**ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ**

**Файл main.c**

1. #include <stdio.h>

2. #include <stdlib.h>

3. #include <string.h>

4. #include "sfs.h"

5.

6. int main() {

7. const char \*diskname = "virtual\_disk.img";

8. char command[256];

9. char data[BLOCK\_SIZE];

10.

11. sfs\_mkfs(diskname);

12. sfs\_mount(diskname);

13.

14. while (1) {

15. printf("\n");

16. sfs\_pwdm();

17. fgets(command, sizeof(command), stdin);

18. command[strcspn(command, "\n")] = 0; // Убираем символ новой строки

19.

20. char \*cmd = strtok(command, " "); // Получаем команду (например, c)

21. char \*arg = strtok(NULL, " "); // Получаем аргумент (например, kaka21)

22.

23. if (cmd == NULL) {

24. printf("Неверный формат команды.(Для справки - help)\n");

25. continue;

26. }

27.

28. if (\*cmd != 'l' && \*cmd != 'e' && strcmp(cmd, "pwd") && strcmp(cmd, "help") && arg == NULL) {

29. printf("Неверный формат команды.(Для справки - help)\n");

30. continue;

31. }

32.

33. if (strcmp(cmd, "c") == 0) {

34. sfs\_create(arg);

35. } else if (strcmp(cmd, "d") == 0) {

36. sfs\_delete(arg);

37. } else if (strcmp(cmd, "w") == 0) {

38. printf("Введите данные для записи в файл '%s': ", arg);

39. fgets(data, sizeof(data), stdin);

40. sfs\_write(arg, data);

41. } else if (strcmp(cmd, "r") == 0) {

42. sfs\_read(arg);

43. } else if (strcmp(cmd, "e") == 0) {

44. sfs\_umount();

45. break;

46. } else if (strcmp(cmd, "mkdir") == 0) {

47. sfs\_create\_dir(arg);

48. } else if (strcmp(cmd, "mv") == 0) {

49. char \*dirname = strtok(NULL, " ");

50. sfs\_move\_to\_dir(arg, dirname);

51. } else if (strcmp(cmd, "ls") == 0) {

52. sfs\_ls\_dir(arg);

53. } else if (strcmp(cmd, "rmdir") == 0) {

54. sfs\_delete\_dir(arg);

55. } else if (strcmp(cmd, "cd") == 0) {

56. if (arg != NULL) {

57. sfs\_cd(arg);

58. } else {

59. printf("Необходимо указать директорию для перехода.\n");

60. }

61. } else if (strcmp(cmd, "pwd") == 0) {

62. sfs\_pwd();

63. } else if (strcmp(cmd, "rm") == 0) {

64. sfs\_delete\_dir\_recursive(arg);

65. } else if (strcmp(cmd, "help") == 0) {

66. help ();

67. } else {

68. printf("Неизвестная команда.(Для справки - help)\n");

69. }

70. }

71.

72. return 0;

73. }

**Файл sfs.c**

1 #include "sfs.h"

2 #include <stdio.h>

3 #include <string.h>

4

5 // Глобальные переменные

6 FILE \*disk = NULL;

7 Superblock superblock;

8 Inode inode\_table[MAX\_FILES];

9 DirectoryEntry directory[MAX\_FILES];

10 int current\_directory\_inode = 0;

11 char current\_directory[MAX\_FILENAME\_LENGTH] = "";

12

13 // Основные функции файловой системы

14

15 void sfs\_mkfs(const char \*diskname) {

16 FILE \*test = fopen(diskname, "rb");

17 if (test) {

18 fclose(test);

19 printf("Файл '%s' уже существует. Используйте mount для доступа.\n", diskname);

20 return;

21 }

22

23 disk = fopen(diskname, "wb+");

24 if (!disk) {

25 printf("Не удалось создать файл диска.\n");

26 return;

27 }

28

29 // Initialize superblock

30 superblock.total\_blocks = MAX\_BLOCKS;

31 superblock.free\_blocks = MAX\_BLOCKS;

32 superblock.total\_inodes = MAX\_FILES;

33 superblock.free\_inodes = MAX\_FILES - 1;

34 memset(superblock.block\_bitmap, 0x00, sizeof(superblock.block\_bitmap));

35

36 // Initialize inodes and directory

37 for (int i = 0; i < MAX\_FILES; i++) {

38 inode\_table[i].is\_used = 0;

39 inode\_table[i].is\_directory = 0;

40 inode\_table[i].directory\_inode\_index = -1;

41 directory[i].inode\_index = -1;

42 memset(directory[i].filename, 0, MAX\_FILENAME\_LENGTH);

43 }

44

45 // Create root directory

46 inode\_table[0].is\_used = 1;

47 inode\_table[0].is\_directory = 1;

48 inode\_table[0].directory\_inode\_index = -1;

49 strncpy(inode\_table[0].filename, "/", MAX\_FILENAME\_LENGTH);

50 directory[0].inode\_index = 0;

51 strncpy(directory[0].filename, "/", MAX\_FILENAME\_LENGTH);

52

53 create\_home\_directory();

54

55 // Write to disk

56 fseek(disk, 0, SEEK\_SET);

57 fwrite(&superblock, sizeof(Superblock), 1, disk);

58 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

59 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

60 fflush(disk);

61

62 printf("Файловая система отформатирована. Корневая директория создана.\n");

63 }

64

65 void sfs\_mount(const char \*diskname) {

66 disk = fopen(diskname, "r+b");

67 if (!disk) {

68 printf("Файл диска не существует. Создать новую файловую систему? (y/n): ");

69 int answer = getchar();

70 while (getchar() != '\n'); // Clear input buffer

71

72 if (answer == 'y' || answer == 'Y') {

73 sfs\_mkfs(diskname);

74 disk = fopen(diskname, "r+b");

75 if (!disk) {

76 printf("Ошибка при создании файловой системы.\n");

77 return;

78 }

79 } else {

80 return;

81 }

82 }

83

84 // Check filesystem validity

85 if (!is\_valid\_filesystem(disk)) {

86 printf("Файл не содержит валидной ФС. Инициализировать? (y/n): ");

87 int answer = getchar();

88 while (getchar() != '\n');

89

90 if (answer == 'y' || answer == 'Y') {

91 fclose(disk);

92 sfs\_mkfs(diskname);

93 disk = fopen(diskname, "r+b");

94 if (!disk) {

95 printf("Ошибка при создании файловой системы.\n");

96 return;

97 }

98 } else {

99 fclose(disk);

100 disk = NULL;

101 return;

102 }

103 }

104

105 // Read structures

106 fseek(disk, 0, SEEK\_SET);

107 if (fread(&superblock, sizeof(Superblock), 1, disk) != 1) {

108 printf("Ошибка чтения суперблока.\n");

109 fclose(disk);

110 disk = NULL;

111 return;

112 }

113

114 fseek(disk, sizeof(Superblock), SEEK\_SET);

115 if (fread(inode\_table, sizeof(Inode), MAX\_FILES, disk) != MAX\_FILES) {

116 printf("Ошибка чтения таблицы inode.\n");

117 fclose(disk);

118 disk = NULL;

119 return;

120 }

121

122 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

123 if (fread(directory, sizeof(DirectoryEntry), MAX\_FILES, disk) != MAX\_FILES) {

124 printf("Ошибка чтения директории.\n");

125 fclose(disk);

126 disk = NULL;

127 return;

128 }

129

130 // Set current directory

131 current\_directory\_inode = 0;

132 strcpy(current\_directory, "/");

133

134 // Try to find /home

135 for (int i = 0; i < MAX\_FILES; i++) {

136 if (directory[i].inode\_index != -1 &&

137 strcmp(directory[i].filename, "home") == 0 &&

138 inode\_table[directory[i].inode\_index].is\_directory) {

139 current\_directory\_inode = directory[i].inode\_index;

140 strcpy(current\_directory, "/home");

141 break;

142 }

143 }

144

145 printf("Файловая система смонтирована. Текущая директория: %s\n", current\_directory);

146 }

147

148 void sfs\_umount() {

149 if (disk) {

150 // Save all changes

151 fseek(disk, 0, SEEK\_SET);

152 fwrite(&superblock, sizeof(Superblock), 1, disk);

153

154 fseek(disk, sizeof(Superblock), SEEK\_SET);

155 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

156

157 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

158 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

159

160 fflush(disk);

161 fclose(disk);

162 disk = NULL;

163

164 printf("Файловая система размонтирована. Все данные сохранены.\n");

165 }

166 }

167

168 // Вспомогательные функции

169 int find\_free\_inode() {

170 for (int i = 0; i < MAX\_FILES; i++) {

171 if (inode\_table[i].is\_used == 0) {

172 return i;

173 }

174 }

175 return -1;

176 }

177

178 int find\_free\_block() {

179 for (int i = 0; i < MAX\_BLOCKS; i++) {

180 if ((superblock.block\_bitmap[i / 8] & (1 << (i % 8))) == 0) {

181 return i;

182 }

183 }

184 return -1;

185 }

186

187 void allocate\_block(int block\_index) {

188 superblock.block\_bitmap[block\_index / 8] &= ~(1 << (block\_index % 8));

189 superblock.free\_blocks--;

190 }

191

192 void free\_block(int block\_index) {

193 superblock.block\_bitmap[block\_index / 8] |= (1 << (block\_index % 8));

194 superblock.free\_blocks++;

195 }

196

197 void print\_current\_directory() {

198 if (current\_directory\_inode == 0) {

199 printf("/\n");

200 return;

201 }

202 char path[MAX\_FILENAME\_LENGTH];

203 build\_path\_from\_inode(current\_directory\_inode, path, MAX\_FILENAME\_LENGTH);

204 printf("%s\n", path);

205 }

206

207 void build\_path\_from\_inode(int inode, char \*path, size\_t path\_size) {

208 if (inode == 0) {

209 strncpy(path, "/", path\_size);

210 return;

211 }

212 char parent\_path[MAX\_FILENAME\_LENGTH] = "";

213 int parent\_inode = inode\_table[inode].directory\_inode\_index;

214 if (parent\_inode != 0) {

215 build\_path\_from\_inode(parent\_inode, parent\_path, sizeof(parent\_path));

216 }

217 for (int i = 0; i < MAX\_FILES; i++) {

218 if (directory[i].inode\_index == inode) {

219 if (strcmp(parent\_path, "/") == 0) {

220 snprintf(path, path\_size, "/%s", directory[i].filename);

221 } else {

222 snprintf(path, path\_size, "%s/%s", parent\_path, directory[i].filename);

223 }

224 break;

225 }

226 }

227 }

228

229 void sfs\_pwd() {

230 printf("Текущая директория: %s\n", current\_directory);

231 }

232

233 void sfs\_pwdm() {

234 printf("%s: ", current\_directory);

235 }

236

237 int is\_valid\_filesystem(FILE \*f) {

238 if (!f) return 0;

239

240 // 1. Проверяем размер файла

241 fseek(f, 0, SEEK\_END);

242 long size = ftell(f);

243 fseek(f, 0, SEEK\_SET);

244

245 if (size < sizeof(Superblock) + sizeof(Inode)\*MAX\_FILES) {

246 return 0;

247 }

248

249 // 2. Проверяем сигнатуру суперблока

250 Superblock sb;

251 if (fread(&sb, sizeof(Superblock), 1, f) != 1) {

252 return 0;

253 }

254

255 // Простые проверки (можно добавить magic number)

256 if (sb.total\_blocks != MAX\_BLOCKS || sb.total\_inodes != MAX\_FILES) {

257 return 0;

258 }

259

260 // 3. Проверяем корневую директорию

261 Inode root\_inode;

262 fseek(f, sizeof(Superblock), SEEK\_SET);

263 if (fread(&root\_inode, sizeof(Inode), 1, f) != 1) {

264 return 0;

265 }

266

267 if (!root\_inode.is\_used || !root\_inode.is\_directory) {

268 return 0;

269 }

270

271 return 1;

272 }

273

274 long get\_block\_offset(int block\_index) {

275 return sizeof(Superblock)

276 + sizeof(Inode) \* MAX\_FILES

277 + sizeof(DirectoryEntry) \* MAX\_FILES

278 + block\_index \* BLOCK\_SIZE;

279 }

280

281 void help() {

282 printf("\n\n\nc <filename> - создание файла с именем filename\n");

283 printf("d <filename> - удаление файла с именем filename\n");

284 printf("w <filename> - открытие файла с именем filename для записи\n");

285 printf("r <filename> - чтение файла с именем filename\n");

286 printf("mkdir <dirname> - создание директории с именем dirname\n");

287 printf("rmdir <dirname> - удаление директории с именем dirname\n");

288 printf("rm <dirname> - рекурсивное удаление директории с именем dirname\n");

289 printf("cd <dirname> - переход в директорию dirname (переход в предыдущую - ..)\n");

290 printf("ls [dirname] - просмотр текущей директории(\* - опционально) или директории с именем dirname\n");

291 printf("mv <filename> <dirname> - перемещение файла filename в директорию dirname\n");

292 printf("pwd - получение пути к текущей директории\n");

293 printf("е - выход из файловой системы\n\n");

294 printf("Для <filename> и <dirname> возможно указание как полного, так и относительного пути в формате:\n dirname\n ./dirname\n ../dirname\n ./dirname1/dirname2\n /home/.../dirname\n\n\n");

295 }

**Файл sfs.h**

1 #ifndef SFS\_H

2 #define SFS\_H

3

4 #include <stdio.h>

5 #include <string.h>

6

7 // Константы

8 #define MAX\_FILENAME\_LENGTH 256

9 #define MAX\_BLOCKS 1024

10 #define MAX\_FILES 128

11 #define BLOCK\_SIZE 4096

12

13 // Структуры

14 typedef struct {

15 int total\_blocks;

16 int free\_blocks;

17 int total\_inodes;

18 int free\_inodes;

19 unsigned char block\_bitmap[MAX\_BLOCKS/8];

20 } Superblock;

21

22 typedef struct {

23 int is\_used;

24 int is\_directory;

25 int directory\_inode\_index;

26 char filename[MAX\_FILENAME\_LENGTH];

27 int size;

28 int block\_count;

29 int blocks[16]; // Максимум 16 блоков на файл

30 } Inode;

31

32 typedef struct {

33 int inode\_index;

34 char filename[MAX\_FILENAME\_LENGTH];

35 } DirectoryEntry;

36

37 // Глобальные переменные (объявлены как extern)

38 extern FILE \*disk;

39 extern Superblock superblock;

40 extern Inode inode\_table[MAX\_FILES];

41 extern DirectoryEntry directory[MAX\_FILES];

42 extern int current\_directory\_inode;

43 extern char current\_directory[MAX\_FILENAME\_LENGTH];

44

45 // Прототипы функций

46 void sfs\_mkfs(const char \*diskname);

47 void sfs\_mount(const char \*diskname);

48 void sfs\_umount();

49 void sfs\_pwdm();

50 void sfs\_pwd();

51 void help();

52 int is\_valid\_filesystem(FILE \*f);

53 void create\_home\_directory();

54 long get\_block\_offset(int block\_index);

55

56 // Функции для файлов

57 void sfs\_create(const char \*filename);

58 void sfs\_write(const char \*filename, const char \*data);

59 void sfs\_read(const char \*filename);

60 void sfs\_delete(const char \*filename);

61

62 // Функции для директорий

63 void sfs\_cd(const char \*dirname);

64 void sfs\_create\_dir(const char \*dirname);

65 void sfs\_ls\_dir(const char \*dirname);

66 void sfs\_delete\_dir(const char \*dirname);

67 void sfs\_delete\_dir\_recursive(const char \*dirname);

68

69 // Вспомогательные функции

70 int find\_free\_inode();

71 int find\_free\_block();

72 void allocate\_block(int block\_index);

73 void free\_block(int block\_index);

74 void print\_current\_directory();

75 void build\_path\_from\_inode(int inode, char \*path, size\_t path\_size);

76 int resolve\_path\_to\_inode(const char \*path, int \*parent\_inode\_index, char \*basename);

77 void sfs\_move\_to\_dir(const char \*file\_input, const char \*dir\_input);

78

79 #endif

**Файл sfs\_file.c**

1 #include "sfs.h"

2

3 // Функции для работы с файлами

4 void sfs\_create(const char \*filename) {

5 // Проверка длины имени

6 if (strlen(filename) >= MAX\_FILENAME\_LENGTH) {

7 printf("Слишком длинное имя файла.\n");

8 return;

9 }

10

11 // Проверка существования файла

12 for (int i = 0; i < MAX\_FILES; i++) {

13 if (directory[i].inode\_index != -1 &&

14 inode\_table[directory[i].inode\_index].directory\_inode\_index == current\_directory\_inode &&

15 strcmp(directory[i].filename, filename) == 0) {

16 printf("Файл '%s' уже существует.\n", filename);

17 return;

18 }

19 }

20

21 // Создание файла

22 int inode\_index = find\_free\_inode();

23 if (inode\_index == -1) {

24 printf("Нет свободных inode.\n");

25 return;

26 }

27

28 int block\_index = find\_free\_block();

29 if (block\_index == -1) {

30 printf("Нет свободных блоков.\n");

31 return;

32 }

33

34 int dir\_entry\_index = -1;

35 for (int i = 0; i < MAX\_FILES; i++) {

36 if (directory[i].inode\_index == -1) {

37 dir\_entry\_index = i;

38 break;

39 }

40 }

41

42 if (dir\_entry\_index == -1) {

43 printf("Нет места в директории.\n");

44 return;

45 }

46

47 // Заполнение структур

48 inode\_table[inode\_index].is\_used = 1;

49 inode\_table[inode\_index].is\_directory = 0;

50 inode\_table[inode\_index].directory\_inode\_index = current\_directory\_inode;

51 strncpy(inode\_table[inode\_index].filename, filename, MAX\_FILENAME\_LENGTH);

52 inode\_table[inode\_index].size = 0;

53 inode\_table[inode\_index].block\_count = 1;

54 inode\_table[inode\_index].blocks[0] = block\_index;

55

56 directory[dir\_entry\_index].inode\_index = inode\_index;

57 strncpy(directory[dir\_entry\_index].filename, filename, MAX\_FILENAME\_LENGTH);

58

59 superblock.free\_inodes--;

60 allocate\_block(block\_index);

61

62 // Сохранение на диск

63 fseek(disk, 0, SEEK\_SET);

64 fwrite(&superblock, sizeof(Superblock), 1, disk);

65 fseek(disk, sizeof(Superblock), SEEK\_SET);

66 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

67 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

68 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

69

70 print\_current\_directory();

71

72 printf("Файл '%s' создан.\n", filename);

73 }

74

75 void sfs\_write(const char \*filename, const char \*data) {

76 for (int i = 0; i < MAX\_FILES; i++) {

77 if (strncmp(directory[i].filename, filename, MAX\_FILENAME\_LENGTH) == 0) {

78 Inode \*inode = &inode\_table[directory[i].inode\_index];

79

80 int data\_written = 0;

81 int size = strlen(data);

82

83 for (int j = 0; j < inode->block\_count; j++) {

84 int block\_size = (size - data\_written > BLOCK\_SIZE) ? BLOCK\_SIZE : size - data\_written;

85

86 fseek(disk, get\_block\_offset(inode->blocks[j]), SEEK\_SET);

87 fwrite(data + data\_written, 1, block\_size, disk);

88

89 data\_written += block\_size;

90 if (data\_written == size) break;

91 }

92 fflush(disk);

93

94 inode->size = size;

95

96 fseek(disk, sizeof(Superblock), SEEK\_SET);

97 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

98

99 fflush(disk);

100

101 printf("Данные в файл '%s' успешно записаны.\n", filename);

102 return;

103 }

104 }

105 printf("Файл '%s' не найден.\n", filename);

106 }

107

108 void sfs\_read(const char \*filename) {

109 for (int i = 0; i < MAX\_FILES; i++) {

110 if (strncmp(directory[i].filename, filename, MAX\_FILENAME\_LENGTH) == 0) {

111 Inode \*inode = &inode\_table[directory[i].inode\_index];

112

113 int data\_read = 0;

114 int size = inode->size;

115 char buffer[size + 1];

116

117 for (int j = 0; j < inode->block\_count; j++) {

118 int block\_size = (size - data\_read > BLOCK\_SIZE) ? BLOCK\_SIZE : size - data\_read;

119

120 fseek(disk, get\_block\_offset(inode->blocks[j]), SEEK\_SET);

121 fread(buffer + data\_read, 1, block\_size, disk);

122

123 data\_read += block\_size;

124 if (data\_read == size) break;

125 }

126

127 buffer[size] = '\0';

128

129 printf("Данные из файла '%s':\n%s\n", filename, buffer);

130 return;

131 }

132 }

133 printf("Файл '%s' не найден.\n", filename);

134 }

135

136 void sfs\_delete(const char \*filename) {

137 if (disk == NULL) {

138 fprintf(stderr, "Ошибка: файловая система не смонтирована.\n");

139 return;

140 }

141

142 if (filename == NULL || strlen(filename) == 0) {

143 printf("Ошибка: не указано имя файла.\n");

144 return;

145 }

146

147 // Ищем файл в текущей директории

148 int dir\_entry\_index = -1;

149 int file\_inode\_index = -1;

150

151 for (int i = 0; i < MAX\_FILES; i++) {

152 if (directory[i].inode\_index != -1 &&

153 inode\_table[directory[i].inode\_index].directory\_inode\_index == current\_directory\_inode &&

154 strcmp(directory[i].filename, filename) == 0) {

155

156 // Проверяем, что это не директория

157 if (inode\_table[directory[i].inode\_index].is\_directory) {

158 printf("Ошибка: '%s' является директорией. Используйте ddir для удаления директорий.\n", filename);

159 return;

160 }

161

162 dir\_entry\_index = i;

163 file\_inode\_index = directory[i].inode\_index;

164 break;

165 }

166 }

167

168 if (dir\_entry\_index == -1) {

169 printf("Файл '%s' не найден в текущей директории.\n", filename);

170 return;

171 }

172

173 Inode \*file\_inode = &inode\_table[file\_inode\_index];

174

175 // Освобождаем все блоки файла

176 for (int i = 0; i < file\_inode->block\_count; i++) {

177 int block\_index = file\_inode->blocks[i];

178

179 if (block\_index < 0 || block\_index >= MAX\_BLOCKS) {

180 printf("Предупреждение: некорректный индекс блока %d, пропускаем.\n", block\_index);

181 continue;

182 }

183

184 // Очищаем блок на диске

185 char zero\_block[BLOCK\_SIZE] = {0};

186 fseek(disk, block\_index \* BLOCK\_SIZE, SEEK\_SET);

187 fwrite(zero\_block, BLOCK\_SIZE, 1, disk);

188

189 // Освобождаем блок в битовой карте

190 free\_block(block\_index);

191 }

192

193 // Освобождаем inode

194 memset(file\_inode, 0, sizeof(Inode));

195 superblock.free\_inodes++;

196

197 // Удаляем запись из директории

198 directory[dir\_entry\_index].inode\_index = -1;

199 memset(directory[dir\_entry\_index].filename, 0, MAX\_FILENAME\_LENGTH);

200

201 // Сохраняем изменения на диск

202 fseek(disk, 0, SEEK\_SET);

203 fwrite(&superblock, sizeof(Superblock), 1, disk);

204

205 fseek(disk, sizeof(Superblock), SEEK\_SET);

206 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

207

208 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

209 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

210

211 fflush(disk); // Гарантируем запись на диск

212

213 printf("Файл '%s' успешно удален.\n", filename);

214 }

**Файл sfs\_dir.c**

1 #include "sfs.h"

2

3 void create\_home\_directory() {

4 // Проверка существования /home

5 for (int i = 0; i < MAX\_FILES; i++) {

6 if (directory[i].inode\_index != -1 &&

7 strcmp(directory[i].filename, "home") == 0) {

8 return;

9 }

10 }

11

12 // Создание /home

13 int home\_inode = find\_free\_inode();

14 if (home\_inode == -1) return;

15

16 inode\_table[home\_inode].is\_used = 1;

17 inode\_table[home\_inode].is\_directory = 1;

18 inode\_table[home\_inode].directory\_inode\_index = 0;

19 strncpy(inode\_table[home\_inode].filename, "home", MAX\_FILENAME\_LENGTH);

20

21 for (int i = 1; i < MAX\_FILES; i++) {

22 if (directory[i].inode\_index == -1) {

23 directory[i].inode\_index = home\_inode;

24 strncpy(directory[i].filename, "home", MAX\_FILENAME\_LENGTH);

25 break;

26 }

27 }

28

29 superblock.free\_inodes--;

30 current\_directory\_inode = home\_inode;

31 strncpy(current\_directory, "/home", MAX\_FILENAME\_LENGTH);

32 printf("Директория /home создана.\n");

33 }

34

35 void sfs\_cd(const char \*dirname) {

36 if (dirname == NULL || strlen(dirname) == 0) {

37 printf("Ошибка: путь не указан.\n");

38 return;

39 }

40

41 char basename[MAX\_FILENAME\_LENGTH];

42 int parent\_inode;

43 int target\_inode = resolve\_path\_to\_inode(dirname, &parent\_inode, basename);

44

45 // Проверяем, что директория существует

46 if (target\_inode == -1) {

47 printf("Директория '%s' не найдена.\n", dirname);

48 return;

49 }

50

51 // Проверяем, что это действительно директория

52 if (!inode\_table[target\_inode].is\_directory) {

53 printf("'%s' не является директорией.\n", dirname);

54 return;

55 }

56

57 // Обновляем текущую директорию

58 current\_directory\_inode = target\_inode;

59

60 // Обновляем строковое представление пути

61 if (target\_inode == 0) {

62 strcpy(current\_directory, "/");

63 } else {

64 // Строим полный путь по inode

65 build\_path\_from\_inode(target\_inode, current\_directory, MAX\_FILENAME\_LENGTH);

66 }

67

68 printf("Текущая директория: %s\n", current\_directory);

69 }

70

71 void sfs\_create\_dir(const char \*dirname) {

72 // Проверка длины имени

73 if (strlen(dirname) >= MAX\_FILENAME\_LENGTH) {

74 printf("Слишком длинное имя директории.\n");

75 return;

76 }

77

78 // Находим родительскую директорию (текущую или корневую)

79 int parent\_inode = current\_directory\_inode;

80 if (strcmp(dirname, "home") == 0) {

81 parent\_inode = 0; // Корень для /home

82 }

83

84 // Проверяем, нет ли уже такой директории

85 for (int i = 0; i < MAX\_FILES; i++) {

86 if (directory[i].inode\_index >= 0 &&

87 inode\_table[directory[i].inode\_index].directory\_inode\_index == parent\_inode &&

88 strcmp(directory[i].filename, dirname) == 0) {

89 printf("Директория '%s' уже существует.\n", dirname);

90 return;

91 }

92 }

93

94 // Ищем свободный inode

95 int inode\_index = find\_free\_inode();

96 if (inode\_index == -1) {

97 printf("Нет свободных inode для новой директории.\n");

98 return;

99 }

100

101 // Ищем свободный блок

102 int block\_index = find\_free\_block();

103 if (block\_index == -1) {

104 printf("Нет свободных блоков для хранения данных.\n");

105 return;

106 }

107

108 // Создаем inode для директории

109 Inode \*inode = &inode\_table[inode\_index];

110 inode->is\_used = 1;

111 inode->is\_directory = 1;

112 inode->directory\_inode\_index = parent\_inode; // Устанавливаем родителя

113 inode->size = 0;

114 inode->block\_count = 1;

115 inode->blocks[0] = block\_index;

116

117 // Находим свободную запись в directory

118 int dir\_entry\_index = -1;

119 for (int i = 0; i < MAX\_FILES; i++) {

120 if (directory[i].inode\_index == -1) {

121 dir\_entry\_index = i;

122 break;

123 }

124 }

125

126 if (dir\_entry\_index == -1) {

127 printf("Нет свободных записей в directory.\n");

128 return;

129 }

130

131 // Заполняем запись в directory

132 DirectoryEntry \*entry = &directory[dir\_entry\_index];

133 strncpy(entry->filename, dirname, MAX\_FILENAME\_LENGTH);

134 entry->inode\_index = inode\_index;

135

136 superblock.free\_inodes--;

137 allocate\_block(block\_index);

138

139 // Обновляем данные на диске

140 fseek(disk, 0, SEEK\_SET);

141 fwrite(&superblock, sizeof(Superblock), 1, disk);

142 fseek(disk, sizeof(Superblock), SEEK\_SET);

143 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

144 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

145 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

146 fflush(disk);

147

148 printf("Директория '%s' успешно создана.\n", dirname);

149 }

150

151 void sfs\_ls\_dir(const char \*dirname) {

152 // Определяем директорию для вывода содержимого

153 int target\_inode;

154 char basename[MAX\_FILENAME\_LENGTH];

155 int parent\_inode;

156

157 if (dirname == NULL || strlen(dirname) == 0) {

158 // Если путь не указан - используем текущую директорию

159 target\_inode = current\_directory\_inode;

160 } else {

161 // Разрешаем переданный путь

162 target\_inode = resolve\_path\_to\_inode(dirname, &parent\_inode, basename);

163 }

164

165 // Проверяем, что объект существует

166 if (target\_inode == -1) {

167 printf("'%s' не найдено.\n", dirname ? dirname : "текущий путь");

168 return;

169 }

170

171 // Проверяем, что это директория (если не корень)

172 if (target\_inode != 0 && !inode\_table[target\_inode].is\_directory) {

173 printf("'%s' не является директорией.\n", dirname ? dirname : "текущий путь");

174 return;

175 }

176

177 // Получаем полное имя директории для вывода

178 char dir\_path[MAX\_FILENAME\_LENGTH];

179 if (target\_inode == 0) {

180 strcpy(dir\_path, "/");

181 } else {

182 build\_path\_from\_inode(target\_inode, dir\_path, MAX\_FILENAME\_LENGTH);

183 }

184

185 printf("Содержимое директории '%s':\n", dir\_path);

186

187 // Выводим содержимое директории

188 int empty = 1;

189 for (int i = 0; i < MAX\_FILES; i++) {

190 if (directory[i].inode\_index != -1) {

191 // Для файлов и поддиректорий этой директории

192 if (inode\_table[directory[i].inode\_index].directory\_inode\_index == target\_inode) {

193 printf("- %s", directory[i].filename);

194 if (inode\_table[directory[i].inode\_index].is\_directory) {

195 printf(" (директория)");

196 } else {

197 printf(" (файл, размер: %d)", inode\_table[directory[i].inode\_index].size);

198 }

199 printf("\n");

200 empty = 0;

201 }

202 }

203 }

204

205 if (empty) {

206 printf("Директория пуста.\n");

207 }

208 }

209

210 void sfs\_delete\_dir(const char \*dirname) {

211 if (disk == NULL) {

212 fprintf(stderr, "Ошибка: файловая система не смонтирована.\n");

213 return;

214 }

215

216 if (dirname == NULL || strlen(dirname) == 0) {

217 printf("Ошибка: не указано имя директории.\n");

218 return;

219 }

220

221 // Разрешаем путь к директории

222 int parent\_inode;

223 char basename[MAX\_FILENAME\_LENGTH];

224 int dir\_inode = resolve\_path\_to\_inode(dirname, &parent\_inode, basename);

225

226 // Проверяем, что директория существует

227 if (dir\_inode == -1) {

228 printf("Директория '%s' не найдена.\n", dirname);

229 return;

230 }

231

232 // Нельзя удалить корневую директорию

233 if (dir\_inode == 0) {

234 printf("Ошибка: нельзя удалить корневую директорию.\n");

235 return;

236 }

237

238 // Проверяем, что это действительно директория

239 if (!inode\_table[dir\_inode].is\_directory) {

240 printf("Ошибка: '%s' не является директорией.\n", dirname);

241 return;

242 }

243

244 // Проверяем, не пытаемся ли удалить текущую директорию

245 if (dir\_inode == current\_directory\_inode) {

246 printf("Ошибка: нельзя удалить текущую директорию.\n");

247 return;

248 }

249

250 // Проверяем, пуста ли директория

251 for (int i = 0; i < MAX\_FILES; i++) {

252 if (directory[i].inode\_index != -1 &&

253 inode\_table[directory[i].inode\_index].directory\_inode\_index == dir\_inode) {

254 printf("Директория '%s' не пуста, невозможно удалить.\n", dirname);

255 return;

256 }

257 }

258

259 // Находим запись в directory для удаляемой директории

260 int dir\_entry\_index = -1;

261 for (int i = 0; i < MAX\_FILES; i++) {

262 if (directory[i].inode\_index == dir\_inode) {

263 dir\_entry\_index = i;

264 break;

265 }

266 }

267

268 if (dir\_entry\_index == -1) {

269 printf("Ошибка: не найдена запись в директории.\n");

270 return;

271 }

272

273 // Получаем полный путь для сообщения

274 char dir\_path[MAX\_FILENAME\_LENGTH];

275 build\_path\_from\_inode(dir\_inode, dir\_path, sizeof(dir\_path));

276

277 // Освобождаем inode

278 inode\_table[dir\_inode].is\_used = 0;

279 superblock.free\_inodes++;

280

281 // Удаляем запись из directory

282 directory[dir\_entry\_index].inode\_index = -1;

283 memset(directory[dir\_entry\_index].filename, 0, MAX\_FILENAME\_LENGTH);

284

285 // Сохраняем изменения на диск

286 fseek(disk, 0, SEEK\_SET);

287 fwrite(&superblock, sizeof(Superblock), 1, disk);

288

289 fseek(disk, sizeof(Superblock), SEEK\_SET);

290 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

291

292 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

293 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

294

295 fflush(disk); // Гарантируем запись на диск

296

297 printf("Директория '%s' успешно удалена.\n", dir\_path);

298 }

299

300 int resolve\_path\_to\_inode(const char \*path, int \*parent\_inode\_index, char \*basename) {

301 if (!path || !parent\_inode\_index || !basename) {

302 printf("[ERROR] Неверные аргументы resolve\_path\_to\_inode.\n");

303 return -1;

304 }

305

306 // Создаем копию пути для безопасной работы

307 char temp\_path[MAX\_FILENAME\_LENGTH \* 2]; // Увеличиваем буфер для сложных путей

308 strncpy(temp\_path, path, sizeof(temp\_path));

309 temp\_path[sizeof(temp\_path) - 1] = '\0';

310

311 //printf("[DEBUG] resolve\_path\_to\_inode: path = '%s'\n", temp\_path);

312

313 char \*token;

314 char \*rest = temp\_path;

315 int current\_inode = (path[0] == '/') ? 0 : current\_directory\_inode;

316 \*parent\_inode\_index = -1;

317 char last\_token[MAX\_FILENAME\_LENGTH] = "";

318

319 // Обрабатываем случай корневого пути

320 if (strcmp(temp\_path, "/") == 0) {

321 strncpy(basename, "/", MAX\_FILENAME\_LENGTH);

322 \*parent\_inode\_index = -1;

323 return 0; // Корневой inode

324 }

325

326 // Разбираем путь по токенам

327 while ((token = strtok\_r(rest, "/", &rest))) {

328 // printf("[DEBUG] token = '%s'\n", token);

329

330 // Пропускаем пустые токены и текущую директорию

331 if (strlen(token) == 0 || strcmp(token, ".") == 0) {

332 continue;

333 }

334

335 // Обработка перехода в родительскую директорию

336 if (strcmp(token, "..") == 0) {

337 if (current\_inode != 0) { // Не выходим за пределы корня

338 \*parent\_inode\_index = inode\_table[current\_inode].directory\_inode\_index;

339 current\_inode = \*parent\_inode\_index;

340 }

341 continue;

342 }

343

344 // Поиск токена в текущей директории

345 int found = 0;

346 for (int i = 0; i < MAX\_FILES; i++) {

347 if (directory[i].inode\_index >= 0 &&

348 inode\_table[directory[i].inode\_index].directory\_inode\_index == current\_inode &&

349 strncmp(directory[i].filename, token, MAX\_FILENAME\_LENGTH) == 0) {

350

351 \*parent\_inode\_index = current\_inode;

352 current\_inode = directory[i].inode\_index;

353 found = 1;

354 strncpy(last\_token, token, MAX\_FILENAME\_LENGTH);

355 break;

356 }

357 }

358

359 if (!found) {

360 // printf("[DEBUG] '%s' не найден в inode %d\n", token, current\_inode);

361 strncpy(basename, token, MAX\_FILENAME\_LENGTH);

362 return -1;

363 }

364 }

365

366 // Обработка случая, когда путь заканчивается слешем (например, "dir/")

367 if (strlen(last\_token) == 0 && path[strlen(path)-1] == '/') {

368 strncpy(basename, "", MAX\_FILENAME\_LENGTH);

369 return current\_inode;

370 }

371

372 // Сохраняем результат

373 strncpy(basename, last\_token, MAX\_FILENAME\_LENGTH);

374 //printf("[DEBUG] Успешно: basename='%s', inode=%d, parent\_inode=%d\n",

375 // basename, current\_inode, \*parent\_inode\_index);

376

377 return current\_inode;

378 }

379

380 void sfs\_move\_to\_dir(const char \*file\_input, const char \*dir\_input) {

381 int file\_parent\_inode, dir\_parent\_inode;

382 char file\_name[MAX\_FILENAME\_LENGTH];

383 char dir\_name[MAX\_FILENAME\_LENGTH];

384

385 // Получаем inode файла и родительской директории

386 int file\_inode\_index = resolve\_path\_to\_inode(file\_input, &file\_parent\_inode, file\_name);

387 if (file\_inode\_index == -1) {

388 printf("Файл '%s' не найден.\n", file\_input);

389 return;

390 }

391

392 // Получаем inode целевой директории

393 int dir\_inode\_index = resolve\_path\_to\_inode(dir\_input, &dir\_parent\_inode, dir\_name);

394 if (dir\_inode\_index == -1 || !inode\_table[dir\_inode\_index].is\_directory) {

395 printf("Директория '%s' не найдена или это не директория.\n", dir\_input);

396 return;

397 }

398

399 // Проверяем, нет ли файла с таким именем в целевой директории

400 for (int i = 0; i < MAX\_FILES; i++) {

401 if (directory[i].inode\_index >= 0 &&

402 inode\_table[directory[i].inode\_index].directory\_inode\_index == dir\_inode\_index &&

403 strcmp(directory[i].filename, file\_name) == 0) {

404 printf("Файл с именем '%s' уже существует в директории '%s'.\n", file\_name, dir\_input);

405 return;

406 }

407 }

408

409 // Удаляем запись из старой директории

410 for (int i = 0; i < MAX\_FILES; i++) {

411 if (directory[i].inode\_index == file\_inode\_index &&

412 inode\_table[directory[i].inode\_index].directory\_inode\_index == file\_parent\_inode) {

413 directory[i].inode\_index = -1; // Помечаем как свободную запись

414 break;

415 }

416 }

417

418 // Добавляем запись в новую директорию

419 for (int i = 0; i < MAX\_FILES; i++) {

420 if (directory[i].inode\_index == -1) { // Находим свободную запись

421 directory[i].inode\_index = file\_inode\_index;

422 strncpy(directory[i].filename, file\_name, MAX\_FILENAME\_LENGTH);

423 inode\_table[file\_inode\_index].directory\_inode\_index = dir\_inode\_index;

424 break;

425 }

426 }

427

428 // Сохраняем изменения

429 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

430 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

431

432 printf("Файл '%s' перемещён в '%s'.\n", file\_input, dir\_input);

433 }

434

435 // Рекурсивное удаление директории

436 void sfs\_delete\_dir\_recursive(const char \*dirname) {

437 if (disk == NULL) {

438 fprintf(stderr, "Ошибка: файловая система не смонтирована.\n");

439 return;

440 }

441

442 if (dirname == NULL || strlen(dirname) == 0) {

443 printf("Ошибка: не указано имя директории.\n");

444 return;

445 }

446

447 // Разрешаем путь к директории

448 int parent\_inode;

449 char basename[MAX\_FILENAME\_LENGTH];

450 int dir\_inode = resolve\_path\_to\_inode(dirname, &parent\_inode, basename);

451

452 // Проверяем, что директория существует

453 if (dir\_inode == -1) {

454 printf("Директория '%s' не найдена.\n", dirname);

455 return;

456 }

457

458 // Нельзя удалить корневую директорию

459 if (dir\_inode == 0) {

460 printf("Ошибка: нельзя удалить корневую директорию.\n");

461 return;

462 }

463

464 // Проверяем, что это действительно директория

465 if (!inode\_table[dir\_inode].is\_directory) {

466 printf("Ошибка: '%s' не является директорией.\n", dirname);

467 return;

468 }

469

470 // Проверяем, не пытаемся ли удалить текущую директорию

471 if (dir\_inode == current\_directory\_inode) {

472 printf("Ошибка: нельзя удалить текущую директорию.\n");

473 return;

474 }

475

476 // Получаем полный путь для сообщений

477 char dir\_path[MAX\_FILENAME\_LENGTH];

478 build\_path\_from\_inode(dir\_inode, dir\_path, sizeof(dir\_path));

479

480 // Рекурсивно удаляем все содержимое директории

481 for (int i = 0; i < MAX\_FILES; i++) {

482 if (directory[i].inode\_index != -1 &&

483 inode\_table[directory[i].inode\_index].directory\_inode\_index == dir\_inode) {

484

485 // Для поддиректорий вызываем рекурсивное удаление

486 if (inode\_table[directory[i].inode\_index].is\_directory) {

487 char subdir\_path[MAX\_FILENAME\_LENGTH];

488 snprintf(subdir\_path, MAX\_FILENAME\_LENGTH, "%s/%s", dir\_path, directory[i].filename);

489 sfs\_delete\_dir\_recursive(subdir\_path);

490 }

491 // Для файлов вызываем обычное удаление

492 else {

493 sfs\_delete(directory[i].filename);

494 }

495 }

496 }

497

498 // Находим запись в directory для удаляемой директории

499 int dir\_entry\_index = -1;

500 for (int i = 0; i < MAX\_FILES; i++) {

501 if (directory[i].inode\_index == dir\_inode) {

502 dir\_entry\_index = i;

503 break;

504 }

505 }

506

507 if (dir\_entry\_index == -1) {

508 printf("Ошибка: не найдена запись в директории.\n");

509 return;

510 }

511

512 // Освобождаем блоки директории (если они есть)

513 for (int i = 0; i < inode\_table[dir\_inode].block\_count; i++) {

514 int block\_index = inode\_table[dir\_inode].blocks[i];

515 if (block\_index >= 0 && block\_index < MAX\_BLOCKS) {

516 free\_block(block\_index);

517 }

518 }

519

520 // Освобождаем inode

521 inode\_table[dir\_inode].is\_used = 0;

522 superblock.free\_inodes++;

523

524 // Удаляем запись из directory

525 directory[dir\_entry\_index].inode\_index = -1;

526 memset(directory[dir\_entry\_index].filename, 0, MAX\_FILENAME\_LENGTH);

527

528 // Сохраняем изменения на диск

529 fseek(disk, 0, SEEK\_SET);

530 fwrite(&superblock, sizeof(Superblock), 1, disk);

531

532 fseek(disk, sizeof(Superblock), SEEK\_SET);

533 fwrite(inode\_table, sizeof(Inode), MAX\_FILES, disk);

534

535 fseek(disk, sizeof(Superblock) + sizeof(Inode) \* MAX\_FILES, SEEK\_SET);

536 fwrite(directory, sizeof(DirectoryEntry), MAX\_FILES, disk);

537

538 fflush(disk); // Гарантируем запись на диск

539

540 printf("Директория '%s' и все её содержимое успешно удалены.\n", dir\_path);

541 }