POSIX

Ваша задача - реализовать классический паттерн producer-consumer с небольшими дополнительными условиями. Программа должна состоять из 3+N потоков:

1. главный
2. producer
3. interruptor
4. N потоков consumer

На стандартный ввод программе подается строка - список чисел, разделённых пробелом (читать можно до конца ввода). Длина списка чисел не задаётся - считывание происходит до перевода каретки.

- Задача producer-потока - получить на вход список чисел, и по очереди использовать каждое значение из этого списка для обновления переменной разделяемой между потоками
- Задача consumer-потоков отреагировать на уведомление от producer и набирать сумму полученных значений. Также этот поток должен защититься от попыток потока-interruptor его остановить. Дополнительные условия:
    1. Функция, исполняющая код этого потока consumer_routine, должна принимать указатель на объект/переменную, из которого будет читать обновления
    2. После суммирования переменной поток должен заснуть на случайное количество миллисекунд, верхний предел будет передан на вход приложения (0 миллисекунд также должно корректно обрабатываться). Вовремя сна поток не должен мешать другим потокам consumer выполнять свои задачи, если они есть
    3. Потоки consumer не должны дублировать вычисления друг с другом одних и тех же значений
    4. В качестве возвращаемого значения поток должен вернуть свою частичную посчитанную сумму
- Задача потока-interruptor проста: пока происходит процесс обновления значений, он должен постоянно пытаться остановить случайный поток consumer (вычисление случайного потока происходит перед каждой попыткой остановки). Как только поток producer произвел последнее обновление, этот поток завершается.

Функция run_threads должна запускать все потоки, дожидаться их выполнения, и возвращать результат общего суммирования.

Для обеспечения межпоточного взаимодействия допускается использование только pthread API. На вход приложения передаётся 2 аргумента при старте именно в такой последовательности:

1. Число потоков consumer
2. Верхний предел сна consumer в миллисекундах

Так-же необходимо реализовать поддержку ключа –debug, при использовании которого каждый consumer-поток будет выводить пару (tid, psum), где tid реализуется с помощью функции get_tid(), а psum это сумма которую посчитал поток. Вывод значений psum происходит при каждом изменении.

Функция get_tid() возвращает идентификатор потока. Идентификатор потока это не проста pthread_self(), а уникальное для каждого потока число в диапозоне от 1 .. 3+N. Значение этого числа предполагается хранить в TLS. Память под сохраняемое значение должно выделяться в heap, а указатель на него в TLS. Так-же функция get_tid должна быть самодостаточной (для использования ее в другом проекте должно быть достаточно только скопировать get_tid и использовать)

В поток вывода должно попадать только результирующее значение, по умолчанию никакой отладочной или запросной информации выводиться не должно.

Шаблон

```c
#include <pthread.h>

void* producer_routine(void* arg) {
  // Wait for consumer to start

  // Read data, loop through each value and update the value,
  // notify consumer, wait for consumer to process
}

void* consumer_routine(void* arg) {
  // notify about start
  // for every update issued by producer, read the value and add to sum
  // return pointer to result (for particular consumer)
}

void* consumer_interruptor_routine(void* arg) {
  // wait for consumers to start

```

```
19    // interrupt random consumer while producer is running
20  }
21
22  int run_threads() {
23    // start N threads and wait until they're done
24    // return aggregated sum of values
25
26    return 0;
27  }
28
29  int get_tid() {
30    // 1 to 3+N thread ID
31
32    return 0;
33  }
34
35  int main() {
36      std::cout << run_threads() << std::endl;
37      return 0;
38  }
```