

Posix threads

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

В файл, чтение которого уже реализовано в шаблоне кода в блокирующем режиме, пишется список чисел, разделённых пробелом (читать можно до переноса строки). 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.
- Завершение приложения происходит посылке сигнала SIGTERM, обработку которого нужно также добавить. В случае, если в этот момент поток, читающий данные с файла, находится в режиме блокирующего чтения, - он также должен корректно завершиться.

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 (`thread id`, `partial summ`). Вывод значений `psum`

происходит при каждом изменении.

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

```
#include <pthread.h>

void read_values() {
    FILE* file = fopen("in.txt");
    ... read();
}

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

    // interrupt random consumer while producer is running
}

int run_threads() {
    // start N threads and wait until they're done
    // return aggregated sum of values

    return 0;
}

int main() {
    std::cout << run_threads() << std::endl;
    return 0;
}
```

From: <http://wiki.osll.ru/> - Open Source & Linux Lab

Permanent link: http://wiki.osll.ru/doku.php/courses:high_performance_computing:producer_consumer?rev=1633207613

Last update: 2021/10/02 23:46



