Лабораторная 4.

Реализация интерактивной оболочки

Лабораторная работа направлена на знакомство студентов с основными концепциями управления процессами и приобретение практического опыта по созданию процессов, перенаправлению ввода/вывода, организации каналов (pipe), а также работы с парсерами.

Требуется реализовать упрощенный вариант интерактивной оболочки командной строки (shell). Необходимо обеспечить базовую функциональность оболочки, включая запуск внешних программ, поддержку каналов (|), перенаправлений ввода/вывода и выполнения в фоне. Реализация должна быть на языке системного программирования (C, C++ или Rust).

Для упрощения задачи вам предоставлен парсер на основе Flex и Bison, который анализирует выражения вида:

```
ls -l
ls -l > /tmp/files
grep foo < /tmp/files
ls -l | wc -l
sleep 100 &
```

и превращает их в структуру данных `struct pipeline`, состоящую из одной или нескольких команд. Например,

```
ls -la | wc -l > /tmp/lines &
```

будет интерпретироваться как:

```
struct pipeline = {
     .first_command = {
          .next = {
               .argv = {"wc", "-l", NULL},
               .argv_cap = 8,
               .argc = 2,
               .output_redir = "/tmp/lines",
               .input_redir = NULL,
```

```
            },
            .argv = {"ls", "-la", NULL},
            .argv_cap = 8,
            .argc = 2,
            .output_redir = NULL,
            .input_redir = NULL,
        },
        .background = 1,
    }
```

Дополнительную информацию можно найти в файле `parse.h`. Кроме того, парсер оболочки также распознаёт следующие встроенные команды: `wait`, `exit`, `kill`, `declare`, `cd`, `pushd`, `popd`, `help`. Они будут преобразованы в `enum builtin_type`, а необязательный аргумент будет передан в `buitin_arg`.

Данный парсер требует установки Flex и Bison:

```
# apt-get install flex bison
```

Задание

Ваша задача — реализовать две функции:

1. `run_pipeline` должна принимать и обрабатывать один экземпляр структуры конвейера.
   a. Если в конвейере содержится более чем одна `struct command`, то каждый стандартный ввод последующей команды должен быть соединён со стандартным выводом предыдущей команды с помощью канала (за исключением первой команды, у которой нет предыдущего стандартного вывода, к которому она могла бы быть подключена).
   b. Если задано `output_redir`, то стандартный вывод команды должен быть перенаправлен в файл по указанному пути. Стандартный вывод должен быть доступен для записи.

c. Если задано `input_redir`, то стандартный ввод команды должен быть перенаправлен из файла по указанному пути. Стандартный ввод должен быть доступен для чтения.

2. `run_builtin` должна выполнять одну из нескольких встроенных команд:
   a. `wait [PID]`: принимает необязательный аргумент `PID` через `builtin_arg` (`builtin_arg` равен `NULL`, если не указан пользователем). Если аргумент указан, то оболочка должна рассматривать его как идентификатор процесса и использовать `waitpid(2)` для этого идентификатора процесса. Если аргумент не указан, следует использовать `waitpid(2)` для ожидания любого дочернего процесса.
   b. `exit [CODE]`: принимает необязательный аргумент `CODE` через `builtin_arg` (`builtin_arg` равен `NULL`, если не указан пользователем). Если указан код выхода, оболочка должна использовать `exit(2)` для завершения работы с заданным кодом выхода. Если аргумент `CODE` не указан, оболочка должна завершить работу с кодом выхода `0`.
   c. `kill PID`: Принимает обязательный аргумент `PID` через `builtin_arg` (`builtin_arg` равен `NULL`, если не указан пользователем). Если аргумент не указан (`builtin_arg` равен `NULL`), должно быть выведено осмысленное сообщение об ошибке. При вызове встроенной функции `kill` оболочка должна отправить сигнал `SIGTERM` процессу с указанным `PID`.
   d. `declare [NAME[=VALUE]]`: Принимает необязательный аргумент `NAME=VALUE` через `builtin_arg` (`builtin_arg` равен `NULL`, если не указан пользователем). Если аргумент указан и содержит как `NAME`, так и `VALUE`, оболочка должна установить переменную окружения `NAME` в значение `VALUE` с помощью `setenv`. Если присутствует только часть `NAME`, оболочка должна удалить

переменную NAME из окружения с помощью unsetenv. Наконец, если аргумент не указан, оболочка должна отобразить имена и значения всех переменных окружения.

e. cd [DIRECTORY]: Принимает необязательный аргумент DIRECTORY через builtin_arg (builtin_arg равен NULL, если не указан пользователем). Если аргумент указан и является каталогом, оболочка должна изменить значение переменной окружения PWD в соответствии с ним. Если аргумент указан, но такой объект в файловой системе не существует или не является каталогом, то должно быть выведено содержательное сообщение об ошибке. Если аргумент не указан (builtin_arg равен NULL), оболочка должна изменить значение PWD, установив его равным значению переменной окружения HOME.

f. pushd DIRECTORY: Принимает обязательный аргумент DIRECTORY через builtin_arg (builtin_arg равен NULL, если не указан пользователем). Если аргумент не указан (builtin_arg равен NULL), то должно быть выведено содержательное сообщение об ошибке. При вызове встроенной функции pushd оболочка должна выполнить те же действия, что и для cd, но перед изменением переменной окружения PWD она должна сохранить её старое значение во внутреннем стеке посещенных каталогов.

g. popd: Не принимает аргументов (builtin_arg равен NULL). При вызове встроенной функции popd оболочка должна взять верхнее значение из внутреннего стека посещенных каталогов и изменить переменную окружения PWD в соответствии с ним.

h. help [NAME]: Принимает необязательный аргумент NAME через builtin_arg (builtin_arg равен NULL, если не указан пользователем). Если аргумент указан и представляет собой имя допустимой встроенной команды, оболочка должна вывести содержательное информационное сообщение об этой команде. Если

аргумент указан, но встроенной команды с таким именем не существует, оболочка должна вывести содержательное сообщение об ошибке. Если аргумент не указан (`builtin_arg` равен `NULL`), оболочка должна вывести краткую информацию о себе и обо всех доступных встроенных командах.

Тестирование

Для удобства тестирования и отладки Вам предоставлен набор скриптов на языке программирования Python, которые выполняют автоматическую проверку части функциональности Вашей оболочки. Скрипты ожидают наличия исполняемого файла с именем `shell` в текущем каталоге. Запустите `make check`, чтобы проверить вашу реализацию на соответствие (неполное!) условиям задания.

Оценивание

- Максимальный **первичный** балл за лабораторную — 10.
- Каждый подпункт из списка выше оценивается в 1 балл, **кроме** 1.a, который **обязателен** к выполнению — без него работа не засчитывается.
- Балл в **итоговый** рейтинг = 0.1 * первичный балл.
- Если студент не может объяснить, что делает та или иная часть программного кода, работа оценивается в 0 баллов.