Отчет Растворов Сергей 13 вар

Отчет по многопоточному приложению "Моделирование работы магазина"

Сценарий предметной области

В данной задаче моделируется работа магазина с тремя отделами, каждый из которых обслуживается отдельным продавцом. Покупатели, заходящие в магазин, взаимодействуют с продавцами следующим образом:

1. **Покупатели**:
    - Покупатели заходят в магазин строго по очереди (покупатель 1 заходит первым, затем покупатель 2 и т. д.). Это гарантирует упорядоченность их появления.
    - После входа покупатель посещает один или несколько отделов в случайном порядке.
    - Если продавец в выбранном отделе занят, покупатель становится в очередь и ожидает своей очереди.
    - После обслуживания в отделе покупатель переходит к следующему отделу или покидает магазин, если все запланированные посещения завершены.
2. **Продавцы**:
    - Каждый отдел магазина обслуживает один продавец.
    - Продавец обслуживает одного покупателя за раз, после чего он становится доступным для следующего клиента.
    - Если в очереди отдела есть клиенты, продавец уведомляет первого клиента об освобождении.

Ситуации, возникающие при моделировании:

- Один покупатель может завершить обслуживание в отделе и перейти в следующий, пока другие покупатели только заходят в магазин или ждут в очереди.
- Покупатели могут обслуживаться в разных отделах одновременно, но очередность их входа в магазин строго контролируется.

Ролевое поведение:

- Покупатели представляют собой субъекты с независимыми планами посещения отделов.

- Продавцы действуют как пассивные участники, предоставляющие обслуживание по мере обращения клиентов.

Модель параллельных вычислений

Для реализации задачи используется **модель потоков POSIX (PThreads)**, которая обеспечивает поддержку многопоточности. Ключевые элементы модели:

1. **Основной поток**:
   - Создает потоки для покупателей и продавцов.
   - Управляет завершением работы программы, дожидаясь завершения всех потоков.
2. **Потоки покупателей**:
   - Каждый покупатель представлен отдельным потоком.
   - Поток покупателя последовательно посещает отделы, блокируя соответствующие мьютексы, чтобы избежать конкурентного доступа.
3. **Потоки продавцов**:
   - Каждый продавец представлен отдельным потоком, работающим с очередью покупателей для своего отдела.
   - Продавец использует условные переменные для уведомления клиентов в очереди.
4. **Синхронизация**:
   - **Мьютексы** используются для обеспечения атомарности операций при доступе к очередям, состояниям продавцов и выводу на консоль.
   - **Условные переменные** используются для управления очередями покупателей в отделах.

Входные данные программы

Входные данные вводятся через консоль при запуске программы:

1. **Количество покупателей**:
   - Пользователь задает число покупателей (целое число больше 0).
   - Это число определяет количество создаваемых потоков покупателей.
2. **Настройки логирования**:
   - Пользователь указывает, включить ли логирование (0 – нет, 1 – да).
   - Если логирование включено, пользователь задает имя файла для записи логов.

- При вводе только имени файла сохранение происходит в директорию к исполняемому файлу т.е. в `cmake-build-debug`

Диапазоны и их интерпретация:

- **Количество покупателей**: варьируется от 1 до максимально возможного значения, допустимого для текущей системы (зависит от ресурсов).
- **Очередность входа покупателей**: всегда строго упорядочена (1, 2, 3, ...).
- **Посещаемые отделы**: выбираются в случайном порядке, что делает последовательность обслуживания каждого покупателя уникальной.

Используемые генераторы случайных чисел

Для симуляции случайности (например, порядка посещения отделов и времени обработки) используются генераторы из стандартной библиотеки C++.

1. **Генератор случайных чисел** (`std::random_device` и `std::mt19937`):
   - Генератор `std::random_device` используется для начальной инициализации генератора псевдослучайных чисел `std::mt19937`.
   - `std::uniform_int_distribution` задает равномерное распределение для случайных чисел.
2. **Диапазоны генерации**:
   - **Порядок посещения отделов**: генерируется случайная перестановка массива `[0, 1, 2]`, представляющего индексы отделов.
   - **Время ожидания (RandomSleep)**:
     - Для покупателей: от 1000 до 3000 мс.
     - Для продавцов: от 1000 до 2000 мс.

Интерпретация случайных данных:

- Время ожидания моделирует реальную задержку в процессе обслуживания.
- Случайный порядок посещения отделов добавляет непредсказуемость в поведение покупателей.

Реализация консольного приложения

1. **Запуск**:
   - Программа запрашивает количество покупателей и настройки логирования.
   - Если включено логирование, записываются все сообщения в указанный файл.

2. **Потоки**:
    - Каждый покупатель и продавец представлен отдельным потоком.
    - Потоки синхронизированы через мьютексы и условные переменные.
3. **Работа с очередями**:
    - Очередь покупателей в каждом отделе представлена `std::queue`.
    - Продавцы управляют очередями через мьютексы и сигнализируют о своей готовности с помощью условных переменных.
4. **Безопасность**:
    - Все операции вывода и изменения общих данных защищены мьютексами.

Пример запуска

Входные данные:

```
Enter the number of customers: 3
Enable logging? (1 – Yes, 0 – No): 1
Enter log file name: shop_log.txt
```

Вывод в консоль:

```
Customer 1 entered the shop.
Customer 2 entered the shop.
Customer 1 is being served in department 0.
Customer 3 entered the shop.
Customer 2 is waiting in line at department 0.
Customer 3 is being served in department 2.
Customer 3 left department 2.
Customer 3 is being served in department 1.
Customer 1 left department 0.
....
Seller in department 1 finished work.
Seller in department 0 finished work.
Seller in department 2 finished work.
```

Содержимое файла `shop_log.txt`:

```
Customer 1 entered the shop.
Customer 2 entered the shop.
Customer 1 is being served in department 0.
Customer 3 entered the shop.
```

```
Customer 2 is waiting in line at department 0.
Customer 3 is being served in department 2.
Customer 3 left department 2.
Customer 3 is being served in department 1.
Customer 1 left department 0.
....
Seller in department 1 finished work.
Seller in department 0 finished work.
Seller in department 2 finished work.
```

Используемые синхропримитивы

Для синхронизации потоков используются:

1. **Мьютексы**:
   - Защищают общие ресурсы (очереди, состояние продавцов, вывод на консоль).
2. **Условные переменные**:
   - Управляют очередью покупателей в каждом отделе.
   - Контролируют порядок входа покупателей в магазин.

Код

```cpp
#include <iostream>
#include <pthread.h>
#include <queue>
#include <vector>
#include <random>
#include <chrono>
#include <unistd.h>
#include <algorithm>
#include <fstream>

using namespace std;

// Constants
constexpr int kNumDepartments = 3;                                    //
Amount of departments
constexpr int kMinCustomerWaitMs = 1000;
constexpr int kMaxCustomerWaitMs = 3000;
constexpr int kMinSellerWaitMs = 1000;
```

```cpp
constexpr int kMaxSellerWaitMs = 2000;

// Global variables
bool exit_flag = false;
pthread_mutex_t department_mutexes[kNumDepartments];                //
Mutexes for departments
pthread_cond_t department_conds[kNumDepartments];                   //
Condition variables for queues
queue<int> department_queues[kNumDepartments];                      //
Queue for each department
pthread_mutex_t cout_mutex = PTHREAD_MUTEX_INITIALIZER;             //
Mutex for safe output
bool sellers_free[kNumDepartments] = {true, true, true};            //
Status of sellers

int current_customer_id = 1;                                        //
Tracks the next customer allowed to enter the shop
pthread_mutex_t customer_order_mutex = PTHREAD_MUTEX_INITIALIZER;   //
Mutex for sequential entry
pthread_cond_t customer_order_cond = PTHREAD_COND_INITIALIZER;

bool enable_logging = false;                                        //
Whether logging enabled
ofstream log_file;                                                  // Log
file stream

// Random sleep function to simulate work
void RandomSleep(int min_ms, int max_ms) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(min_ms, max_ms);
    usleep(dist(gen) * 1000);
}

// Safe print function for console output and optional logging
void SafeCout(const string& message) {
    pthread_mutex_lock(&cout_mutex);
    cout << message << endl;
    if (enable_logging && log_file.is_open()) {
        log_file << message << endl;
    }
    pthread_mutex_unlock(&cout_mutex);
}

// Customer thread function
void* CustomerThread(void* arg) {
    const int customer_id = *static_cast<int*>(arg);
    delete static_cast<int*>(arg);

    // Ensure customers enter the shop in sequence
```

```
    pthread_mutex_lock(&customer_order_mutex);
    while (customer_id != current_customer_id) {
        pthread_cond_wait(&customer_order_cond, &customer_order_mutex);
    }
    SafeCout("Customer " + to_string(customer_id) + " entered the shop.");
    current_customer_id++;
    pthread_cond_broadcast(&customer_order_cond);                    //
Notify other customers
    pthread_mutex_unlock(&customer_order_mutex);

    // Randomize the order in which departments are visited
    vector<int> departments(kNumDepartments);
    iota(departments.begin(), departments.end(), 0);
    random_device rd;
    mt19937 gen(rd());
    shuffle(departments.begin(), departments.end(), gen);

    // Visit each department
    for (const int dep : departments) {
        pthread_mutex_lock(&department_mutexes[dep]);

        if (!sellers_free[dep]) {
            SafeCout("Customer " + to_string(customer_id) +
                     " is waiting in line at department " + to_string(dep)
+ ".");
            department_queues[dep].push(customer_id);

            // Wait until it's this customer's turn
            while (!sellers_free[dep] || department_queues[dep].front() !=
customer_id) {
                pthread_cond_wait(&department_conds[dep],
&department_mutexes[dep]);
            }

            department_queues[dep].pop();
        }

        sellers_free[dep] = false;                                   //
Seller becomes busy
        pthread_mutex_unlock(&department_mutexes[dep]);

        SafeCout("Customer " + to_string(customer_id) +
                 " is being served in department " + to_string(dep) +
".");
        RandomSleep(kMinCustomerWaitMs, kMaxCustomerWaitMs);         //
Simulate service time

        pthread_mutex_lock(&department_mutexes[dep]);
        sellers_free[dep] = true;                                    //
Seller becomes free
```

```
        pthread_cond_signal(&department_conds[dep]);                // Notify next customer in line
        pthread_mutex_unlock(&department_mutexes[dep]);

        SafeCout("Customer " + to_string(customer_id) +
                 " left department " + to_string(dep) + ".");
    }

    SafeCout("Customer " + to_string(customer_id) + " left the shop.");
    pthread_exit(nullptr);
}

// Seller thread function
void* SellerThread(void* arg) {
    int department_id = *static_cast<int*>(arg);
    delete static_cast<int*>(arg);

    while (!exit_flag) {
        pthread_mutex_lock(&department_mutexes[department_id]);
        if (!department_queues[department_id].empty()) {
            pthread_cond_signal(&department_conds[department_id]);
        }
        pthread_mutex_unlock(&department_mutexes[department_id]);

        RandomSleep(kMinSellerWaitMs, kMaxSellerWaitMs);            // Simulate idle time
    }

    SafeCout("Seller in department " + to_string(department_id) + " finished work.");
    pthread_exit(nullptr);
}

int main() {
    int num_customers;
    string log_file_name;

    // Input: number of customers
    cout << "Enter the number of customers: ";
    cin >> num_customers;

    // Input: whether logging is enabled
    cout << "Enable logging? (1 - Yes, 0 - No): ";
    int log_choice;
    cin >> log_choice;
    enable_logging = (log_choice == 1);

    // Input: log file name if logging is enabled
    if (enable_logging) {
        cout << "Enter log file name: ";
```

```
        cin >> log_file_name;
        log_file.open(log_file_name);
        if (!log_file.is_open()) {
            cerr << "Error opening log file. Logging disabled." << endl;
            enable_logging = false;
        }
    }

    // Initialize department mutexes and condition variables
    for (int i = 0; i < kNumDepartments; ++i) {
        pthread_mutex_init(&department_mutexes[i], nullptr);
        pthread_cond_init(&department_conds[i], nullptr);
    }

    // Create customer threads
    vector<pthread_t> customer_threads(num_customers);
    for (int i = 0; i < num_customers; ++i) {
        int* id = new int(i + 1);
        if (pthread_create(&customer_threads[i], nullptr, CustomerThread,
id) != 0) {
            cerr << "Error creating thread for customer " << (i + 1) <<
endl;
            delete id;
        }
    }

    // Create seller threads
    vector<pthread_t> seller_threads(kNumDepartments);
    for (int i = 0; i < kNumDepartments; ++i) {
        int* id = new int(i);
        if (pthread_create(&seller_threads[i], nullptr, SellerThread, id)
!= 0) {
            cerr << "Error creating thread for seller in department " << i
<< endl;
            delete id;
        }
    }

    // Wait for all customer threads to finish
    for (const auto& thread : customer_threads) {
        pthread_join(thread, nullptr);
    }

    // Signal sellers to stop and wait for them to finish
    exit_flag = true;
    for (const auto& thread : seller_threads) {
        pthread_join(thread, nullptr);
    }

    // Destroy mutexes and condition variables
```

```
    for (int i = 0; i < kNumDepartments; ++i) {
        pthread_mutex_destroy(&department_mutexes[i]);
        pthread_cond_destroy(&department_conds[i]);
    }
    pthread_mutex_destroy(&cout_mutex);
    pthread_mutex_destroy(&customer_order_mutex);
    pthread_cond_destroy(&customer_order_cond);

    // Close the log file if it was opened
    if (log_file.is_open()) {
        log_file.close();
    }

    return 0;
}
```