Герман Александр, ФИТ-4

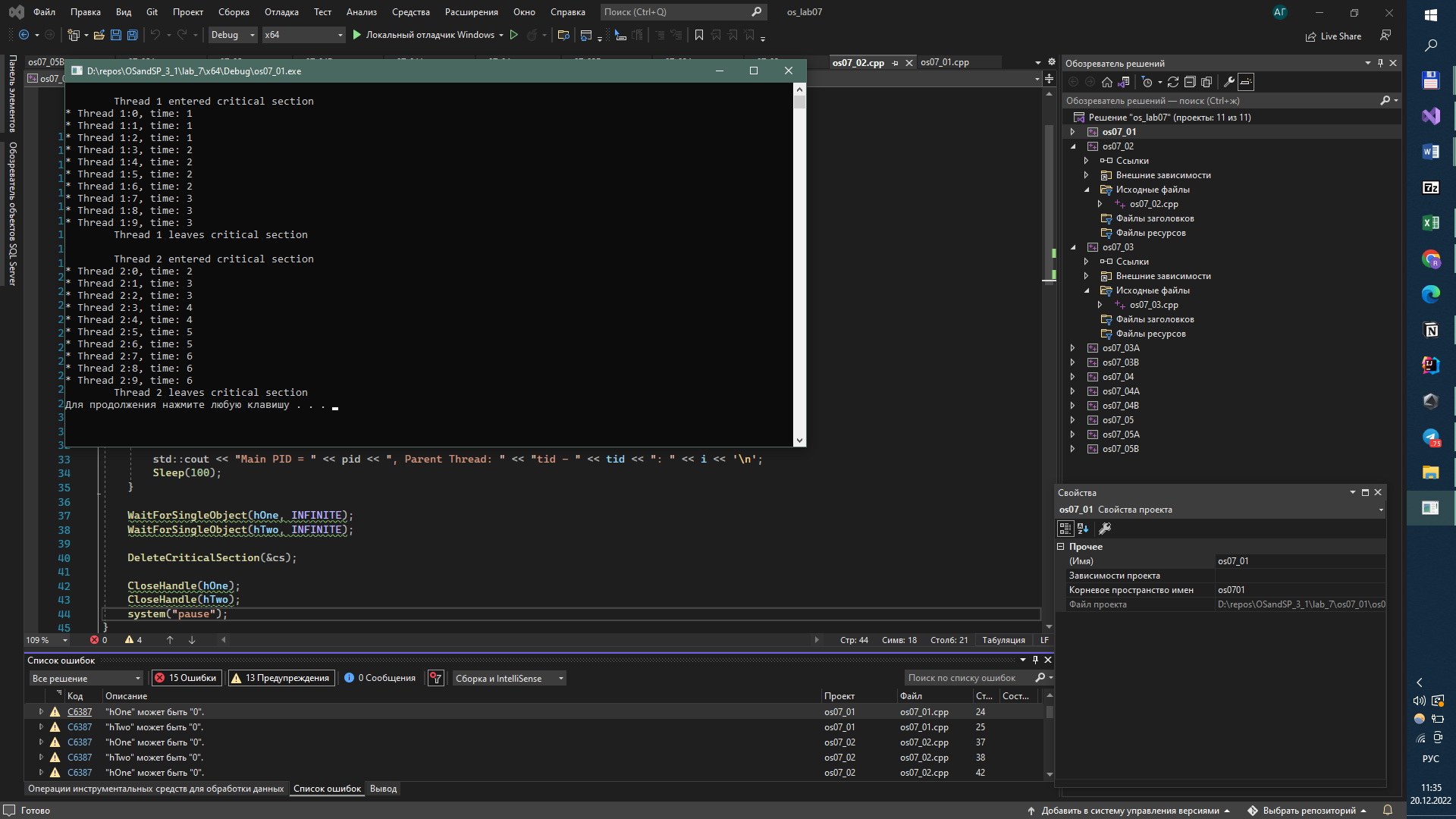
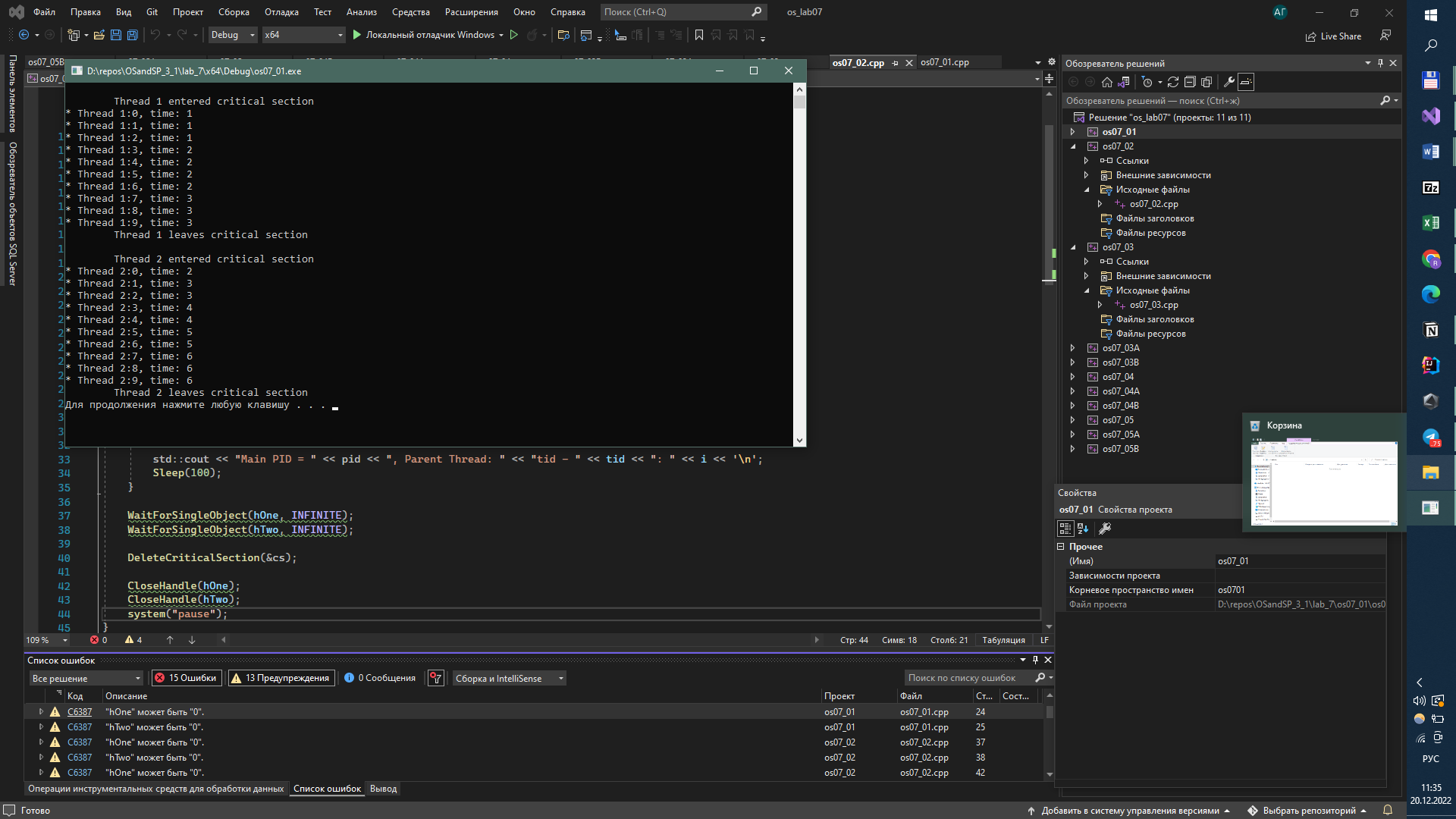
**Лабораторная работа №7**

«Синхронизация»

**Задание 01. Windows**

1. Напишите ассемблерный код с применением команд BTS или BTR, демонстрирующий реализацию механизма синхронизации двух потоков одного процесса и поясните его работу.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  DWORD WINAPI ChildThrOne();  DWORD WINAPI ChildThrTwo();  void EnterCriticalSectionAsm();  void LeaveCriticalSectionAsm();  int check = 0;  int main() {  DWORD OneId = NULL, TwoId = NULL;  HANDLE hOne = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThrOne, NULL, 0, &OneId);  HANDLE hTwo = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThrTwo, NULL, 0, &TwoId);  WaitForSingleObject(hOne, INFINITY);  WaitForSingleObject(hTwo, INFINITY);  CloseHandle(hOne);  CloseHandle(hTwo);  system("pause");  return 0;  }  DWORD WINAPI ChildThrOne() {  int start = clock();  EnterCriticalSectionAsm();  cout << "\n\tThr\_1 now entered critical section\n";  for (int i = 0; i < 10; i++) {  cout << "\* Thr\_1:" << i << ", time: " << clock() - start << endl;  }  cout << "\tThr\_1 now leaves critical section\n";  LeaveCriticalSectionAsm();  return 0;  }  DWORD WINAPI ChildThrTwo() {  int start = clock();  EnterCriticalSectionAsm();  cout << "\n\tThr\_2 now entered critical section\n";  for (int i = 0; i < 10; i++) {  cout << "\* Thr\_2:" << i << ", time: " << clock() - start << endl;  }  cout << "\tThr\_2 now leaves critical section\n";  LeaveCriticalSectionAsm();  return 0;  }  void EnterCriticalSectionAsm() {  // lock устанавливает сигнал LOCK и запрещает доступ к системной шине другим процессорам на время выполнения данной команды  // BTS сохраняет значение бита из check со смещением 0 во флаге CF, а затем устанавливает этот бит в 1  \_asm {  CriticalSection:  lock bts check, 0;  jc CriticalSection  }  }  void LeaveCriticalSectionAsm() {  // BTR сохраняет значение бита из check со смещением 0 во флаге CF, а затем обнуляет этот бит  \_asm lock btr check, 0  } |

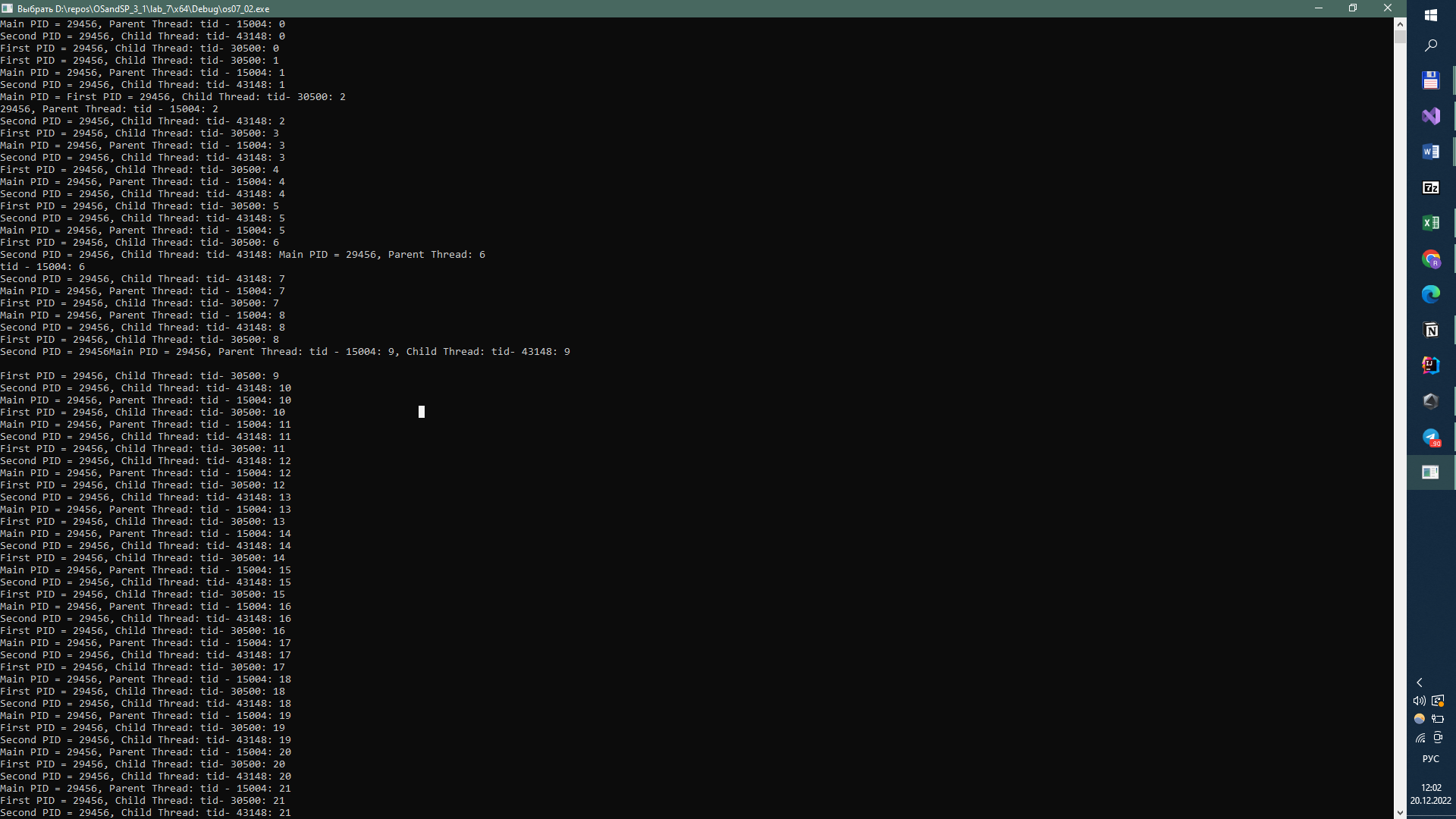
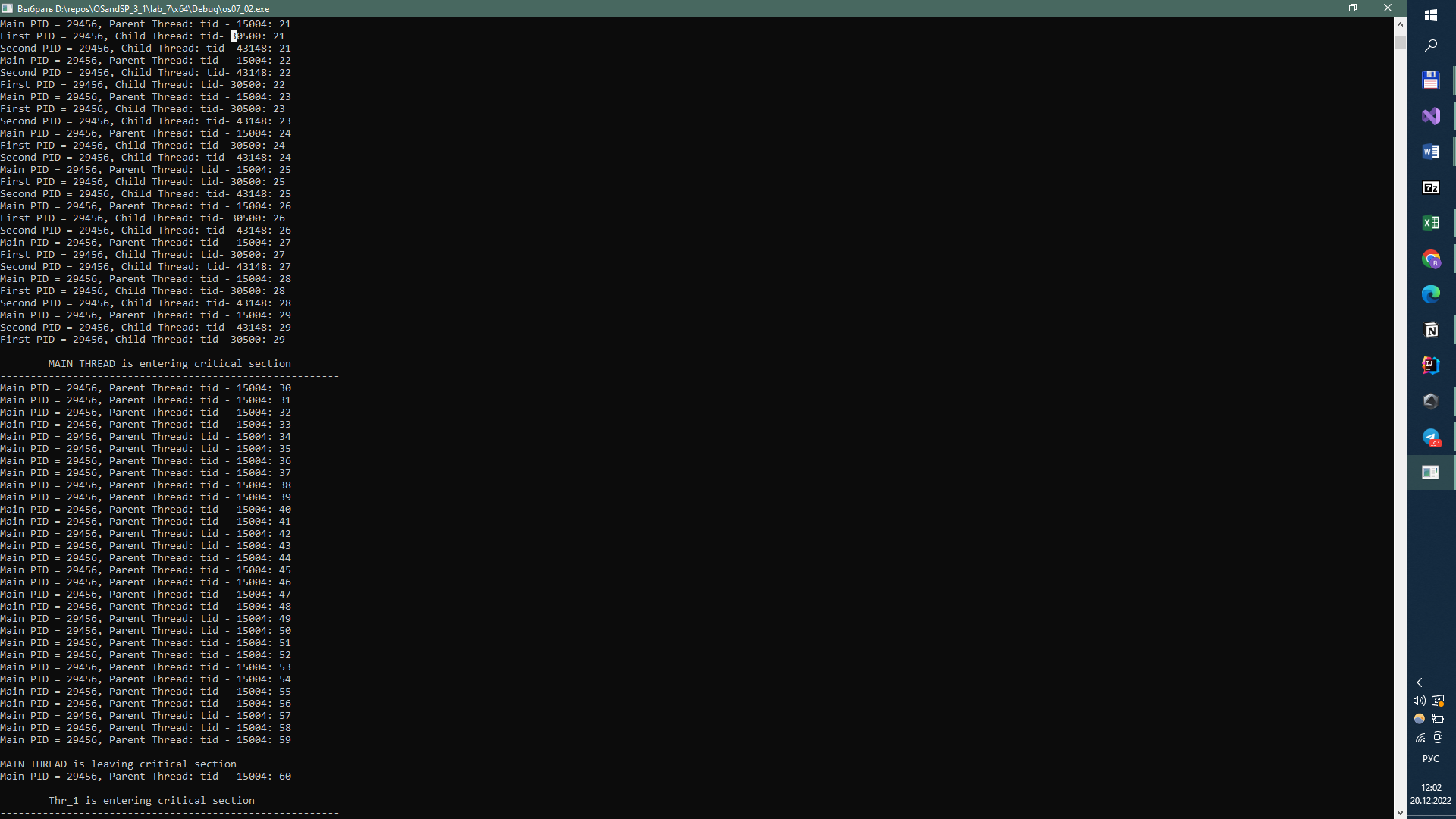
При запуске потоков вызывается функция EnterCriticalSectionAssem(), которая содержит вставку ассемблерного кода. Команда BTS сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем устанавливает этот бит в 1. Команда jc выполняет ветвление (цикл) по адресу, если флаг переноса равен "1", в противном случае выполняется следующая команда. Этот переход будет осуществляться до тех пор, пока не будет вызвана функция LeaveCriticalSectionAssem, в которой находится команда BTR, которая сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем обнуляет этот бит. То есть поток А установит 1 и поток Б будет ходить по этому циклу, в то время, как А будет выполнять свою работу. Потом А сбросит и поток Б сможет задать 1 и продолжить свою работу.

**Задание 02. Windows**

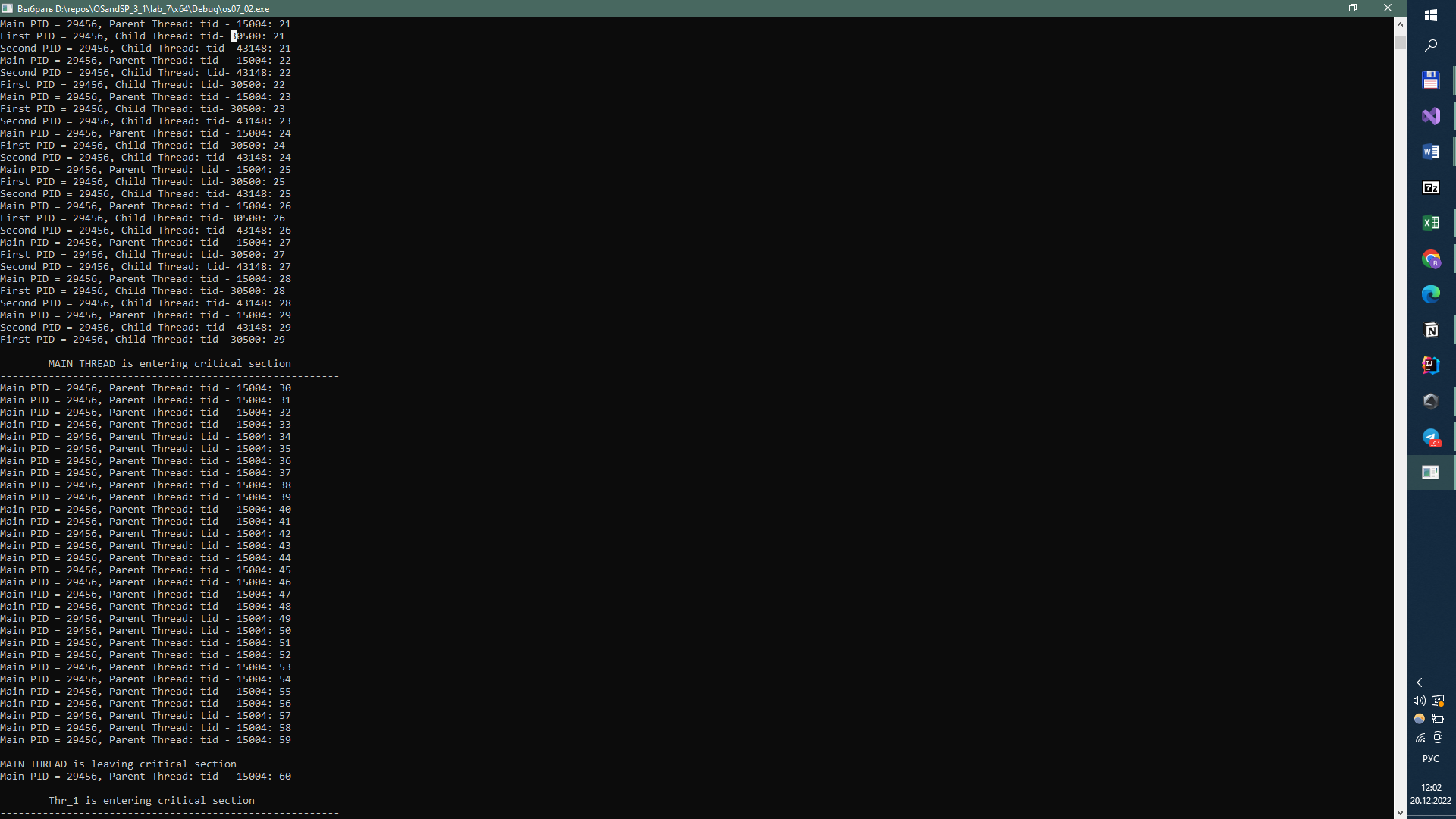
1. Разработайте приложение **OS07\_02**, запускающее два дочерних потока **A** и **B**.
2. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS07\_02** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма **critical section.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  //Поочередно с 30 до 60  DWORD pid = NULL;  DWORD WINAPI ChildThrOne();  DWORD WINAPI ChildThrTwo();  CRITICAL\_SECTION cs;  int main() {  pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  DWORD OneId = NULL, TwoId = NULL;  HANDLE hOne = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThrOne, NULL, 0, &OneId);  HANDLE hTwo = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThrTwo, NULL, 0, &TwoId);  // инициализация критической секци  InitializeCriticalSection(&cs);  for (int i = 0; i < 90; i++) {  if (i == 30) {  EnterCriticalSection(&cs);  std::cout << "\n\tMAIN THREAD is entering critical section\n";  std::cout << "--------------------------------------------------------\n";  }  if (i == 60) {  std::cout << "\n\MAIN THREAD is leaving critical section\n";  LeaveCriticalSection(&cs);  }  std::cout << "Main PID = " << pid << ", Parent Thread: " << "tid - " << tid << ": " << i << '\n';  Sleep(100);  }  WaitForSingleObject(hOne, INFINITE);  WaitForSingleObject(hTwo, INFINITE);  DeleteCriticalSection(&cs);  CloseHandle(hOne);  CloseHandle(hTwo);  system("pause");  }  DWORD WINAPI ChildThrOne() {  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 90; i++) {  if (i == 30) {  EnterCriticalSection(&cs);  std::cout << "\n\tThr\_1 is entering critical section\n";  std::cout << "--------------------------------------------------------\n";  }  if (i == 60) {  std::cout << "\n\tThr\_1 is leaving critical section\n";  LeaveCriticalSection(&cs);  }  std::cout << "First PID = " << pid << ", Child Thread: tid- " << tid << ": " << i << '\n';  Sleep(100);  }  LeaveCriticalSection(&cs);  return 0;  }  DWORD WINAPI ChildThrTwo() {  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 90; i++) {  if (i == 30) {  EnterCriticalSection(&cs);  std::cout << "\n\tThr\_2 is entering critical section\n";  std::cout << "--------------------------------------------------------\n";  }  if (i == 60) {  std::cout << "\n\tThr\_2 is leaving critical section\n";  LeaveCriticalSection(&cs);  }  std::cout << "Second PID = " << pid << ", Child Thread: tid- " << tid << ": " << i << '\n';  Sleep(100);  }  LeaveCriticalSection(&cs);  return 0;  } |

Цикл до 30 итерации(входа в критическую секцию):

Цикл в критической секции:

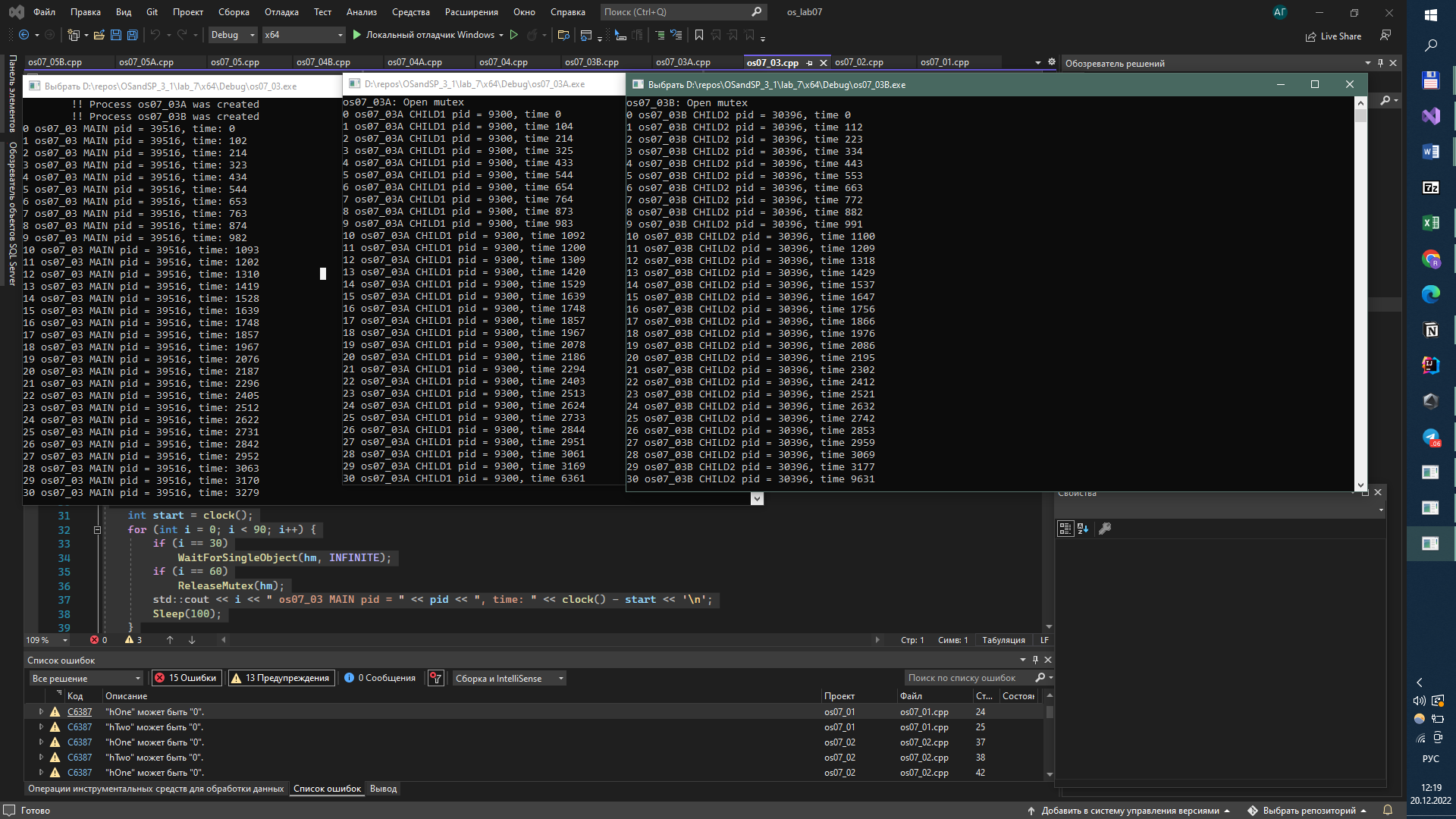


**Задание 03. Windows**

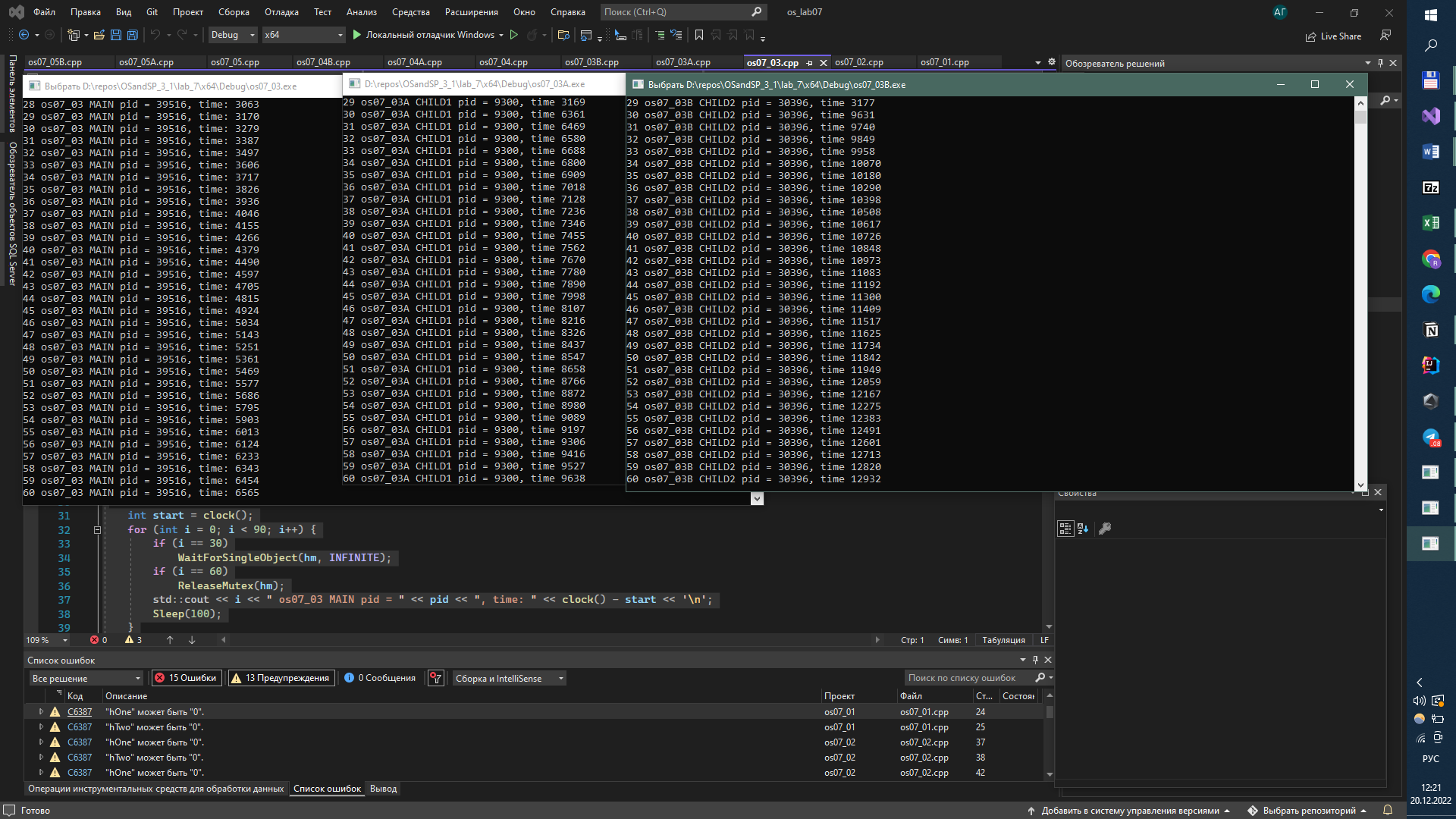
1. Разработайте приложение **OS07\_03**, запускающее два дочерних процесса **OS07\_03A** и **OS07\_03B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS07\_03** синхронизирует выполнение процессов **OS07\_03**, **OS07\_03A** и **OS07\_03B** с помощью механизма **mutex.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main() {  LPCWSTR pathA = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_03A.exe";  LPCWSTR pathB = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_03B.exe";  DWORD pid = GetCurrentProcessId();  STARTUPINFO si1;  STARTUPINFO si2;  PROCESS\_INFORMATION pi1;  PROCESS\_INFORMATION pi2;  ZeroMemory(&si1, sizeof(STARTUPINFO)); si1.cb = sizeof(STARTUPINFO);  ZeroMemory(&si2, sizeof(STARTUPINFO)); si2.cb = sizeof(STARTUPINFO);  // чтобы остальные объекты нашли этот мьютекс, ему дается имя  // false - создавший его поток не является владельцем  HANDLE hm = CreateMutex(NULL, FALSE, L"Mutex");  if (CreateProcess(pathA, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1))  std::cout << "\t!! Process os07\_03A was created\n";  else  std::cout << "\t!! Process os07\_03A wasn't created\n";  if (CreateProcess(pathB, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2))  std::cout << "\t!! Process os07\_03B was created\n";  else  std::cout << "\t!! Process os07\_03B wasn't created\n";  int start = clock();  for (int i = 0; i < 90; i++) {  if (i == 30)  WaitForSingleObject(hm, INFINITE);  if (i == 60)  ReleaseMutex(hm);  std::cout << i << " os07\_03 MAIN pid = " << pid << ", time: " << clock() - start << '\n';  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(hm);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  system("pause");  return 0;  } |

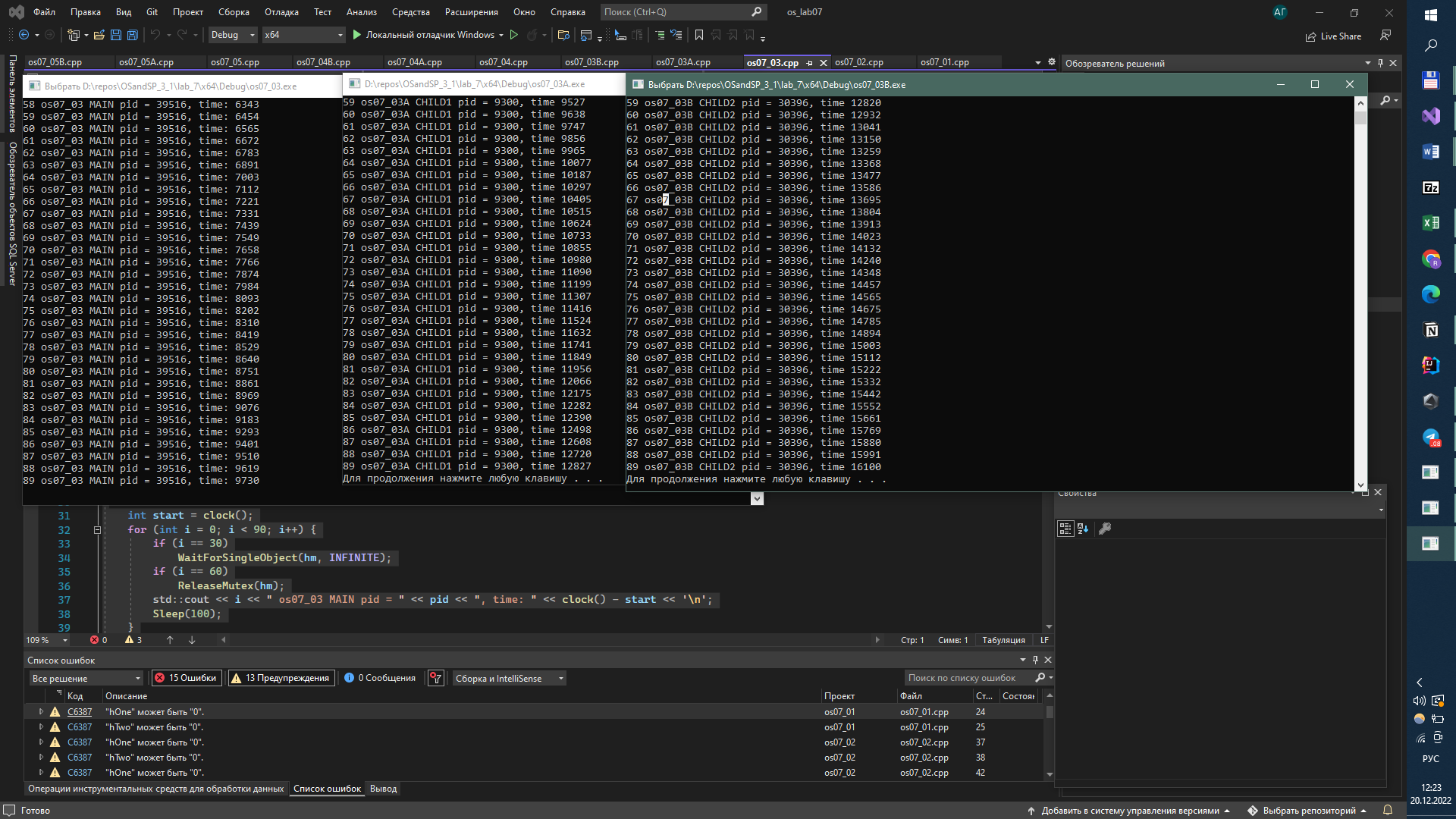
До 30 итерации(входа в мьютекс):



[30;60] итерации цикла:



[60;90] итерации цикла:

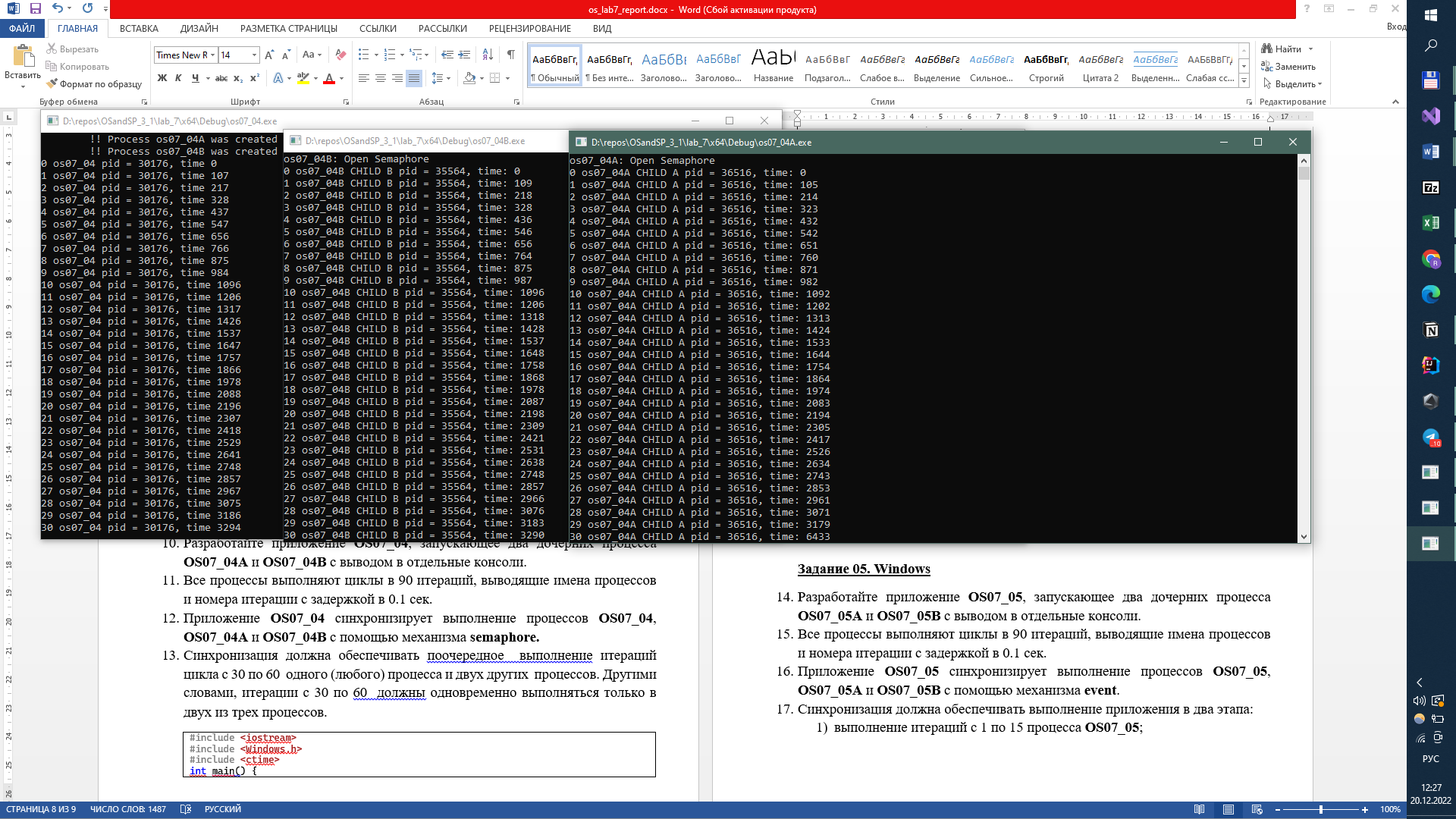


**Задание 04. Windows**

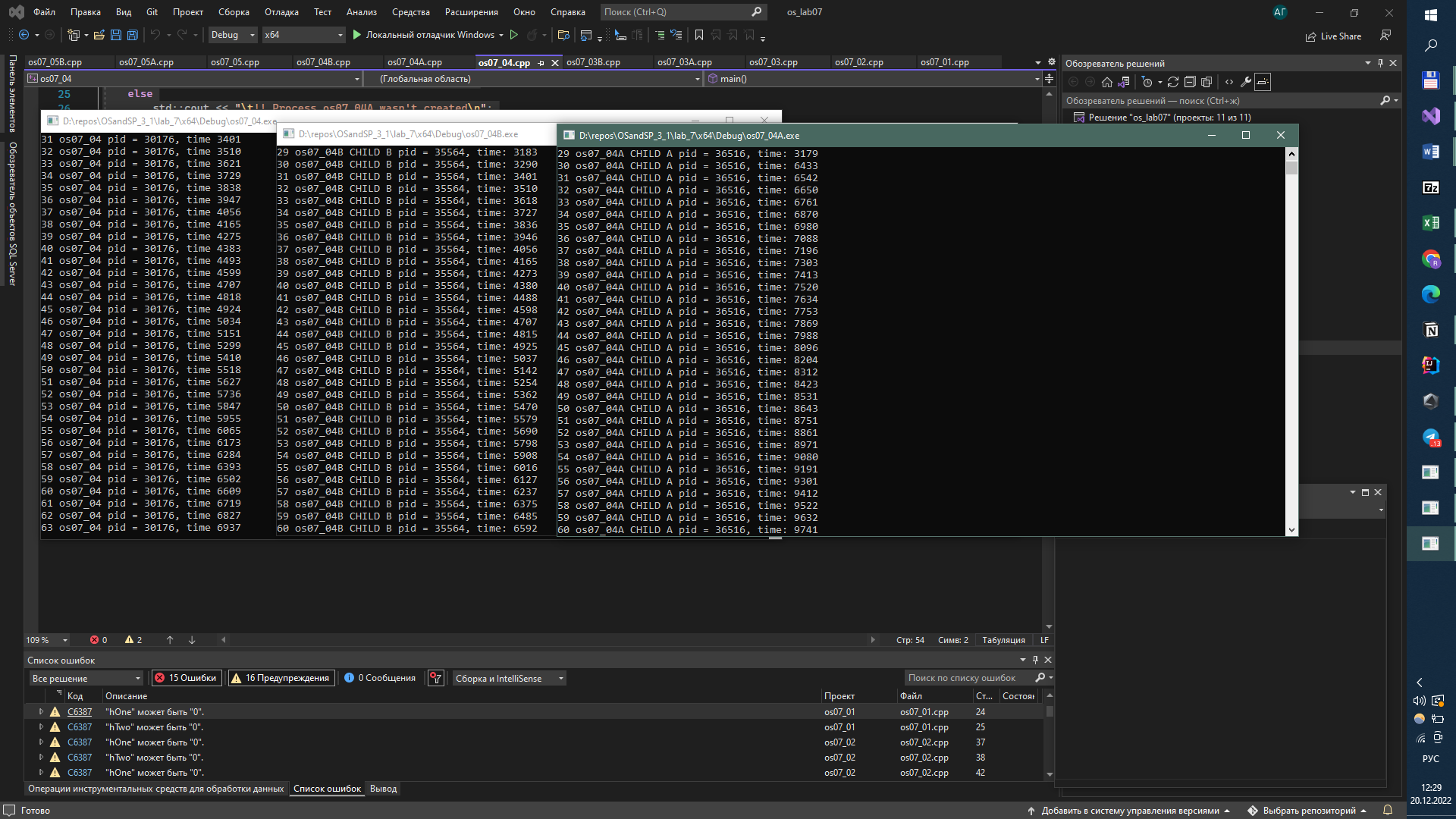
1. Разработайте приложение **OS07\_04**, запускающее два дочерних процесса **OS07\_04A** и **OS07\_04B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS07\_04** синхронизирует выполнение процессов **OS07\_04**, **OS07\_04A** и **OS07\_04B** с помощью механизма **semaphore.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 одного (любого) процесса и двух других процессов. Другими словами, итерации с 30 по 60 должны одновременно выполняться только в двух из трех процессов.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  #include <ctime>  int main() {  // атрибуты защиты, кол-во свободных ресурсов при инициализации, всего, имя  HANDLE hs = CreateSemaphore(NULL, 2, 3, L"Semaphore");  LPCWSTR pathA = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_04A.exe";  LPCWSTR pathB = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_04B.exe";  DWORD pid = GetCurrentProcessId();  STARTUPINFO si1;  STARTUPINFO si2;  PROCESS\_INFORMATION pi1;  PROCESS\_INFORMATION pi2;  ZeroMemory(&si1, sizeof(STARTUPINFO));  ZeroMemory(&si2, sizeof(STARTUPINFO));  si1.cb = sizeof(STARTUPINFO);  si2.cb = sizeof(STARTUPINFO);  if (CreateProcess(pathA, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1))  std::cout << "\t!! Process os07\_04A was created\n";  else  std::cout << "\t!! Process os07\_04A wasn't created\n";  if (CreateProcess(pathB, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2))  std::cout << "\t!! Process os07\_04B was created\n";  else  std::cout << "\t!! Process os07\_04B wasn't created\n";  LONG prevcount = 0;  int start = clock();  for (int i = 0; i < 90; i++) {  if (i == 30)  WaitForSingleObject(hs, INFINITE);  if (i == 60)  ReleaseSemaphore(hs, 1, &prevcount);  std::cout << i << " os07\_04 pid = " << pid << ", time " << clock() - start << '\n';  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(hs);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  system("pause");  return 0;  } |

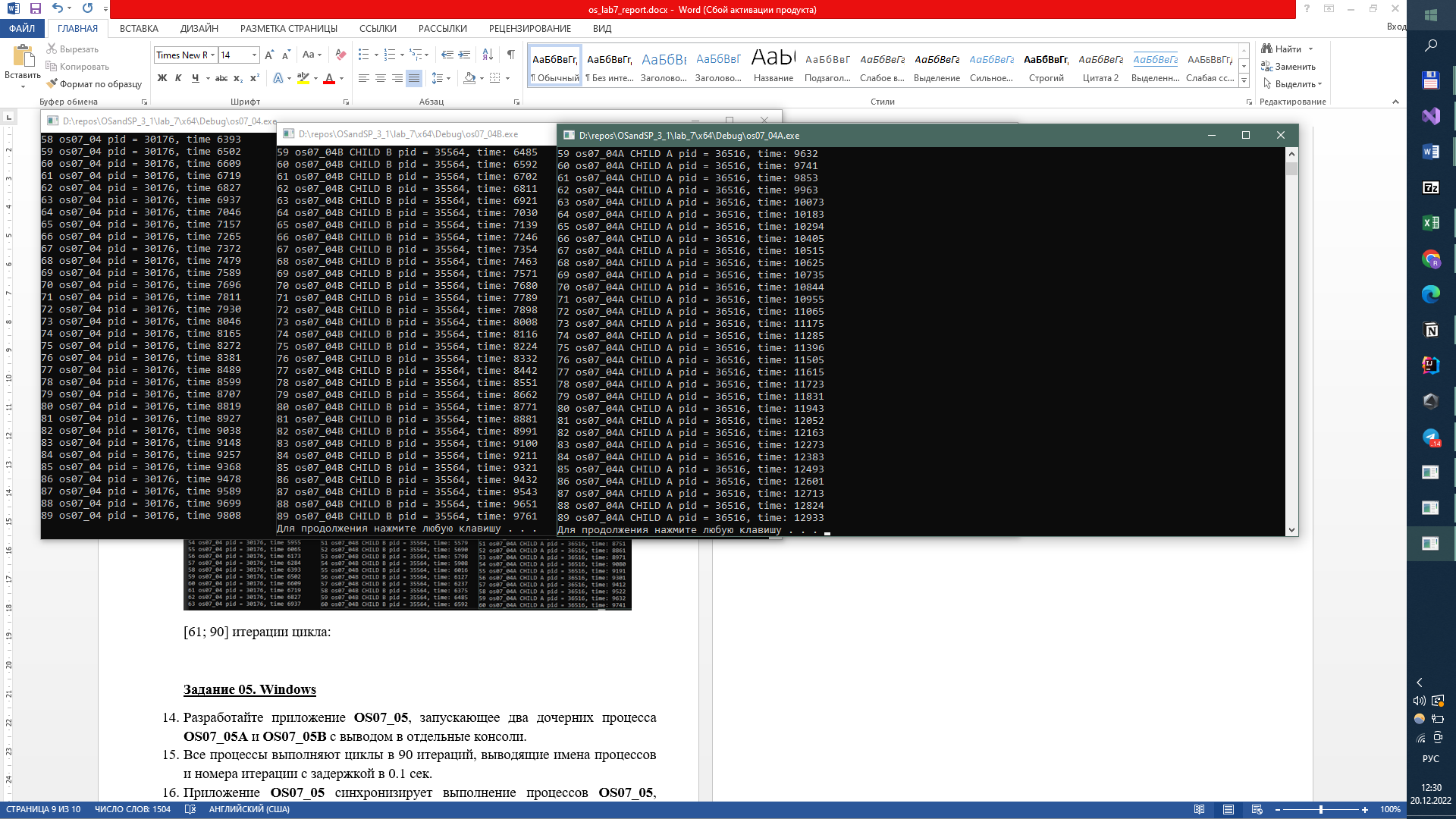
С 0 по 30 итерацию (До входа в семафор):



[31; 60] итерации цикла:



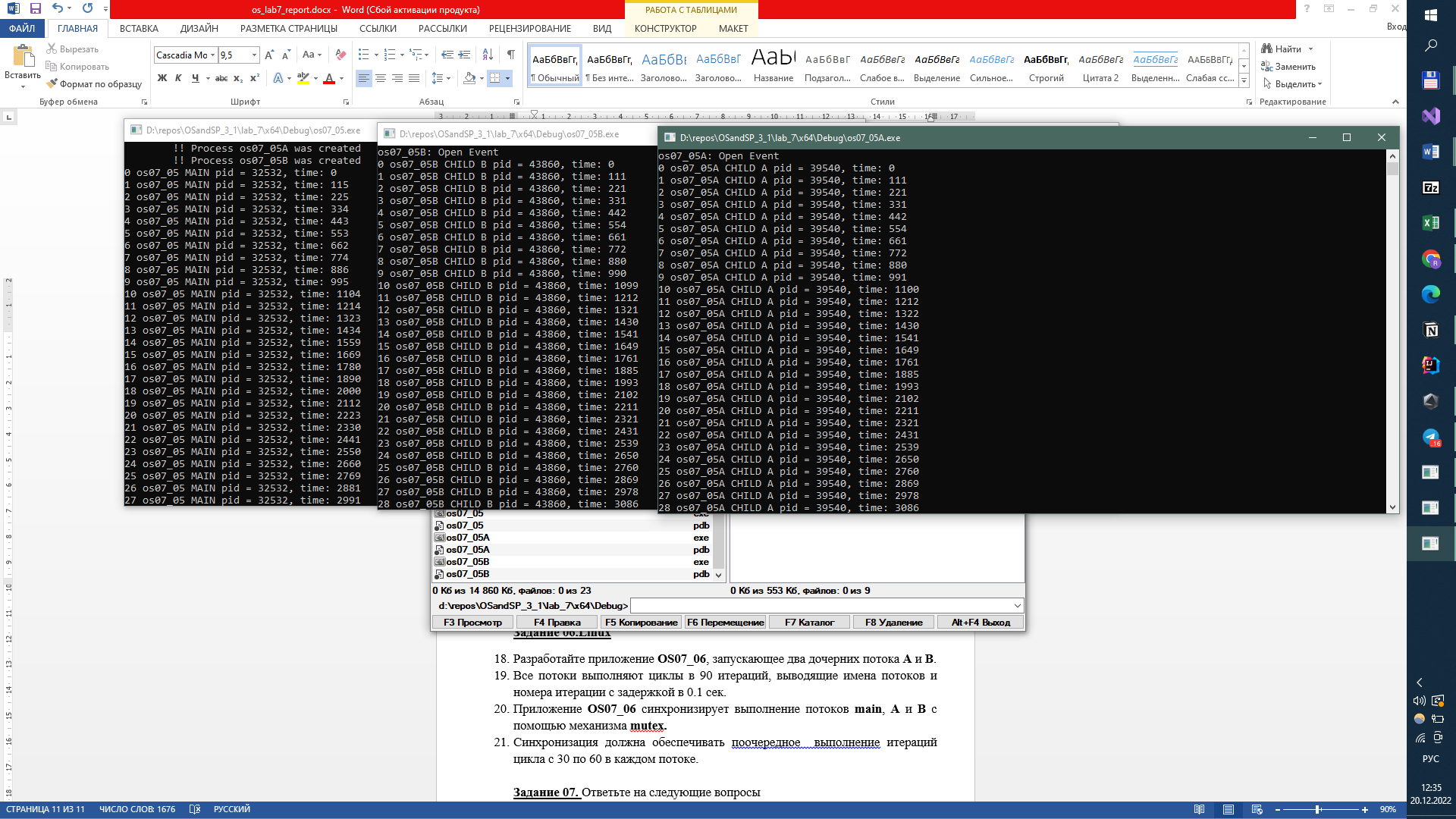
[61; 90] итерации цикла:



**Задание 05. Windows**

1. Разработайте приложение **OS07\_05**, запускающее два дочерних процесса **OS07\_05A** и **OS07\_05B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS07\_05** синхронизирует выполнение процессов **OS07\_05**, **OS07\_05A** и **OS07\_05B** с помощью механизма **event**.
4. Синхронизация должна обеспечивать выполнение приложения в два этапа:
5. выполнение итераций с 1 по 15 процесса **OS07\_05**;
6. одновременное выполнение всех трех процессов: **OS07\_05** – продолжает выполнение итераций; процессы **OS07\_05A** и **OS07\_05B** выполняются начиная с первой итерации.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  #include <ctime>  int main() {  DWORD pid = GetCurrentProcessId();  // атрибут защиты, ручной сброс, начальное состояние несигнальное, имя  HANDLE he = CreateEvent(NULL, TRUE, FALSE, L"Event");  LPCWSTR pathA = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_05A.exe";  LPCWSTR pathB = L"D:\\repos\\OSandSP\_3\_1\\lab\_7\\x64\\Debug\\os07\_05B.exe";  STARTUPINFO si1;  STARTUPINFO si2;  PROCESS\_INFORMATION pi1;  PROCESS\_INFORMATION pi2;  ZeroMemory(&si1, sizeof(STARTUPINFO));  ZeroMemory(&si2, sizeof(STARTUPINFO));  si1.cb = sizeof(STARTUPINFO);  si2.cb = sizeof(STARTUPINFO);  if (CreateProcess(pathA, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1))  std::cout << "\t!! Process os07\_05A was created\n";  else  std::cout << "\t!! Process os07\_05A wasn't created\n";  if (CreateProcess(pathB, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2))  std::cout << "\t!! Process os07\_05B was created\n";  else  std::cout << "\t!! Process os07\_05B wasn't created\n";  int start = clock();  LONG prevcount = 0;  for (int i = 0; i < 90; i++) {  // устанавливает объект события в сигнальное состояние  if (i == 15)  PulseEvent(he);  std::cout << i << " os07\_05 MAIN pid = " << pid << ", time: " << clock() - start << '\n';  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(he);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  system("pause");  return 0;  } |



**Задание 06.Linux**

1. Разработайте приложение **OS07\_06**, запускающее два дочерних потока **A** и **B**.
2. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS07\_06** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма **mutex.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.

**Задание 07.** Ответьте на следующие вопросы

1. Дайте определение понятию «синхронизация потоков».
2. Объясните понятие «взаимная блокировка».
3. Перечислите механизмы авторизации OS.
4. Поясните в чем разница между механизмом **mutex** и **semaphore**.
5. Почему **mutex,** **semaphore, event** создают объект ядра OS, а **critical section** нет.