Задание 1. Обработчик данных POSIX

В этом задании вам необходимо разработать обработчик задач. Задачи которые нужно обрабатывать это подсчет чисел Фибоначчи, вычисление степени числа, bubble сортировка, а так-же stop сообщение которое говорит что обработчик должен завершиться после того как доделает всю работу. Так-же все потоки должны уметь корректно завершаться с использование pthread_cancel

Формат входящих сообщений

```
typedef enum
{
    FIBONACCI,
    POW,
    BUBBLE_SORT_UINT64,
    STOP
} EType;

typedef struct
{
    uint8_t Type;
    uint64_t Size;
    uint8_t *Data;
} TMessage;
```

Псевдокод для записи и чтения:

Псевдокод чтения/записи

```
CreateFib(uint64_t N)
{
    TMessage message;
    message.Type = EType::FIBONACCI;
    message.Size = sizeof(N);
    message.Data = calloc(message.Size);
    Write(message.Data, &N, message.Size);
    return message;
}

Write(File file, TMessage message)
{
    Write(file, &message.Type, sizeof(uint8_t));
    Write(file, &message.Size, sizeof(uint64_t));
    Write(file, message.Data, message.Size);
```

```
}

Read(File file)
{
    TMessage message;
    Read(file, &message.Type, sizeof(uint8_t));
    Read(file, &message.Size, sizeof(uin64_t));
    message.Data = calloc(message.Size);
    Read(file, message.Data, message.Size);
    return message;
}
```

Для TMessage можно использовать C++ библиотеку Google protobuf.

Здесь важно не забыть сделать структуру с выравниванием 1 иначе могут возникнуть проблемы при разработке генератор.

На стандартный ввод программе подается поток данных из TMessage которые нужно обрабатывать, а результат работы записывать в специальный файл в отдельном потоке Writer. Считывать входной поток необходимо в отдельном потоке, назовем его Reader.

Что касается обработки данных, то здесь необходимо реализовать три стратегии обработки данных. Стратегии задаются с помощью опции утилите –strategy. Стратегии обработки следующие:

1. **per_thread** - поток Reader создает на каждую задачу отдельный поток в котором происходят вычисления, после этого поток который выполнял вычисления отправляет результат потоку Writer который записывает результат в файл

2. **per_task** - На каждый тип задачи необходимо создать свой поток. Поток Reader отправляет задачу в поток который обрабатывает только такой тип задач. После выполнения задачи поток отправляет результат потоку Writer который записывает результат в файл

3. **thread_pool** - поток Reader отправляет задачу в thread pool. Количство потоков в thread pool должно задаваться с помощью опции утилите –count-threads. После выполнения задачи thread pool отправляет результат потоку Writer который записывает результат в файл

При выполнении работы необходимо использовать язык C и библиотеку pthread для работы с потоками и их синхронизации. Так-же необходимо добавить метрики времени работы и загрузки системы в перцентилях. Метрики необходимо репортить каждые n миллисекунд в отдельный файл. n должно задаваться как параметр к запуску утилиты.

Задание 2. Генератор данных

На любом языке программирования необходимо разработать генератор данных с различными распределениями, которые будут указываться с помощью опции –mode, а параметры для распределения передаваться с помощью опции –param. Придумайте три или более распределений, которые будут генерироваться вашим инструментом.

Задание 3. Тестирование

1. Для разработанных программ в заданиях 1,2 необходимо написать тесты
2. Покрытие тестами должно превышать 70%. Для измерения покрытия необходимо использовать специализированные инструменты. Например clang-coverage. Результаты выдачи инструментов необходимо приложить в отчет
3. Необходимо провести тестирование производительности разработанной инфраструктуры и на графиках показать при каких входных данных, какой алгоритм лучше выбрать
4. При проведении измерений нужно использовать k-ые порядковые статистики. Например 1,25,50,70,90,99,99.9,99.99,99.999. Измерения необходим провести для каждого этапа. Время ожидания в очередях и время на каждый этап (чтение, запись, исполнение)

Отчет

1. Статистика демонстрирующая покрытие вашего кода тестами
2. Графики производительности различных алгоритмов обработки данных. Должно получиться как минимум три графика, а в лучшем случае больше
3. На оценку 3 достаточно реализовать обработку: per_thread

   На оценку 4 достаточно реализовать обработку: per_thread, per_task

   На оценку 5 необходимо реализовать все алгоритмы обработки данных