalls <source> <dest>
```

- `<source>` — путь к исходному файлу;
- `<dest>` — путь к создаваемому (перезаписываемому) файлу.

При неправильном числе аргументов программа выводит подсказку:

```
Usage: file_copy_syscalls <source> <dest>
```

и завершает работу с кодом возврата `1`.

Используемые системные вызовы и функции

Для работы с файлами используются **только системные вызовы**:

- `open` — открытие исходного файла и создание/открытие целевого файла;
- `read` — чтение данных из файла;
- `write` — запись данных в файл;
- `close` — закрытие файловых дескрипторов;
- `fstat` — получение информации о файле (включая режим доступа `st_mode`).

Дополнительно используются функции стандартной библиотеки C:

- `perror` — вывод сообщений об ошибках на основе `errno`;
- `strlen` — вычисление длины строк для вызова `write` при выводе сообщений;
- типы и структуры: `struct stat`, `mode_t`, `ssize_t`, `off_t`.

Флаги и макросы:

- `STDERR_FILENO` — макрос, обозначающий файловый дескриптор стандартного потока ошибок. В программе используется как первый аргумент системного вызова `write` для вывода сообщений об ошибках: `write(STDERR_FILENO, ...)`.
- `O_RDONLY` — флаг для системного вызова `open`, задающий режим “только чтение”. В программе используется при открытии исходного файла: `open(source_path, O_RDONLY)`.
- `O_WRONLY | O_CREAT | O_TRUNC` — комбинация флагов для `open`:
  - `O_WRONLY` — открыть только для записи;
  - `O_CREAT` — создать файл, если он не существует;
  - `O_TRUNC` — обнулить содержимое файла, если он уже существует.
    В программе используется при открытии/создании целевого файла: `open(dst_path, O_WRONLY | O_CREAT | O_TRUNC, src_mode)`.

Алгоритм работы программы

1. **Проверка аргументов.**
   Если `argc != 3`, выводится сообщение об использовании и программа завершает работу с кодом `1`.
   - `argv[0]` — строка с именем программы: `"./file_copy_syscalls"`
   - `argv[1]` — первый аргумент: `"test.txt"` (путь к исходному файлу)
   - `argv[2]` — второй аргумент: `"copy.txt"` (путь к целевому файлу)
2. **Открытие исходного файла.**

   ```
   int source_fd = open(source_path, O_RDONLY);
   ```

   Возвращаемое значение:

   - `>= 0` — **успех**, это файловый дескриптор (0, 1, 2, 3, ...);
   - `-1` — **ошибка**, при этом устанавливается глобальная переменная `errno`.

   Поэтому:

   - `source_fd < 0` означает: `open` вернул `-1` => файл открыть не удалось.
     При ошибке вызывается `perror("open source")` и программа завершается.
3. **Получение режима доступа исходного файла.**

   ```
   struct stat st;
   fstat(src_fd, &st);
   mode_t source_mode = st.st_mode & 0777;
   ```

`struct stat` — структура, определённая в `<sys/stat.h>`, в ней хранится информация о файле:

- размер (`st_size`),
- права доступа (`st_mode`),
- тип файла и прочее.
- `fstat` заполняет структуру `stat` .

Побитовое `& 0777` оставляет только биты прав (`rwx` для пользователя, группы и остальных), без лишних флагов (тип файла и т.п.).

4. **Создание/открытие целевого файла.**

```
int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, src_mode);
```

- `O_WRONLY` — только запись;
- `O_CREAT` — создать файл, если он не существует;
- `O_TRUNC` — обнулить существующий файл;
- `source_mode` — устанавливает те же права доступа, что у исходного файла:
  - если исходный файл был исполняемым (скрипт, бинарник) — копия тоже исполнима;
  - если исходный файл был обычным текстовым (`0644`) — копия тоже обычный текстовый файл, без `x` .

5. **Копирование содержимого (32-байтовый буфер).**

Объявляется фиксированный буфер:

```
char buffer[32];
```

Далее основной цикл:

- Чтение:

```
ssize_t n_read = read(src_fd, buffer, sizeof(buffer));
```

  - Если `n_read < 0` — ошибка чтения;
  - Если `n_read == 0` — достигнут конец файла, выходим из цикла;
  - Иначе прочитано от 1 до 32 байт.

- Запись:

```
ssize_t total_written = 0;
while (total_written < n_read) {
    ssize_t n_written = write(dst_fd,
                              buffer + total_written,
                              n_read - total_written);
```

```
    ...
}
```

- Учитывается возможность частичной записи `write` (когда возвращает меньше, чем запрошенное количество байт).
- Цикл продолжается, пока не будет записано ровно `n_read` байт.

Таким образом, один и тот же 32-байтовый буфер **циклически используется**: читаем → записываем → снова читаем → снова записываем и т.д., пока весь файл не будет скопирован.

6. **Закрытие файлов.**

   В конце работы (или при возникновении ошибок) оба файловых дескриптора закрываются через `close`.

Обработка ошибок

В программе предусмотрена базовая обработка ошибок:

- ошибки открытия файлов (`open`) → `perror("open source")`, `perror("open dest")`;
- ошибки `fstat`, `read`, `write` → соответствующий `perror(...)` и завершение программы с ненулевым кодом возврата;
- проверка возвращаемых значений `read` и `write`:
  - `n < 0` — ошибка;
  - `n == 0` при чтении — конец файла;
  - `n == 0` при записи — рассматривается как аномальная ситуация, о которой выводится сообщение.

Соответствие условиям и ожидаемая оценка

Реализован следующий функционал:

1. **Системные вызовы вместо стандартных средств ввода-вывода.**
   Для работы с файлами используются только `open`, `read`, `write`, `close`, `fstat`.
2. **Работа с текстовыми и бинарными файлами.**
   Программа копирует произвольные байты, не меняя содержимого.
3. **Буфер 32 байта с циклическим использованием.**
   Вместо большого буфера, превышающего размер файла, реализован улучшенный

вариант: копирование по 32 байта с повторным использованием буфера (что соответствует опциональному пункту на +1 балл).

4. **Копирование прав доступа (включая признак исполнимости).**
   - Используется `fstat` для получения `st_mode` исходного файла.
   - При создании целевого файла права доступа выставляются такими же, как у исходного (`source_mode`).
   - Исполняемые файлы и скрипты остаются исполнимыми; обычные текстовые файлы **не становятся** исполняемыми.