Эмулятор "Умного дома" на базе сокетов

EASY

Автор проекта: Ключникова А.В.

Цель проекта

Разработать распределённую систему эмуляции IoT-устройств (датчиков и исполнительных механизмов) с взаимодействием через локальную сеть (TCP/UDP).

Проект должен отрабатывать навыки:

- Работы с сетевыми протоколами на низком уровне (сырые сокеты).
- Обработки данных без динамической памяти (как в embedded).
- Синхронизации и обработки ошибок в ресурсо-ограниченной среде.

Состав системы

Система состоит из 3 типов программ (каждая - отдельный процесс):

Устройство	**Роль**
Датчик температуры	Генерирует случайные значения (20–30°C), отправляет на сервер раз в 1 сек.

Устройство	Роль
Контроллер света	Получает команды (вкл/выкл), хранит состояние в битовой маске.
Сервер-агрегатор	Принимает данные от датчиков, отправляет команды устройствам, логирует.

Технические требования

Общие

- **Язык**: Чистый C (стандарт C11/C17).
- **Сборка**: Makefile или CMake.
- **Запрещено**: Использовать готовые библиотеки для сетей (только системные вызовы: `socket`, `bind`, `sendto и т.д.`).
- **Платформа**: Linux

Датчики температуры

- **Протокол**: UDP
- **Данные**: JSON-подобный формат:

```
{"dev": "temp1", "val": 23.5, "ts": 1234567890}
```

- **Особенности**:
  - Задержка 500 мс перед отправкой (эмуляция АЦП).
  - При недоступности сервера — повторная отправка через 2 сек.

Контроллер света

- **Протокол:** TCP
- **Команды**: Пример сообщения с командой от сервера:

```
{"cmd": "set_light", "id": 1, "state": "on"}
```

- **Особенности**:
  - Управление N количеством светильников (например, 8, 16, 32).
  - Состояние хранится в **битовой маске** (1 бит = 1 светильник).
  - Пример ответа серверу:

    ```
    {"status": "ok", "new_state": 0b1010}
    ```

Сервер-агрегатор

- **Функции:**
  - Принимает данные от датчиков по UDP.
  - Управляет контроллерами по TCP.
  - Логирует события в отдельный файл `.`
- **Особенности:**
  - Обработка до 5 одновременных TCP-подключений.

Дополнительные задания

1. **Исключить использование** `malloc и printf`.

   Задание напрямую связано с ограничениями в embedded-разработке, где критически важны:
   - Детерминированное поведение (никаких неожиданностей в runtime)
   - Минимальное потребление памяти (куча часто отсутствует или сильно ограничена)
   - Скорость выполнения (динамическая аллокация медленнее статической)

   Запрет malloc (и других функций кучи): В реальных микроконтроллерах куча (heap) либо отсутствует, либо её размер — несколько КБ. malloc может фрагментировать память и через условно час работы устройство упадёт из-за "Out of Memory". Альтернативой является использование статических буферов (глобальные или на стеке, если известен размер) и пул объектов (заранее выделенные массив структур).

   Запрет printf (и других stdio-функций): printf тянет за собой тяжёлую библиотеку (до 10+ КБ в бинарнике). Также эта функция использует динамическую память внутри (например, для форматирования чисел). Альтернативой для функции являются write / send - для вывода в

сокет/файл и самописные функции.

2. **Реализация простого шифрования (XOR + Base64)**

   Задание включает работу с бинарными данными и кодированием.
   Требования:

   - Все сообщения между устройствами должны шифроваться алгоритмом:
     - XOR каждого байта с ключом 0xAA.
     - Кодирование результата в Base64 (реализация без внешних библиотек).
   - Сервер и клиенты должны автоматически шифровать/дешифровать данные.

3. **Эмуляция аварийного события (перегрев > 28°C)**

   Задание предназначено для отработки реакции на критические события в реальном времени.
   Требования:

   - Если сервер получает значение температуры > 28°C, он отправляет специальную команду контроллеру.
   - Контроллер света отключает все лампы и отвечает на запрос.

4. **Кольцевой буфер (Ring Buffer) для логов**

   Задание предназначено для оптимизации работы с памятью (как в embedded-устройствах).
   Требования:

   - Сервер хранит последние 100 сообщений в кольцевом буфере.
   - При переполнении старые данные перезаписываются.
   - Логирование в файл происходит только при явном сигнале (например, раз в 10 сек).

5. **Эмуляция потери пакетов**

   Задание предназначено для отработки устойчивости к сбоям в сети.
   Реализовать режим работы сервера или/и контроллера, когда объект игнорирует каждый n-й пакет (эмуляция потери).

6. **Визуализация данных через Python-скрипт**

   Задание введено с целью научиться взаимодействию между Embedded-системой и внешними инструментами.
   Требования:

   - Визуализация статистики изменения температуры в реальном времени
   - Визуализация статистики использования света в помещении в реальном времени

Проверка и критерии оценки

Критерии	Балл
Корректная работа UDP/TCP	3
Нет утечек памяти	2
Обработка ошибок сети	2
Читаемый код + документация	2
Допзадания	+ 1-3

Рекомендуемые инструменты

- **Отладка**:
  - `Wireshark` для анализа сетевого трафика.
  - `nc` (netcat) для ручной отправки команд.
- **Тестирование**:
  - Скрипт на Bash, который запускает N датчиков.