**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

Дисциплина: Обьектно-ориентированное программирование

по теме «Потоки в C++»

Выполнил: ст. группы ПВ-21  
Ковалев Павел

Проверил: Буханов Д.Г.

Белгород 2020

**Цель работы**: изучение основных возможностей потоков управления и потоков ввода-вывода. Получение навыков работы со стандартными средствами управления потоками в С++11. Знакомство с классом Thread и стандартными средствами синхронизации потоков.

**Вариант 8**

Один поток удаляет лишние пробелы в строке, а другой подсчитывает количество слов в тексте. Произвести синхронный вывод при каждой итерации. Показать выполнение работы программы в синхронном и асинхронном режимах.

1. #include <QCoreApplication>
2. #include <QTextCodec>
3. #include <mutex>
4. #include <thread>
5. #include <chrono>
6. #include <iostream>
7. #include <string>
8. #include <limits>
9. #include <windows.h>
10. #undef max
11. *using* *namespace* std;
12. *class* **Thread** {
13. *private*:
14. mutex& m;
15. bool sync;
16. string& text;
17. unsigned& count;
18. *virtual* void ***body***() = 0;
19. *public*:
20. **Thread**(mutex& m, bool sync, string& text, unsigned& count) : m(m), sync(sync), text(text), count(count) {};
21. void **lock\_mutex**(){ *this*->m.lock(); }
22. void **unlock\_mutex**(){ *this*->m.unlock(); }
23. bool **get\_sync**(){ *return* *this*->sync; }
24. string& **get\_text**(){ *return* *this*->text; }
25. unsigned **get\_count**(){ *return* *this*->count; }
26. void **set\_count**(unsigned new\_val){ *this*->count = new\_val; }
27. void *operator*()(){ *this*->*body*(); }
28. };
29. *class* **CountThread** : *public* Thread {
30. void ***body***();
31. *public*:
32. **CountThread**(mutex& m, bool sync, string& text, unsigned& count) : Thread(*m*, sync, *text*, *count*) {};
33. };
34. *class* **DelThread** : *public* Thread {
35. void ***body***();
36. *public*:
37. **DelThread**(mutex& m, bool sync, string& text, unsigned& count) : Thread(*m*, sync, *text*, *count*) {};
38. };
39. *class* **OutThread** : *public* Thread {
40. void ***body***();
41. *public*:
42. **OutThread**(mutex& m, bool sync, string& text, unsigned& count) : Thread(*m*, sync, *text*, *count*) {};
43. };
44. void CountThread::***body***() {
45. *while* (*this* -> get\_count() != 1) {
    1. *if* (*this*->get\_sync()) *this*->lock\_mutex();
    2. *this*->set\_count(0);
    3. int state;
    4. state = 0;
    5. *for* (size\_t i = 0; i < *this*->get\_text().size(); i++) {
       1. *if* (*this*->get\_text()[i] == ' ') {
       2. state = 0;
       3. }
       4. *else* *if* (state == 0) {
       5. state = 1;
       6. *this*->set\_count(*this*->get\_count()+1);
       7. }
    6. }
    7. *if* (*this*->get\_sync()) *this*->unlock\_mutex();
    8. this\_thread::sleep\_for(chrono::seconds(1));
46. }
47. }
48. void DelThread::***body***() {
49. size\_t i = 0;
50. *while* (i < *this*->get\_text().size()) {
    1. *if* (*this*->get\_sync()) *this*->lock\_mutex();
    2. *if* (*this*->get\_text()[i] == ' ') {
       1. *for* (size\_t j = i + 1; j < *this*->get\_text().size(); j++) {
       2. *this*->get\_text()[j - 1] = *this*->get\_text()[j];
       3. }
       4. *this*->get\_text().resize(*this*->get\_text().size() - 1);
    3. }
    4. i++;
    5. *if* (*this*->get\_sync()) *this*->unlock\_mutex();
    6. this\_thread::sleep\_for(std::chrono::seconds(1));
51. }
52. }
53. void OutThread::***body***() {
54. *while* (1)
55. {
    1. *this*->lock\_mutex();
    2. cout << QString::fromUtf8("В тексте ").toLocal8Bit().data() << *this*->get\_count() << QString::fromUtf8(" слов.").toLocal8Bit().data() << endl;
    3. cout << *this*->get\_text() << endl;
    4. *this*->unlock\_mutex();
    5. std::this\_thread::sleep\_for(std::chrono::seconds(1));
56. }
57. }
58. int **main**(int argc, char \*argv[])
59. {
60. QCoreApplication a(*argc*, argv);std::mutex m;
61. #ifdef Q\_OS\_WIN32
62. QTextCodec::setCodecForLocale(QTextCodec::codecForName("IBM 866"));
63. #endif
64. #ifdef Q\_OS\_LINUX
65. QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));
66. #endif
67. string text;
68. unsigned count;
69. int answer;
70. *do* {
    1. cout << QString::fromUtf8("Работа в синхронном или асинхронном режиме?\n1-синхронный, 2 - асинхронный: ").toLocal8Bit().data();
    2. cin >> answer;
71. }*while* (answer != 1 && answer != 2);
72. bool sync = answer == 1;
73. cout << QString::fromUtf8("Введите текст: ").toLocal8Bit().data();
74. cin.clear();
75. cin.ignore(numeric\_limits<streamsize>::max(), '\n');
76. getline(*cin*, *text*);
77. CountThread ct(ref(*m*), sync, ref(*text*), ref(*count*));
78. DelThread dt(ref(*m*), sync, ref(*text*), ref(*count*));
79. OutThread ot(ref(*m*), sync, ref(*text*), ref(*count*));
80. thread th1(*ct*); *//1* *поток*
81. thread th2(*dt*); *//2* *поток*
82. thread th3(*ot*); *//поток* *вывода*
83. th1.detach();
84. th2.detach();
85. th3.join();
86. *return* a.exec();
87. }

Один поток удаляет лишние пробелы в строке, а другой подсчитывает количество слов в тексте. Произвести синхронный вывод при каждой итерации. Показать выполнение работы программы в синхронном и асинхронном режимах.

1. #include <QCoreApplication>
2. #include <QTextCodec>
3. #include <iostream>
4. #include <string>
5. #include <limits>
6. #include <cstdlib>
7. #include <ctime>
8. #include <functional>
9. #include <Windows.h>
10. *using* *namespace* std;
11. #undef max
12. #define PAUSE 500
13. *const* int OFFSET = ('a' - 'A');
14. *class* **Thread** {
15. *private*:
16. CONST HANDLE hMutex;
17. bool sync;
18. string& text;
19. *public*:
20. *friend* void **start\_thread**(Thread&);
21. *virtual* void ***body***() = 0;
22. **Thread**(CONST HANDLE hMutex, bool sync, string& text) : hMutex(hMutex), sync(sync), text(text) {};
23. void **lock\_mutex**() { WaitForSingleObject(*this*->hMutex, INFINITE); }
24. void **unlock\_mutex**() { ReleaseMutex(*this*->hMutex); }
25. bool **get\_sync**() { *return* *this*->sync; }
26. string& **get\_text**() { *return* *this*->text; }
27. };
28. *class* **UpThread** : *public* Thread {
29. *public*:
30. void ***body***();
31. **UpThread**(CONST HANDLE hMutex, bool sync, string& text) : Thread(hMutex, sync, *text*) {};
32. };
33. *class* **DownThread** : *public* Thread {
34. *public*:
35. void ***body***();
36. **DownThread**(CONST HANDLE hMutex, bool sync, string& text) : Thread(hMutex, sync, *text*) {};
37. };
38. *class* **OutThread** : *public* Thread {
39. void ***body***();
40. *public*:
41. **OutThread**(CONST HANDLE hMutex, bool sync, string& text) : Thread(hMutex, sync, *text*) {};
42. };
43. void UpThread::***body***() {
44. *while* (1) {
    1. *if* (*this*->get\_sync()) *this*->lock\_mutex();
    2. int rand\_number = rand() % *this*->get\_text().size();
    3. *if* (*this*->get\_text()[rand\_number] > 'Z') {
       1. *this*->get\_text()[rand\_number] -= OFFSET;
    4. }
    5. *if* (*this*->get\_sync()) *this*->unlock\_mutex();
    6. Sleep(PAUSE);
45. }
46. }
47. void DownThread::***body***() {
48. *while* (1) {
    1. *if* (*this*->get\_sync()) *this*->lock\_mutex();
    2. int rand\_number = rand() % *this*->get\_text().size();
    3. *if* (*this*->get\_text()[rand\_number] < 'a'){
       1. *this*->get\_text()[rand\_number] += OFFSET;
    4. }
    5. *if* (*this*->get\_sync()) *this*->unlock\_mutex();
    6. Sleep(PAUSE);
49. }
50. }
51. void OutThread::***body***() {
52. *while* (1)
53. {
    1. *this*->lock\_mutex();
    2. cout << endl;
    3. cout << *this*->get\_text();
    4. cout << endl;
    5. *this*->unlock\_mutex();
    6. Sleep(PAUSE);
54. }
55. }
56. DWORD WINAPI **start\_thread**(CONST LPVOID lpParam) {
57. ((Thread\*)lpParam)->*body*();
58. ExitThread(0);
59. }
60. int **main**(int argc, char \*argv[])
61. {
62. QCoreApplication a(*argc*, argv);
63. #ifdef Q\_OS\_WIN32
64. QTextCodec::setCodecForLocale(QTextCodec::codecForName("IBM 866"));
65. #endif
66. #ifdef Q\_OS\_LINUX
67. QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));
68. #endif
69. srand(time(*nullptr*));
70. string text;
71. int answer;
72. *do* {
    1. cout << QString::fromUtf8("Работа в синхронном или асинхронном режиме?\n1-синхронный, 2 - асинхронный: ").toLocal8Bit().data();
    2. cin >> answer;
73. } *while* (answer != 1 && answer != 2);
74. bool sync = answer == 1;
75. cout << QString::fromUtf8("Введите текст: ").toLocal8Bit().data();
76. cin.clear();
77. cin.ignore(numeric\_limits<std::streamsize>::max(), '\n');
78. getline(*cin*, *text*);
79. CONST HANDLE hMutex = CreateMutex(NULL, FALSE, NULL);
80. UpThread ut(hMutex, sync, ref(*text*));
81. DownThread dt(hMutex, sync, ref(*text*));
82. OutThread ot(hMutex, sync, ref(*text*));
83. HANDLE hThreads[3];
84. hThreads[0] = CreateThread(NULL, 0, &start\_thread, &ut, 0, NULL);
85. hThreads[1] = CreateThread(NULL, 0, &start\_thread, &dt, 0, NULL);
86. hThreads[2] = CreateThread(NULL, 0, &start\_thread, &ot, 0, NULL);
87. WaitForMultipleObjects(3, hThreads, TRUE, INFINITE);
88. CloseHandle(hThreads[0]);
89. CloseHandle(hThreads[1]);
90. CloseHandle(hThreads[2]);
91. CloseHandle(hMutex);
92. ExitProcess(0);
93. *return* a.exec();
94. }