Тестовое задание на позицию Серверный разработчик C++

Краткое описание

Есть логи, полученные от игровых серверов. Необходимо их распарсить и агрегировать данные.

Развернутое описание

Имеем директорию, в которой лежат N лог-файлов (например, 10 штук), названных fileM.log .
Нумерация - с единицы.
В каждом файле логов лежит “много” записей (например, 1кк).
Каждая запись находится на новой строке и представляет собой действие игрока:

{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
{"ts_fact" : 1489494303, "fact_name": "fact1", "actor_id": 111222, "props": {"prop1": 11, "prop2": 22,.. "prop10": 1010}}
...

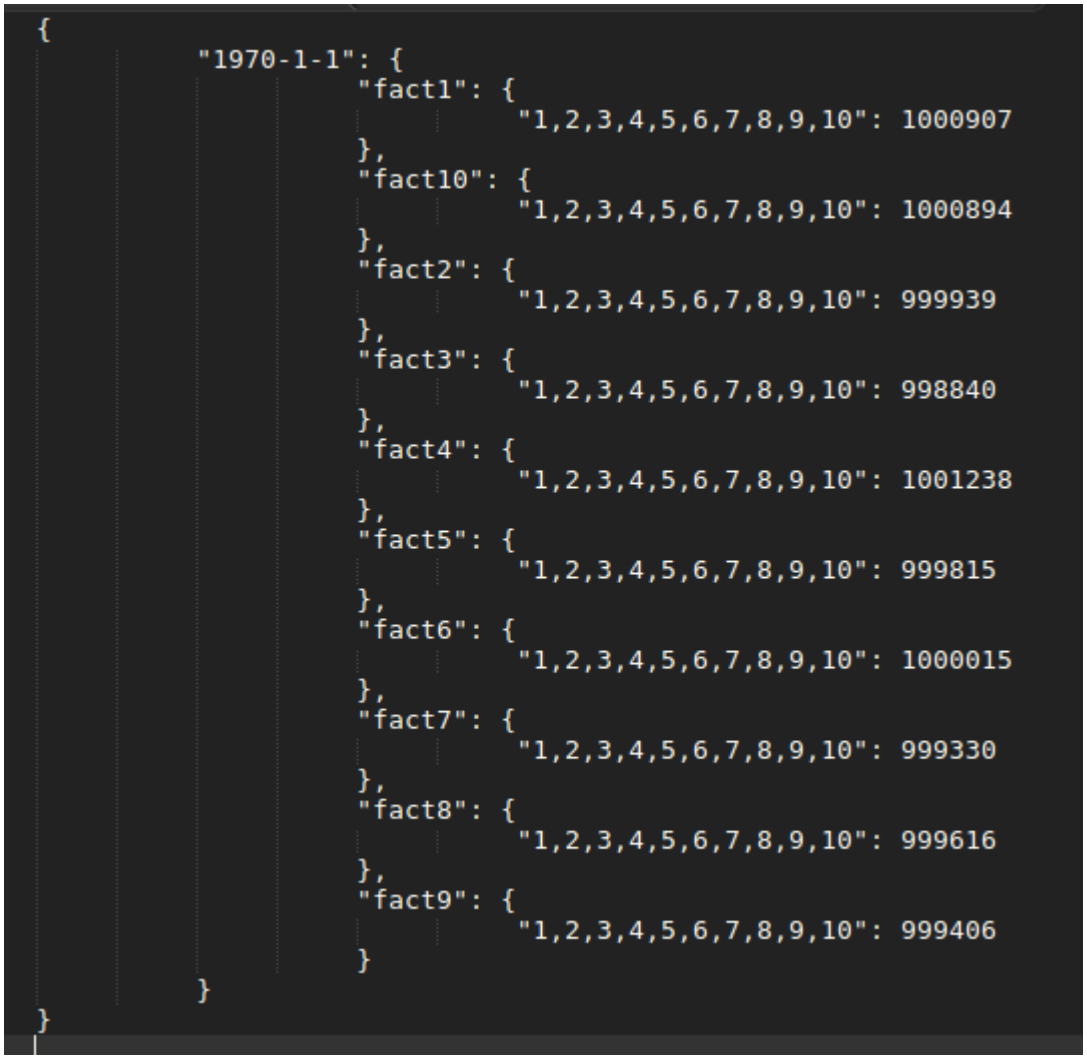
Значения полей:

- ts_fact: когда сделано действие, timestamp, uint32
- fact_name: имя действия, str
- actor_id: автор действия, uint32
- props: свойства действия, prop1 - prop10

Приложение читает все логи в директории и заполняет данными такую структуру:

- для каждых UTC+0-суток:
    - для каждого fact_name:
        - для каждой комбинации (prop1, .., prop10)
            - хранится количество таких фактов в логах

После обработки логов структура пишется в файл agr.txt. Формат - на ваше усмотрение, например такой:



```
{
        "1970-1-1": {
                "fact1": {
                        "1,2,3,4,5,6,7,8,9,10": 1000907
                },
                "fact10": {
                        "1,2,3,4,5,6,7,8,9,10": 1000894
                },
                "fact2": {
                        "1,2,3,4,5,6,7,8,9,10": 999939
                },
                "fact3": {
                        "1,2,3,4,5,6,7,8,9,10": 998840
                },
                "fact4": {
                        "1,2,3,4,5,6,7,8,9,10": 1001238
                },
                "fact5": {
                        "1,2,3,4,5,6,7,8,9,10": 999815
                },
                "fact6": {
                        "1,2,3,4,5,6,7,8,9,10": 1000015
                },
                "fact7": {
                        "1,2,3,4,5,6,7,8,9,10": 999330
                },
                "fact8": {
                        "1,2,3,4,5,6,7,8,9,10": 999616
                },
                "fact9": {
                        "1,2,3,4,5,6,7,8,9,10": 999406
                }
        }
}
```

Для ускорения предлагается сделать приложение многопоточным. Например, параллельно обрабатывать несколько файлов. Реализация на ваше усмотрение.

Приложение принимает следующие параметры.

- Полный путь к директории с логами.
- Число логов в директории.
- Если приложение многопоточное: параметры многопоточности.

Критерии качества в порядке важности.

1. Структурированный код. Без элементов языка C.
2. Скорость исполнения.
3. Требуемая память.

Тесты будут проводиться на общем количестве записей от 10кк.
Реализация не должна быть наивной (например требовать десятки гигабайт оперативной памяти при исполнении).

Приложение должно собираться на gcc >= 6.2.0, cmake >= 3.5.1 и работать на Ubuntu 18.04 . Если перед сборкой необходимо установить какие-то пакеты или выполнить какие либо действия - опишите это в файле README в каталоге с проектом.

Для парсинга json используем готовую библиотеку. Свой парсинг писать **не надо**.
Другие сторонние библиотеки лучше не использовать.
Boost использовать нельзя.