Вылиток А.А.

МОДЕЛИРОВАНИЕ РАБОТЫ ИНТЕРПРЕТАТОРА SHELL

(программа My_Shell)

Интерпретатор команд shell — это (интерактивная) программа, воспринимающая и выполняющая команды, вводимые пользователем с терминала или находящиеся в командном файле (т.е. текстовом файле, содержащем последовательность команд). Команды делятся на «внутренние», которые shell выполняет непосредственно, и «внешние», для выполнения которых создаются отдельные процессы. Имя любого исполняемого файла Unix является «внешней» командой shell. Кроме того, shell позволяет соединять выполняющиеся команды каналами (создавать «конвейер»), перенаправлять ввод-вывод команд в файлы, выполнять команды в асинхронном режиме. Интерпретатор должен выдавать приглашение на ввод очередной команды, например “=> ” или “$ ”. При наступлении ситуации «конец файла» (Ctrl-D при вводе с клавиатуры) интерпретатор завершается.

Примеры существующих командных оболочек для операционных систем Unix: bash, sh, csh, ksh, zsh; для Windows: Console (cmd.exe), Power Shell (powershell.exe), для MS DOS: command.com

Синтаксис команд shell (за исключением управляющих команд типа "if", "for" и некоторых других) определяется следующими правилами:

⟨команда_shell⟩::= ⟨список_команд⟩

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

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

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

⟨простая_команда>::=⟨имя_файла⟩ {⟨аргумент⟩}
[**<**⟨имя_файла⟩] [[один из **>**, **>>**] ⟨имя_файла⟩]

Аргумент команды — это имя файла, флаг, ключ и т. пр. Конструкция "{…}"означает повторение несколько раз фрагмента между фигурными скобками или его отсутствие. Конструкция "[один из ...]" означает обязательное присутствие одного (и только одного) из перечисленных в ней элементов. Конструкция "[...]" означает возможное присутствие одного из перечисленных в ней через запятую элементов.

Для выполнения большинства команд интерпретатор создает новый процесс и заменяет его тело на соответствующий исполняемый файл. Это так называемые внешние команды. Однако есть и внутренние, для выполнения которых не создается новых процессов. Необходимо реализовать следующие внутренние команды: cd – изменяет текущую директорию, pwd – печатает полный путь к текущей директории и exit – завершает работу интерпретатора.

- cd — переводит пользователя в домашнюю директорию, имя которой хранится в переменной $HOME (см. ниже)
- cd path/folder — переводит пользователя в указанный каталог (путь может быть как абсолютным, так и относительным).

Пустая строка (не содержащая не пробельных символов) игнорируется и выдается приглашение к вводу следующей команды.

На этапе лексического анализа должны корректно обрабатываться следующие символы:

• кавычки (позволяют вставить пробел в аргумент или имя программы), например: cat ”new file” – вывести на экран файл с именем ”new file”

• \ (экранирование символа, отменяет специальное действие последующего символа, например \” трактуется как собственно символ кавычка, а не ограничитель закавыченной подстроки), последовательность \\ позволяет ввести обычный одинарный слеш

• комментарии: при наличии во входной строке символа # (не внутри кавычек, не экранированного обратным слешем) все символы, стоящие после решётки игнорируются.

Примеры простых команд:

```
cd

gcc -o test test.c

printf "%d %d %d" 10 20 30

##
```

Также на этапе лексического анализа необходимо реализовать подстановку четырёх переменных:

• $HOME (домашняя директория пользователя),

• $SHELL (путь к исполняемому в данный момент шеллу),

• $USER (имя пользователя, запустившего процесс),

• $EUID (идентификатор пользователя, с правами которого работает процесс).

Если одна из этих последовательностей встречена в команде, нужно заменить её на соответствующее ей значение (строку). Скажем, вместо $HOME подставится /user/mydir, если домашняя директория имеет именно такое название. Как реализовать эти подстановки? С помощью специальных функций (см. справочник man). Например, обращение к функции getenv(”HOME”) возвращает значение переменной окружения HOME, т.е. имя домашней директории. Окружение наследуется от bash при запуске вашего shell и доступно как третий параметр с функции main (также см. man). Для реализации других замен полезны функции geteuid(), getlogin().

Операции **>**, **>>** позволяют перенаправить в файл результаты работы любой команды, предназначенные для стандартного вывода. (Операция **>>** добавляет результаты в конец указанного файла). Если файл не существовал, то он создается. Операция **<** переназначает входной поток, так что данные, которые выполняемая команда должна считывать из входного потока, будут взяты из файла.

Через точку с запятой (;) указываются последовательно выполняемые команды (конвейеры). Символ && (||) означает, что следующая за ним команда будет выполнена только в том случае, если предыдущая команда завершилась успешно (неуспешно), т.е. возвратила нулевое (ненулевое) значение. Прерывание сигналом также считается неуспешным завершением.

Операция & указывает, что предшествующая ей команда (конвейер) выполняется в фоновом режиме, т.е. shell, не ожидая завершения данной команды, выполняет следующую. Команда, выполняемая в фоновом режиме, не должна прерываться сигналом SIGINT, не должна читать со стандартного входного потока. Когда фоновая команда выполнится, shell обычно сообщает, что она завершена перед тем, как выполнить очередную введенную пользователем команду.

Конвейер позволяет запустить указанные в нем команды в виде отдельных параллельно работающих процессов, соединив каналом стандартный выходной поток каждого процесса

(кроме последнего) со стандартным входным потоком следующего процесса. Работа конвейера завершается с окончанием работы последней команды в нем.

Круглые скобки означают, что для выполнения указанного в них списка команд создается новый процесс shell (т.н. subshell), который исполнит эти команды и возвратит в отцовский процесс статус (код) завершения последней выполненной команды (т.е. то число, которое программа команды вернула с помощью функции exit или оператора return в функции main).

Примеры команд shell:

1) `(mv f1.c f2.c; cp f2.c f3.c) &`
2) `sort -n < source | grep 'xyz'| cat > result`
3) `( ls -R dir1 | grep 'mydir'; (pwd; cd .. ; pwd) | sort) | wc`

Для выполнения задания по моделированию интерпретатора команд можно ограничить синтаксис входного языка; описать полученный вариант в виде БНФ или синтаксической диаграммы. Реализованный вариант синтаксиса иметь с собой (в виде БНФ) при сдаче задания.

Обязательно должны быть реализованы следующие конструкции:

- конвейер
  pr1 | pr2 | ... | prN
  для произвольного N≥2;

можно считать, что аргументов у prI ($1 \le I \le N$) нет, но возможна реализация с произвольным числом аргументов у каждого процесса в конвейере

- перенаправление ввода-вывода
  <, >, >>
  (в том числе для pr1 и prN в конвейере)
  Примеры:
  ```
  pr < data > res
  pr1 | pr2 > res.txt
  ```

- запуск в фоновом режиме &
  (в том числе и для конвейеров)
  Примеры:
  ```
  pr arg1 arg2 &
  pr1 | pr2 | pr3 > res.all &
  ```

Должна быть реализована хотя бы одна из следующих операций:

1) pr1 ; pr2 ; ... ; prN
   (последовательное выполнение команд prI – как если бы они были переданы интерпретатору по одной команде в строке)

2) pr1 && pr2
   (выполнить pr1; в случае успеха выполнить pr2)

3) pr1 || pr2
   (выполнить pr1; в случае неудачи выполнить pr2)

Приоритет операции ‘|’ выше, чем приоритет операции ‘;’ ,т.е. например, pr1;pr2|pr3 даст тот же эффект, что и pr1;(pr2|pr3). Однако, допустимо (pr1;pr2)|pr3, что приведет к конкатенации результатов работы pr1 и pr2, которые будут переданы процессу pr3 как входные данные. Операции ‘&’ и ‘;’ имеют равный приоритет.
Операции ‘||’ и ‘&&’ имеют равный приоритет, который ниже, чем у ‘|’, но выше, чем у ‘&’ и ‘;’.
Наивысший приоритет у ‘<’, ‘>>’, ’>’.

Желательно реализовать хотя бы одну из следующих возможностей:

a) возможность исполнения командных файлов

b) возможность выполнения команды cmd путем передачи ее в качестве параметра при вызове my_shell:

```
my_shell cmd
```

c) выполнение команд, указанных в круглых скобках в рамках дочернего процесса my_shell

Пример: `(ls|wc)|cat`

Также по желанию (если будет время но это) можно реализовать

d) управляющие конструкции "if", "for" и другие, т.е получить полноценный командный язык программирования

Обработка ошибок

Для ошибочных команд должно выдаваться сообщение об ошибке (хотя бы краткое “error”), но возможно и с пояснением, в чем ошибка. Ошибки могут быть лексические, синтаксические, семантические.

Во всех неясных случаях семантики тех или иных конструкций, ориентируйтесь на поведение bash, читая man и экспериментируя с bash.