Оглавление

Введение

Целью данной работы является создание функциональной и удобной для конечных пользователей и администраторов системы контроля доступа в учебные помещения.

Под «системой контроля доступа» подразумевается программно-аппаратный комплекс, способный определять личность пользователя по изображению его лица (аутентификация) и, исходя из наличия «разрешения на доступ» в конкретное помещение, открывать или не открывать вход в помещение (авторизация); управляемый, настраиваемый и контролируемый администраторами удаленно.

Программная компонента системы контроля доступа

Разделение на отдельные процессы

СКД обслуживается несколькими программными процессами, выполняемыми параллельно на одном одноплатном компьютере. Следующие подсистемы были выделены в отдельные процессы:

1. Процесс биометрической аутентификации (распознавание лиц),
2. Процесс интерфейса администрирования (телеграм-бот),
3. Процесс физического пользовательского интерфейса (обработка нажатий физических кнопок, открытия двери, сигналы пользователям и т. д),
4. Главный процесс (основная логика СКД),
5. База данных.

Главный процесс является только «процессом-сервером», то есть, обрабатывает события, получаемые от других процессов. Процесс физического пользовательского интерфейса является как генератором событий для главного процесса, так и сам ждет команд от него (открыть дверь, проморгать что-то светодиодом и т. п.). Процессы аутентификации и интерфейса администрирования являются только «клиентами».

Взаимодействие процессов

На данный момент все программные модули СКД представлены скриптами, написанными на языке *Python*, поэтому для межпроцессного взаимодействия будут использованы объекты класса *Pipe*, предоставляемого модулем *multiprocessing. connection*. Этот класс являются оберткой над стандартной абстракцией операционных систем — сокетами. В связи с этим, в будущем, даже при изменении программного стека какой-либо подсистемы, протокол взаимодействия редактировать не придется.

Взаимодействие будет происходить путем передачи сообщений формата JSON через отдельные «трубы» для каждого клиентского процесса. Главный процесс в бесконечном цикле проверяет их на наличие входящих сообщений.

В будущем при масштабировании системы возможен безпроблемный переход на протокол HTTP с сохранением протокола взаимодействия.

Процесс аутентификации – главный процесс

Процесс аутентификации отправляет в главный процесс информацию об обнаруженном пользователе и помещении, в которое пытаются попасть. Если был обнаружен известный пользователь (вероятность совпадения > 0.9), то параметр *«is_known»* будет содержать значение *true*, также, в объекте будет присутствовать параметр *«user_id»*, содержащий идентификатор распознаного пользователя. Если был обнаружен новый пользователь или вероятность совпадения < 0.9, то параметр *«is_known»* будет содержать значение *false*, будет присутствовать массив похожих пользователей *«similar_ids»* (вероятность совпадения > 0.6) и путь к файлу с изображением лица пользователя. На рисунке 1 расположен пример событий, отправленных процессом аутентификации.

```
// Обнаружен неизвестный пользователь
{
  "room_id": 1231 // идентификатор помещения
  "is_known": false,
  "img_path": "/home/user/AccessControlSystem/FaceImages/temp/21.png",
  "similar_ids": [
    {
      "id": 45, // идентификатор похожего пользователя
      "prob": 0.67 // вероятность совпадения
    },
    {
      "id": 89,
      "prob": 0.76
    }
  ]
}

// Обнаружен известный пользователь
{
  "room_id": 329
  "is_known": true,
  "user_id": 32
}
```

Рисунок 1 – Пример событий, полученных от процесса аутентификации

В ответ главный процесс в случае известного пользователя отправляет сообщение с ответом об успешной обработке события. В случае события, вызванного обнаружением лица неизвестного пользователя, встает вопрос, нужно ли сохранить его модель. Главный процесс после обращения к администратору отвечает сообщением, нужно ли сохранить или обновить существующую модель (рисунок 3).

```
// Пользователь уже известен, отработано успешно
{
  "status": "OK"
}

// Пользователь уже известен, нужно обновить модель лица
{
  "status": "OK",
  "is_known": true,
  "user_id": 213
}

// Это новый пользователь, нужно сохранить его модель с идентификатором 213
{
  "status": "OK",
  "is_known": false,
  "user_id": 213
}

// Это новый пользователь, сохранять его модель не требуется
{
  "status": "OK",
  "is_known": false,
  "user_id": -1
}
```

Рисунок 2 – Ответ при успешной обработке события обнаружения неизвестного пользователя

Процесс интерфейса администрирования – главный процесс

Процесс администрирования может вызвать следующие события-команды (рисунок 4):

- открыть дверь указанного помещения прямо сейчас,
- запланировать открытие двери указанного помещения в указанное время,
- запретить открытие двери указанного помещения на определенное до определенного времени.

```
// Открыть дверь помещения с идентификатором 42 прямо сейчас
{
  "command": "open_now",
  "params": {
    "room_id": 42
  }
}

// Запланировать открытие двери помещения с указанным
// идентификатором на указанное время
{
  "command": "open_at_time",
  "params": {
    "room_id": 42,
    "date": "31.12.2020",
    "time": "23:59"
  }
}

// Запретить открытие двери помещения с указанным идентификатором
// до указанного времени
{
  "command": "forbid_open_until_time",
  "params": {
    "room_id": 42,
    "date": "31.12.2020",
    "time": "23:59"
  }
}
```

Рисунок 4 – События, получаемые от процесса интерфейса администрирования

На подобные события главный процесс отвечает статусом выполнения операции планирования и массивом запланированных открытий для заданного помещения.

Процесс физического пользовательского интерфейса – главный процесс

Процесс физического пользовательского интерфейса может вызвать событие определенного нажатия на кнопку, передав тип и количество нажатий. На рисунке 5 приведен пример вызываемого события и ответа главного процесса.

Главный процесс отвечает «OK». Также, в ответе может присутствовать команда физическому интерфейсу (см. пункт «Главный процесс – процесс физического пользовательского интерфейса»).

```
// События отправляемое в главный процесс при двух коротких нажатиях на кнопку
{
  "press_type": "short" // short|long|hold - быстрое|длинное нажатие|удержание
  "click_quantity": 2
}

// Ответ на событие
{
  "status": "OK",
  "command": {
    ...
  }
}
```

Рисунок 5 – Событие, получаемое от процесса интерфейса физического взаимодействия

Главный процесс – процесс физического пользовательского интерфейса

Команда главного процесса содержит информацию об обратной связи, воспроизводимой пользователю подключенными светодиодами и пьезодинамиком (рисунок 6).

```
// Команда процессу физического пользовательского интерфейса,
// вызываемая самостоятельно, либо являющаяся частью ответа на событие
{
  "open": true,
  "lights": {
    "led_1": {
      "times_number": 2, // Количество морганий
      "mode": "short" // short - короткие моргания, long - длинные моргания
    },
    "led_2": {
      "times_number": 1,
      "mode": "long"
    },
  },
  "buzzer": {
    ...
    // Возможна воспроизведение какого-либо сигнала аудиосигнала
    // при помощи пьезодинамика
  }
}
```

Рисунок 6 – Команда процессу физического пользовательского интерфейса

Структура программного кода

Работая над столь крупным по кодовой базе проектом, я стремился разделить его на смысловые и программные модули, отражая это в структуре директорий, пакетов и файлов скриптов. Рисунок, отражающий текущую структуру проекта расположен в приложении Б.

Сущности базы данных

Была доработана созданная во время выполнения летней практики схема программных сущностей базы данных: были добавлены новые сущности, в частности, сущность задачи и модели лица пользователя. В дальнейшем будет происходить уточнение содержания сущностей. На рисунке 7 представлена существующая на данный момент времени «рабочая» схема программных сущностей.

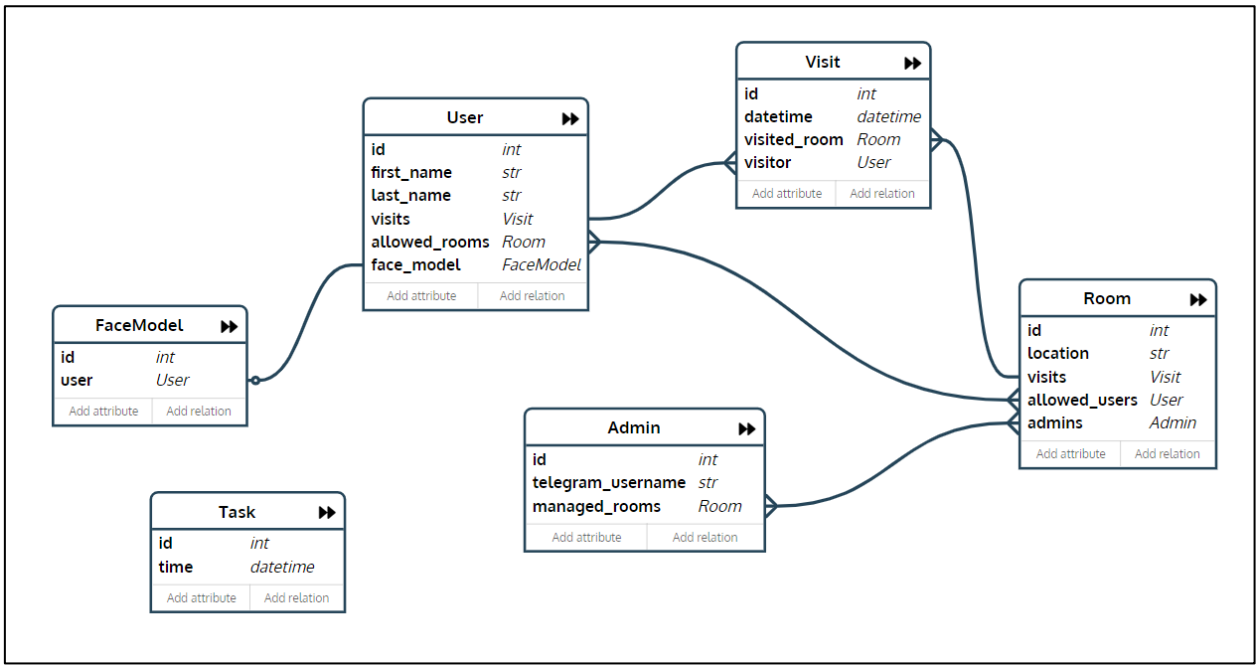

Рисунок 7 – Схема программных сущностей базы данных

Реализация главного процесса

Главный процесс будет представлен бесконечным циклом, в котором поочередно проверяются

- все соединения с другими процессами на наличие новых входных сообщений,
- наступление запланированные событий (задач),
- возвращение ответов от вызванных сопрограмм.

На рисунке 8 представлен листинг фрагмента скрипта главного процесса, содержащий имплементацию описанного выше бесконечного цикла (полное содержание кода главного процесса располагается в приложении А)

```
while True:
    # Проверяем и выполняем своевременные задачи раз в ON_TIME_TASKS_CHECK_PERIOD секунд
    if int(time()) % ON_TIME_TASKS_CHECK_PERIOD == 0:
        perform_on_time_tasks(on_time_tasks, datetime.now())

    # Выполняем «обычные» задачи
    if not common_tasks.empty():
        for _ in range(PER_ITERATION_TASKS_NUMBER):
            perform_common_task(common_tasks)

    if face_id.poll():
        # сообщение от подсистемы аутентификации
        request = face_id.recv()

        if response := face_id_handler(request, tasks):
            # если обработчик вернул None, это означает,
            # что ответ еще не готов и будет отпарвлен позже
            face_id.send(response)

    if hardware.poll():
        # сообщение от подсистемы физического пользовательского интерфейса
        request = hardware.recv()
        response = hardware_handler(request, tasks)

    if telebot.poll():
        # сообщение от подсистемы интерфейса администрирования
        request = telebot.recv()
        response = telebot_handler(request, tasks)
```

Рисунок 8 – Листинг. Главный цикл главного процесса программы СКД

Реализация обработчиков подсистем

Обработчик подсистемы аутентификации

Обработчик системы аутентификации интересен тем, что для него возможен случай, когда нужно выполнить операцию, занимающую значительный отрезок времени (опрос администраторов). Данная операция выполняется асинхронно в отдельном потоке: ответ подсистемы аутентификации в этом случае отправляется не сразу из главного цикла, а по завершению вызванной в отдельном потоке функции. В листинге на рисунке 9 представлен фрагмент файла *subsystem_handlers.py*, отвечающий за обработку событий (сообщений) от подсистемы биометрического аутентификации.

Обработчики остальных указанных подсистем будут разработаны и имплементированы в коде в будущем.

```python
def face_id_handler(request: dict or list,
                    tasks: Tasks,
                    subsystem: Subsystem) -> dict or list or None:
    """Обработчки событий (сообщенией), вызываемых подсистемой аутентификации"""
    room_id = request['room_id']
    is_known = request['is_known']

    if is_known:
        # Обнаружен известный пользователь
        user_id = request['user_id']
        hardware_commands.open_door(user_id, room_id)
        return {'status': 'OK'}
        # TODO: Добавить обработку исключений при невозможности открытия помещения
    else:
        # Обнаружен неизвестный пользователь
        similar_ids = {user['id']: user['prob'] for user in request['similar_ids']}
        img_path = request['img_path']

        # Совершаем опрос администраторов в отдельном потоке
        Thread(target=_face_id_ask_admins,
               args=(room_id, img_path, similar_ids, subsystem.send)).start()
        return None
        # Ответ будет отправлен позже


def _face_id_ask_admins(room_id: int, img_path: str,
                        similar_ids: dict[int, float],
                        send_func: Callable) -> None:
    """
    Функция, выполняемая в отдельном потоке,
    т.к. опрос администарторов занимает определенное количество времяни
    """
    result = telebot_commands.ask_admins(room_id, img_path, similar_ids)
    answer = {'status': 'OK', 'is_known': result.is_known, 'user_id': result.user_id}
    send_func(answer)
```

Рисунок 9 – Листинг. Обработчик событий (сообщений) от подсистемы биометрической аутентификации

Реализации подсистемы интерфейса администрирования

Подсистема интерфейса администрирования по большей своей части была разработана при выполнении летней практики. На данный момент ее нужно интегрировать как подсистему. Это будет сделано в будущем.

Реализация подсистемы биометрического аутентификации

Основанная на созданной другими студентами программе биометрической аутентификации подсистема будет интегрирована в программу СКД в будущем.