Лабораторная работа 06

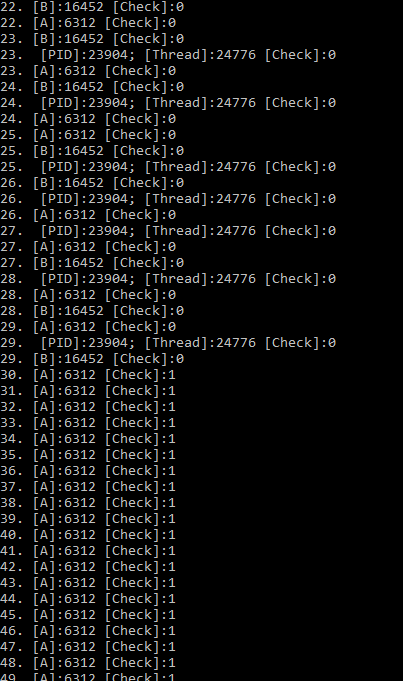
Синхронизация

OC, ПОИТ-3

**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.

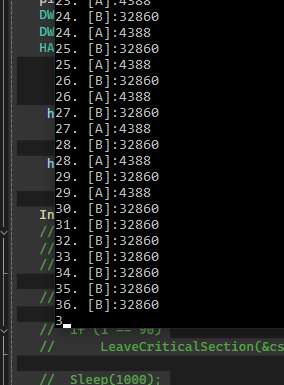
|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  int status;  DWORD pid = NULL;  void EnterCriticalSectionAssembly()  {  \_asm // спецификатор  {  CriticalSection: // метка  // префикс для обеспечения атомарного выполнения команды  lock bts status, 0;  jc CriticalSection // Если бит уже был установлен, переходит к метке CriticalSection и ждет  }  }  void LeaveCriticalSectionAssembly()  {  \_asm lock btr status, 0 // которая проверяет значение бита 0 и сбрасывает его в 0.  // Сброс бита 0 переменной означает, что поток выходит из критической секции и освобождает ее для других потоков  }  DWORD WINAPI A() {  DWORD tid = GetCurrentThreadId();  for (int i = 1; i <= 90; i++) {  if (i == 30)  EnterCriticalSectionAssembly();  cout << i << ". " << "[A]:" << tid << " [Check]:" << check << endl;  if (i == 60)  LeaveCriticalSectionAssembly();  Sleep(100);  }  cout << "---------> A: FINISHED\n";  return 0;  }  DWORD WINAPI B() {  DWORD tid = GetCurrentThreadId();  for (int i = 1; i <= 90; i++) {  if (i == 30)  EnterCriticalSectionAssembly();  cout << i << ". " << "[B]:" << tid << " [Check]:" << check << endl;  if (i == 60)  LeaveCriticalSectionAssembly();  Sleep(100);  }  cout << "---------> B: FINISHED\n";  return 0;  }  int main()  {  pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  DWORD ChildId = NULL;  HANDLE hChild[2];  hChild[0] = CreateThread(NULL, 0,  (LPTHREAD\_START\_ROUTINE)A, NULL, 0, &ChildId);  hChild[1] = CreateThread(NULL, 0,  (LPTHREAD\_START\_ROUTINE)B, NULL, 0, &ChildId);  for (int i = 1; i <= 90; i++) {  if(i==30)  EnterCriticalSectionAssembly();  cout << i << ". [PID]:" << pid << "; [Thread]:" << tid << " [Check]:"<< check << endl;  if (i == 90)  LeaveCriticalSectionAssembly();  Sleep(100);  }  cout << "---------> MAIN: FINISHED\n";  WaitForMultipleObjects(2, hChild, TRUE, INFINITE);  CloseHandle(hChild[0]);  CloseHandle(hChild[1]);  return 0;   1. } |



**Задание 02. Windows**

1. Разработайте приложение **OS06\_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 **OS06\_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.

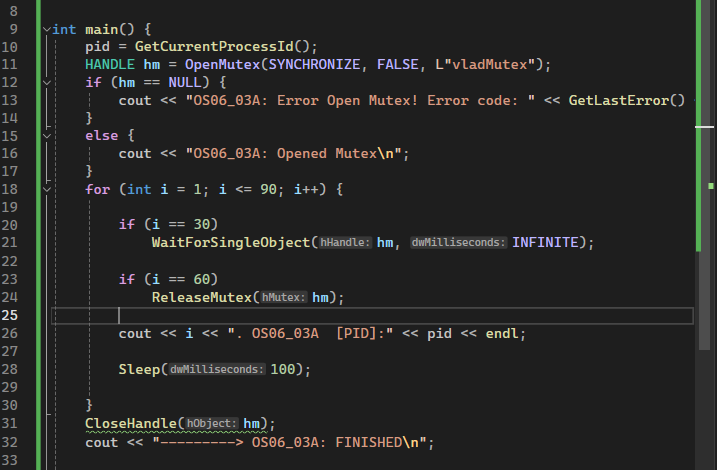
|  |
| --- |
| #include <iostream>  #include <Windows.h>  using namespace std;  DWORD pid = NULL;  CRITICAL\_SECTION cs;  DWORD WINAPI A() {  DWORD tid = GetCurrentThreadId();  for (int i = 1; i <= 90; i++) {  if (i == 30)  EnterCriticalSection(&cs);  cout << i << ". " << "[A]:" << tid << endl;  if (i == 60)  LeaveCriticalSection(&cs);  Sleep(1000);  }  cout << "---------> A: FINISHED\n";  return 0;  }  DWORD WINAPI B() {  DWORD tid = GetCurrentThreadId();  for (int i = 1; i <= 90; i++) {  if (i == 30)  EnterCriticalSection(&cs);  cout << i << ". " << "[B]:" << tid << endl;  if (i == 60)  LeaveCriticalSection(&cs);  Sleep(1000);  }  cout << "---------> B: FINISHED\n";  return 0;  }  int main() {  pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  DWORD ChildId = NULL;  HANDLE hChild[2];  hChild[0] = CreateThread(NULL, 0,  (LPTHREAD\_START\_ROUTINE)A,NULL,0,&ChildId);  hChild[1] = CreateThread(NULL, 0,  (LPTHREAD\_START\_ROUTINE)B, NULL, 0, &ChildId);  InitializeCriticalSection(&cs);  for (int i = 1; i <= 90; i++) {  if(i==30)  EnterCriticalSection(&cs);  cout << i << ". MAIN [PID]:" << pid << "; [Thread]:" << tid << endl;  if (i == 60)  LeaveCriticalSection(&cs);  Sleep(1000);  }  cout << "---------> MAIN: FINISHED\n";  WaitForMultipleObjects(2,hChild,TRUE,INFINITE);  DeleteCriticalSection(&cs);  CloseHandle(hChild[0]);  CloseHandle(hChild[1]);  return 0;  } |

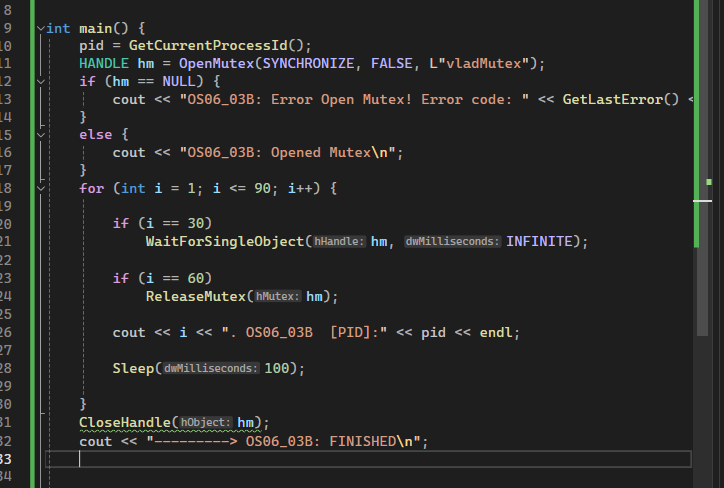


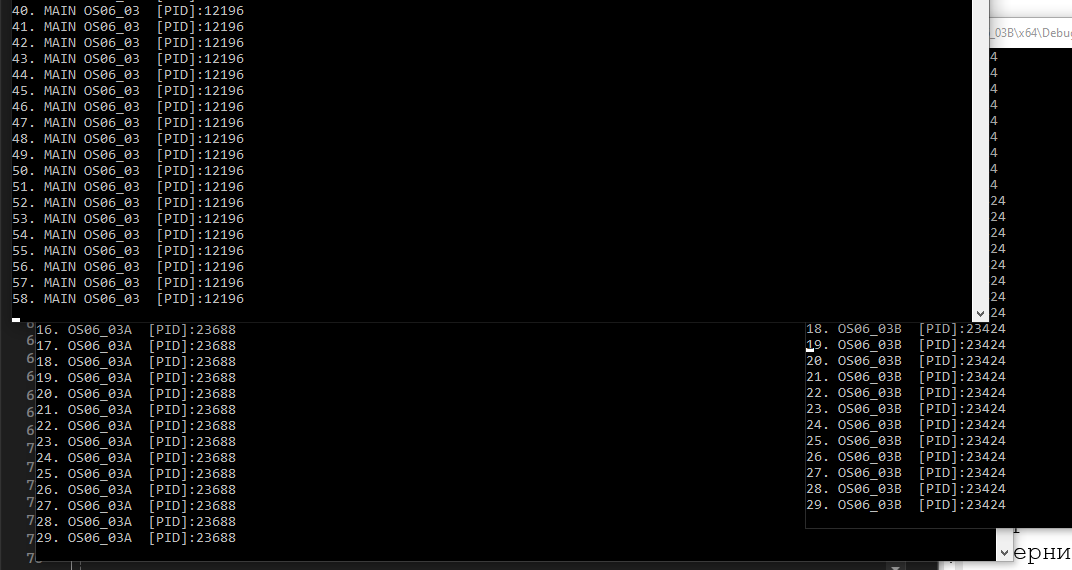
**Задание 03. Windows**

1. Разработайте приложение **OS06\_03**, запускающее два дочерних процесса **OS06\_03A** и **OS06\_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 **OS06\_03** синхронизирует выполнение процессов **OS06\_03**, **OS06\_03A** и **OS06\_03B** с помощью механизма **mutex.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60.

|  |
| --- |
| 1. #include <iostream> 2. #include <Windows.h> 3. using namespace std; 4. DWORD pid = NULL; 5. int main() { 6. pid = GetCurrentProcessId(); 7. LPCWSTR an1 = L"C:\\Users\\Влад\\Desktop\\OS06\_03A\\x64\\Debug\\OS06\_03A.exe"; 8. LPCWSTR an2 = L"C:\\Users\\Влад\\Desktop\\OS06\_03B\\x64\\Debug\\OS06\_03B.exe"; 9. PROCESS\_INFORMATION pi1; 10. PROCESS\_INFORMATION pi2; 11. HANDLE hm = CreateMutex(NULL, FALSE, L"vladMutex"); 12. if (hm == NULL) { 13. cout << "Error creating mutex: " << GetLastError() << endl; 14. return 1; 15. } 16. else { 17. cout << "Mutex created \n"; 18. } 19. { 20. STARTUPINFO si; 21. ZeroMemory(&si, sizeof(STARTUPINFO)); 22. si.cb = sizeof(STARTUPINFO); 23. if (CreateProcess( 24. an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi1 25. )) { 26. cout << "------> OS06\_03A: Created\n"; 27. } 28. else { 29. cout << "------> OS06\_03A: Not Created\n"; 30. } 31. } 32. { 33. STARTUPINFO si; 34. ZeroMemory(&si, sizeof(STARTUPINFO)); 35. si.cb = sizeof(STARTUPINFO); 36. if (CreateProcess( 37. an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi2 38. )) { 39. cout << "------> OS06\_03B: Created\n"; 40. } 41. else { 42. cout << "------> OS06\_03B: Not Created\n"; 43. } 44. } 45. for (int i = 1; i <= 90; i++) { 46. if (i == 30) { 47. WaitForSingleObject(hm, INFINITE); 48. } 49. if (i == 60) { 50. ReleaseMutex(hm); 51. } 52. cout << i << ". MAIN OS06\_03 [PID]:" << pid << endl; 53. Sleep(100); 54. } 55. WaitForSingleObject(pi1.hProcess, INFINITE); 56. WaitForSingleObject(pi2.hProcess, INFINITE); 57. CloseHandle(hm); 58. CloseHandle(pi1.hProcess); 59. CloseHandle(pi2.hProcess); 60. cout << "---------> MAIN: FINISHED\n"; 61. return 0; 62. } |



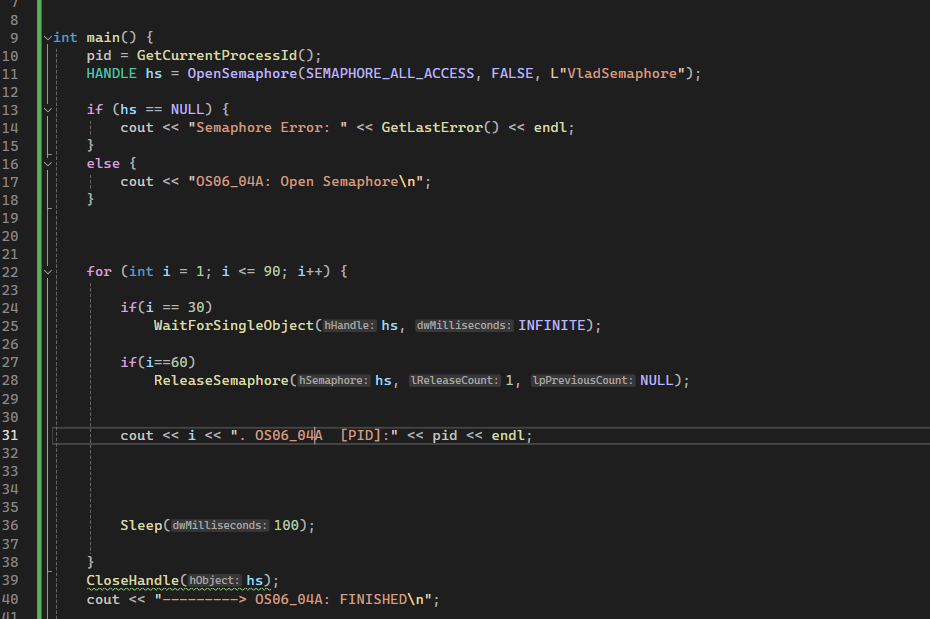
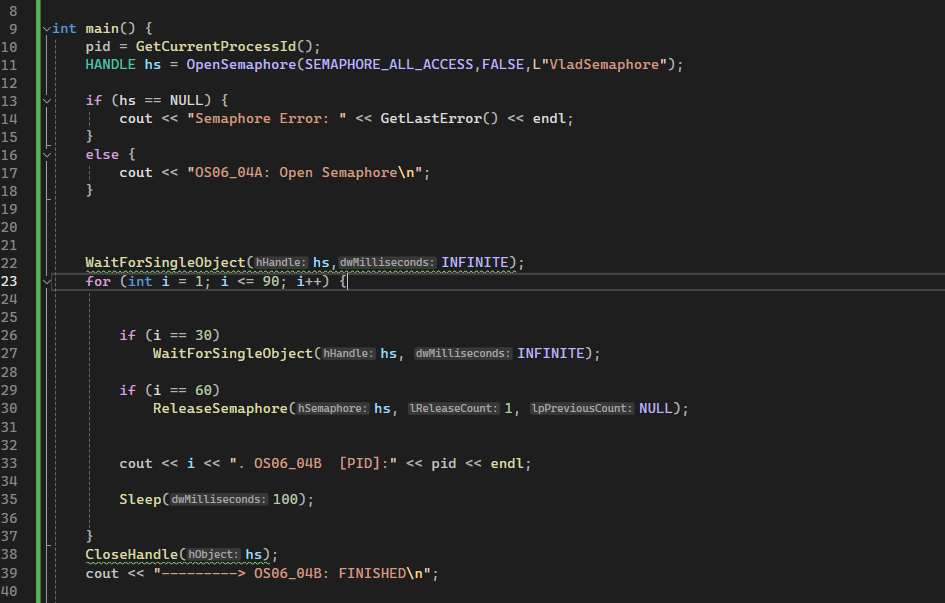


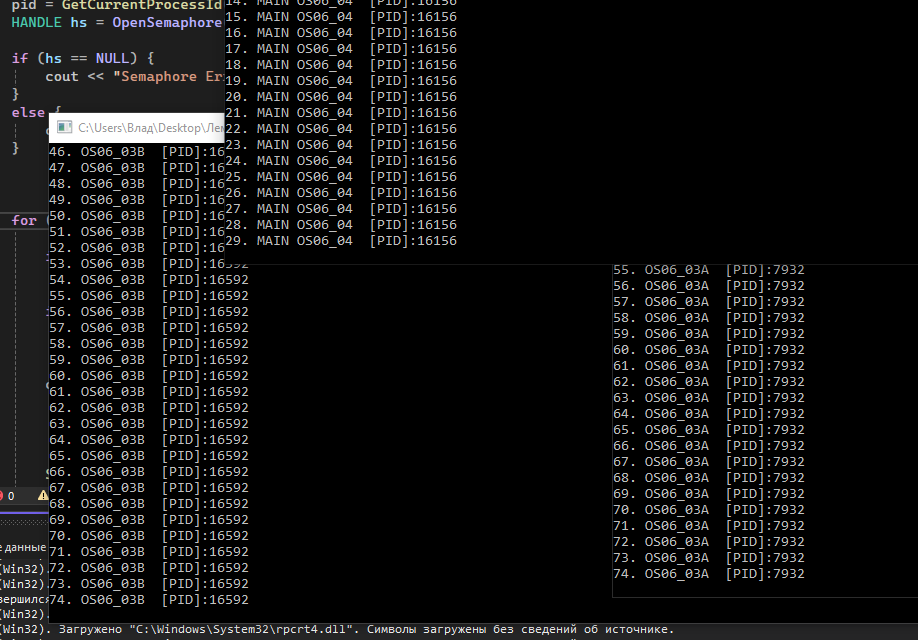


**Задание 04. Windows**

1. Разработайте приложение **OS06\_04**, запускающее два дочерних процесса **OS06\_04A** и **OS06\_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 **OS06\_04** синхронизирует выполнение процессов **OS06\_04**, **OS06\_04A** и **OS06\_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.

|  |
| --- |
| 1. #include <iostream> 2. #include <Windows.h> 3. using namespace std; 4. DWORD pid = NULL; 5. int main() { 6. pid = GetCurrentProcessId(); 7. LPCWSTR an1 = L"C:\\Users\\Влад\\Desktop\\Лемешевский\_ОС\_ИСиТ\_3\_2\\Lab\_06\\Lab06\\x64\\Debug\\OS06\_04A.exe"; 8. LPCWSTR an2 = L"C:\\Users\\Влад\\Desktop\\Лемешевский\_ОС\_ИСиТ\_3\_2\\Lab\_06\\Lab06\\x64\\Debug\\OS06\_04B.exe"; 9. PROCESS\_INFORMATION pi1; 10. PROCESS\_INFORMATION pi2; 12. HANDLE hs = CreateSemaphore(NULL,2,2,L"VladSemaphore"); 13. if (hs != NULL) { 14. cout << "Semaphore created\n"; 15. } 16. { 17. STARTUPINFO si; 18. ZeroMemory(&si, sizeof(STARTUPINFO)); 19. si.cb = sizeof(STARTUPINFO); 20. if (CreateProcess( 21. an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi1 22. )) { 23. cout << "------> OS06\_04A: Created\n"; 24. } 25. else { 26. cout << "------> OS06\_04A: Not Created\n"; 27. } 28. } 29. { 30. STARTUPINFO si; 31. ZeroMemory(&si, sizeof(STARTUPINFO)); 32. si.cb = sizeof(STARTUPINFO); 33. if (CreateProcess( 34. an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi2 35. )) { 36. cout << "------> OS06\_04B: Created\n"; 37. } 38. else { 39. cout << "------> OS06\_04B: Not Created\n"; 40. } 41. } 42. for (int i = 1; i <= 90; i++) { 43. if (i == 30) { 44. WaitForSingleObject(hs, INFINITE); // Войти в критическую секцию 45. } 46. cout << i << ". MAIN OS06\_04 [PID]:" << pid << endl; 47. Sleep(100); 48. if (i == 60) { 49. ReleaseSemaphore(hs, 1, NULL); // Выйти из критической секции 50. } 51. Sleep(100); 52. } 53. LONG prevcount; 54. ReleaseSemaphore(hs,1,&prevcount); 55. WaitForSingleObject(pi1.hProcess, INFINITE); 56. WaitForSingleObject(pi2.hProcess, INFINITE); 57. CloseHandle(hs); 58. CloseHandle(pi1.hProcess); 59. CloseHandle(pi2.hProcess); 60. cout << "---------> MAIN: FINISHED\n"; 61. return 0; 62. } |

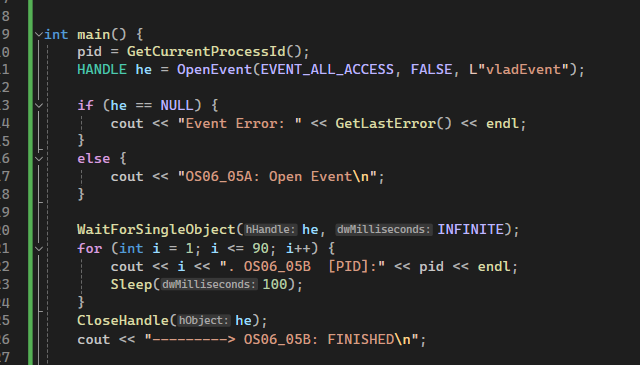


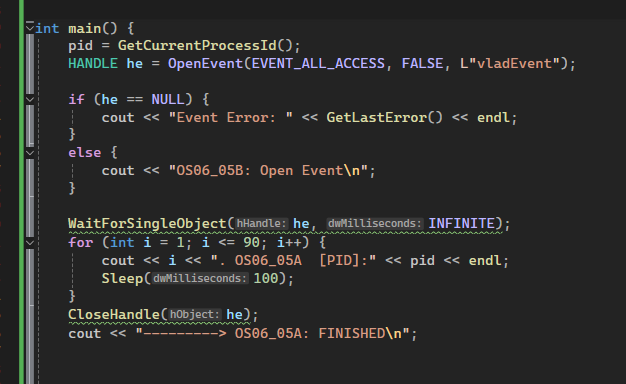


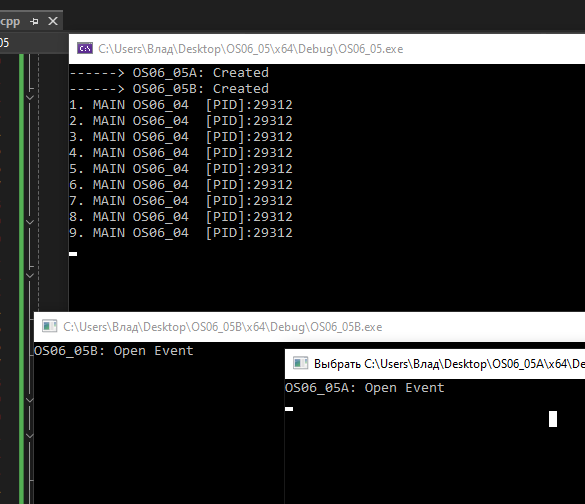
**Задание 05. Windows**

1. Разработайте приложение **OS06\_05**, запускающее два дочерних процесса **OS06\_05A** и **OS06\_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 **OS06\_05** синхронизирует выполнение процессов **OS06\_05**, **OS06\_05A** и **OS06\_05B** с помощью механизма **event**.
4. Синхронизация должна обеспечивать выполнение приложения в два этапа:
5. выполнение итераций с 1 по 15 процесса **OS06\_05**;
6. одновременное выполнение всех трех процессов: **OS06\_05** – продолжает выполнение итераций; процессы **OS06\_05A** и **OS06\_05B** выполняются начиная с первой итерации.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  using namespace std;  DWORD pid = NULL;  int main() {  pid = GetCurrentProcessId();  LPCWSTR an1 = L"C:\\Users\\Влад\\Desktop\\OS06\_05A\\x64\\Debug\\OS06\_05A.exe";  LPCWSTR an2 = L"C:\\Users\\Влад\\Desktop\\OS06\_05B\\x64\\Debug\\OS06\_05B.exe";  PROCESS\_INFORMATION pi1;  PROCESS\_INFORMATION pi2;  HANDLE he = CreateEvent(NULL, TRUE, FALSE, L"vladEvent");  {  STARTUPINFO si;  ZeroMemory(&si, sizeof(STARTUPINFO));  si.cb = sizeof(STARTUPINFO);  if (CreateProcess(  an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi1  )) {  cout << "------> OS06\_05A: Created\n";  }  else {  cout << "------> OS06\_05A: Not Created\n";  }  }  {  STARTUPINFO si;  ZeroMemory(&si, sizeof(STARTUPINFO));  si.cb = sizeof(STARTUPINFO);  if (CreateProcess(  an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi2  )) {  cout << "------> OS06\_05B: Created\n";  }  else {  cout << "------> OS06\_05B: Not Created\n";  }  }  for (int i = 1; i <= 90; i++) {  if (i == 16) {  SetEvent(he); // Разрешает другим процессам начать выполнение после 15 итераций  }  cout << i << ". MAIN OS06\_04 [PID]:" << pid << endl;  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(he);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  cout << "---------> MAIN: FINISHED\n";  return 0;  } |

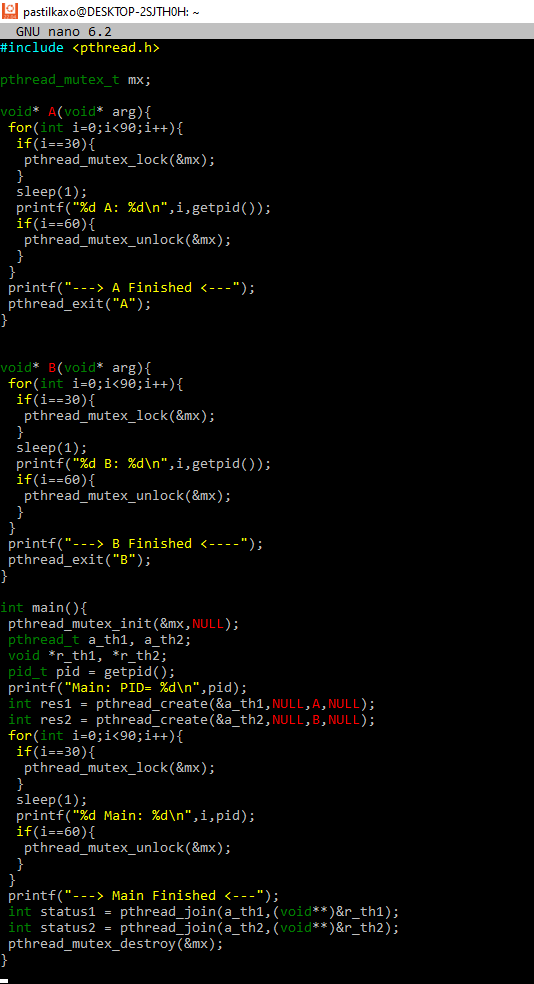


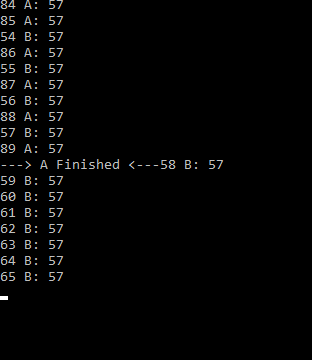
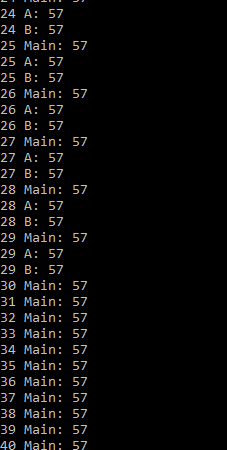




**Задание 06.Linux**

1. Разработайте приложение **OS06\_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 **OS06\_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».

Синхронизация потоков — это механизм управления выполнением потоков для предотвращения конфликтов при совместном доступе к общим ресурсам. Синхронизация необходима для упорядочивания выполнения потоков и обеспечения корректности и безопасности данных, когда несколько потоков могут одновременно обращаться к одному и тому же ресурсу.

1. Объясните понятие «взаимная блокировка».

Взаимная блокировка (или "deadlock") — это состояние, при котором два или более потока или процесса зацикливаются, ожидая освобождения ресурсов, занятых друг другом. Взаимная блокировка приводит к полной остановке работы этих потоков или процессов, так как ни один из них не может продолжить выполнение без освобождения требуемых ресурсов.

1. Перечислите механизмы авторизации OS.

Механизмы авторизации, используемые в операционных системах, включают:

* **ACL (Access Control List)**: список управления доступом, задающий права пользователей и групп на доступ к файлам и другим ресурсам.
* **RBAC (Role-Based Access Control)**: управление доступом на основе ролей, где пользователям назначаются роли с определенными разрешениями.
* **DAC (Discretionary Access Control)**: дискреционное управление доступом, при котором владельцы ресурсов могут предоставлять доступ другим пользователям.
* **MAC (Mandatory Access Control)**: мандатное управление доступом, устанавливающее правила доступа, независимые от воли пользователя, например, на основе уровней безопасности.

1. Поясните в чем разница между механизмом **mutex** и **semaphore**.

 **Mutex (взаимное исключение)** используется для обеспечения эксклюзивного доступа к ресурсу. Только один поток может удерживать mutex в определенный момент времени. Mutex часто используется для синхронизации потоков внутри одного процесса.

 **Semaphore (семафор)** ограничивает количество потоков, которые могут одновременно использовать ресурс. Например, семафор с значением "2" позволяет доступ двум потокам одновременно. Семафоры могут применяться для синхронизации потоков как в рамках одного, так и нескольких процессов.

1. Почему **mutex,** **semaphore, event** создают объект ядра OS, а **critical section** нет.

Mutex, semaphore и event — это объекты ядра, потому что они могут использоваться для синхронизации потоков между разными процессами, а также предоставляют межпроцессную синхронизацию. Объекты ядра имеют контекст, общий для всех процессов и управляются на уровне операционной системы, что позволяет им быть доступными в пространстве ядра.  
Критическая секция, наоборот, предназначена для синхронизации потоков внутри одного процесса и не требует поддержки на уровне ядра. Это делает ее более быстрой и менее ресурсоемкой, поскольку она работает только в пространстве пользователя.

1. Объясните понятие «критическая секция».

Критическая секция – это некоторый участок кода, который в каждый момент времени может выполняться только одним из потоков. Если код, используемый для инициализации массива, поместить в критическую секцию, то другие потоки не смогут войти в этот участок кода до тех пор, пока первый поток не завершит его выполнение.

Event - это объекты синхронизации, которые используются для координации потоков в многозадачном окружении. Эти объекты позволяют одному или нескольким потокам «ожидать» (или блокироваться) до тех пор, пока другое событие не произойдет.

**Семафоры** — это объекты синхронизации, которые ограничивают доступ к ресурсу для определенного числа потоков одновременно. Семафор хранит счетчик, указывающий, сколько потоков могут одновременно иметь доступ к ресурсу. В отличие от мьютекса, который позволяет доступ только одному потоку, семафор может разрешать доступ нескольким потокам одновременно (например, если ресурс поддерживает параллельное использование).

### BTS и BTR

* **BTS** (Bit Test and Set) — команда, которая проверяет значение определенного бита в операнде и устанавливает его в 1.
  + В этой программе lock bts check, 0 проверяет значение первого бита переменной check и, если он не установлен, выставляет его в 1. Если бит уже установлен, это указывает, что критическая секция занята, и поток ждет.
* **BTR** (Bit Test and Reset) — команда, которая проверяет значение определенного бита и сбрасывает его в 0.
  + В коде lock btr check, 0 освобождает критическую секцию, сбрасывая бит check в 0, после чего другие потоки могут получить доступ к секции.