SHELL

2 вариант

Шпилевой Георгий
207 группа

Моделирование работы интерпретатора SHELL

(программа My_Shell)

Входной язык: подмножество командного языка SHELL (определяется вариантом).

Поток команд:

1. командный файл, т.е. каждая строка файла - это отдельная команда, которая должна быть выполнена интерпретатором (имя файла - аргумент в командной строке при вызове интерпретатора);
2. стандартный входной поток команд;
3. для исполнения каждой команды требуется запуск программы

ВНИМАНИЕ!!! "побочный" эффект выполнения уже обработанных команд (например, перенаправление ввода-вывода) не должен влиять на выполнение последующих команд.

Входной язык (варианты):
Общая часть (одинаковая для всех вариантов):

• # конвейер $pr_1 | pr_2 | ... | pr_N$ для произвольного N>=2; считать, что аргументов у

pr_I (1 <= I<= N) нет (но возможна реализация с произвольным числом аргументов у каждого процесса)

• # перенаправление ввода-вывода <, >, >> (в том числе для pr_1 и pr_N в конвейере)

Например, pr < data > res pr1 | pr2 > res.txt

• # запуск в фоновом режиме & (в том числе и для конвейеров)

Например, pr arg1 arg2 & pr1 | pr2 | pr3 > res.all &

Вариантная часть:

В каждый вариант входит (как минимум) один из подпунктов каждого пункта, отмеченного римской цифрой. Звездочкой отмечены более сложные подпункты. Части подпунктов, содержащие слово «возможно», могут быть опущены при выборе варианта; их реализация усложняет вариант. Вариант определяет преподаватель.

I.

1. mv old_file new_file 2. cp file copy_file

II.

1. wc filename
результат: filename строк слов символов (возможен список имен файлов; в этом случае подобная информация выдаются о каждом файле)

2. grep substring filename

результат: строки файла filename, содержащие substring как подстроку (возможен флаг -v; в этом случае результат - это строки, ко- торые не содержат substring как подстроку)

3. cmp filename1 filename2
результат: информация о первом различии в содержимом двух файлов

Например, filename1 differs from filename2: line 5 char 36 *4. sort filename

сортировка строк файла в соответствии с кодировкой ASCII возможны флаги:

-r обратный порядок
-f не различать большие и малые буквы -n числовой порядок
+n начать сортировку с (n+1)-ой строки

III.

1. cat filenames возможен флаг:

-n с нумерацией строк (если файлов несколько, то нумерация сквозная)

2. tail filename
вывод 10 последних строк файла возможны флаги:

-n n последних строк

+n с n-ой строки и до конца файла 3. od filename

вывод содержимого файла по 10 символов в строке с указа- нием номера первого символа в каждой десятке

Например, 000001abcd\nefghi 000011 j k \t l m n

возможен флаг:
-b с указанием восьмеричных кодов символов

IV.

1. pr_1 ; pr_2 ; ... ; prN
последовательное выполнение команд pr_I - как если бы они были переданы интерпретатору по одной команде в строке

ВНИМАНИЕ !!! приоритет операции | выше, чем приоритет операции **;** однако, возможно использование скобок: напрмер, (pr1; pr2) | pr3, что приве- дет к конкатенации результатов работы pr1 и pr2, которые будут переданы про- цессу pr3 как входные данные.

2. pr1 && pr2
выполнить pr1; в случае успеха выполнить pr2

3. pr1 || pr2
выполнить pr1; в случае неудачи выполнить pr2

0. Про посылку

Чтобы получить исполняемый файл Shell:
make -f makeprog
Запустить задачу:
./prog

1.Реализация

1.1 Дерево процессов

В библиотеке tree_shell.c лежат функции создания дерева процессов и обработки дерева процессов. Подробнее о нем можно почитать дальше в этом файле.

1.2 Стек

Реализован в библиотеке stack.c.
описание можно посмотреть в readme P_3

1.3 Парсинг строки

В библиотеке parsing.c реализованы функции чтения команды из потока ввода, функции разбиения этой строки на команды и их аргументы

2. Описание типов

2.1 Cmd_inf (tree_shell.h) – ячейка дерева процессов

```
typedef struct cmd_inf {
        char ** argv; // список из имени команды и аргументов
        char *infile; // переназначенный файл стандартного ввода
        char *outfile; // переназначенный файл стандартного вывода
        int backgrnd; // ==1, если команда подлежит выполнению в
        фоновом режиме
        struct cmd_inf* psubcmd; // команды для запуска в дочернем
        struct cmd_inf* pipe; // следующая команда после “|”
        struct cmd_inf* next; // следующая после “;” (или после “&”)
        int append;
} Cmd_inf;
```

2.2 Stack (stack.h)

Стек

```
typedef struct Stack
{
        void *data;
        size_t size;
        size_t cap;
} Stack;
```

1) Указатель на массив байтов
2) размер данных в стеке
3) вместимость стека в байтах

2.3

```
enum
{
        //const for flag_operation
        START = 0,
        PIPE = 1,     //|
        LESS = 2,     //<
        MORE = 3,     //>
        SEMICOLON = 4,//;
        AMPER = 5,    //&
        BRACK_OPN = 6,//(
        BRACK_CLS = 7,//)
        DUB_MORE = 8  //>>
};
```

3. Функции

3.1 Функции tree_shell.c

1. init_cmd_inf

Заголовок:

```
int
init_cmd_inf(Cmd_inf *p);
```

Инициализирует ячейку дерева процессов.

2. mk_pipe

Заголовок:

```
Cmd_inf *
mk_pipe(Cmd_inf *prev);
```

Создает ячейку дерева процессов и кладет в поле pipe структуры *prev адрес этой новой ячейки. Функция возвращает указатель на эту новую ячейку, в случае неудачи возвращает NULL.

3. next_cmd

Заголовок:

```
Cmd_inf *
next_cmd(Cmd_inf *prev);
```

Создает ячейку дерева процессов и кладет в поле next структуры *prev адрес этой новой ячейки. Функция возвращает указатель на эту новую ячейку, в случае неудачи возвращает NULL.

4. brack_open

Заголовок:

```
Cmd_inf *
brack_open(Stack *stack, Cmd_inf *prev);
```

Создает ячейку дерева процессов и кладет в поле psubcmd структуры *prev адрес этой новой ячейки и пушает в стек адрес ячейки, на которую указывает prev. Функция возвращает указатель на эту новую ячейку, в случае неудачи возвращает NULL.

5. brack_close

Заголовок:

```
Cmd_inf *
brack_close(Stack *stack);
```

Достает из стека адрес ячейки, который был положен при встрече открывающей скобки при парсинге. Возвращает этот указатель.

6. final_tree

Заголовок:

```
void
final_tree(Cmd_inf *head);
```

Освобождает память, занимаемую деревом,

7. split_line_and_mktree

Заголовок:

```
Cmd_inf *
split_line_and_mktree(char *line, int size);
```

Функция получает указатель на строку, содержащую введнные команды и создает по ним дерево процессов. Возвращает указатель на первую ячейку дерева. В случае неудачи возвращает NULL.

8. print_tree

Заголовок:

```
void
print_tree(Cmd_inf *tree, int i);
```

Функция для отладки. Печатает графическую модель дерева процессов.

9. calculate_tree

Заголовок:

```
int
calculate_tree(Cmd_inf *proc, int in, int out, int flag);
```

Функция выполняет дерево процессов, аргументы in out и flag нужны для последующих вызовов рекурсии. in и out для дескрипторов ввода и вывода, а flag для определенных ситуаций.

3.2 Функции parsing.c

1. read_line

Заголовок:

```
char *
read_line(int *size);
```

Функция читает вводимый текст с экрана и возвращает указатель на строку с вводимым текстом. В качестве побочного эффекта кладет по адресу size размер этой строки. В случае неудачи возвращает 0.

2. split_command

Заголовок:

```
char **
split_command(char *line, int pos, int size);
```

Функция получает указатель на строку, позицию в ней и размер этой строки. Функция начинает расщеплять строку на команду и ее аргументы, начиная с данной позиции, пока не встретит служебный символ. Создает массив указателей на строки и возвращает адрес начала этого массива. В случае неудачи возвращает NULL.

3. get_file_name

Заголовок:

```
char *
get_file_name(char *line, int pos, int size);
```

Функция получает указатель на строку, позицию в ней и размер этой строки. Функция возвращает указатель на строку, содержащую имя файла, которое находилось в строке начиная с позиции pos.

3.3 Функции stack.c

1. inti_stack

Заголовок:

```
int
init_stack(Stack *stack);
```

Инициализирует стек с указателем stack нулевыми размерами и нулевым указателем на массив данных.
Возвращает 0, в случае успеха, иначе 1.

2. final_stack

Заголовок:

```
void
final_stack(Stack *stack);
```

Функция освобождает память, выделенную под стек (указатель stack)

3. pop_stack

Заголовок:

```
int
pop_stack(Stack *stack, void *data, size_t pop_size);
```

Добавляет в стек (stack) данные с указателем data и размера size. Возвращает 0, в случае успеха, иначе 1;

4. **push_stack**

   Заголовок:

   ```
   int
   push_stack(Stack *stack, const void *data, size_t push_size);
   ```

Функция пушит в стек (stack) данные с указателем data и размера size. Возвращает 0, в случае успеха, иначе 1.