Лабораторная работа 2

Выполнил Куприянов АА, P33113

Задача

Разработать консольное приложение на языке Ruby, который будет взаимодействовать с shared библиотекой для работы с файловой системой XFS

Выполнение

Давай по новой, Миша, все *!

Первые трудности с которыми я столкнулся это переписывание существующих методов из первой лабы. Переписывать сильно их конечно не надо было, но приходилось, например, вместо вывода в `stderr` использовать буфер, что потом уже печатать его в `UI`

```
void xfs_pwd(char* output_buf, struct xfs* xfs) {
    strcpy(output_buf, xfs->path);
    strcat(output_buf, "\n");
}
```

Там кто-то снизу стучится

Для работы с низкоуровневой библиотекой использовалась библиотека FFI (https://github.com/ffi/ffi).

Во-первых, необходимо было зааттачить функции с маппингами типов. Подробнее о маппингах можно посмотреть здесь: https://github.com/ffi/ffi/wiki/Types

Создаем своего рода враппер, через который будем общаться с нашей библиотекой

```
module Utils
  extend FFI::Library
  ffi_lib 'shared/libxfs.so'
  attach_function(
    :init_output_buffer,
    :init_output_buffer,
    [],
    :pointer
  )
  attach_function(
    :destroy_output_buffer,
    :destroy_output_buffer,
    [:pointer],
    :void
  )
  attach_function(
    :get_disks_and_partitions,
    :get_disks_and_partitions,
    [:pointer],
    :void
  )
```

```
  attach_function(
    :execute_xfs_operation,
    :execute_xfs_operation,
    [:int, :pointer, :int, :pointer, :pointer],
    :void
  )
  attach_function(
    :xfs_init,
    :init,
    [:string, :pointer, :pointer],
    :pointer
  )
  attach_function(
    :init_argv,
    :init_argv,
    [:int],
    :pointer
  )
  attach_function(
    :add_argv,
    :add_argv,
    [:int, :pointer, :string],
    :int
  )
  attach_function(
    :destroy_argv,
    :destroy_argv,
    [:pointer, :int],
    :void
  )
  attach_function(
    :get_command_by_str,
    :get_command_by_str,
    [:string],
    :int
  )
end
```

Тысячи и тысячи историй

Думаю, нет особого смысла рассказывать про UI составляющую, так как она не представляет особого интереса. Выступает в виде интерфейса между пользователем и низкоуровневой библиотекой

Про использование низкоуровневых библиотек

Думаю, это представляет большой интерес для программ, которые выполняют множество вычислений и требуют высокой производительности. Так как работа в таком виде мягко говоря не доставляет особого удовольствия. Возьмем те же маппинги, писать под каждый метод для больших библиотек и проектов весьма геморно

Я нашел Рубин!

Мое личное мнение про руби: имеет право на существование. Язык весьма интересный основанный на полной инкапсуляции и парадигм ООП. Даже сравнивая с джава, руби намного ближе к ООП. Поэтому думаю он зайдет любителям чистого ООП и его принципов. Хотя конечно же это зависит от самих программистов, так как плохой код отличный от того что хотели разработчики рубина весьма легко.

Вывод

Использование shared библиотек языками высокого уровня имеет место быть, но нужно учесть, что это добавляет больше кода, а соответственно больше шансов допустить ошибку. Поэтому нужно заранее определиться, что важнее производительность или поддержка кода и в какой степени