Отчет
Пирогов Даниил Игоревич
БПИ-245

Написанная программа выполняет копирование файлов с использованием системных вызовов Linux. Реализованы оба прописанных в ТЗ режима работы: с большим буфером , т.е. размером больше файла и с малым буфером 32 байта с циклическим чтением и записью. Для исполняемых файлов автоматически сохраняются права доступа.

Общая функциональность выполнена на оценку 8

Исходный код:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 32

int copy_big(const char* from, const char* to, int keep_exe);
int copy_small(const char* from, const char* to, int keep_exe);
int is_exe(const char* path);

int main(int argc, char* argv[]) {
  int small_buf = 0;
  int keep_exe = 0;
  int opt;

  while ((opt = getopt(argc, argv, "sxh")) != -1) {
     switch (opt) {
        case 's':
           small_buf = 1;
           break;
        case 'x':
           keep_exe = 1;
           break;
        case 'h':
           printf("Использование: %s <исходный_файл> <целевой_файл> [опции]\n",
argv[0]);
           printf("-s    использовать буфер 32 байта\n");
           printf("-x    сохранить права на выполнение\n");
           printf("-h    справка\n");
```

```
            return 0;
        default:
            fprintf(stderr, "Неизвестная опция: %c\n", opt);
            return 1;
      }
    }

    if (argc - optind < 2) {
      fprintf(stderr, "Ошибка: укажите исходный и целевой файлы\n");
      return 1;
    }

    const char* from = argv[optind];
    const char* to = argv[optind + 1];

    if (access(from, F_OK) != 0) {
      printf("Файл %s не существует\n", from);
      return 1;
    }

    if (!keep_exe && is_exe(from)) {
      printf("Файл является исполняемым, сохраняем права доступа\n");
      keep_exe = 1;
    }

    int ret;
    if (small_buf) {
      ret = copy_small(from, to, keep_exe);
    } else {
      ret = copy_big(from, to, keep_exe);
    }

    if (ret == 0) {
      printf("Копирование завершено\n");
    } else {
      printf("Ошибка копирования\n");
      return 1;
    }

    return 0;
}

int copy_big(const char* from, const char* to, int keep_exe) {
    struct stat st;
```

```
int fd_from, fd_to;
ssize_t size, rd, wr;
char* buf;

if (stat(from, &st) < 0) {
    printf("Не удалось получить информацию о файле\n");
    return -1;
}

size = st.st_size;

if ((fd_from = open(from, O_RDONLY)) < 0) {
    printf("Не удалось открыть файл для чтения\n");
    return -1;
}

buf = malloc(size + 1);
if (!buf) {
    printf("Не удалось выделить память\n");
    close(fd_from);
    return -1;
}

rd = read(fd_from, buf, size);
if (rd == -1) {
    printf("Ошибка чтения файла\n");
    free(buf);
    close(fd_from);
    return -1;
}

mode_t mode;
if (keep_exe) {
    mode = st.st_mode;
} else {
    mode = 0666;
}

if ((fd_to = open(to, O_WRONLY | O_CREAT | O_TRUNC, mode)) < 0) {
    printf("Не удалось открыть файл для записи\n");
    free(buf);
    close(fd_from);
    return -1;
}
```

```
    wr = write(fd_to, buf, rd);
    if (wr != rd) {
        printf("Не удалось записать все данные\n");
        free(buf);
        close(fd_from);
        close(fd_to);
        return -1;
    }

    free(buf);
    if (close(fd_from) < 0) {
        printf("Ошибка закрытия исходного файла\n");
    }
    if (close(fd_to) < 0) {
        printf("Ошибка закрытия целевого файла\n");
    }

    return 0;
}

int copy_small(const char* from, const char* to, int keep_exe) {
    struct stat st;
    int fd_from, fd_to;
    ssize_t rd, wr;
    char buf[BUF_SIZE];

    if (stat(from, &st) < 0) {
        printf("Не удалось получить информацию о файле\n");
        return -1;
    }

    if ((fd_from = open(from, O_RDONLY)) < 0) {
        printf("Не удалось открыть файл для чтения\n");
        return -1;
    }

    mode_t mode;
    if (keep_exe) {
        mode = st.st_mode;
    } else {
        mode = 0666;
    }
```

```
    if ((fd_to = open(to, O_WRONLY | O_CREAT | O_TRUNC, mode)) < 0) {
        printf("Не удалось открыть файл для записи\n");
        close(fd_from);
        return -1;
    }

    do {
        rd = read(fd_from, buf, BUF_SIZE);
        if (rd == -1) {
            printf("Ошибка чтения файла\n");
            close(fd_from);
            close(fd_to);
            return -1;
        }

        if (rd > 0) {
            wr = write(fd_to, buf, rd);
            if (wr != rd) {
                printf("Не удалось записать все данные\n");
                close(fd_from);
                close(fd_to);
                return -1;
            }
        }
    } while (rd == BUF_SIZE);

    if (close(fd_from) < 0) {
        printf("Ошибка закрытия исходного файла\n");
    }
    if (close(fd_to) < 0) {
        printf("Ошибка закрытия целевого файла\n");
    }

    return 0;
}

int is_exe(const char* path) {
    return access(path, X_OK) == 0;
}
```