ACS. Individual Homework 4

ACS
Александр Васюков | БПИ235 (239)

Вариант 34

`Задача о сельской библиотеке. В библиотеке имеется N книг, каждая из книг в одном экземпляре. M читателей регулярно заглядывают в библиотеку, выбирая для чтения от одной до трех книг и читая их некоторое количество дней. Если желаемой книги нет, то читатель, взяв существующие, дожидается от библиотекаря информации об ее появлении и приходит в библиотеку, чтобы специально забрать ее. Возможна ситуация, когда несколько читателей конкурируют из-за этой популярной книги. Создать многопоточное приложение, моделирующее заданный процесс.`

Решение претендует на 8 баллов

Файлы хранятся на GitHub по ссылке: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/ACS/Homework/IHW_4
Есть 6 файлов:

1. `main.cpp` - основная программа.
2. `config.txt` - файл с данными о количестве книг и читателей.
3. Файлы с результатами тестов `outputN.txt`, где N = [1, 2, 3, 4].

Логика

Есть M читателей и N книг.
Каждый читатель по очереди подходит к библиотекарю, чтобы взять от 1 до 3 книг.
Если книга свободна, то читатель берёт её и читает некоторое время.
Если книга занята, то читатель становится в очередь за ней и ждёт, пока она станет свободной, после чего берёт читать.

Модель параллельного выполнения

1. **Общие потоки:**
   - Поток библиотекаря:
     - Обрабатывает запросы от читателей из очереди `librarian_queue`.
     - Выбираются случайные книги и добавляются к читателю в очередь книг `books_queue`.
     - Запускает поток для обработки книги, если он еще не запущен.
   - Потоки для книг:
     - Создаются для каждой книги, если к ней поступил запрос.
     - Обрабатывают читателей из очереди для своей книги `books_queue[id]`.
     - Ждут, пока в очереди книги не появится запрос или библиотека не закроется.
2. **Очереди:**
   - Очередь читателей `librarian_queue`:
     - Ожидает запросов читателей, которые библиотекарь обрабатывает.
   - Очереди для книг `books_queue`:
     - Для каждой книги поддерживается своя очередь запросов от читателей.
3. **Синхронизация:**
   - Мьютекс `queueMutex`:
     - Защищает доступ к общим ресурсам: очередям, массиву `book_requested` и счетчику `all_requests`.
   - Условные переменные:
     - `librarianCond`: для уведомления библиотекаря о новых запросах.
     - `booksCond[i]`: для уведомления потока книги о новых запросах на чтение.
4. **Этапы выполнения:**
   - Добавление читателей:
     - В главном потоке читатели добавляются в очередь библиотекаря `librarian_queue`, сигнализируя об этом библиотекарю.
   - Обработка запросов библиотекарем:
     - Библиотекарь извлекает читателя из очереди и резервирует для него случайные книги.
     - Добавляет запросы в соответствующие очереди книг и создает поток книги, если он еще не активен.
   - Чтение книг:
     - Поток книги обрабатывает запросы из очереди своей книги.
     - Имитация чтения (задержка `usleep`), затем возврат книги и уменьшение счетчика запросов.
   - Закрытие библиотеки:
     - Главный поток ждет, пока все запросы не будут обработаны.

- Уведомляет библиотекаря и потоки книг о завершении работы, чтобы они завершились корректно.

5. **Многопоточность и конкуренция:**
   - Потоки библиотекаря и книг работают независимо, синхронизируясь через мьютексы и условные переменные.
   - Библиотекарь и потоки книг могут одновременно работать с разными очередями, не блокируя друг друга.
6. **Закрытие программы:**
   - После завершения всех запросов библиотека закрывается.
   - Потоки библиотекаря и книг корректно завершаются, освобождаются ресурсы `pthread_cond_destroy`.

Случайные числа

Используются рандомные числа для выбора количества книг (от 1 до 3), а также для длительности чтения книги от 1 до 10 секунд.

Пример работы программы

Тест 1

С использованием файла -f config.txt, в котором записаны есть строка: `3 5`.

```
/Users/alexvasyukov/Desktop/iFolder/Code/ACS/multithreading/cmake-build-debug/multithreading -f config.txt
Do you want to use your file? (Yes/No): Yes
Use data from file.
Input output file name: output1.txt
Library simulation started. Data will be written to output1.txt
============================
Librarian is seeing reader 1
Librarian gave to reader 1 book 0
Librarian gave to reader 1 book 4
Librarian is seeing reader 2
Reader 1 is reading book 0
Reader 1 is reading book 4
Librarian gave to reader 2 book 1
Librarian gave to reader 2 book 2
Librarian gave to reader 2 book 3
Librarian is seeing reader 3
Reader 2 is reading book 3
Reader 2 is reading book 1
Reader 2 is reading book 2
Reader 2 returned book 2
Reader 1 returned book 0
Librarian gave to reader 3 book 1
Librarian gave to reader 3 book 4
Reader 2 returned book 3
Reader 1 returned book 4
Reader 3 is reading book 4
Reader 2 returned book 1
Reader 3 is reading book 1
Reader 3 returned book 1
Reader 3 returned book 4
Library is now closed.

Process finished with exit code 0
```

Тест 2

С использованием файла -f config.txt, в котором записаны есть строка: `5 3`.

```
/Users/alexvasyukov/Desktop/iFolder/Code/ACS/multithreading/cmake-build-debug/multithreading -f config.txt
Do you want to use your file? (Yes/No): Yes
Use data from file.
Input output file name: output2.txt
Library simulation started. Data will be written to output2.txt
============================
Librarian is seeing reader 1
Librarian gave to reader 1 book 0
Librarian gave to reader 1 book 1
Librarian is seeing reader 2
Reader 1 is reading book 1
Reader 1 is reading book 0
Librarian gave to reader 2 book 0
Librarian gave to reader 2 book 1
Librarian is seeing reader 3
Librarian gave to reader 3 book 0
Librarian gave to reader 3 book 1
Librarian is seeing reader 4
Reader 1 returned book 0
Reader 2 is reading book 0
Librarian gave to reader 4 book 0
Librarian gave to reader 4 book 2
Librarian is seeing reader 5
```

```
 Reader 4 is reading book 2
 Librarian gave to reader 5 book 1
 Librarian gave to reader 5 book 2
 Reader 1 returned book 1
 Reader 2 is reading book 1
 Reader 4 returned book 2
 Reader 5 is reading book 2
 Reader 2 returned book 1
 Reader 3 is reading book 1
 Reader 2 returned book 0
 Reader 3 is reading book 0
 Reader 5 returned book 2
 Reader 3 returned book 1
 Reader 5 is reading book 1
 Reader 3 returned book 0
 Reader 4 is reading book 0
 Reader 5 returned book 1
 Reader 4 returned book 0
 Library is now closed.

 Process finished with exit code 0
```

Тест 3

С использованием файла -f config.txt, в котором записаны есть строка: `2 2`.

```
/Users/alexvasyukov/Desktop/iFolder/Code/ACS/multithreading/cmake-build-debug/multithreading -f config.txt
Do you want to use your file? (Yes/No): Yes
Use data from file.
Input output file name: output3.txt
Library simulation started. Data will be written to output3.txt
============================
Librarian is seeing reader 1
Librarian gave to reader 1 book 0
Librarian gave to reader 1 book 1
Librarian is seeing reader 2
Reader 1 is reading book 0
Reader 1 is reading book 1
Librarian gave to reader 2 book 0
Librarian gave to reader 2 book 1
Reader 1 returned book 1
Reader 2 is reading book 1
Reader 2 returned book 1
Reader 1 returned book 0
Reader 2 is reading book 0
Reader 2 returned book 0
Library is now closed.

Process finished with exit code 0
```

Тест 4

Автоматические данные: количество книг - 5000, количество читателей - 100.

```
/Users/alexvasyukov/Desktop/iFolder/Code/ACS/multithreading/cmake-build-debug/multithreading -f config.txt
Do you want to use your file? (Yes/No): No
Use automatic data.
Input output file name: output4.txt
Library simulation started. Data will be written to output4.txt
============================
Librarian is seeing reader 1
Librarian gave to reader 1 book 1212
Librarian gave to reader 1 book 3920
Librarian gave to reader 1 book 4031
Librarian is seeing reader 2
Reader 1 is reading book 4031
Reader 1 is reading book 3920
Reader 1 is reading book 1212
Reader 1 returned book 4031
Reader 1 returned book 1212
Librarian gave to reader 2 book 1576
Librarian gave to reader 2 book 2290
Librarian gave to reader 2 book 4300
Librarian is seeing reader 3
Reader 2 is reading book 4300
Reader 2 is reading book 1576
Reader 2 is reading book 2290
```

Более подробные данные в файле `output4.txt`.

Выполненные условия:

На 4-5

1. Соблюдены общие требования к отчету.
2. В отчете приведен сценарий, описывающий одновременное поведение представленных в условии задания сущностей в терминах предметной области.
3. Описана модель параллельных вычислений, используемая при разработке многопоточной программы.
4. Описаны входные данные программы, включающие вариативные диапазоны, возможные при многократных запусках.
5. Реализовано консольное приложение, решающее поставленную задачу с использованием одного варианта изученных синхропримитивов.

6. Ввод данных в приложение реализован с консоли во время выполнения программы.
7. Для используемых генераторов случайных чисел описаны их диапазоны и то, как интерпретируются данные этих генераторов.
8. Вывод программы информативный, отражающий все ключевые протекающие в ней события в терминах предметной области.
9. В программе присутствуют комментарии, поясняющие выполняемые действия и описание используемых объектов и переменных.
10. Результаты работы программы представлены в отчете

На 6-7

1. В отчете подробно описан обобщенный алгоритм, используемый при реализации программы исходного словесного сценария. В котором показано, как на программу отображается каждый из субъектов предметной области.
2. В программу добавлена генерация случайных данных в допустимых диапазонах.
3. Реализован ввод исходных данных из командной строки при запуске программы вместо ввода параметров с консоли во время выполнения программы.
4. Результаты изменений отражены в отчете.

На 8

1. В программу, наряду с выводом в консоль, добавлен вывод результатов в файл. Имя файла для вывода данных задается в командной строке как один из ее параметров.
2. Результаты работы программы должны выводиться на экран и записываться в файл.
3. Наряду с вводом исходных данных через командную строку добавлен альтернативный вариант их ввода из файла, который по сути играет роль конфигурационного файла. Имя этого файла задается в командной строке вместо параметров, которые в этом случае не вводятся. Управление вводом в командной строке осуществляется с использованием соответствующих ключей.
4. Приведено не менее трех вариантов входных и выходных файлов с различными исходным данными и результатами выполнения программы.
5. Ввод данных из командной строки расширен с учетом введенных изменений.
6. Результаты изменений отражены в отчете.

Код программы

```
#include <pthread.h>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <set>
#include <queue>
#include <unistd.h>

#define NUM_READERS_DEFAULT 100
#define NUM_BOOKS_DEFAULT 5000
```

```
std::ofstream out;
std::queue<int> librarian_queue;
std::queue<int> books_queue[NUM_BOOKS_DEFAULT];

pthread_mutex_t queueMutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t librarianCond = PTHREAD_COND_INITIALIZER;
pthread_cond_t booksCond[NUM_BOOKS_DEFAULT];

bool book_requested[NUM_BOOKS_DEFAULT] = {false};
pthread_t books_threads[NUM_BOOKS_DEFAULT];

bool library_open = true;
int all_requests = 0;
int NUM_READERS = NUM_READERS_DEFAULT;
int NUM_BOOKS = NUM_BOOKS_DEFAULT;

// Проверяет, пусты ли очереди всех книг
bool isAllQueuesEmpty() {
    for (int i = 0; i < NUM_BOOKS; ++i) {
        if (!books_queue[i].empty()) {
            return false;
        }
    }
    return true;
}
```

```
// Генерация случайной книги
int randomBook() {
    return rand() % NUM_BOOKS;
}

// Поток обработки книги
void* bookReading(void* arg) {
    int id = *(int*)arg;
    delete (int*)arg;

    while (true) {
        pthread_mutex_lock(&queueMutex);

        // Пока очередь книги пуста и библиотека открыта, ждать
        while (books_queue[id].empty() && library_open) {
            pthread_cond_wait(&booksCond[id], &queueMutex);
        }
        // Если очередь книги пуста и библиотека закрыта, то завершаем работу книги
        if (books_queue[id].empty() && !library_open) {
            pthread_mutex_unlock(&queueMutex);
            break;
        }

        // Получаем читателя книги
        int reader = books_queue[id].front();
        books_queue[id].pop();

        std::cout << "Reader " << reader << " is reading book " << id << '\n';
        out  << "Reader " << reader << " is reading book " << id << '\n';
        pthread_mutex_unlock(&queueMutex);

        // Симуляция чтения книги
        usleep(1000 + rand() % 10 * 1000);

        pthread_mutex_lock(&queueMutex);
        std::cout << "Reader " << reader << " returned book " << id << '\n';
        out  << "Reader " << reader << " returned book " << id << '\n';
        pthread_mutex_unlock(&queueMutex);

        pthread_mutex_lock(&queueMutex);
        // Удаляем 1 запрос на чтение
        --all_requests;
        pthread_mutex_unlock(&queueMutex);
    }
    return nullptr;
}

// Поток библиотекаря
void* librarian(void* arg) {
    while (true) {
        pthread_mutex_lock(&queueMutex);

        // Пока очередь к библиотекарю пуста и библиотека открыта, ждать
        while (librarian_queue.empty() && library_open) {
            pthread_cond_wait(&librarianCond, &queueMutex);
        }

        // Если очередь к библиотекарю пуста и библиотека закрыта, ждать
        if (librarian_queue.empty() && !library_open) {
            pthread_mutex_unlock(&queueMutex);
            break;
        }

        // Получаем читателя
        int reader = librarian_queue.front();
        librarian_queue.pop();

        std::cout << "Librarian is seeing reader " << reader << '\n';
        out << "Librarian is seeing reader " << reader << '\n';
        pthread_mutex_unlock(&queueMutex);

        // Имитируем работу библиотекаря с читателем
usleep(2000);

        // Читатель выбирает до 3 книг
        std::set<int> books_reserved;
```

```
        for (int i = 0; i < 3; ++i) {
            int book = randomBook();
            books_reserved.insert(book);
        }

        // Заполняем очередь к книге
for (int book : books_reserved) {
            pthread_mutex_lock(&queueMutex);
            books_queue[book].push(reader);
            pthread_cond_signal(&booksCond[book]);
            ++all_requests;

            if (!book_requested[book]) {
                book_requested[book] = true;
                int* book_id = new int(book);
                pthread_create(&books_threads[book], nullptr, bookReading, book_id);
            }

            std::cout << "Librarian gave to reader " << reader << " book " << book << '\n';
            out << "Librarian gave to reader " << reader << " book " << book << '\n';
            pthread_mutex_unlock(&queueMutex);
        }
    }
    return nullptr;
}

// Чтение конфигурационного файла
void readConfigFile(const std::string& filename, int& readers, int& books) {
    std::ifstream config(filename);
    if (!config.is_open()) {
        throw std::runtime_error("Cannot open configuration file");
    }

    config >> readers >> books;
    config.close();
}

int main(int argc, char* argv[]) {
    srand(time(nullptr));

    bool use_config = false;
    std::string choice;
    std::cout << "Do you want to use your file? (Yes/No): ";
    std::cin >> choice;
    if (choice == "Yes") {
        std::cout << "Use data from file.\n";
        use_config = true;
    } else {
        std::cout << "Use automatic data.\n";
    }

    if (use_config) {
        if (argc != 3) {
            std::cout << "You can use: ./miltithreading [-f configFile]\nData in the file: NUM_READERS
NUM_BOOKS\n";
            use_config = false;
        } else {
            // Обработка аргументов командной строки
            std::string output_filename;
            try {
                if (std::string(argv[1]) == "-f") {
                    std::string configFileName = argv[2];
                    readConfigFile(configFileName, NUM_READERS, NUM_BOOKS);
                }
            } catch (const std::runtime_error&) {
                use_config = false;
            }
        }
    }

    std::string filename;
    std::cout << "Input output file name: ";
    std::cin >> filename;

    out.open(filename, std::ios::out | std::ios::trunc);
    if (!out.is_open()) {
        std::cerr << "Failed to open " << filename << std::endl;
```

```
        return 1;
    }

    std::cout << "Library simulation started. Data will be written to " << filename << '\n';
    std::cout << "============================\n";

    pthread_t librarian_thread;
    int librarian_id = 1;

    for (int i = 0; i < NUM_BOOKS; ++i) {
        pthread_cond_init(&booksCond[i], nullptr);
    }

    pthread_create(&librarian_thread, nullptr, librarian, &librarian_id);

    for (int i = 1; i <= NUM_READERS; ++i) {
        pthread_mutex_lock(&queueMutex);
        librarian_queue.push(i);
        pthread_cond_signal(&librarianCond);
        pthread_mutex_unlock(&queueMutex);
        usleep(1000);
    }

    while (true) {
        pthread_mutex_lock(&queueMutex);
        if (all_requests == 0 && isAllQueuesEmpty()) {
            library_open = false;
            pthread_cond_broadcast(&librarianCond);
            for (int i = 0; i < NUM_BOOKS; ++i) {
                pthread_cond_broadcast(&booksCond[i]);
            }
            pthread_mutex_unlock(&queueMutex);
            break;
        }
        pthread_mutex_unlock(&queueMutex);
        usleep(500);
    }

    pthread_join(librarian_thread, nullptr);
    for (int i = 0; i < NUM_BOOKS; ++i) {
        if (book_requested[i]) {
            pthread_join(books_threads[i], nullptr);
        }
        pthread_cond_destroy(&booksCond[i]);
    }

    std::cout << "Library is now closed." << std::endl;
    out << "Library is now closed." << std::endl;

    out.close();
    return 0;
}
```