Разработка прототипа механизма межпроцессного взаимодействия для Linux операционных на базе unix сокетов, предоставляющего разработчикам RESTful интерфейс взаимодействия.

Проблематика

В операционных системах на базе Linux чаще всего используется механизм межпроцессного взаимодействия DBus, который предоставляет разработчикам довольно большой функционал. Однако он имеет ряд ключевых недостатков:

- Высокий порог входа для разработчиков, так как DBus имеет под собой непривычную для первого взгляда и довольно сложную концепцию сервисов, объектов и интерфейсов.
- Свой формат сообщений, непривычный для разработчиков, которые взаимодействуют с ним впервые.
- Отсутствие официальных библиотек, предоставляющих API для взаимодействия с DBus, что приводит к отсутствию стандартизированных подходов при разработке различных проектов.

Большинство из данных проблем связано с высоким порогом входа для разработчиков. Эту проблему можно решить, если предоставить разработчикам простой и проверенный временем RESTful интерфейс взаимодействия, который используется в WEB, для реализации межпроцессного взаимодействия.

Ключевые преимущества RESTful интерфейса:

- Простота концепции
- Привычный для разработчиков формат сообщений – JSON
- Наличие наработанных годами подходов и практик построения удобного API для взаимодействия компонентов.

В Linux системе имеются unix-сокеты, которые отлично подходят для решения задачи межпроцессного взаимодействия на базе RESTful интерфейса, так как суть концепции – обмен JSON сообщениями.

Также имеется много библиотек и алгоритмов для бинарной сериализации и десериализации JSON сообщений, что положительно скажется на производительности при общении двух процессов между собой.

Задачи

1. Провести анализ и изучить концепции механизма межпроцессного взаимодействия DBus.
2. Разработать C++ библиотеку предоставляющий простой и удобный API для построения клиент-серверного межпроцессного взаимодействия с RESTful интерфейсом.
3. Сравнить скорость P2P взаимодействия процессов используя аналогичные по логике проекты на базе DBus и на базе разработанного механизма.
4. Провести анализ полученных результатов
5. Выделить перспективы развития разработанного механизма, например:
   5.1. Добавление middleware для фильтрации входящих запросов
   5.2. Добавление Polling API в качестве альтернативы сигналам в DBus
   5.3. Создание модуля ядра для обеспечения более безопасного и производительного взаимодействия между процессами
6. Итоги

Примерный API библиотеки

(Концепция на базе популярной REST библиотеки express.js)

```
int server() {
  IPC::Server server("com.example.server");

  server.on("/users", [](const IPC::Request &req, const IPC::Response &res) {
    res.send(IPC::Json{{"users", std::vector<int>{0, 1, 2}}});
  });

  server.on("/users/create", [](const IPC::Request &req, const IPC::Response &res)
  {
    IPC::Json body = req.body();
    IPC::Cred credentials = req.credentials();

    if (credentials.uid() != 0) {
      res.error(IPC::Error{
          "ErrorClientNotAuthorized",
          "The client has unauthorized user id"
      });
      return;
    }

    if (!create_user_with_id(body["uid"])) {
      res.error(IPC::Error{
          "ErrorInternal",
          "This is error message"
      });
      return;
    }

    res.send(IPC::Json{{"ok", true}});
  });

  return server.serve();
}



int client() {
    IPC::Client client("com.example.server");

    client.call("/users", [](const IPC::Error &error, const IPC::Json &reply) {
        if (error) {
            std::cout << "Failed to request users ("
                      << error.message() << ")" << std::endl;
        } else {
            for (const auto &uid : reply.at<std::vector<int>>("users")) {
                std::cout << "Uid: " << uid << std::endl;
            }
        }
    });
}
```