**Содержание**

[**Задание** 2](#_Toc6923452)

[**Ход выполнения задачи** 2](#_Toc6923453)

[**Вывод** 8](#_Toc6923454)

[**Приложение** 9](#_Toc6923455)

# **Задание**

Составить программу моделирования классической задачи параллельного программирования «Спящий парикмахер»

# **Ход выполнения задачи**

Монитор — в языках программирования высокоуровневый механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к неразделяемым ресурсам.

Если семафор - это просто счётчик, то монитор – это высокоуровневый механизм, в состав которого может входить семафор.

**Описание алгоритма**

В задаче моделируется обслуживание клиентов парикмахерской. В парикмахерской имеется два помещения: комната ожидания с ограниченным количеством мест и рабочая комната с единственным креслом, в котором располагается обслуживаемый клиент. Посетители заходят в парикмахерскую – если комната ожидания заполнена, то они разворачиваются и уходят; иначе занимают свободные места и засыпают, ожидая своей очереди к парикмахеру. Парикмахер, если есть клиенты, приглашает одного из них в рабочую комнату, и подстригает его. После стрижки клиент покидает парикмахерскую, а парикмахер приглашает следующего посетителя и так далее. Если клиентов нет (комната ожидания пуста), то парикмахер садится в своё рабочее кресло и засыпает. Будит его очередной появляющийся посетитель парикмахерской. В данной задаче места ожидания и рабочее кресло парикмахера являются ресурсами, за доступ к которым конкурируют потоки-посетители. Доступ к этим ресурсам предоставляется строго последовательно: сначала посетитель должен найти место в комнате ожидания, и только затем занять очередь к парикмахеру. При этом предоставление рабочего кресла для обслуживания производит отдельный процесс-парикмахер.

**Исходными данными для модели являются:**

- количество мест в комнате ожидания queueMaxLength;

- количество потоков customerThreads, моделирующих действия посетителей;

- интенсивность появления новых клиентов;

- продолжительность стрижки.

Первые два параметра запрашиваются у пользователя, остальные два заданы в исходном коде программы.

В процессе работы программы эти же сообщения выводятся на экран.

Создание требуемого количества параллельных потоков выполняется следующим образом.

В главной функции программы main содержится:

**#pragma omp parallel sections num\_threads(2)**

**#pragma omp section { }**

определяющие две секции программы, выполняющиеся параллельно. В первой секции вызывается функция customer, моделирующая действия посетителя, во второй секции вызывается функция barber, моделирующая действия парикмахера.

В функции customer содержится:

#pragma omp parallel num\_threads(customerThreads)

создающая столько параллельных потоков.

Для организации очереди посетителей используется структура данных customerStruct, содержащая три поля:

- index: абсолютный номер посетителя;

- state: статус посетителя;

- pNext: указатель на следующую структуру в очереди.

Каждый поток, моделирующий действия посетителя, располагает собственной копией этой структуры, а у тех посетителей, которые в данный момент ждут в очереди, эти структуры связаны в однонаправленный список при помощи полей pNext. В поле index записывается новое значение каждый раз, когда в парикмахерскую приходит новый посетитель (на шаге 2 в алгоритме действий посетителя). Нумерация посетителей сквозная, то есть когда тот же самый поток смоделирует приход следующего посетителя, он получит уже другой номер. Количество потоков определяет количество посетителей, которые могут зайти в парикмахерскую одновременно. Поле state содержит одно из четырёх значений статуса посетителя:

0 – посетитель не в парикмахерской;

1 – посетитель в парикмахерской и ожидает обслуживания;

2 – парикмахер пригласил посетителя в рабочую комнату;

3 – посетитель ещё в парикмахерской, стрижка уже окончена.

Чтобы добавление посетителя в очередь и удаление его оттуда производилось корректно, эти операции в каждый момент времени должны выполняться только одним процессом, а если их запросит другой процесс, то последний должен быть задержан до тех пор, пока первый процесс не выполнит свои действия. Для этого используется переменная-замок queueLock типа omp\_lock\_t, для работы с которой используются следующие функции OpenMP:

- инициализация переменной: omp\_init\_lock(&queueLock);

- установка замка: omp\_set\_lock(&queueLock);

- снятие замка: omp\_unset\_lock(&queueLock);.

**Тест № 1. Количество посетителей и возможности парикмахера сбалансированы.**

Исходные данные:

- количество мест в комнате ожидания: 5;

- количество потоков, моделирующих действия посетителей: 5.

На рисунке приведен скриншот (PrtScr) экрана пользовательского интерфейса при задании параметров моделирования.

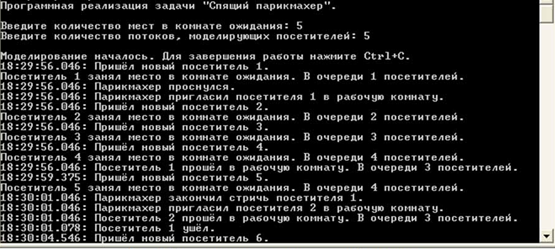


Рис. Скриншот экрана пользовательского интерфейса при задании параметров

**Тест № 2. Количество посетителей превышает возможности парикмахера.**

Исходные данные:

- количество мест в комнате ожидания: 5;

- количество потоков, моделирующих действия посетителей: 7.

Далее приводится трассировка результатов моделирования задачи (до момента прерывания работы пользователем):

18:31:43.187: Пришёл новый посетитель 1.

Посетитель 1 занял место в комнате ожидания. В очереди 1 посетителей.

18:31:43.187: Парикмахер проснулся.

18:31:43.187: Парикмахер пригласил посетителя 1 в рабочую комнату.

18:31:43.187: Пришёл новый посетитель 2.

Посетитель 2 занял место в комнате ожидания. В очереди 2 посетителей.

18:31:43.187: Пришёл новый посетитель 3.

Посетитель 3 занял место в комнате ожидания. В очереди 3 посетителей.

18:31:43.187: Пришёл новый посетитель 4.

Посетитель 4 занял место в комнате ожидания. В очереди 4 посетителей.

18:31:43.203: Пришёл новый посетитель 5.

Посетитель 5 занял место в комнате ожидания. В очереди 5 посетителей.

18:31:43.203: Пришёл новый посетитель 6.

Свободных мест нет. Посетитель 6 ушёл.

18:31:43.203: Посетитель 1 прошёл в рабочую комнату. В очереди 4 посетителей.

18:31:46.671: Пришёл новый посетитель 7.

Посетитель 7 занял место в комнате ожидания. В очереди 5 посетителей.

18:31:46.859: Пришёл новый посетитель 8.

Свободных мест нет. Посетитель 8 ушёл.

18:31:46.953: Пришёл новый посетитель 9.

Свободных мест нет. Посетитель 9 ушёл.

18:31:48.187: Парикмахер закончил стричь посетителя 1.

18:31:48.187: Парикмахер пригласил посетителя 2 в рабочую комнату.

18:31:48.203: Посетитель 2 прошёл в рабочую комнату. В очереди 4 посетителей.

18:31:48.234: Посетитель 1 ушёл.

18:31:48.656: Пришёл новый посетитель 10.

Посетитель 10 занял место в комнате ожидания. В очереди 5 посетителей.

18:31:51.687: Пришёл новый посетитель 11.

Свободных мест нет. Посетитель 11 ушёл.

18:31:53.031: Пришёл новый посетитель 12.

Свободных мест нет. Посетитель 12 ушёл.

18:31:53.187: Парикмахер закончил стричь посетителя 2.

18:31:53.187: Парикмахер пригласил посетителя 3 в рабочую комнату.

18:31:53.203: Посетитель 3 прошёл в рабочую комнату. В очереди 4 посетителей.

18:31:53.218: Посетитель 2 ушёл.

18:31:54.531: Пришёл новый посетитель 13.

Посетитель 13 занял место в комнате ожидания. В очереди 5 посетителей.

18:31:56.687: Пришёл новый посетитель 14.

Свободных мест нет. Посетитель 14 ушёл.

18:31:58.031: Пришёл новый посетитель 15.

Свободных мест нет. Посетитель 15 ушёл.

18:31:58.187: Парикмахер закончил стричь посетителя 3.

18:31:58.187: Парикмахер пригласил посетителя 4 в рабочую комнату.

18:31:58.203: Посетитель 4 прошёл в рабочую комнату. В очереди 4 посетителей.

18:31:58.218: Посетитель 3 ушёл.

18:31:59.531: Пришёл новый посетитель 16.

Посетитель 16 занял место в комнате ожидания. В очереди 5 посетителей.

18:32:01.687: Пришёл новый посетитель 17.

Свободных мест нет. Посетитель 17 ушёл.

18:32:03.031: Пришёл новый посетитель 18.

Свободных мест нет. Посетитель 18 ушёл.

18:32:03.203: Парикмахер закончил стричь посетителя 4.

18:32:03.203: Парикмахер пригласил посетителя 5 в рабочую комнату.

18:32:03.203: Посетитель 5 прошёл в рабочую комнату. В очереди 4 посетителей.

18:32:03.234: Посетитель 4 ушёл.

18:32:04.531: Пришёл новый посетитель 19.

Посетитель 19 занял место в комнате ожидания. В очереди 5 посетителей.

18:32:06.687: Пришёл новый посетитель 20.

Свободных мест нет. Посетитель 20 ушёл.

18:32:08.031: Пришёл новый посетитель 21.

Свободных мест нет. Посетитель 21 ушёл.

18:32:08.203: Парикмахер закончил стричь посетителя 5.

18:32:08.203: Парикмахер пригласил посетителя 7 в рабочую комнату.

18:32:08.203: Посетитель 7 прошёл в рабочую комнату. В очереди 4 посетителей.

18:32:08.234: Посетитель 5 ушёл.

18:32:09.531: Пришёл новый посетитель 22.

Посетитель 22 занял место в комнате ожидания. В очереди 5 посетителей.

18:32:11.703: Пришёл новый посетитель 23.

Свободных мест нет. Посетитель 23 ушёл.

18:32:13.031: Пришёл новый посетитель 24.

Свободных мест нет. Посетитель 24 ушёл.

18:32:13.203: Парикмахер закончил стричь посетителя 7.

18:32:13.203: Парикмахер пригласил посетителя 10 в рабочую комнату.

18:32:13.203: Посетитель 10 прошёл в рабочую комнату. В очереди 4 посетителей.

18:32:13.234: Посетитель 7 ушёл.

18:32:14.531: Пришёл новый посетитель 25.

Посетитель 25 занял место в комнате ожидания. В очереди 5 посетителей.

**Тест № 3. Количество посетителей ниже того, которое может обслужить парикмахер**.

Исходные данные:

- количество мест в комнате ожидания: 5;

- количество потоков, моделирующих действия посетителей: 2.

- время стрижки (заданное в исходном коде программы) сокращено с 5 до 3 секунд.

# **Вывод**

В результате выполнения настоящей работы реализована модель классической задачи многопоточного программирования «Спящий парикмахер». Количество параллельных потоков, моделирующих действия посетителей парикмахерской, может быть произвольно задано пользователем.

# **Приложение**

#include "stdafx.h"

#include <omp.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <process.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

using namespace std;

FILE \*stream; // Файловый поток.

struct customerStruct // Структура, использующаяся для организации { очереди из посетителей.}

int index; // - абсолютный номер посетителя;

int state; // - статус посетителя:

// 0 - посетитель не в парикмахерской;

// 1 - посетитель в парикмахерской

// и ожидает обслуживания;

// 2 - парикмахер пригласил посетителя

// в рабочую комнату;

// 3 - посетитель ещё в парикмахерской,

// но стрижка уже окончена.

customerStruct \*pNext; // - указатель на следующий элемент очереди.

/\* Процедура printTimeStamp для определения и вывода текущего времени. \*/

void printTimeStamp()

{

SYSTEMTIME timeStamp; // Текущее время.

GetLocalTime (&timeStamp); // Определить текущее время.

printf("%02d:%02d:%02d.%03d", // Вывести текущее время на экран.

timeStamp.wHour, timeStamp.wMinute,

timeStamp.wSecond, timeStamp.wMilliseconds);

timeStamp.wHour, timeStamp.wMinute,

timeStamp.wSecond, timeStamp.wMilliseconds);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Процедура customer, моделирующая действия посетителей. \*/

/\* Принимаемые параметры: \*/

/\* pFirst - указатель на первого посетителя в очереди \*/

/\* или nullptr, если очередь пуста; \*/

/\* pLast - указатель на последнего посетителя в очереди \*/

/\* или nullptr, если очередь пуста; \*/

/\* queueLength - количество посетителей в очереди в данный момент; \*/

/\* queueMaxLength - количество мест в комнате ожидания; \*/

/\* customerThreads - количество параллельных потоков \*/

/\* для моделирования посетителей; \*/

/\* totalCustomers - общий счётчик посетителей \*/

/\* (в начале работы равен нулю); \*/

/\* queueLock - замок для блокировки операций с очередью посетителей. \*/

/\* Глобальные переменные: \*/

/\* stream - файловый поток. \*/

/\* Возвращаемый результат: нет \*/

/\* Изменяемые переменные: pFirst, pLast, queueLength, totalCustomers, \*/

/\* queueLock. \*/

void customer(customerStruct\*& pFirst,

customerStruct\*& pLast,

int &queueLength,

int queueMaxLength,

int customerThreads,

int &totalCustomers,

omp\_lock\_t queueLock){

#pragma omp parallel num\_threads(customerThreads) // Создать параллельные

{ // потоки.

customerStruct self;

while(true) // Выполнять в бесконечном цикле.

{

self.state = 0; // Начальный статус посетителя:

// "Не в парикмахерской".

self.pNext = nullptr;

Sleep(rand() % 5000); // Выждать случайное время

// в пределах 5 секунд.

omp\_set\_lock(&queueLock); // Установить замок для блокировки

// операций с очередью.

self.index = ++totalCustomers; // Сохранить абсолютный номер

// посетителя.

#pragma omp critical // Вывести сообщение на экран и в файл.

{

printTimeStamp();

printf(": Пришёл новый посетитель %d.\n", self.index);

fprintf(stream, ": Пришёл новый посетитель %d.\n", self.index);

} // #pragma omp critical

if (queueLength < queueMaxLength) // Если в комнате ожидания

{ // есть свободное место, то

// добавить посетителя в очередь:

if (pFirst == nullptr) // - если очередь пуста,

pFirst = pLast = &self; // то сделать посетителя первым

// и последним в очереди;

else // - иначе

{

(\*pLast).pNext = &self; // добавить его в конец очереди.

pLast = &self;

}

queueLength++; // Увеличить счётчик количества

// посетителей в очереди.

self.state = 1; // Изменить текущий статус посетителя.

#pragma omp critical // Вывести сообщение на экран и в файл.

{

printf("Посетитель %d занял место в комнате ожидания. ", self.index);

printf("В очереди %d посетителей.\n", queueLength);

fprintf(stream,

"Посетитель %d занял место в комнате ожидания. ", self.index);

fprintf(stream, "В очереди %d посетителей.\n", queueLength);

} // #pragma omp critical

omp\_unset\_lock(&queueLock); // Снять замок для блокировки

// операций с очередью.

while(self.state != 2) // Дождаться, пока парикмахер

Sleep(1); // пригласит посетителя для стрижки.

omp\_set\_lock(&queueLock); // Установить замок для блокировки

// операций с очередью.

pFirst = (\*pFirst).pNext; // Удалить посетителя из очереди

if (pFirst == nullptr) // (к этому моменту посетитель

pLast = nullptr; // будет в очереди первым).

queueLength--; // Уменьшить счётчик количества

// посетителей в очереди.

#pragma omp critical // Вывести сообщение на экран и в файл.

{

printTimeStamp();

printf(": Посетитель %d прошёл в рабочую комнату. ", self.index);

printf("В очереди %d посетителей.\n", queueLength);

fprintf(stream,

": Посетитель %d прошёл в рабочую комнату. ", self.index);

fprintf(stream, "В очереди %d посетителей.\n", queueLength);

} // #pragma omp critical

omp\_unset\_lock(&queueLock); // Снять замок для блокировки

// операций с очередью.

while(self.state != 3) // Дождаться, пока парикмахер

Sleep(100); // закончит стрижку.

#pragma omp critical // Вывести сообщение на экран и в файл.

{

printTimeStamp();

printf(": Посетитель %d ушёл.\n", self.index);

fprintf(stream, ": Посетитель %d ушёл.\n", self.index);

} // #pragma omp critical

} // if (queueLength < queueMaxLength)

else // Если в комнате ожидания

{ // нет свободного места, то

#pragma omp critical // вывести сообщение на экран.

{

printf("Свободных мест нет. Посетитель %d ушёл.\n", self.index);

fprintf(stream,

"Свободных мест нет. Посетитель %d ушёл.\n", self.index);

omp\_unset\_lock(&queueLock); // Снять замок для блокировки

// операций с очередью.

} // #pragma omp critical

} // else\_if (queueLength < queueMaxLength)

} // while(true)

} // #pragma omp parallel

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Процедура barber, моделирующая действия парикмахера. \*/

/\* Принимаемые параметры: \*/

/\* pFirst - указатель на первого посетителя в очереди. \*/

/\* Глобальные переменные: \*/

/\* stream - файловый поток. \*/

/\* Возвращаемый результат: нет \*/

/\* Изменяемые переменные: структура по указателю pFirst. \*/

void barber(customerStruct\*& pFirst)

{

int state = 0; // Текущий статус парикмахера.

// Начальное значение 0

// означает: "Парикмахер спит".

customerStruct \*pCurrent = nullptr; // Указатель на посетителя,

// которого в данный момент

// обслуживает парикмахер.

while(true) // Выполнять в бесконечном цикле.

if (pFirst != nullptr) // Если в комнате ожидания есть

{ // посетители, то

if (state == 0) // Если парикмахер спал,

{ // то вывести сообщение

#pragma omp critical // на экран и в файл.

{

printTimeStamp();

printf(": Парикмахер проснулся.\n");

fprintf(stream, ": Парикмахер проснулся.\n");

} // #pragma omp critical

} // if (state = 0)

state = 1; // Изменить статус парикмахера

// на "Парикмахер работает".

pCurrent = pFirst; // Сохранить ссылку на первого

// посетителя в очереди.

#pragma omp critical // Вывести сообщение на экран

{ // и в файл.

printTimeStamp();

printf(": Парикмахер пригласил посетителя %d в рабочую комнату.\n",

(\*pCurrent).index);

fprintf(stream,

": Парикмахер пригласил посетителя %d в рабочую комнату.\n",

(\*pCurrent).index);

} // #pragma omp critical

(\*pCurrent).state = 2; // Изменить статус обслуживаемого

// посетителя на "Посетитель

// приглашён для стрижки".

Sleep(5000); // Происходит стрижка.

#pragma omp critical // Вывести сообщение на экран

{ // и в файл.

printTimeStamp();

printf(": Парикмахер закончил стричь посетителя %d.\n",

(\*pCurrent).index);

fprintf(stream, ": Парикмахер закончил стричь посетителя %d.\n",

(\*pCurrent).index);

} // #pragma omp critical

(\*pCurrent).state = 3; // Изменить статус обслуживаемого

// посетителя на "Стрижка окончена".

} // if (pFirst != nullptr)

else // Если в комнате ожидания

{ // нет посетителей, то:

if (state == 1) // Если парикмахер работал,

{ // то вывести сообщение

#pragma omp critical // на экран и в файл.

{

printTimeStamp();

printf(": Больше нет посетителей. Парикмахер заснул.\n");

fprintf(stream,

": Больше нет посетителей. Парикмахер заснул.\n");

} // #pragma omp critical

} // if (state = 1)

state = 0; // Изменить статус парикмахера

// на "Парикмахер спит".

} // else\_if (pFirst != nullptr)

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Главная процедура main. \*/

/\* Выполняемые действия: \*/

/\* 2) Инициализировать замок для блокировки операций \*/

/\* с очередью посетителей. \*/

/\* 4) Запросить у пользователя следующие исходные данные: \*/

/\* - количество мест в комнате ожидания queueMaxLength; \*/

/\* - количество потоков customerThreads, моделирующих \*/

/\* действия посетителей. \*/

/\* 5) Создать два параллельных потока и в одном из них вызвать \*/

/\* процедуру моделирования посетителей, а в другом - процедуру \*/

/\* моделирования парикмахера. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

setlocale (LC\_ALL,"Russian");

srand (time(NULL));

customerStruct \*pFirst = nullptr, // Указатель на первого посетителя в очереди.

\*pLast = nullptr; // Указатель на последнего посетителя в очереди.

int queueLength = 0, // Длина очереди в текущий момент.

queueMaxLength = 0, // Количество мест в комнате ожидания

// (задаётся пользователем).

totalCustomers = 0, // Общий счётчик посетителей.

customerThreads = 0; // Количество потоков, моделирующих действия

// посетителей (задаётся пользователем).

omp\_lock\_t queueLock; // Замок для блокировки операций

// с очередью посетителей.

fopen\_s(&stream, "log.txt", "w"); // Определить файл для вывода информации.

omp\_init\_lock(&queueLock); // Инициализировать замок.

omp\_set\_nested(true); // Включить обработку вложенных директив OpenMP.

/\* Запросить у пользователя исходные данные. \*/

printf ("Лабораторная работа N 7.\n");

printf ("Программная реализация задачи \"Спящий парикмахер\".\n\n");

printf ("Введите количество мест в комнате ожидания: ");

cin >> queueMaxLength;

printf ("Введите количество потоков, моделирующих посетителей: ");

cin >> customerThreads;

printf ("\nМоделирование началось. Для завершения работы нажмите Ctrl+C.\n");

#pragma omp parallel sections num\_threads(2) // Создать два параллельных потока.

{

#pragma omp section // Вызвать процедуру моделирования

{ // посетителей в первом потоке.

customer(pFirst, pLast,

queueLength, queueMaxLength,

customerThreads, totalCustomers,

queueLock);

}

#pragma omp section // Вызвать процедуру моделирования

{ // парикмахера во втором потоке.

barber(pFirst);

}

} // #pragma omp parallel sections

}