

В осеннем семестре второго курса в рамках изучения языка программирования **C** будет выдано два практических задания. Первое задание является обязательным к выполнению для получения зачёта.

Задания разделены на несколько логических этапов, каждый из которых предполагает свой срок сдачи. Разумеется задание можно сдать полностью (или несколько этапов), но при этом все крайние сроки должны быть соблюдены.

Задание 1: Разработка интерпретатора команд

Краткое описание задания

В качестве первого задания практикума предлагается написать программу на языке **C** с использованием программного интерфейса (*API*) ОС семейства *Unix*. Данная программа представляет собой упрощённый интерпретатор пользовательских команд, похожий на *bash*. Отметим, что все стандартные команды, в числе которых *ls*, *pwd*, *chmod*, *rm*, *mv*, *cp*, *find*, *grep* и др. уже есть в системе и, как правило, располагаются в каталоге */bin/*. Последнее означает, что сами команды нужно не реализовывать, а вызывать с помощью программного интерфейса (*API*) операционной системы.

Упрощённое описание интерпретатора команд

С точки зрения пользователя выполнение программы интерпретатора представляет собой цикл из трёх последовательных шагов: вывод приглашения к вводу команды, чтение и анализ команды и собственно выполнение команды или печать сообщения об ошибке.

На первом шаге в стандартный поток вывода программы (*STDOUT*) печатается приглашение к вводу команды, которое, обычно, оканчивается символом доллара *\$*, а также содержит различную дополнительную информацию (имя или путь к текущей директории, имя пользователя и т.п.). Примерами приглашений к вводу являются: *scripts \$*, *antontodua@antontodua:~/backend-cpp\$* и др.

На следующем (втором) шаге интерпретатор целиком считывает команду пользователя со стандартного потока ввода (*STDIN*). Чтение команды обычно завершается символом перевода строки *\n*, который возникает при нажатии клавиши *ENTER*. Примерами пользовательских команд являются: *ls\n*, *cd ..\n*, *mv a.txt task.txt\n* и др.

Если команда не уместится в одну строку допустимо использование символа обратного слэша ** непосредственно перед переводом строки *\n*, который в этом случае будет проигнорирован. Другими словами, команды *mv a.txt task.txt\n*, *mv a.txt\\n task.txt\n* и *mv a.txt \\ntask.txt\n* являются эквивалентными.

Текст команды состоит из нескольких слов (*лексем*), разделённых одним или несколькими пробельными символами. Первое слово задаёт исполняемый файл (программу), который должен быть запущен для выполнения команды. Второе и последующие слова задают аргументы командной строки, которые будут переданы программе при запуске. Разумеется, передача аргументов командной строки не является обязательной. К примеру, команда *ls\n* не содержит аргументов командной строки, а команда *mv a.txt task.txt\n* — содержит сразу два — *a.txt* и *task.txt*.

Иногда в аргументе команды хочется передать пробельный символ или один из специальных символов (например, вертикальной черты *|*). В подобных случаях аргумент частично или полностью заключается в

кавычки (" двойные или ' одинарные). К примеру, следующие вызовы команды `echo` с одним аргументом полностью эквивалентны: `echo "Hello, World!", echo 'Hello, World!', echo Hello,' 'World!, echo Hello', 'World!, echo Hello", "World!, echo "Hello, "World!, echo Hello', World!'`. Отличие двойных " кавычек от одинарных ' состоит в том, что двойные кавычки допускают экранирование символов (по аналогии со строковыми константами в языке **C**), например, экранирование самих двойных кавычек `\"` и символа обратного слэша `\\`.

На последнем (третьем) шаге интерпретатор выполняет считанную команду или сообщает пользователю о невозможности выполнения, т.к. команда содержит ошибку. Распространённой ошибкой является отсутствие исполняемого файла, например, попытка выполнить команду `zxcvb qwe` приведёт к ошибке: `-bash: zxcvb: command not found`. Помимо существования файла, также требуется, чтобы у него были права на исполнение, в этом случае выполнение команды по сути представляет собой вызов найденного файла с заданными аргументами.

Требование к выполнению задания

Разработка интерпретатора команд разделена на несколько логических этапов:

1. Печать стандартного приглашения к вводу команды и чтение команды (перевод текста команды в некоторую программную структуру данных, пригодную для дальнейшего выполнения команды)
2. Выполнение команды, прочитанной на предыдущем шаге, включая команды `cd` — смена текущей директории и `exit` — завершение работы интерпретатора
3. Поддержка режима выполнения команд в фоновом режиме с помощью символа амперсанда `&`, например: `echo a & echo b & echo c`
4. Поддержка перенаправления стандартных потоков ввода и вывода с помощью символов `<`, `>` и `>>`, например: `echo abc >> abc.txt`
5. Поддержка конвейерного режима выполнения команд с помощью символа вертикальной черты `|`, например: `find -type f -name '*.swp' | xargs rm -rf`

Стоит отметить, что для сдачи первого этапа задания достаточно считать текст команды в структуру достаточную для реализации второго этапа, т.е. обрабатывать символы амперсанда `&`, перенаправления стандартных потоков `<`, `>`, `>>` и вертикальной черты `|` не требуется. Однако, выполнение первого этапа с учётом данных особенностей, позволит сильно облегчить разработку последующих этапов.

Ещё раз подчеркнём, что сами команды (вроде `cat`) реализовывать не нужно, они уже есть в операционной системе! Для первого этапа достаточно просто считывать со стандартного потока команду (имя файла) и её аргументы. К примеру, если пользователь напечатал `find -type f -name '*.swp'`, то результат чтения этой команды можно сохранить в виде массива строк: `find, -type, f, -name, *.swp`. Такая структура данных (массив строк) подходит для вызова стандартной функции `execvp`, необходимой для выполнения команды.

Для выполнения второго и последующих этапов задания также понадобятся и другие (помимо `execvp`) функции **API** операционной системы, в их числе: `fork`, `wait`, `chdir`, `waitpid`, `open`, `close`, `dup`, `dup2`, `pipe`. Объявления большинства необходимых функций **API** располагаются в стандартном заголовочном файле `unistd.h`.