



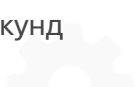
## Академия Бэкенда, 1 год, 2023, Языковая часть (Python)

### 1 задание

Ограничение времениОграничение памяти

5 секунд

256 МБ



### Предисловие

Вам предлагается на выполнение задание на моделирование "Умного дома". Время на выполнение этого задания - 96 часов. Не обращайте внимание на то, что на странице задачи таймер отсчитывает 24 часа!

Выполнить его вы должны на том языке программирования, на направление по которому планируете поступать (Scala, Python, Java, Go). Если вы пишете на python, java, go, вам доступна только стандартная библиотека соответствующих языков. Если вы пишете на scala, помимо стандартной библиотеки разрешается использовать [sttp](https://sttp.softwaremill.com/en/stable/examples.html#use-the-simple-synchronous-client) (<https://sttp.softwaremill.com/en/stable/examples.html#use-the-simple-synchronous-client>).

В задании не потребуется знание каких-то языково-специфичных конструкций или особенностей стандартной библиотеки или API, за исключением работы с HTTP POST запросами на сервер. От вас потребуется знание основ программирования, таких как работа со строками, последовательностями байт и битами внутри байта, и базовых структур данных (массивов, списков).

Текст задания может сначала показаться большим и страшным, и чтобы справиться со сложностью вы

**08:17:47**

Выполнено: 0 из 1

Завершить

[Компиляторы и значения ошибок](#)[Как сдавать экзамен](#)

можете разделить выполнение задания на несколько этапов.

Научитесь подключаться к серверу и выполнять **HTTP POST** запросы на сервер, передавая ему уже готовые закодированные сообщения.

Научитесь кодировать и декодировать **URL-encoded padded base64** для отправки на сервер и приёма данных от сервера.

Научитесь кодировать и декодировать пакеты из бинарной формы в ваше внутреннее представление.

Научитесь выявлять структуру сети устройств с помощью **WHOISHERE**.

Научитесь управлять одной лампой и одним выключателем.

Реализуйте весь требуемый функционал.

Для вашего удобства вам предоставляется готовая (консольная) программа - модельный сервер, на котором вы можете отлаживаться. Кроме того эта программа умеет кодировать и декодировать пакеты. Для того, чтобы посмотреть доступные опции работы программы, воспользуйтесь опцией **-help**. Модельный сервер располагается здесь: <https://github.com/blackav/smart-home-binary>

## Условие задачи

Напишите программу, которая реализует функции хаба умного дома. В умном доме размещены *сенсоры*, то есть устройства, измеряющие параметры окружающей среды, *актуаторы*, то есть устройства, воздействующие на среду в доме, *таймер*, передающий показания текущего времени, и *хаб*, то есть устройство, которое управляет всеми остальными устройствами в умном доме.

Все устройства находятся в общей коммуникационной среде (сети). Информация по сети передаётся с помощью пакетов. Пакет может

отправляться одному устройству, либо всем устройствам одновременно (**broadcast**). Каждое устройство, включая хаб, имеет свой уникальный 14-битный адрес в сети. Коммуникационная среда ненадёжна, то есть пакеты могут в сети теряться либо портиться, но пакеты не дублируются, то есть ситуация, когда получатель получает один и тот же пакет несколько раз, невозможна.

Для моделирования функциональности хаба умного дома ваша программа должна получать данные из сети, и в качестве реакции на полученные данные отправлять данные в сеть. В рамках данной задачи получение данных из сети и отправка данных в сеть моделируется с помощью **HTTP POST** запроса на специальный сервер, который моделирует функционирование остальных устройств умного дома.

## Описание формата пакетов

Для описания формата данных, передаваемых по сети, используются следующие типы данных:

**byte** — беззнаковое 8-битное значение (октет).

**bytes** — массив байтов переменного размера, этот тип должен конкретизироваться в описании конкретных форматов пакетов в зависимости от типа пакета и типа устройства.

**string** — строка. Первый байт строки хранит ее длину, затем идут символы строки, в строке допустимы символы (байты) с кодами 32-126 (включительно).

**varuint** — беззнаковое целое число в формате ULEB128.

**[]T** — массив элементов типа T, первый байт памяти, отводимой под массив, это длина массива, далее следуют байты, кодирующие элементы массива.

**struct** — структура, состоящая из полей произвольного типа. Структура может иметь

переменный размер, поскольку поля структуры могут иметь переменный размер, например `varuint`, `string`, и т. п.

Каждый пакет, передаваемый по сети, описывается следующим образом:

```
type packet struct {  
    length byte  
    payload bytes  
    crc8 byte  
};
```

Где:

**length** — это размер поля **payload** в октетах (байтах);

**payload** — данные, передаваемые в пакете, конкретный формат данных для каждого типа пакета описывается ниже;

**crc8** — контрольная сумма поля **payload**, вычисленная по алгоритму cyclic redundancy check 8 ([http://www.sunshine2k.de/articles/coding/crc/understanding\\_crc.html](http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html)).

Полезная нагрузка (поле **payload**) имеет следующую структуру:

```
type payload struct {  
    src varuint  
    dst varuint  
    serial varuint  
    dev_type byte  
    cmd byte  
    cmd_body bytes  
};
```

Где:

**src** - это 14-битный "адрес" устройства-отправителя;

**dst** - 14-битный “адрес” устройства-получателя, причем адреса 0x0000 и 0x3FFF (16383) зарезервированы. Адрес 0x3FFF означает “широковещательную” рассылку, то есть данные адресованы всем устройствам одновременно;

**serial** - это порядковый номер пакета, отправленного устройством, от момента его включения. **serial** нумеруется с 1;

**dev\_type** - это тип устройства, отправившего пакет;

**cmd** - это команда протокола;

**cmd\_body** - формат которого зависит от команды протокола.

Тип устройства **dev\_type** и команда **cmd** в совокупности определяют данные, которые передаются в **cmd\_body**, как будет описано далее.

## Описание команд протокола (cmd)

**0x01 - WHOISHERE** — отправляется устройством, желающим обнаружить своих соседей в сети. Адрес рассылки **dst** должен быть широковещательным 0x3FFF. Поле **cmd\_body** описывает характеристики самого устройства в виде структуры:

```
type device struct {
    dev_name string
    dev_props bytes
};
```

Содержимое **dev\_props** определяется в зависимости от типа устройства.

**0x02 - IAMHERE** — отправляется устройством, получившим команду **WHOISHERE**, и содержит информацию о самом устройстве в поле **cmd\_body**. Команда **IAMHERE** отправляется строго в ответ на **WHOISHERE**. Команда отправляется на широковещательный адрес.

**0x03 - GETSTATUS** — отправляется хабом какому-либо устройству для чтения состояния устройства. Если устройство не поддерживает команду GETSTATUS (например, таймер), команда игнорируется.

**0x04 - STATUS** — отправляется устройством хабу и как ответ на запросы **GETSTATUS**, **SETSTATUS**, и самостоятельно при изменении состояния устройства. Например, переключатель отправляет сообщение **STATUS** в момент переключения. В этом случае адресом получателя устанавливается устройство, которое последнее отправило данному устройству команду **GETSTATUS**. Если такой команды ещё не поступало, сообщение **STATUS** не отправляется никому.

**0x05 - SETSTATUS** — отправляется хабом какому-либо устройству, чтобы устройство изменило свое состояние, например, чтобы включилась лампа. Если устройство не поддерживает изменение состояния (например, таймер), команда игнорируется.

**0x06 - TICK** — тик таймера, отправляется таймером. Периодичность отправления не гарантируется, но если на некоторый момент времени запланировано событие, то срабатывание события должно наступать, когда время, передаваемое в сообщении **TICK** становится больше или равно запланированному. Поле `cmd_body` содержит следующие данные:

```
type timer_cmd_body struct {  
    timestamp varuint  
};
```

Поле **timestamp** содержит текущее время в миллисекундах от 1970-01-01T00:00:00Z (java timestamp). Таким образом точность измерения времени составляет 1 мс, но тики таймера могут

отправляться значительно реже, например, раз в 100 мс. Обратите внимание, что таймер показывает модельное время, а не реальное астрономическое время. Работа вашей программы, моделирующей хаб умного дома, должна быть привязана к модельному, а не к астрономическому времени.

Командами **GETSTATUS**, **STATUS**, **SETSTATUS** обмениваются два устройства, одно из которых - ваш хаб. Широковещательные адреса не допускаются.

Устройство должно ответить на широковещательный запрос **WHOISHERE** не позднее, чем через 300 мс. Устройство должно ответить на запросы **GETSTATUS** или **SETSTATUS**, адресованные данному устройству, не позднее, чем через 300 мс. Команда **TICK** не требует ответа.

## Типы устройств

**0x01 - SmartHub** — это устройство, которое моделирует ваша программа, оно единственное устройство этого типа в сети;

**0x02 - EnvSensor** — датчик характеристик окружающей среды (температура, влажность, освещенность, загрязнение воздуха);

**0x03 - Switch** — переключатель;

**0x04 - Lamp** — лампа;

**0x05 - Socket** — розетка;

**0x06 - Clock** — часы, которые широковещательно рассылают сообщения **TICK**. Часы гарантированно присутствуют в сети и только в одном экземпляре.

## Обработка команд в зависимости от типа устройства

**SmartHub**

**WHOISHERE, IAMHERE** — `dev_props` пуст

## EnvSensor

**WHOISHERE, IAMHERE** — `dev_props` имеет следующую структуру:

```
type env_sensor_props struct {  
    sensors byte  
    triggers [] struct {  
        op byte  
        value varuint  
        name string  
    }  
};
```

Поле **sensors** содержит битовую маску поддерживаемых сенсоров, где значения каждого бита обозначают следующее:

**0x1** — имеется датчик температуры (сенсор 0);

**0x2** — имеется датчик влажности (сенсор 1);

**0x4** — имеется датчик освещенности (сенсор 2);

**0x8** — имеется датчик загрязнения воздуха (сенсор 3).

Поле **triggers** — массив пороговых значений сенсоров для срабатывания. Здесь операция **op** имеет следующий формат:

**Бит 0** (младший) — включить или выключить устройство;

**Бит 1** — сравнивать по условию меньше (0) или больше (1);

**Биты 2-3** — тип сенсора

**value** — это пороговое значение сенсора;

**name** — имя устройства, которое



должно быть включено или  
выключено.

**Обратите внимание**, что **EnvSensor** не управляет устройствами непосредственно, функция управления переложена на хаб (вашу программу). Когда значение какого-либо сенсора переходит через пороговое, **EnvSensor** сам отправляет сообщение **STATUS** по адресу устройства, которое последним запрашивало **GETSTATUS** у **EnvSensor**. Таким образом нет необходимости в постоянном опросе значений сенсоров со стороны хаба.

**STATUS** — поле **cmd\_body** содержит показания всех поддерживаемых устройством сенсоров как массив целых чисел.

```
type env_sensor_status_cmd_body struct {  
    values []varuint  
};
```

Показания всегда идут в порядке **температура-влажность-освещенность-загрязнение воздуха**. Например, если сенсор поддерживает датчики температуры и освещенности, то сначала всегда передаётся температура, а затем освещенность. Физические измерения передаются в следующей форме:

**температура** измеряется в 0.1 Кельвина, то есть температура -273.1 кодируется значением 0, а температура 0 по Цельсию кодируется значением 2731.

*Примечание: температура в Кельвинах и в Цельсиях связана следующим соотношением:  $C = K + 273.1$ , где  $C$  — температура в Цельсиях, а  $K$  — температура в Кельвинах;*

**относительная влажность** измеряется в промилле, то есть принимает значения от

0 до 1000;

**освещенность** измеряется в десятых долях люкса;

**загрязнение воздуха** измеряется в десятых долях показателя PM2.5.

## Switch

**WHOISHERE, IAMHERE** — **dev\_props**

является массивом строк. Каждая строка — это имя (**dev\_name**) устройства, которое подключено к данному выключателю. Включение выключателя должно включать все устройства, а выключение должно выключать. За это отвечает хаб (ваша программа).

**STATUS** — поле **cmd\_body** имеет размер 1 байт и содержит значение 0, если переключатель находится в положении **OFF**, и 1, если переключатель находится в положении **ON**.

## Lamp и Socket

**WHOISHERE, IAMHERE** — массив **dev\_props** пустой (имеет нулевой размер);

**STATUS** — поле **cmd\_body** имеет размер 1 байт и содержит значение 0, если переключатель находится в положении **OFF**, и 1, если переключатель находится в положении **ON**;

**SETSTATUS** — поле **cmd\_body** должно иметь размер 1 байт и содержать 0 для выключения устройства и 1 для включения устройства.

## Работа вашей программы

Вашей программе в аргументе командной строки передается **URL**, который нужно использовать для обмена данными с сетью, и 14-битный адрес

вашего устройства-хаба в сети в шестнадцатеричном виде. Для отправки данных в сеть необходимо выполнить **POST**-запрос передав в теле запроса выдаваемые в сеть пакеты данных в виде **URL-encoded Base64** строки. В ответ **POST**-запрос вернет пакеты данных, принятые из сети. Каждую порцию данных таким образом можно прочитать только один раз. В случае успешного чтения очередной порции данных сервер вернет код ответа **HTTP 200**. В случае, когда данных больше нет, сервер вернет код ответа **HTTP 204**. При получении кода 204 ваша программа должна завершить свою работу с кодом завершения 0.

Пример запуска вашей программы:

```
solution http://localhost:12183 ef0
```

Если при взаимодействии с сервером возникла какая-либо ошибка на уровне сети или протокола **HTTP**, или сервер вернул код ответа, отличный от 200 или 204, ваша программа должна завершить выполнение с кодом завершения 99.

Каждая порция входных данных закодирована с помощью **URL-encoded unpadding Base64**, причем все пробельные символы (пробелы, табуляции, переводы строк) являются незначимыми и должны быть проигнорированы. Если порция данных закодирована в base64 некорректно, она должна быть пропущена, и ваша программа должна перейти к обработке следующей порции данных.

Ваша программа должна моделировать работу хаба по следующему алгоритму:

В начале работы хаб собирает информацию о сети с помощью запроса **WHOISHERE** и запрашивает начальное состояние всех устройств

Далее хаб ожидает сообщений от сенсоров, и

для каждого сообщения от сенсора  
выполняются действия, прописанные в  
**dev\_props** сенсора

Все подключенные устройства кроме хаба и таймера в процессе работы могут быть отключены. Отключенное устройство перестает отвечать на сообщения, адресованные этому устройству. Устройства могут быть включены в процессе работы, и могут появиться новые устройства. Повторно или вновь включенное устройство посылает в сеть запрос **WHOISHERE**, на который хаб должен ответить. Конфигурация устройства (**dev\_props**) может отличаться от конфигурации, которая была в прошлый раз, в этом случае хаб должен начать работать с новой конфигурацией.

Если хаб отправил команду какому-то конкретному устройству и не получил от него ответа в течение 300мс, это устройство считается выключенным и на него больше не отправляется команд, а сообщения от него игнорируются, пока это устройство само не объявит о своем включении с помощью запроса **WHOISHERE**.

## Замечание

Ваша программа должна полностью размещаться в одном файле исходного кода. Ваша программа **не должна** ничего считывать со стандартного потока ввода или выводить на стандартный поток вывода, кроме того закрывать эти потоки нельзя. Весь обмен данными ведется с помощью **HTTP POST** запросов.

## Примеры данных

Ввод

Вывод

Решение

Язык

Python3 3.10.6



Решение

1

[Отправить](#) [Можно пересдать еще раз](#)  
[Оферта](#) [Сведения об образовательной организации](#)

## Предыдущие решения

По вопросам пишите на почту [edu@tinkoff.ru](mailto:edu@tinkoff.ru)

---

Здесь будет список решений

© 2023, АНО ДПО «Тинькофф Образование»