Документация к файлу main.c
Командный интерпретатор

Разработчик: [Ваше имя]

16 ноября 2025 г.

Аннотация

Данный документ описывает структуру и функциональность главного модуля командного интерпретатора, реализованного в файле `main.c`. Модуль отвечает за основной цикл работы shell'а, обработку сигналов и взаимодействие с пользователем.

Содержание

1 Обзор модуля

Файл `main.c` является центральным модулем командного интерпретатора, который реализует основной цикл **REPL** (Read-Eval-Print Loop). Модуль координирует работу всех компонентов системы: чтение ввода, парсинг команд и их выполнение.

2 Структура модуля

2.1 Заголовочные файлы

```
#include "shell.h"
```

Модуль подключает главный заголовочный файл, содержащий объявления всех структур данных и функций, используемых в проекте.

2.2 Глобальные определения

- Макросы определены в `shell.h`
- Структуры данных: `command_t`, `command_sequence_t`

3 Функции модуля

3.1 Функция `check_child`

```
void check_child(int sig) {
    int status;
    pid_t pid;

    while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
        printf("[%d] Finished with status %d\n", pid, WEXITSTATUS(status));
    }
}
```

Назначение: Обработчик сигнала `SIGCHLD` для отслеживания завершения фоновых процессов.

Параметры:

- `sig` - номер сигнала (всегда `SIGCHLD`)

Особенности реализации:

- Использует `waitpid()` с флагом `WNOHANG` для неблокирующей проверки
- Обрабатывает все завершившиеся процессы в цикле
- Выводит информационное сообщение о завершении процесса

3.2 Функция `print_dir`

```
char* print_dir() {
    long size = pathconf(".", _PC_PATH_MAX);
    if (size == -1) {
        size = 4096;
    }

    char *cwd = malloc((size_t)size);
    if (cwd == NULL) {
        perror("malloc");
        exit(2);
    }
```

```
12
13      cwd = getcwd(cwd, (size_t)size);
14      if (cwd == NULL) {
15          perror("getcwd");
16          free(cwd);
17          exit(2);
18      }
19
20      return cwd;
21  }
```

Назначение: Получение текущей рабочей директории для отображения в приглашении.

Возвращаемое значение: Указатель на строку с текущим путем (требует освобождения).

Особенности реализации:

- Динамически определяет максимальную длину пути
- Резервирует значение 4096 байт при ошибке определения
- Корректно обрабатывает ошибки выделения памяти
- Возвращает готовую для использования строку

3.3 Функция `read_line`

```
1   char *read_line(void) {
2       char *line = NULL;
3       size_t linesize = 0;
4       char* dir = print_dir();
5
6       printf("%s> ", dir);
7       fflush(stdout);
8       free(dir);
9
10      ssize_t num_chars_read = getline(&line, &linesize, stdin);
11
12      if (num_chars_read <= 0) {
13          free(line);
14          return NULL;
15      }
16
17      if (line[num_chars_read - 1] == '\n') {
18          line[num_chars_read - 1] = '\0';
19      }
20
21      return line;
22  }
```

Назначение: Чтение командной строки от пользователя.

Возвращаемое значение: Указатель на введенную строку или `NULL` при EOF.

Особенности реализации:

- Использует `getline()` для безопасного чтения строк
- Автоматически выделяет достаточный объем памяти

- Удаляет символ новой строки из конца ввода
- Выводит приглашение с текущей директорией

3.4 Функция `main`

```c
int main(void) {
    char *input;

    signal(SIGCHLD, check_child);

    printf("Shell R v7.2\n");
    printf("Type 'exit' to quit. Or use Ctrl + D\n");

    while ((input = read_line()) != NULL) {
        if (strlen(input) == 0) {
            free(input);
            continue;
        }

        //

        command_t *cmd = parse_input(input);

        if (cmd != NULL) {
            //                          -
                                print_command(cmd);

            //
            execute_command(cmd);

            free_command(cmd);
        }

        free(input);
    }

    printf("\nGoodbye!\n");
    return 0;
}
```

Назначение: Главная функция программы, реализующая основной цикл работы.
Логика работы:

1. Установка обработчика сигналов для фоновых процессов
2. Вывод приветственного сообщения
3. Запуск основного цикла REPL:
    - Чтение команды
    - Пропуск пустых строк
    - Парсинг ввода
    - Выполнение команды
    - Освобождение ресурсов
4. Корректное завершение программы

4 Взаимодействие с другими модулями

4.1 Зависимости

- `shell.h` - основные структуры и объявления
- `parcer.c` - функции парсинга команд
- `executor.c` - функции выполнения команд
- `cmdfrombash.c` - встроенные команды

4.2 Поток данных

1. `read_line()` → Ввод от пользователя
2. `parse_input()` → Разбор на структуру `command_t`
3. `execute_command()` → Выполнение команды
4. `free_command()` → Освобождение памяти

5 Особенности реализации

5.1 Обработка сигналов

- Используется `signal(SIGCHLD, check_child)` для отслеживания дочерних процессов
- Обеспечивает корректную работу фоновых процессов

5.2 Управление памятью

- Все динамически выделенные ресурсы освобождаются
- Используется `getline()` для безопасного чтения строк
- Проверка ошибок выделения памяти

5.3 Обработка ошибок

- Проверка возвращаемых значений системных вызовов
- Корректная обработка конца файла (Ctrl+D)
- Освобождение памяти при ошибках

6 Примеры использования

6.1 Нормальный workflow

```
/home/user> ls -la
[вывод команды ls]
/home/user> exit
Goodbye!
```

6.2 Фоновые процессы

```
/home/user> sleep 10 &
[1234] Started in fonius
/home/user> [1234] Finished with status 0
```

7 Заключение

Модуль `main.c` успешно реализует основной цикл командного интерпретатора, обеспечивая стабильную работу, корректное управление ресурсами и удобное взаимодействие с пользователем. Архитектура модуля позволяет легко расширять функциональность shell'а.