ОТЧЕТ

Пшеничников Глеб 212

В программе реализованы следующие возможные структуы команд: <команда_shell> ::= <список_команд>

<список_команд> ::= <команда> | <конвеер> | <последовательность> | <конвеер_или> | <конвеер_и> [&]

<команда> ::= <простая_команда> | (<список_команд>) [**<** <имя_файла>] [[один из >, >>] <имя_файла>]

<простая_команда> ::= <имя_файла> {<аргумент>} (<список_команд>) [**<** <имя_файла>] [[один из >, >>] <имя_файла>]

<конвейер> ::= <команда> { **|** <команда>}

<последовательность> ::= <команда> { [один из **;**,**&**] <команда>}

<конвеер_или> ::= <команда> { **||** <команда>}

<конвеер_и> ::= <команда> { **&&** <команда>}

Примеры тестов, в которых нет ошибок (они работают так же, как в shell): pwd > fpwd cd cd.. yes | head yes | yes | yes | head sleep 5 ; pwd sleep 5 & ; pwd sleep 5 & sleep 6 & sleep 7 & pwd; ls; ls -l echo 12345 >> rt.txt echo < f1.txt

Примеры тестов, в которых выдается ошибка: pwr pwd < f3.txt (нет файла) | ls | pwd ;;

TASK5

Пшеничников Глеб 212

Вся программма разделена на 4 модуля

- **main.c** - главный модуль, координирующий работу модулей и совершающий диалог с пользователем
- **list.c** - создает список из слов, которые выделяет из стандратного входного потока
- **tree.c** - создает дерево по созданнмк списку (элементами дерева являются команды и информация о них)
- **exec.c** - запускает программы, которые указаны в дереве

Соединение модулей просиходит при помощи Makefile

main.c

Модкль содержит функции:
`void inv()` приглашает пользователя к вводу (печатает имя директории, откуда запускалась программа и символ «%»)

`void zombi()` удаляет процессы, которые когда-то были запущены на фоне и сейчас закончили свое выполнение, для этого используется функция waitpid() с флагом NO…. Этот флага позволяет не дожидаясь завершения процесса, определить статус выполнения команды (то есть определить завершилась ли программа или нет). Если программа завершилась, то она удаляется сигналом SIGKILL и соответствующее звено удаляется из списка

`void jdem()` ждет завершения всех фоновых процессов (которые на данный момент еще не завершились). Используется перед завершением программы

`void rfv()` реакция на сигнал **SIGINT**

В основной программе задается реакция на сигнала **SIGINT** и запускается бесконечный цикл, который работае по следующему алгоритму:

1. Отчищается дерево

2. Отчищется списко из слов

3. Удаляются зомби

4. Печается приглашение к вводу

5. Создается список из слов

6. Производится проверка на пустой список (в этом случае идем к началу цикла), на конец программы при помощи команды *exit* или ctrl + D (в этом случае очищается лист, ожидается завршение всех фоновых процессов и завершается пограмма), на '#' (в этом случае команда воспринимается как комментарий)

7. Создается дерево по списку слов

8. Удаляются зомби

9. Выполняются команды по построенному дереву

list.c

Модуль содержит функции:
`void nullbuf()` удаление буфера

`void addsym()` добавление символа в буфер

`void addword()` добавление буфера в список

`void clearlist(char **lst)` удаление списка

`void null_list()` обнуление списка

`void termlist()` завершение списка

`void printlist(char **lst)` печать списка

`int symset(int c)` возвращает 1 если любой сивол кроме специальных

`int getsym()` возвращает символ из строки

`char **create_list()` главная функция, создающая список

В *list.h* занесены следующие функции:

```
void printlist(char **lst);
void clearlist(char **lst);
char **create_list();
```

tree.c

Описание типа *дерева*:

```
typedef struct cmd_inf
{
    char ** argv; // список из имени команды и аргументов
    char *infile; // переназначенный файл стандартного ввода
    int typeout; // =1 если >>
    char *outfile; // переназначенный файл стандартного вывода
    int backgrnd; // =1, если команда подлежит выполнению в фоновом режиме cmd_inf* psubcmd; // команды для запуска в дочернем shell
   // int num; // кол-во процессов в pr1 | pr2 | pr3...
    tree psubcmd; //запуск дочернего
    tree pipe; // следующая команда после “|”
    tree next_one; // следующая после “;” (или после “&”)
    tree next_two; //следущая после "||"
    tree next_three; //следующая после  "&&"
}info;
```

Модуль содержит функции: `tree create_tree()` проверяет на правильный синтаксис (ксо первым символом в строке был спецсимвол - ошибка) и создает дерево. После построения проверяет на наличие ошибки: если она есть, дерево удаляется и программа возвращает управление main.c; если ошибки нет то возвращается построенное дерево

`tree null_tree()` создает новое звено для дерева и обнуляет всего его поля

`tree postr()` Заполняет звено информацией о команде. Сначала проверяет на спецсимвол (если спецсимвол - первый элемент, то ошибка) Далее считывается из списка имя самой команды и накаливаются аргументы в поле tr->argv[]. Далее, если есть, в звене указываются файлы для перенаправления ввода/вывода. При этом проверяется корректность и уместность использования стволов '>', '>>', '<' (сначала должен указываться файл для перенаправления ввода, потом для вывода). Далее определяется, выполняется ли команда в фоновом режиме (если да, то в поле звена backgrbd заносится 1, иначе 0). Далее определяется, как связана данная команда с отставными (при помощи |, &, ;, ||, &&) и из соответсвующего поля звена вызывается эта же функция

`void print_tree(tree tr)` вводит информацию о дереве. Рекурсивно обходит все дерево (выводя информацию о перенаправления ввода/вывода, о статусе типе выполнения (на фоне или нет) и выводит аргументы команды). Программа вызывается рекурсивно только из тех полей звена, в которых есть ссылка на другое звено

`void clear_tree(tree tr)` рекурсивно удаляет дерево (идем по каждому звену и очищаем его)

В *tree.h* занесены следующие функции:

```
tree create_tree();
void print_tree(tree tr);
void clear_tree(tree tr);
```

exec.c

Описание типа списка для фоновых прцессов:

```
typedef struct backgrndList {
    int pid;
    struct backgrndList *next;
} intlist;
typedef intlist *bckgrnd;
```

Модуль содержит функции:
`void pwd(tree tr)` моделирует команду pwd (при этом указаны возможные ошибки во время выполнения команды) - печатает путь до текущей директории

`void cd(tree tr)` моделирует команду cd (при этом указаны возможные ошибки во время выполнения команды) - меняет текущую директорию

`void cmd(tree tr)` вызывается в дочернем процессе. Подготавливает процесс для выполнения (перенаправляет ввод/вывод) и запускает программу при помощи execvp()

`int wait_status(tree tr, int pid)` вызывается из родительского процесса. Родитель ждет завершения сына, если сын был запущен не в фоном режиме. Если же сын запущен на фоне, то его pid заносится в список фоновых процессов. Так же функция возвращает статус завершения сына (если он был запущен не на фоне)

`int konez(tree tr)` возвращает 1, если находимя на последнем звене в дереве

`int vnutr(tree tr)` возвращает 1, команда является внутреней (pwd, cd, ext...)

`void go(tree tr)` реализует конвейер. Определяет, как соединены команды. Если при помощи pipe: отдельно выполняется первая команда (в ней канал для чтения закрывается,запись переводится в канал). Программы в середине переправляют чтение и запись в канал. И наконец последняя программа (тоже рассматривается отдельно) закрывает канала на запись, перенапряжение чтение на канал. После выполнения всех программ программа ждет завершения всех этих процессов

Если при помощи ';' или при помощи '&': программы запускаются независимо друг от друга (в цикле).Так же определятся стату выполнения (если на фоне то ставится реакция SIG_IGN на сигнал SIGINT). Потом после каждой программы вызывается функция, которая ждет завершения программы (если процесс выполняется не на фоне)

Если при помощи ‘||’: программа запускается в процессе и проверяется статус ее завершения (если завершилась корректно то дальше программы не выполняются)

Если при помощи ‘*&&*’: программа запускается в процессе и проверяется статус ее завершения (если завершилась некорректно, то дальше программы не выполняются)

`void do_exec(tree tr)` запускает выполнение функций

В *exec.h* занесены следующие функции:

```
void do_exec(tree tr);
```

Для соединения моделей были созданы list.h, tree.h, exec.h, в которых указаны функции, используемые в других модулях.