Точилина Полина БПИ199.

Микропроект 2, Вариант 25

Задача о читателях и писателях.

Базу данных разделяют два типа процессов – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в непротиворечивом состоянии (т.е. отношения между данными имеют смысл). Каждая отдельная транзакция переводит БД из одного непротиворечивого состояния в другое. Для предотвращения взаимного влияния транзакций процесс-писатель должен иметь исключительный доступ к БД. Если к БД не обращается ни один из процессов-писателей, то выполнять транзакции могут одновременно сколько угодно читателей. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры.

Для более удобного представления программы, условие реализовано на ситуации буквальных писателей и читателей некоторых книг.

Используемая модель

Источник информации

В модели есть два типа потоков: 1 - читатель, 2 - писатель. Одновременный доступ получают сколько угодно читателей, но у писателя доступ эксклюзивный.

Для взаимодействия между потоками использованы семафоры.

Код программы

includes

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <algorithm>
#include <string>
#include <vector>
#include <thread>
#include <string>
```

Глобальные переменные

```
std::vector<std::pair<std::string, std::string>> db; // книги
int started = 0, book_cnt, write_cnt, read_cnt; // количество начатых книг
std::vector<pthread_t> threads; // потоки

sem_t out; // доступ к cout
sem_t rc; // доступ к readCount
sem_t start; // доступ к started
sem_t dbAccess; // доступ к базе
int readCount = 0; //количество активных читателей
```

Считывание целочисленных значений

Переписан stoi, чтобы не было исключений.

Написан метод, который считывает числа в интервале и сравнивает stoi с начальной строкой, чтобы не пропускать строки, которые просто начинаются с числа.

```
bool stoi(const std::string& str, int* p_value, std::size_t* pos = 0, int base = 10) {
    // обертка чтобы не было исключений
    try {
        *p_value = std::stoi(str, pos, base);
        return true;
    }
    catch (const std::exception& e)
    {
        return false;
    }
}

int readInt(int min = INT_MIN, int max = INT_MAX) {
    int result;
    bool correctInput = false;
    std::string tmp;
    std::cin >> tmp;
    correctInput = stoi(tmp, &result);

    while (!correctInput || std::to_string(result) != tmp || result < min || result > max) {
        std::cout << "Некорректный ввод. Попробуйте ещё раз\n";
        std::cin >> tmp;
        correctInput = stoi(tmp, &result);
    }

    return result;
}
```

Функция потоков-читателей

Инициализация

Инициализируется рандомное время видимой работы потока, номер потока и его имя для вывода.

```
void* reader(void* args) {
    // время работы в миллисекундах [3000, 5000]
    int time = 3000 + std::rand() % 2000;
    int number = *((int*)args);
    std::string name = "Читатель" + std::to_string(number);
```

Цикл

Поток заходит в цикл с выходом при завершении написания всех книг потоками-авторами.

Если до этого на данном участке не было других потоков-читателей, то поток блокирует доступ потокам-писателям к базе данных.

```
while (true) {
    if (db.size() == 0)
        continue;
    sem_wait(&rc);
    // увеличиваем число активных читателей
    readCount++;
    // если это первый читатель, то закрываем вход писателям
    if (readCount == 1)
        sem_wait(&dbAccess);
    sem_post(&rc);

    // выходим
    if (db.size() == book_cnt) {
        // выводим информацию о выходе
        sem_wait(&out);
        std::cout << name << " узнал, что время выбора книг закончено" << std::endl;
        sem_post(&out);
        return nullptr;
    }
```

Выполнение задания

Поток получает рандомную запись в базе данных на чтение.

Если на данном участке не осталось потоков-читателей, то поток открывает доступ потокам-писателям к базе данных.

Поток спит, симулируя чтение книги.

```
        // выводим информацию о заходе
        sem_wait(&out);
        std::cout << name << " выбирает книгу" << std::endl;
        sem_post(&out);

        // получаем книгу
        auto bookName = db[std::rand() % db.size()];

        // освобождаем семафор
        sem_wait(&rc);
        readCount--;
        // если это последний читатель, открываем вход писателям
        if (readCount == 0)
            sem_post(&dbAccess);
        sem_post(&rc);

        // выводим информацию о старте
        sem_wait(&out);
        std::cout << name << " начал читать книгу " << bookName.second << " автора " << bookName.first << std::endl;
        sem_post(&out);

        // читаем книгу
        std::this_thread::sleep_for(std::chrono::milliseconds(time));

        // выводим информацию об окончании
        sem_wait(&out);
        std::cout << name << " закончил читать книгу " << bookName.second << " автора " << bookName.first << std::endl;
        sem_post(&out);
    }
}
```

Функция потоков-писателей

Инициализация

Инициализируется рандомное время видимой работы потока, номер потока и его имя для вывода.

```
void* writer(void* args) {
    // время работы в миллисекундах [3000, 5000]
    int time = 3000 + std::rand() % 2000;
    int number = *((int*)args);
    std::string name = "Автор" + std::to_string(number);
```

Цикл

Поток заходит в цикл с выходом при условии, что начато книг столько, сколько должно быть завершено. (Чтобы не написать лишних книг).

Поток засыпает, симулируя написание книги.

Поток добавляет новую запись в базу данных.

```
    while (true) {
        sem_wait(&start);
        // если все книги написаны (некоторые дописываются)
        if (started >= book_cnt) {
            sem_post(&start);
            return nullptr;
        }

        started++;
        // создаем название новой книги
        std::string bookname = "Книга" + std::to_string(started);
        sem_post(&start);

        // выводим информацию о старте
        sem_wait(&out);
        std::cout << name << " начал писать книгу: " << bookname << std::endl;
        sem_post(&out);

        // пишем книгу
        std::this_thread::sleep_for(std::chrono::milliseconds(time));

        // выводим информацию об окончании
        sem_wait(&out);
        std::cout << name << " закончил писать книгу: " << bookname << std::endl;
        sem_post(&out);

        // добавляем книгу в базу
        sem_wait(&dbAccess);
        db.push_back({ name, bookname });
        // выводим информацию о добавлении книги в базу
        sem_wait(&out);
        std::cout << name << " передал в издательство книгу: " << bookname << std::endl;
        // все книги изданы
        if (db.size() == book_cnt)
            std::cout << "Все книги изданы!" << std::endl;
        sem_post(&out);
        sem_post(&dbAccess);
    }
}
```

Функция main

Приветствие

Инициализируется локаль, рандом и семафоры.

Выводятся сообщения пользователю и происходит ввод начальных значений.

```
int main()
{
    setlocale(LC_ALL, "rus");
    srand(time(0));

    std::cout << "Мы рады приветствовать вас в нашей программе \"Почувствуй себя издателем\"!\n"
        << "Сегодня вам предстоит издать некоторое количество книг.\n"
        << "Сколько книг вы хотели бы издать? ( > 0 )\n";
    book_cnt = readInt(1);
    std::cout << "Сколько у вас есть писателей? ( > 0 )\n";
    write_cnt = readInt(1);;
    std::cout << "Сколько у вас значится читателей? ( > 0 )\n";
    read_cnt = readInt(1);;
    std::cout << "Скорость написания и прочтения книг будет сгенерировна автоматически\n"
        "Писатель пишет книгу за [3, 5] секунд\nЧитатель читает книгу за [3, 5] секунд\n\n";

    sem_init(&rc, 0, 1);
    sem_init(&out, 0, 1);
    sem_init(&start, 0, 1);
    sem_init(&dbAccess, 0, 1);

    std::vector<int*> names;
```

Основная часть

Создаются потоки, симулирующие писателей.

Создаются потоки, симулирующие читателей.

Ожидается окончание работы всех потоков.

Уничтожаются семафоры.

```
    for (int i = 0; i < write_cnt; i++) {
        pthread_t thread;
        names.push_back(new int(i + 1));
        pthread_create(&thread, NULL, writer, (void*)names[i]);
        threads.push_back(std::move(thread));
    }

    std::vector<int*> names2;

    for (int i = 0; i < read_cnt; i++) {
        pthread_t thread;
        names2.push_back(new int(i + 1));
        pthread_create(&thread, NULL, reader, (void*)names2[i]);
        threads.push_back(std::move(thread));
    }

    for (int i = 0; i < threads.size(); i++)
        pthread_join(threads[i], NULL);

    sem_destroy(&rc);
    sem_destroy(&out);
    sem_destroy(&start);
    sem_destroy(&dbAccess);

    std::cout << "Симуляция издательства окончена.";
}
```

Тестирование программы

Программа приветствует пользователя

```
Мы рады приветствовать вас в нашей программе "Почувствуй себя издателем"!
Сегодня вам предстоит издать некоторое количество книг.
Сколько книг вы хотели бы издать? ( > 0 )
```

Протестируем проверку ввода целого числа

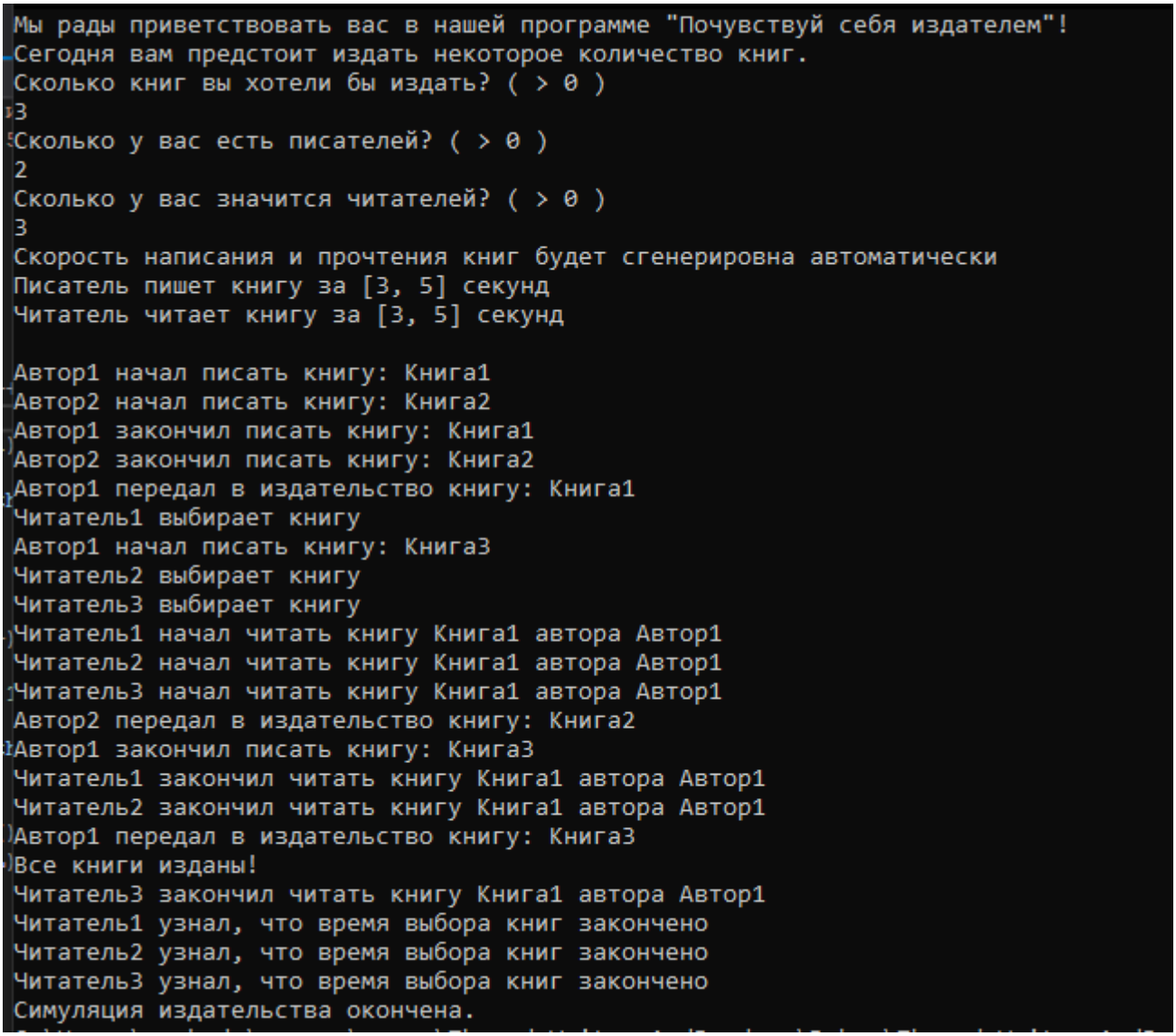
```
Сколько книг вы хотели бы издать? ( > 0 )
fg
Некорректный ввод. Попробуйте ещё раз
3,5
Некорректный ввод. Попробуйте ещё раз
10000000000000000000000000000000000000000
Некорректный ввод. Попробуйте ещё раз
3
Сколько у вас есть писателей? ( > 0 )
```

Протестируем программу при вводе одинакого количесвта писателей и книг

```
Мы рады приветствовать вас в нашей программе "Почувствуй себя издателем"!
Сегодня вам предстоит издать некоторое количество книг.
Сколько книг вы хотели бы издать? ( > 0 )
2
Сколько у вас есть писателей? ( > 0 )
2
Сколько у вас значится читателей? ( > 0 )
2
Скорость написания и прочтения книг будет сгенерировна автоматически
Писатель пишет книгу за [3, 5] секунд
Читатель читает книгу за [3, 5] секунд

Автор1 начал писать книгу: Книга1
Автор2 начал писать книгу: Книга2
Автор1 закончил писать книгу: Книга1
Автор2 закончил писать книгу: Книга2
Автор1 передал в издательство книгу: Книга1
Читатель1 выбирает книгу
Читатель2 выбирает книгу
Читатель1 начал читать книгу Книга1 автора Автор1
Читатель2 начал читать книгу Книга1 автора Автор1
Автор2 передал в издательство книгу: Книга2
Все книги изданы!
Читатель1 закончил читать книгу Книга1 автора Автор1
Читатель2 закончил читать книгу Книга1 автора Автор1
Читатель1 узнал, что время выбора книг закончено
Читатель2 узнал, что время выбора книг закончено
Симуляция издательства окончена.
```

Протестируем программу на стандартных данных



Протестируем программу при вводе писателей больше чем книг

Лишние книги не были начаты, лишние писатели сразу вышли из цикла.

```
Мы рады приветствовать вас в нашей программе "Почувствуй себя издателем"!
Сегодня вам предстоит издать некоторое количество книг.
Сколько книг вы хотели бы издать? ( > 0 )
3
Сколько у вас есть писателей? ( > 0 )
2
Сколько у вас значится читателей? ( > 0 )
3
Скорость написания и прочтения книг будет сгенерировна автоматически
Писатель пишет книгу за [3, 5] секунд
Читатель читает книгу за [3, 5] секунд

Автор1 начал писать книгу: Книга1
Автор2 начал писать книгу: Книга2
Автор1 закончил писать книгу: Книга1
Автор2 закончил писать книгу: Книга2
Автор1 передал в издательство книгу: Книга1
Читатель1 выбирает книгу
Автор1 начал писать книгу: Книга3
Читатель2 выбирает книгу
Читатель3 выбирает книгу
Читатель1 начал читать книгу Книга1 автора Автор1
Читатель2 начал читать книгу Книга1 автора Автор1
Читатель3 начал читать книгу Книга1 автора Автор1
Автор2 передал в издательство книгу: Книга2
Автор1 закончил писать книгу: Книга3
Читатель1 закончил читать книгу Книга1 автора Автор1
Читатель2 закончил читать книгу Книга1 автора Автор1
Автор1 передал в издательство книгу: Книга3
Все книги изданы!
Читатель3 закончил читать книгу Книга1 автора Автор1
Читатель1 узнал, что время выбора книг закончено
Читатель2 узнал, что время выбора книг закончено
Читатель3 узнал, что время выбора книг закончено
Симуляция издательства окончена.
```