P05 SHELL

Волчанинов Алексей 207

1. ***О парсинге строки***

Для разбора строки используется алгоритм рекурсивного спуска, который по ходу разбора строки строит граф из структур, каждая из которых содержит своеобразное описание команды. Соответственно, на выходе, имеем граф, удобно сконфигурированный для запуска команд, описанных в его вершинах. Структура, описывающая команду выглядит так: (рис.1). Назначение полей: argv – массив строк – аргументов (сформирован под execvp), arg_amount – кол-во аргументов в массиве (если структура описывает процесс скобок, то массив аргументов будет пустым), infile – имя входного фалйа, если он есть, outfile – имя выходного файла, если имеется, if_append – флаг типа записи в файл(один – в конец, ноль – пересоздать), is_error – служебный флаг – идентификатор синтаксической ошибки, psubcmd – указатель на первый процесс в скобках (собственно струткура, описывающая процесс скобок содержит ненулевое поле ненулевым), pipe – указатель на следующий процесс в конвейере, если таковой имеется, next – указатель на следующий (после ; или &) процесс, флаг backgrnd – поднят если &, опущен если ;.

```
struct comand_info{
    char **argv;
    int arg_amount;
    char *infile;
    char *outfile;
    int if_append;
    int backgrnd;
    int is_error;
    struct comand_info *psubcmd;
    struct comand_info *pipe;
    struct comand_info *next;
};
```

Рис 1. Структура-команда

Рекурсивный спуск состоит из пяти основных функций: shell_comand, comand_list, pipe_comand, comand, simple_comand. Shell_comand – начальная функция спуска, command_list – функция, реализующая список команд, pipe_comand – «находит» конвейеры, command – определяет является ли команда простой или командой в скобках и осуществляет различные действия в зависимости от этого. Simple_comand – функция, отвечающая за заполнение полей argv, infile, outfile в простой команде. Грамматика, взятая за основу рекурсивного спуска прилагается. Каждая функция возвращает указатель на некоторую промежуточно-сформированную структуру команды.

⟨команда_shell⟩::= ⟨список_команд⟩

⟨список_команд⟩::= ⟨конвейер⟩ { [один из &, &&, ||, ;] ⟨конвейер⟩} [&, ;]

⟨конвейер⟩::= ⟨команда⟩ { | ⟨команда⟩}

⟨команда⟩::= ⟨простая_команда⟩ | (⟨список_команд⟩)[, >>] ⟨имя_файла⟩]

⟨простая_команда>::= ⟨имя_файла⟩ {⟨аргумент⟩} [, >>] ⟨имя_файла⟩]

Рис. 2 Грамматика

Итого, на выходе алгоритма рекурсивного спуска мы имеем: либо сформированный граф из структур, который надо «запустить на выполнение», либо нулевой указатель и сообщение о синтаксической ошибку на стандартном потоке ошибок.

p.s. Файлом, именем команды или аргументом считается любая последовательность символов без служебных символов (&, ;, (,), |, <, >) и пробельных символов (то есть эти символы разделяют имена файлов, команд и аргументов.

2. ***О Запуске на Выполнение***

   Для того, чтобы выполнять команды, используется следующая концепция: на вход подается первая(входная) вершина графа из структур, после чего функция main_proc осуществляет выполнение команд по следующему алгоритму: есть основной цикл, который продолжается пока у структуры есть поле next (каждый раз переходим на следующую структуру, по указателю). Основной шаг цикла содержится в функции main_step_proc. Есть 2 концептуально разных варианта: либо между двумя разделителями (; или &) стоит конвейер, либо просто команда. Если поле pipe ненулевое, то мы имеем дело с конвейером и запускаем его обработку (функция pipe_proc), иначе просто создаем сына. Функция основного шага возвращает массив с пидами сынов, порожденных ею. Скобки реализованы следующим образом: сформировав все потоки ввода-вывода, мы решаем что мы будем делать с сыном: если запускаемый процесс – скобки, то мы рекурсивно вызываем main_proc в сыне от первой команды в скобках, если же это простая команда, то делаем execvp. Каждый проход цикла мы освобождаем таблицу процессов от зомби, проходя неблокирующим waitpid по всем массивам ранее созданных сыновей. Также, если разделитель был ;, то мы таргетировано ждем всех сыновей которые запущены за последний проход цикла. Функция file_proc ответственна за формирование ввода-вывода, link_proc – используется в pipe_proc. ***ЗАМЕЧАНИЕ:*** Так как сказано, что сирот быть не должно, перед самым завершением, отец дожидается всех сыновей (даже тех, что были запущены в фоне). Функция main_proc возвращает сформированный код завершения процесса-отца.

3. ***О тестировании***

   Программа тестировалось под санитайзерами на множестве тестов, некоторые из них приложены в файле tests.txt. В коде содержатся комментарии, частично более подробно описывающие работу программы.

4. ***О makefile***

   Программа разбита на 3 модуля: модуль с парсингом строки и созданием дерева (файлы lexems1.c и lexems1.h), модуль с функциями запуска команд (файлы making_tree_all.c и making_tree_all.h), а так же головной модуль с запуском основных функций из предыдущих модулей (файл main_proc.c). Соответственно в мэйкфайле есть цели для сборки объектников каждого модуля. Целью по умолчанию стоит компиляция всего кода в исполняемый файл solution, также имеется цель clean, которая очищает все временные и бинарные файлы.