Документация к файлу cmdfrombash.c
Встроенные команды shell'a

Разработчик: [Ваше имя]

16 ноября 2025 г.

Аннотация

Данный документ описывает структуру и функциональность модуля встроенных команд, реализованного в файле `cmdfrombash.c`. Модуль содержит реализации команд, которые выполняются непосредственно в процессе shell'a без создания дочерних процессов, включая управление директориями, переменными окружения и завершение работы.

Содержание

1 Обзор модуля

Файл `cmdfrombash.c` реализует встроенные команды shell'а, которые обеспечивают:

- Управление текущей рабочей директорией
- Завершение работы shell'а
- Управление переменной окружения PATH
- Просмотр и модификация системных настроек
- Базовые операции с окружением выполнения

2 Структура модуля

2.1 Заголовочные файлы

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

Модуль использует только главный заголовочный файл, так как все необходимые системные вызовы уже объявлены в `shell.h`.

3 Функции встроенных команд

3.1 Функция `from_bash_cd`

```
int from_bash_cd(char **args) {
    if (args[1] == NULL) {
        return 1;
    }

    if (args[2] != NULL) {
        perror("cd: too many arguments\n");
        return 1;
    }

    if (chdir(args[1]) != 0) {
        perror("cd");
        return 1;
    }
    return 0;
}
```

Назначение: Смена текущей рабочей директории.
Синтаксис: `cd <директория>`
Параметры:

- `args[1]` - целевая директория

Логика работы:

1. Проверка наличия аргумента (директории)
2. Проверка на избыточное количество аргументов
3. Вызов системной функции `chdir()`
4. Обработка ошибок изменения директории

Возвращаемые значения:

- 0 - успешное изменение директории
- 1 - ошибка (нет аргумента, слишком много аргументов, несуществующая директория)

Особенности:

- Команда изменяет текущую рабочую директорию всего процесса shell'a
- Все последующие команды выполняются в новой директории
- Поддерживает как абсолютные, так и относительные пути

3.2 Функция `from_bash_exit`

```
int from_bash_exit(char **args) {
    int exit_code = 0;

    if (args[1] != NULL) {
        exit_code = atoi(args[1]);
    }

    exit(exit_code);
}
```

Назначение: Завершение работы shell'a.
Синтаксис: `exit [код_завершения]`
Параметры:

- `args[1]` - необязательный код завершения (по умолчанию 0)

Логика работы:

1. Проверка наличия аргумента с кодом завершения
2. Преобразование строки в число с помощью `atoi()`
3. Вызов системной функции `exit()` с соответствующим кодом

Особенности:

- Команда немедленно завершает работу всего shell'a
- Поддерживает пользовательские коды завершения
- По умолчанию возвращает код 0 (успешное завершение)

3.3 Функция `from_path`

```c
int from_path(char **args) {
    char *path = getenv("PATH");
    if (path == NULL) {
        printf("PATH is not set\n");
    } else {
        printf("PATH=%s\n", path);
    }
    return 0;
}
```

Назначение: Вывод текущего значения переменной окружения PATH.
Синтаксис: `path`
Логика работы:

1. Получение значения переменной PATH с помощью `getenv()`
2. Проверка на NULL (переменная не установлена)
3. Вывод значения в формате `PATH=<значение>`

Возвращаемые значения:

- 0 - всегда успешное выполнение

Особенности:

- Не изменяет переменную PATH, только отображает её значение
- Показывает все директории, в которых shell ищет исполняемые файлы

3.4 Функция `set_path`

```c
int set_path(char **args) {
    if (args[1] == NULL) {
        fprintf(stderr, "setpath: missing argument\n");
        fprintf(stderr, "Usage: setpath <new_path>\n");
        return 1;
    }

    if (args[2] != NULL) {
        fprintf(stderr, "setpath: too many arguments\n");
        fprintf(stderr, "Usage: setpath <new_path>\n");
        return 1;
    }

    if (setenv("PATH", args[1], 1) != 0) {
        perror("setpath");
        return 1;
    }

    printf("PATH set to: %s\n", args[1]);
    return 0;
}
```

Назначение: Установка нового значения переменной окружения PATH.
Синтаксис: `setpath <новый_путь>`
Параметры:

- `args[1]` - новое значение переменной PATH

Логика работы:

1. Проверка наличия обязательного аргумента
2. Проверка на избыточное количество аргументов
3. Установка переменной PATH с помощью `setenv()`
4. Вывод подтверждающего сообщения

Возвращаемые значения:

- 0 - успешная установка PATH
- 1 - ошибка (нет аргумента, слишком много аргументов, ошибка setenv)

Особенности:

- Полностью заменяет текущее значение PATH
- Использует `setenv()` с флагом перезаписи (1)
- Выводит понятные сообщения об ошибках с usage information

3.5 Функция `add_to_path`

```
int add_to_path(char **args) {
    if (args[1] == NULL) {
        fprintf(stderr, "addpath: missing directory\n");
        fprintf(stderr, "Usage: addpath <directory>\n");
        return 1;
    }

    char *current_path = getenv("PATH");
    if (current_path == NULL) {
        //          PATH                          ,

        if (setenv("PATH", args[1], 1) != 0) {
            perror("addpath");
            return 1;
        }
    } else {
        //
                PATH
        size_t new_len = strlen(current_path) + strlen(args[1]) + 2;
        char *new_path = malloc(new_len);
        if (new_path == NULL) {
            perror("addpath: malloc");
            return 1;
        }

        snprintf(new_path, new_len, "%s:%s", args[1], current_path);

        if (setenv("PATH", new_path, 1) != 0) {
            perror("addpath");
            free(new_path);
            return 1;
```

```
        }

        free(new_path);
    }

    printf("Added '%s' to PATH\n", args[1]);
    printf("New PATH: %s\n", getenv("PATH"));
    return 0;
}
```

Назначение: Добавление директории в начало переменной PATH.
Синтаксис: `addpath <директория>`
Параметры:

- `args[1]` - директория для добавления

Логика работы:

1. Проверка наличия обязательного аргумента
2. Получение текущего значения PATH
3. Если PATH не установлен - создание нового
4. Если PATH установлен - добавление директории в начало
5. Формирование новой строки PATH с разделителем ":"
6. Установка обновленного значения PATH

Возвращаемые значения:

- 0 - успешное добавление в PATH
- 1 - ошибка (нет аргумента, ошибка malloc, ошибка setenv)

Особенности:

- Добавляет директорию в **начало** PATH (высший приоритет)
- Корректно обрабатывает случай когда PATH не установлен
- Динамически выделяет память для новой строки PATH
- Выводит старое и новое значение PATH для подтверждения

3.6 Функция `reset_path`

```
int reset_path(char **args) {
    const char *default_path = "/home/ruslan/.vscode-server/bin/
    ac4cbdf48759c7d8c3eb91ffe6bb04316e263c57/bin/remote-cli:/usr/local/sbin:/
    usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/
    mnt/c/Program Files (x86)/Common Files/Oracle/Java/javapath:/mnt/c/
    Windows/System32:/mnt/c/Windows:/mnt/c/Windows/System32/wbem:/mnt/c/
    Windows/System32/WindowsPowerShell/v1.0:/mnt/c/Windows/System32/OpenSSH:/
    mnt/c/Program Files (x86)/NVIDIA Corporation/PhysX/Common:/mnt/c/Code
    Write/FreePascal/bin/i386-Win32:/mnt/c/ProgramData/chocolatey/bin:/mnt/c/
    Program Files/Git/cmd:/mnt/c/Users/123/Desktop/MASM/bin:/mnt/c/msys64/
    ucrt64/bin:/mnt/c/Users/123/AppData/Local/Microsoft/WindowsApps:/mnt/c/
    Users/123/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/123/
    AppData/Local/Programs/Python/Python311:/snap/bin";
```

```
    if (setenv("PATH", default_path, 1) != 0) {
        perror("resetpath");
        return 1;
    }

    printf("PATH reset to default: %s\n", default_path);
    return 0;
}
```

Назначение: Сброс переменной PATH к значению по умолчанию.
Синтаксис: `resetpath`
Логика работы:

1. Использование предопределенного значения PATH по умолчанию
2. Установка переменной PATH с помощью `setenv()`
3. Вывод подтверждающего сообщения

Возвращаемые значения:

- 0 - успешный сброс PATH
- 1 - ошибка установки переменной

Особенности:

- Восстанавливает PATH к жестко заданному значению по умолчанию
- Значение по умолчанию включает системные директории и пути WSL
- Полезно для восстановления работоспособности после некорректных изменений PATH

4 Главная функция диспетчеризации

4.1 Функция `execute_bash_cmd`

```
int execute_bash_cmd(char **args) {
    if (args[0] == NULL) {
        return 1;
    }

    if (strcmp(args[0], "cd") == 0) {
        return from_bash_cd(args);
    } else if (strcmp(args[0], "exit") == 0) {
        return from_bash_exit(args);
    } else if (strcmp(args[0], "path") == 0) {
        return from_path(args);
    } else if (strcmp(args[0], "setpath") == 0) {
        return set_path(args);
    } else if (strcmp(args[0], "addpath") == 0) {
        return add_to_path(args);
    } else if (strcmp(args[0], "resetpath") == 0) {
        return reset_path(args);
    }

    return -1; //
}
```

Назначение: Определение и вызов соответствующей встроенной команды.
Параметры:

- `args` - массив строк с командой и аргументами

Логика работы:

1. Проверка на пустую команду
2. Последовательное сравнение `args[0]` с именами встроенных команд
3. Вызов соответствующей функции-обработчика
4. Возврат -1 если команда не является встроенной

Возвращаемые значения:

- 0/1 - результат выполнения встроенной команды
- -1 - команда не является встроенной

Поддерживаемые команды:

- `cd` - смена директории
- `exit` - завершение shell'а
- `path` - просмотр PATH
- `setpath` - установка PATH
- `addpath` - добавление в PATH
- `resetpath` - сброс PATH

5 Примеры использования

5.1 Пример 1: Работа с директориями

```
$ pwd
/home/user
$ cd /var/log
$ pwd
/var/log
$ cd /nonexistent
cd: No such file or directory
```

5.2 Пример 2: Управление PATH

```
$ path
PATH=/usr/local/bin:/usr/bin:/bin
$ setpath /custom/bin:/other/bin
PATH set to: /custom/bin:/other/bin
$ addpath /new/bin
Added '/new/bin' to PATH
New PATH: /new/bin:/custom/bin:/other/bin
$ resetpath
PATH reset to default: [длинная строка с путями по умолчанию]
```

5.3 Пример 3: Завершение работы

```
$ exit
Goodbye!
$ exit 1
# Завершение с кодом 1
```

6 Особенности реализации

6.1 Обработка аргументов

- Все функции проверяют наличие обязательных аргументов
- Проверка на избыточное количество аргументов
- Вывод понятных сообщений об ошибках с usage information

6.2 Управление памятью

- Функция `add_to_path` динамически выделяет память для новой строки PATH
- Корректное освобождение памяти при ошибках
- Использование безопасных функций работы со строками (`snprintf`)

6.3 Обработка ошибок

- Проверка возвращаемых значений системных вызовов
- Использование `perror()` для системных ошибок
- Возврат соответствующих кодов ошибок

6.4 Безопасность

- Проверка границ строк при работе с путями
- Использование `snprintf` вместо `sprintf`
- Валидация входных аргументов

7 Взаимодействие с другими модулями

7.1 Зависимости

- `shell.h` - объявления структур и констант
- `executor.c` - вызов `execute_bash_cmd()` для определения типа команды

7.2 Поток данных

1. `executor.c` → Вызов `execute_bash_cmd()` с разобранными аргументами
2. `execute_bash_cmd()` → Определение и вызов соответствующей встроенной команды
3. Функция-обработчик → Выполнение команды и возврат статуса
4. Возврат статуса выполнения в `executor.c`

8 Расширяемость

Система встроенных команд легко расширяема. Для добавления новой команды необходимо:

1. Реализовать функцию-обработчик с сигнатурой `int function_name(char **args)`
2. Добавить проверку в функцию `execute_bash_cmd()`
3. Объявить функцию в `shell.h`

9 Заключение

Модуль `cmdfrombash.c` реализует основные встроенные команды shell'а, которые обеспечивают:

- Управление файловой системой (команда `cd`)
- Контроль работы shell'а (команда `exit`)
- Гибкое управление переменными окружения (команды `path`, `setpath`, `addpath`, `resetpath`)
- Понятные сообщения об ошибках и usage information
- Надежную обработку граничных случаев и ошибок

Встроенные команды выполняются эффективно без создания дочерних процессов, что делает их быстрыми и позволяющими напрямую влиять на состояние shell'а. Модуль успешно интегрируется с системой выполнения команд и обеспечивает базовую функциональность управления shell'ом.