Требования

Суть задания заключается в создании части транзакционной системы, которая отвечает за получение структурированных данных, их обработку и сохранение результатов обработки в текстовых файлах. Под транзакцией далее подразумевается процесс получения данных пользователя, их обработки и создания текстового файла.

Требования к такой системе описываются аббревиатурой ACID:

1) Атомарность – транзакция не должна быть зафиксирована частично. Либо она выполняет целиком, либо никак. То есть, в случае любого сбоя не должно быть частично сформированных или пустых файлов.
2) Согласованность – транзакция фиксирует только допустимые результаты. То есть, в случае успешного завершения транзакции актуальный файл должен называться “username.txt”, а уже не актуальные файлы – “old_ username_yyyy-mm-ddThh:mm.txt”.
3) Изолированность – необходимо предусмотреть возможность распараллелить работу системы, чтобы параллельные транзакции не влияли друг на друга.
4) Стойкость – в случае сбоя, результаты всех корректно завершенных транзакций должны сохраниться, а результаты незавершенных (или некорректно завершенных) транзакций не должны влиять на состояние системы.

Также в ТЗ указана необходимость эффективной работы программы и минимально возможное количество записей на диск.

Проектирование

Перед написанием самого кода необходимо спроектировать работу системы, которая удовлетворяет всем этим свойствам, продумав крайние случаи и возможные сбои.

Для соблюдения атомарности предлагается сначала записывать новый отчет в “черновой” файл, с названием “new_username.txt”. Таким образом, если сбой произойдет во время записи файла, то никакого противоречия не случится: частично сформированный файл не будет называться так, как должен называться актуальный отчет.

Это можно реализовать двумя способами: более эффективным и более аккуратным. Первый способ – создавать черновые файлы прямо в директории tasks, тогда для актуализации нового отчета достаточно переименовать старый и новый отчеты. Второй способ – создавать черновые файлы в директории program_data, но тогда после записи файла необходимо будет перед переименованием перенести его в директорию tasks. Для данной небольшой

системы подходит и первый способ, но для лучшей масштабируемости не стоит засорять директорию с актуальными отчетами, поэтому выбран второй способ.

Процесс работы программы разделяется на 3 этапа: проверка корректности последнего завершения работы, получение данных и выполнение транзакций.

Транзакция выполняется в 6 этапов:

1) Получение данных транзакции
2) Создание текста для записи
3) Создание в program_data чернового файла с этим текстом
4) Переименование старого файла (если он есть)
5) Переименование и перенос чернового файла в директорию tasks

Предотвращение и обработка ошибок

1) В задании не указан userId. Предлагается преобразовать данные о заданиях в формат класса defaultdict, установив в качестве значения по умолчанию число -1. Тогда эти задания не достанутся никому из пользователей и будут просто проигнорированы.
2) В информации о пользователе не хватает каких-то данных, необходимых для формирования файла. Предлагается отслеживать такую ситуацию при создании текста для этого пользователя. Если данных не хватает, просто не создавать такой файл, игнорировать его. При необходимости можно вместо игнорирования описывать эту проблему в текстовом файле log.txt.
3) Некорректное завершение скрипта. Если сбой произошел во время чтения файла, то все хорошо: контекстный менеджер with автоматически закроет файл. Из всех возможных моментов сбоя только один может стать проблемой: когда старый файл уже переименован, а новый – еще не перенесен из program_data в tasks. Единственная возможная обработка такого случая: это добавить в скрипт проверку корректности последнего завершения работы. То есть, каждый файл program_data (если они есть) проверяется на то, заполнился ли он до конца (в нем должно быть либо три строки, если задач нет, либо количество задач + 7 строк). Если он заполнен, то переносится в program_data, старый файл переименовывается. Иначе выводится сообщение о его удалении.
4) Проблемы с загрузкой данных (проблемы со связью или с работой сервиса). Единственная возможная обработка такой ошибки – вывести сообщение о ней и прекратить работу скрипта.
5) Неоднократное обновление информации о задачах пользователя в течении минуты. Если трижды за минуту поступит информация о задачах пользователя, то не получится сохранить старый отчет с правильным

форматом. Предлагается в таком случае сохранять менее старый отчет с припиской “v2”. Если информация о задачах пользователя обновляется чаще трех раз в минуту, возникает ошибка. В зависимости от бизнес-процессов и теоретической возможности такой ситуации может быть разное решение. Если ситуация с таким частым обновлением информации невозможна, то проблемы нет. Если такая ситуация возможна, то лучшее решение – указывать в названии файла еще и секунды.

Описание некоторых решений в коде

Основной файл с кодом, script.py, построен таким образом, чтобы его можно было использовать, как библиотечный код. Несмотря на усложнение кода в самом файле, так его гораздо проще использовать в другом скрипте.

Функция main принимает в качестве необязательных параметров функцию получения данных и функцию обработки данных, а также два булевых значения: проводить ли проверку корректности последнего завершения программы, и показывать ли время выполнения скрипта. Поэтому для переопределения способа обработки данных не нужно переписывать функцию в исходном файле, достаточно использовать в другом файле любую стороннюю функцию.

Это реализовано в файле parallel.py, в котором просто переопределена функция параллельной обработки данных, и для этого понадобилось только импортировать функцию транзакции и саму функцию main.

Код выделен в функции по смысловому принципу, чтобы между разными действиями было легко что-то добавить, или чтобы можно было их подменить. Для более вариативной задачи можно легко преобразовать этот код в паттерн «Шаблонный метод».

Программа проверялась на Ubuntu 18.04.