Отчет
Доп_дз_11
Пирогов Даниил Игоревич
БПИ-245

Сначала создаётся обычный файл a. Потом строится цепочка символьных ссылок: l0001 ссылается на a, l0002 ссылается на l0001, l0003 ссылается на l0002 и тд. После создания каждой ссылки программа пытается открыть её через open(). Как только он возвращает ошибку, то предел достигнут. Программа выводит max_ok - максимальную глубину, на которой файл ещё открывался, fail_at - глубину, на которой открытие впервые не получилось, а также errno и текст ошибки. Все файлы создаются в отдельной временной папке, которая делается через mkdir с уникальным именем на основе pid, чтобы не засорять рабочую директорию и чтобы потом было проще всё удалить. Все файловые дескрипторы закрываются, результаты системных вызовов проверяются, а в конце файлы и папка удаляются, если не указан режим сохранения.
Запуск программы

./dop_maxDepth

Чтобы увеличить число попыток, если лимит не достигнут:

./dop_maxDepth -n 5000

Оставить временную директорию с созданными файлами для проверки:

./dop_maxDepth -n 200 -k

```
daniilpirogov@MacBook-Air-Daniil sem11 % gcc -std=c11 -Wall -Wextra -O2 dop_maxDepth.c -o dop_maxDepth
./dop_maxDepth -n 5000

max_ok=32
fail_at=33
errno=62 (Too many levels of symbolic links)
daniilpirogov@MacBook-Air-Daniil sem11 %
```

Исходный код:

```
#define _POSIX_C_SOURCE 200809L

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

```
static void die(const char *m) {
    perror(m);
    exit(1);
}

static void mk_tmpdir(char *out, size_t out_sz) {
    pid_t pid = getpid();

    const char *bases[] = { "/tmp", "." };

    for (int b = 0; b < 2; b++) {
        for (int k = 0; k < 10000; k++) {
            int w = snprintf(out, out_sz, "%s/symlink-depth-%ld-%d",
                             bases[b], (long)pid, k);
            if (w < 0 || (size_t)w >= out_sz) die("snprintf");

            if (mkdir(out, 0700) == 0) return;
            if (errno == EEXIST) continue;
            break;
        }
    }

    die("mkdir tmpdir");
}

int main(int argc, char **argv) {
    int n = 1000;
    int keep = 0;

    for (int i = 1; i < argc; i++) {
        if (!strcmp(argv[i], "-n") && i + 1 < argc) {
            n = atoi(argv[++i]);
            if (n <= 0) {
                fprintf(stderr, "Ошибка некорректное значение -n\n");
                return 2;
            }
        } else if (!strcmp(argv[i], "-k")) {
            keep = 1;
        } else {
            fprintf(stderr, "Использование: %s [-n N] [-k]\n", argv[0]);
            return 2;
        }
    }

    char here[PATH_MAX];
    if (!getcwd(here, sizeof(here))) die("getcwd");

    char dir[PATH_MAX];
    mk_tmpdir(dir, sizeof(dir));

    if (chdir(dir) != 0) die("chdir");

    int fd = open("a", O_CREAT | O_TRUNC | O_WRONLY, 0600);
    if (fd < 0) die("open a");
    if (write(fd, "x", 1) != 1) die("write a");
    if (close(fd) != 0) die("close a");

    char **name = calloc((size_t)n + 1, sizeof(char *));
    if (!name) die("calloc");
```

```
    const char *prev = "a";
    int ok = 0, bad = 0, bad_e = 0;

    for (int i = 1; i <= n; i++) {
        char buf[64];
        snprintf(buf, sizeof(buf), "l%04d", i);

        name[i] = strdup(buf);
        if (!name[i]) die("strdup");

        if (symlink(prev, name[i]) != 0) die("symlink");

        int t = open(name[i], O_RDONLY);
        if (t >= 0) {
            close(t);
            ok = i;
            prev = name[i];
        } else {
            bad = i;
            bad_e = errno;
            break;
        }
    }

    if (bad == 0) {
        printf("max_ok=%d (Лимит не достигнут, n=%d)\n", ok, n);
    } else {
        printf("max_ok=%d\n", ok);
        printf("fail_at=%d\n", bad);
        printf("errno=%d (%s)\n", bad_e, strerror(bad_e));
    }

    if (!keep) {
        for (int i = n; i >= 1; i--) {
            if (name[i]) unlink(name[i]);
        }
        unlink("a");

        if (chdir(here) != 0) perror("chdir back");
        if (rmdir(dir) != 0) perror("rmdir");
    } else {
        if (chdir(here) != 0) perror("chdir back");
        printf("dir=%s\n", dir);
    }

    for (int i = 1; i <= n; i++) free(name[i]);
    free(name);

    return 0;
}
```