Тестовое задание 2

Описание задания

Есть источник кадров (например, карта захвата) и несколько клиентов (например запись в несколько файлов, отображение на экране и т.п.). Требуется сделать объект (сплиттер), который буферизирует входящие кадры (чтобы не было пропусков, если какой-то клиент не успел вовремя забрать кадр), и позволяет клиентам забирать кадры из этого объекта в том же порядке, в котором они пришли, с минимальной задержкой и без потери кадров.

Задача:

- Реализовать C++ класс который решает эту задачу.
- Написать unit-тесты для этого класса.

Абстракция:

- Класс, который реализует очередь с одним входом и несколькими динамическими выходами (клиентами).

Требования:

- В любой момент клиент может быть добавлен или удалён.
- Клиент получает только те данные, которые были отправлены в сплиттер после подключения клиента.
- Очередь не должна блокироваться, если кто-то из клиентов не забирает данные (т.е. работа одного клиента не должна оказывать существенного влияния на работу других клиентов).
- Задержка (количество буферов) для каждого клиента не должна превышать заданного значения.
- Класс должен быть потоково безопасным - т.е. отправлять, забирать данные, удалять, добавлять клиентов, закрывать объект и т.п. можно из любого потока.
- Возвращаемые значения методов должны позволять идентифицировать тип ошибки.

Интерфейс

```
// Создание объекта сплиттера - задаётся максимальное количество буферов в очереди, и максимальное количество клиентов.
std::shared_ptr<ISplitter> SplitterCreate(IN size_t _zMaxBuffers, IN size_t _zMaxClients);

// ISplitter интерфейс

bool   ISplitter::SplitterInfoGet(OUT size_t* _pzMaxBuffers, OUT size_t* _pzMaxClients);

// Кладём данные в очередь. Если какой-то клиент не успел ещё забрать свои данные, и количество буферов (задержка) для него больше максимального значения, то ждём пока не освободятся буфера (клиент заберет данные) не более _nTimeOutMsec (**). Если по истечению
```

времени данные так и не забраны, то удаляем старые данные для этого клиента, добавляем новые (по принципу FIFO) (*). Возвращаем код ошибки, который дает понять что один или несколько клиентов “пропустили” свои данные.

```
int32_t ISplitter::SplitterPut(IN const std::shared_ptr<std::vector<uint8_t>>& _pVecPut, IN int32_t _nTimeOutMsec);

// Сбрасываем все буфера, прерываем все ожидания. (после вызова допустима дальнейшая работа)
int32_t ISplitter::SplitterFlush();

// Добавляем нового клиента - возвращаем уникальный идентификатор клиента.
bool   ISplitter::SplitterClientAdd(OUT uint32_t* _punClientID);

// Удаляем клиента по идентификатору, если клиент находиться в процессе ожидания буфера, то прерываем ожидание.
bool   ISplitter::SplitterClientRemove(IN uint32_t _unClientID);

// Перечисление клиентов, для каждого клиента возвращаем его идентификатор, количество буферов в очереди (задержку) для этого клиента а также количество отброшенных буферов.
bool   ISplitter::SplitterClientGetCount(OUT size_t* _pnCount);
bool   ISplitter::SplitterClientGetByIndex(IN size_t _zIndex, OUT uint32_t* _punClientID, OUT size_t* _pzLatency, OUT size_t* _pzDropped);

// По идентификатору клиента возвращаем задержку
bool   ISplitter::SplitterClientGetById(IN uint32_t _unClientID, OUT size_t* _pzLatency, OUT size_t* _pzDropped);

// По идентификатору клиента запрашиваем данные, если данных пока нет, то ожидаем не более _nTimeOutMsec (**) пока не будут добавлены новые данные, в случае превышения времени ожидания - возвращаем ошибку.
int32_t ISplitter::SplitterGet(IN uint32_t _nClientID, OUT std::shared_ptr<std::vector<uint8_t>>& _pVecGet, IN int32_t _nTimeOutMsec);

// Закрытие объекта сплиттера - все ожидания должны быть прерваны все вызовы возвращают соответствующую ошибку. Все клиенты удалены. (после вызова допустимо добавление новых клиентов и дальнейшая работа)
void   ISplitter::SplitterClose();
```

(*)Пусть количество буферов (максимальная задержка) равно 2. Мы положили в сплиттер буфера 1,2,3,4,5,6,7,8,9,10 (с интервалом в 100 msec, максимальное время ожидания в **SplitterPut** - 50 msec).
- Клиент 1 сразу получил 1,2,3 а затем 500 msec “спал”, то после того как проснется он должен получить 7,8,9,10 (4, 5, 6 будут потеряны)
- Остальные клиенты должны в это время получить все буфера 1,2,3,4,5,6,7,8,9,10 с максимальной задержкой 50 msec (для буферов 6, 7, 8,).

(**) Отрицательное значение _nTimeOutMsec означает что ждём или пока не появятся/освободяться данные или до вызова Flush/Close