РЕАЛИЗАЦИЯ КОМАНДНОГО ИНТЕРПРЕТАТОРА ДЛЯ UNIX-ПОДОБНЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Д. П. Лебедев

Белорусский государственный университет, г. Минск;
daniel.lebedzeu@gmail.com;
науч. рук. — В. В. Горячкин, канд. физ.-мат. наук, доц.

В статье рассматриваются вопросы разработки командного интерпретатора, способного работать под управлением Unix-подобных операционных систем. Уделено внимание изучению места командных интерпретаторов среди средств организации пользовательского интерфейса и современных командных оболочек Unix-систем. Кроме этого, освещены такие аспекты разработки интерпретатора, как проектирование входного языка интерпретатора на основе выдвинутых требований, простота разбора этого языка и внутреннего устройства его анализаторов, а также техническая реализация конструкций, поддерживаемых им. Практическая значимость проведенного исследования состоит в возможности использования разработанного интерпретатора в реальных ситуациях и его улучшения при необходимости.

Ключевые слова: Unix; пользовательский интерфейс; командный интерпретатор; формальный язык; процесс разбора.

Среди видов пользовательского интерфейса выделяется *интерфейс командной строки*, также именуемый *текстовым*. Суть его состоит в том, что пользователь вводит с клавиатуры *команды,* предписывающие выполнить то или иное действие. После ввода команды исполняются компьютером, а результаты выводятся на экран. Для этого используется специальная программа, называемая *командным интерпретатором* или *оболочкой командной строки*.

Одно из важнейших мест интерфейс командной строки в целом и командные интерпретаторы в частности занимают в операционных системах семейства Unix. Такое положение обусловлено прежде всего тем, что уже в начале истории Unix в целом завершилось формирование классической *командной оболочки Unix* вместе с достаточно богатым набором ее возможностей, включавшим в том числе поддержку управляющих конструкций (ветвлений и циклов) [1].Классическая оболочка Unixтакже называется *Bourneshell* по имени своего создателя Стивена Борна или *sh*по названию своего бинарного файла.

Современные интерпретаторы также поддерживают многие другие возможности, не связанные напрямую с исполнением команд, — автодополнение команды (в том числе программируемое), хранение истории команд и прочее.

В современных Unix-системах наиболее широко распространены следующие командные интерпретаторы:

- *bash* (*BourneAgainShell*) — наиболее популярный интерпретатор для Linux, который обратно совместим с *sh*, но поддерживает и ряд собственных расширений;
- *ksh*(*KornShell*) — альтернативный *sh*-совместимый интерпретатор, который отличаетсяпрежлелучшей поддержкой сценариев;
- *tcsh* — интерпретатор, основанный на *csh*(*CShell*), основной особенностью которого является синтаксис входного языка, приближенный к синтаксису Си, основная оболочка для пользователя rootв системе FreeBSD;
- *fish*(*FriendlyInteractiveShell*) — интерпретатор, рассчитанный прежде всего на интерактивное использование и поддерживающий такие возможности, как подсветка синтаксиса, отображение подсказки и краткая справка по команде;
- *zsh*(*ZShell*) — *sh*-совместимый интерпретатор, который объединяет в себе возможности всех предыдущих [2].

Разработанный интерпретатор отличается минимализмом по возможностям в сравнении с рассмотренными: так, не поддерживаются не только автодополнение или история команд, но даже управляющие конструкции. Тем не менее, основные возможности интерпретаторов поддерживаются на уровне входного языка, что уже позволяет использовать его для решения реальных задач. Кроме того, за счет дизайна интерпретатора возможно расширять его возможности, не влияя при этом на уже реализованную функциональность.

Интерпретатор состоит из следующих частей: модуль разбора, состоящий из лексического и синтаксического анализаторов, модуль функциональности, который реализует возможности интерпретатора, и главный модуль, который связывает два предыдущих модуля воедино.

Реализованные возможности интерпретатора включают в себя:

- запуск одиночных команд и перенаправление потоков ввода-вывода для них, а также открытие/дублирование потоков ввода-вывода под произвольным номером дескриптора;
- объединение команд в так называемые связки:
  - конвейеры — связки, в которых стандартный вывод команды непосредственно связан со стандартным выводом следующей команды;
  - условные связки, в которых исполнение очередной команды зависит от того, успешно ли выполнилась предыдущая команда;
  - безусловные связки: фоновое выполнение предыдущей команды, последовательное выполнение команд.

Все связки, реализованные в интерпретаторе, являются левоассоциативными, то есть составляющие их команды и связки

исполняются слева направо. Кроме того, для команд и связок действует приоритет: сначала исполняются одиночные команды, затем конвейеры, затем условные связки, затем безусловные.

Эти возможности командного интерпретатора нашли свое воплощение в конструкциях входного языка.

Разбор этого языка разделяется на две части: лексический и синтаксический анализ. Такое разбиение обычно для процесса разбора вообще, поскольку позволяет достичь сразу нескольких выгод: упрощения целевой грамматики, разделения обязанностей и возможности генерировать один и тот же поток лексем при разных кодировках[3].

Для генерации лексем используется отдельный лексический анализатор, представляющий собой детерминированный конечный автомат, который сохраняет свое состояние и накопленную часть лексемы. Когда автомат решит, что очередная лексема готова, он передает ее наружу. Также каждому состоянию соответствует свой обработчик. Таким образом, добавление новых лексем представляет собой простой процесс: достаточно написать новый обработчик состояния или состояний, соответствующих новой лексеме, а также добавить все нужные переходы, не меняя при этом существующих.

Для построения синтаксического анализатора используется контекстно-свободная грамматика, нетерминальные символы которой соответствуют той или иной конструкции (команде, связке), а терминальные — той или иной лексеме (слову, составляющему команду, а также служебным символам). Грамматика входного языка принадлежит классу SLR(1) и, следовательно, допускает сравнительно простой алгоритм табличного разбора, а вместе с тем и простое и детерминированное построение новой таблицы по известному алгоритму (см. [3]), что дает возможность быстрого расширения грамматики.

Исполнение команды происходит путем обхода подготовленного дерева разбора в глубину. Подготовка состоит в том, что из дерева удаляются все бесполезные вершины, которые усложняют реализацию конструкций языка. При этом при посещении очередной вершины дерева вызываются обработчики, соответствующие ее типу, что опять-таки позволяет поддержать добавление новых конструкций.

Если для исполнения одиночной команды необходимо запустить программу, то интерпретатор порождает для нее отдельный процесс системным вызовом *fork*и, после подготовительных действий запускает программу вызовом *execvp*. Использование именно этого вызова оправдано тем, что он позволяет запускать все программы из

директорий, перечисленных в переменной окружения PATH[4], то есть практически все программы, установленные в системе.

В подготовительные действия входит в том числе перенаправление потоков ввода-вывода, которое состоит из открытия необходимых файлов и дублирования дескрипторов при помощи системного вызова *dup2*.

Конвейер организуется при помощи связывания стандартных потоков, составляющих команду, посредством анонимного канала, при этом каналов необходимо ровно на один меньше, чем число команд в конвейере.

Для организации условных связок необходимо поддерживать статус последней команды, что можно сделать, если запомнить статус, возвращенный системным вызовом *waitpid*; при этом успешно завершившейся командой считается та команда, которая завершилась сама (то есть не по сигналу) с кодом 0.

Наконец, фоновые команды организуются следующим образом. Для их исполнения порождается отдельный процесс, который и проинтерпретирует поддерево разбора, соответствующее команде. При этом интерпретатор не ожидает его выполнения, но может получить информацию о его исполнении в будущем.

Исходный код интерпретатора размещен автором в GitHub-репозитории, который можно просмотреть по ссылке [5].

Реализованный интерпретатор, с одной стороны, достаточно минималистичен, то есть включает в себя лишь необходимый минимум функций. Как следствие, после компиляции получается бинарный файл, размер которого невелик относительно других интерпретаторов (порядка 120 КБ против более чем 1 МБ у того же *bash*). Таким образом, он может найти свое применение в ситуациях, когда размер доступной памяти имеет значение (например, во встраиваемых системах). Его также можно относительно просто обогащать новыми конструкциями за счёт особенностей его архитектуры (таких как наличие отдельных обособленных друг от друга обработчиков).

Библиографические ссылки

1. *StephenR. Bourne.* TheUnixshell // BYTE. 1983. Volume 8, Number 10.
2. Which Linux Shell Is Best? 5 Common Shells Compared // makeuseof.com. URL: https://www.makeuseof.com/tag/best-linux-shells (дата обращения: 27.05.2022).
3. *AlfredV. Ahoetal.*Compilers: principles, techniques, andtools. 2nded. Boston: Pear-sonEducation, 2007. 1009 p.
4. exec(3) — Linux manual page. URL: https://man7.org/linux/man-pages/man3/exec.3.html (дата обращения: 27.05.2022).
5. alphaver/dipsh. URL: https://github.com/alphaver/dipsh (дата обращения: 27.05.2022).