Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №6**

Выполнил:

Студент 3 курса 4 группы ФИТ

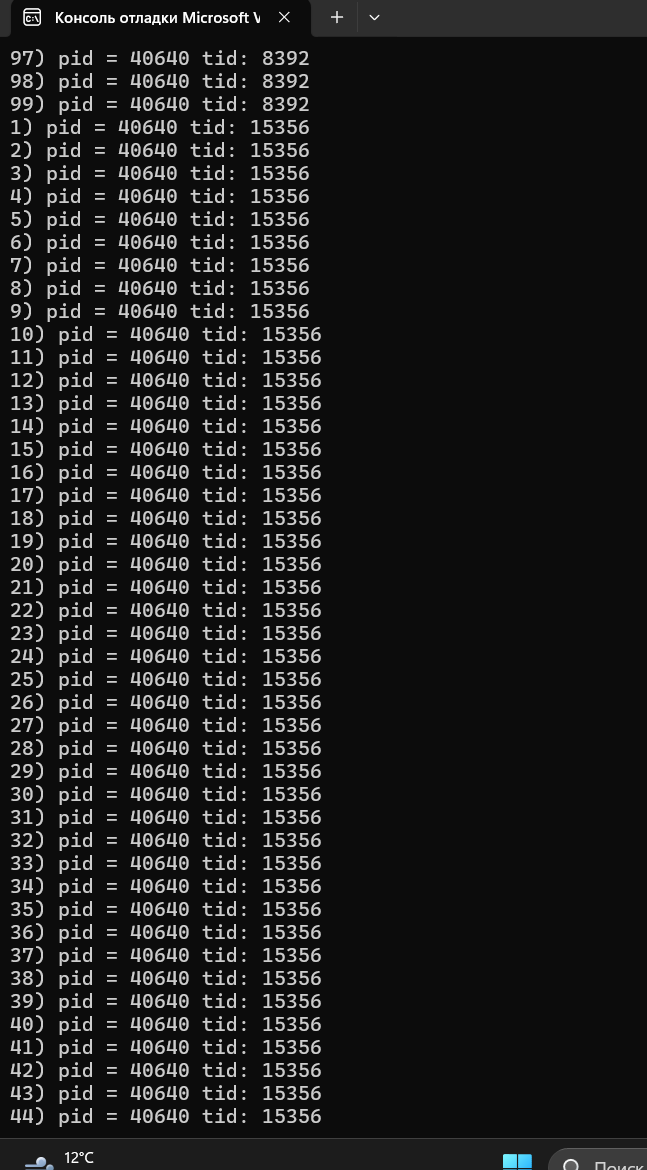
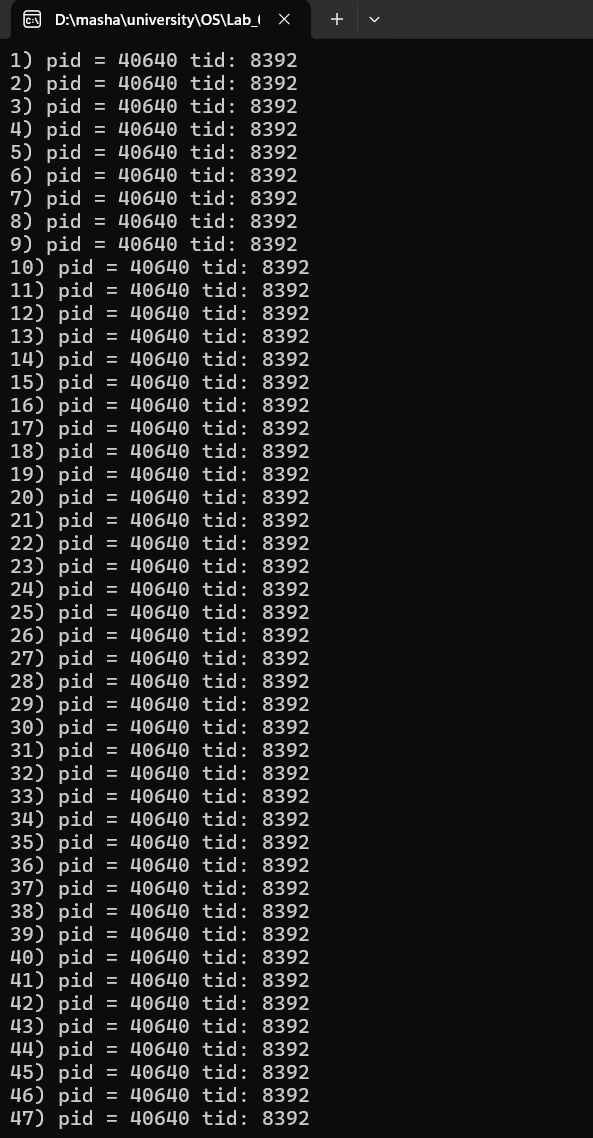
Сосновец Мария Игоревна

2024 г.

**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.

|  |
| --- |
| 1. #include <Windows.h> 2. #include <iostream> 3. using namespace std; 4. DWORD pid = NULL; 5. int check; 6. void EnterCriticalSectionAsm() 7. { 8. \_asm 9. { 10. CriticalSection: 11. lock bts check, 0; 12. jc CriticalSection 13. } 14. } 15. void LeaveCriticalSectionAsmчё() 16. { 17. \_asm lock btr check, 0 18. } 19. DWORD WINAPI ChildThread() 20. { 21. DWORD tid = GetCurrentThreadId(); 22. EnterCriticalSectionAsm(); 23. for (short i = 1; i < 100; i++) 24. { 25. cout << i << ") pid = " << pid << " tid: " << tid << ": " << endl; 26. Sleep(40); 27. } 28. LeaveCriticalSectionAsm(); 29. return 0; 30. } 31. int main() 32. { 33. pid = GetCurrentProcessId(); 34. DWORD tid = GetCurrentThreadId(); 35. DWORD ChildId = NULL; 36. HANDLE hChild = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThread, NULL, 0, &ChildId); 37. EnterCriticalSectionAsm(); 38. for (short i = 1; i < 100; i++) 39. { 40. cout << i << ") pid = " << pid << " tid: " << tid << ": " << endl; 41. Sleep(40); 42. } 43. LeaveCriticalSectionAsm(); 44. WaitForSingleObject(hChild, INFINITE); 45. CloseHandle(hChild); 46. return 0; 47. } |



**Задание 02. Windows**

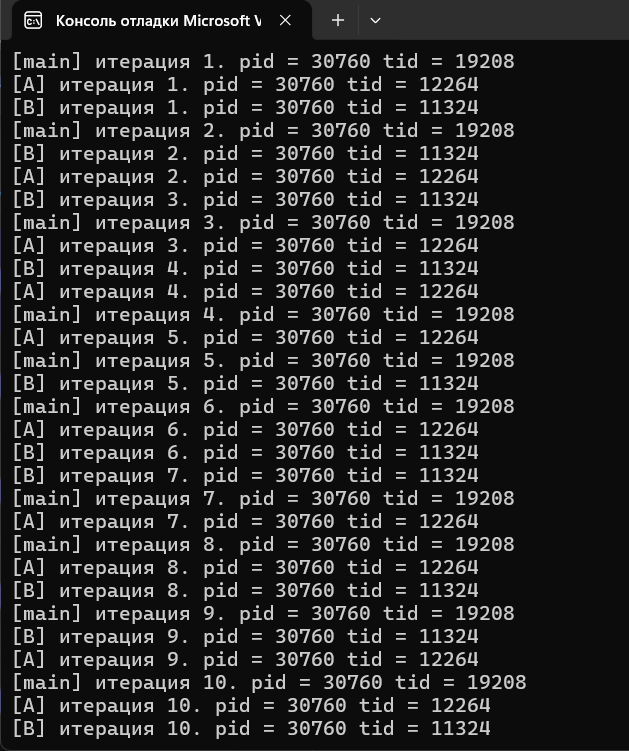
Разработайте приложение **OS06\_02**, запускающее два дочерних потока **A** и **B**.

Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.

Приложение **OS06\_02** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма **critical section.**

Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.

|  |
| --- |
| #include <iostream>  #include <windows.h>  #include <stdio.h>  using namespace std;  CRITICAL\_SECTION cs;  void WINAPI LoopFunction(const char\* displayed\_name)  {  int pid = GetCurrentProcessId();  int tid = GetCurrentThreadId();  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  EnterCriticalSection(&cs);  printf("[%s] итерация %d. pid = %d tid = %u\n", displayed\_name, i, pid, tid);  if (i == 60)  LeaveCriticalSection(&cs);  Sleep(100);  }  }  int main()  {  setlocale(LC\_ALL, "rus");  InitializeCriticalSection(&cs);  const int size = 2;  DWORD ChildId1, ChildId2;  HANDLE threads[size];  threads[0] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)LoopFunction, (LPVOID)"A", 0, &ChildId1);  threads[1] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)LoopFunction, (LPVOID)"B", 0, &ChildId2);  LoopFunction("main");  WaitForMultipleObjects(size, threads, TRUE, INFINITE);  for (int i = 0; i < size; ++i)  CloseHandle(threads[i]);  DeleteCriticalSection(&cs);  return 0;  } |



**Задание 03. Windows**

Разработайте приложение OS06\_03, запускающее два дочерних процесса OS06\_03A и OS06\_03B с выводом в отдельные консоли.

Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.

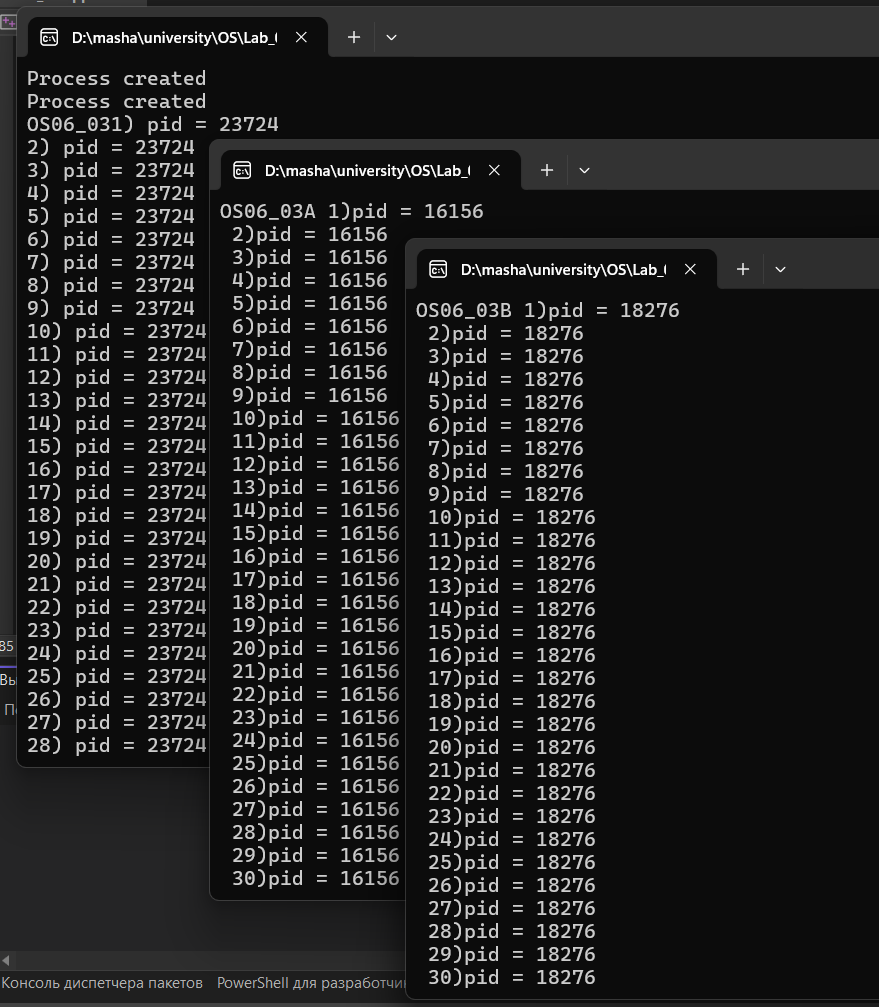
Приложение OS06\_03 синхронизирует выполнение процессов OS06\_03, OS06\_03A и OS06\_03B с помощью механизма mutex.

Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60.

|  |
| --- |
| #include <iostream>  #include <Windows.h>  using namespace std;  PROCESS\_INFORMATION createProcess(LPCWSTR path)  {  STARTUPINFO si;  PROCESS\_INFORMATION pi;  ZeroMemory(&si, sizeof(STARTUPINFO));  si.cb = sizeof(STARTUPINFO);  if (CreateProcessW(path, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi))  cout << "Process created\n";  else  cout << "Error CreateProcessW\n";  return pi;  }  int main()  {  int pid = GetCurrentProcessId();  const int size = 2;  HANDLE mutex;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_03A\\x64\\Debug\\OS06\_03A.exe").hProcess; позволяет одному процессу блокировать ресурс, пока он его использует  processes[1] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_03B\\x64\\Debug\\OS06\_03B.exe").hProcess;  mutex = CreateMutex(NULL, FALSE, L"OS06\_03");  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  WaitForSingleObject(mutex, INFINITE);  else if (i == 60)  ReleaseMutex(mutex);  printf("OS06\_03 %d. PID = %d\n", i, pid);  Sleep(100);  }  WaitForMultipleObjects(size, processes, TRUE, INFINITE);  for (int i = 0; i < size; i++)  CloseHandle(processes[i]);  CloseHandle(mutex);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE mutex = OpenMutex(SYNCHRONIZE, FALSE, L"OS06\_03");  printf("OS06\_03A");  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  WaitForSingleObject(mutex, INFINITE);  else if (i == 60)  ReleaseMutex(mutex);    printf(" %d)pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(mutex);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE mutex = OpenMutex(SYNCHRONIZE, FALSE, L"OS06\_03");  printf("OS06\_03B");  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  WaitForSingleObject(mutex, INFINITE);  else if (i == 60)  ReleaseMutex(mutex);    printf(" %d)pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(mutex);  system("pause");  return 0;  } |



**Задание 04. Windows**

Разработайте приложение **OS06\_04**, запускающее два дочерних процесса **OS06\_04A** и **OS06\_04B** свыводом в отдельные консоли.

Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.

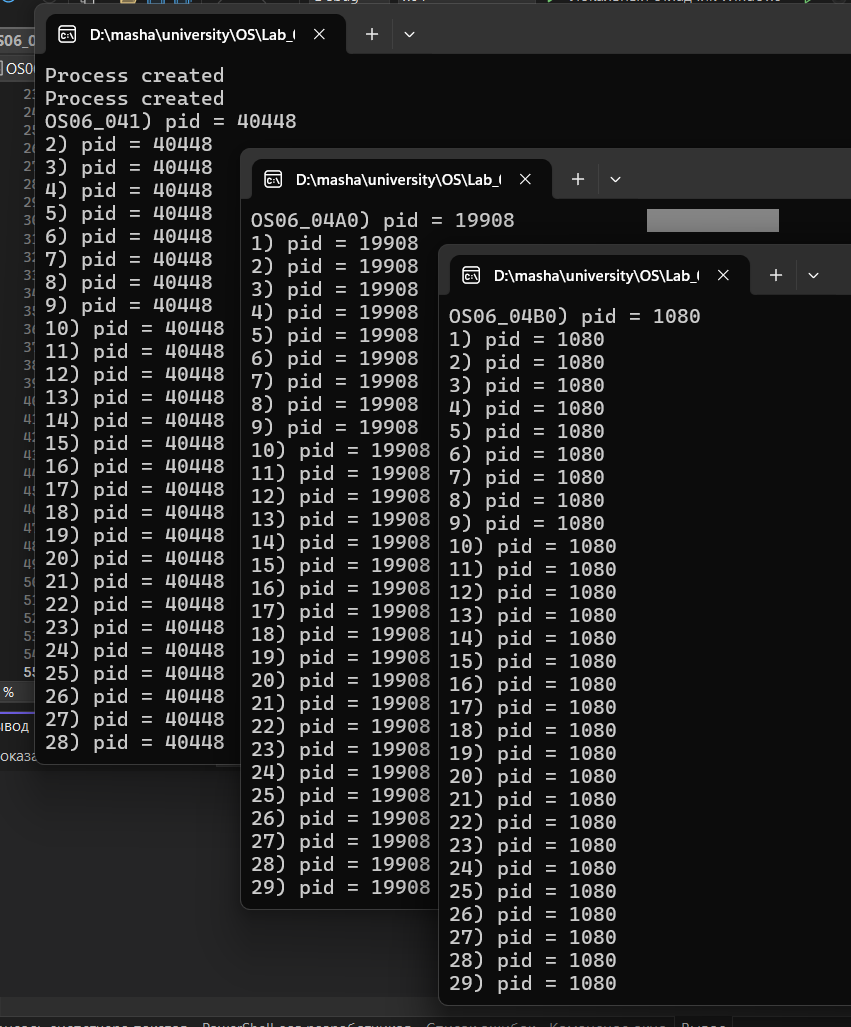
Приложение **OS06\_04** синхронизирует выполнение процессов **OS06\_04**, **OS06\_04A** и **OS06\_04B** с помощью механизма **semaphore.**

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>  using namespace std;  PROCESS\_INFORMATION createProcess(LPCWSTR path)  {  STARTUPINFO si;  PROCESS\_INFORMATION pi;  ZeroMemory(&si, sizeof(STARTUPINFO));  si.cb = sizeof(STARTUPINFO);  if (CreateProcessW(path, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi))  cout << "Process created\n";  else  cout << "Error CreateProcessW\n";  return pi;  }  int main()  {  int pid = GetCurrentProcessId();  const int size = 2;  HANDLE semaphore;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_04A\\x64\\Debug\\OS06\_04A.exe").hProcess;  processes[1] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_04B\\x64\\Debug\\OS06\_04B.exe").hProcess;  semaphore = CreateSemaphore(NULL, 2, 2, L"OS06\_04");  printf("OS06\_04");  for (int i = 1; i <= 90; i++)  {  if (i == 30)  WaitForSingleObject(semaphore, INFINITE);  else if (i == 60)  ReleaseSemaphore(semaphore, 1, NULL);  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  WaitForMultipleObjects(size, processes, TRUE, INFINITE);  for (int i = 0; i < size; i++)  CloseHandle(processes[i]);  CloseHandle(semaphore);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE semaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"OS06\_04");  printf("OS06\_04A");  for (int i = 0; i < 90; i++)  {  if (i == 30)  WaitForSingleObject(semaphore, INFINITE);  else if (i == 60)  ReleaseSemaphore(semaphore, 1, NULL);  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(semaphore);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE semaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"OS06\_04");  printf("OS06\_04B");  for (int i = 0; i < 90; i++)  {  if (i == 30)  WaitForSingleObject(semaphore, INFINITE);  else if (i == 60)  ReleaseSemaphore(semaphore, 1, NULL);  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(semaphore);  system("pause");  return 0;  } |



**Задание 05. Windows**

Разработайте приложение **OS06\_05**, запускающее два дочерних процесса **OS06\_05A** и **OS06\_05B** свыводом в отдельные консоли.

Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.

Приложение **OS06\_05** синхронизирует выполнение процессов **OS06\_05**, **OS06\_05A** и **OS06\_05B** с помощью механизма **event**.

Синхронизация должна обеспечивать выполнение приложения в два этапа:

1. выполнение итераций с 1 по 15 процесса **OS06\_05**;
2. 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;  PROCESS\_INFORMATION createProcess(LPCWSTR path)  {  STARTUPINFO startupInfo;  PROCESS\_INFORMATION processInfo;  ZeroMemory(&startupInfo, sizeof(STARTUPINFO));  startupInfo.cb = sizeof(STARTUPINFO);  if (CreateProcessW(path, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &startupInfo, &processInfo))  cout << "Process created.\n";  else  cout << "Error CreateProcessW\n";  return processInfo;  }  int main()  {  int pid = GetCurrentProcessId();  const int size = 2;  HANDLE event;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_05A\\x64\\Debug\\OS06\_05A.exe").hProcess;  processes[1] = createProcess(L"D:\\masha\\university\\OS\\Lab\_06\\OS06\_05B\\x64\\Debug\\OS06\_05B.exe").hProcess;  event = CreateEvent(NULL, FALSE, FALSE, L"OS06\_05");  printf("OS06\_05");  for (int i = 1; i <= 90; i++)  {  if (i == 15)  SetEvent(event);  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  WaitForMultipleObjects(size, processes, TRUE, INFINITE);  for (int i = 0; i < size; i++)  CloseHandle(processes[i]);  CloseHandle(event);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE event = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"OS06\_05");  printf("OS06\_05A");  WaitForSingleObject(event, INFINITE);  SetEvent(event);  for (int i = 1; i <= 90; i++)  {  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(event);  system("pause");  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE event = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"OS06\_05");  printf("OS06\_05B");  WaitForSingleObject(event, INFINITE);  SetEvent(event);  for (int i = 1; i <= 90; i++)  {  printf("%d) pid = %d\n", i, pid);  Sleep(100);  }  CloseHandle(event);  system("pause");  return 0;  } |

**Задание 06.Linux**

Разработайте приложение **OS06\_06**, запускающее два дочерних потока **A** и **B**.

Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.

Приложение **OS06\_06** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма **mutex.**

Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <unistd.h>  #include <sys/types.h>  #include <syscall.h>  pthread\_mutex\_t mx;  const char\* thread\_names[] = {"main", "A", "B"};  только один поток может выполнять критическую секцию.  void\* threadA(void\* arg) {  pid\_t pid = getpid();  pid\_t tid = (pid\_t)syscall(SYS\_gettid);  for (int i = 1; i <= 90; i++) {  if (i == 30) {  pthread\_mutex\_lock(&mx);  }  if (i == 60) {  pthread\_mutex\_unlock(&mx);  }  usleep(100000);  printf("[%s] %d) pid = %d tid = %u\n", thread\_names[1], i, pid, (unsigned int)tid);  if (i == 60) {  pthread\_mutex\_lock(&mx);  pthread\_mutex\_unlock(&mx);  }  }  pthread\_exit(NULL);  }  void\* threadB(void\* arg) {  pid\_t pid = getpid();  pid\_t tid = (pid\_t)syscall(SYS\_gettid);  for (int i = 1; i <= 90; i++) {  if (i == 30) {  pthread\_mutex\_lock(&mx);  }  if (i == 60) {  pthread\_mutex\_unlock(&mx);  }  usleep(100000);  printf("[%s] %d) pid = %d tid = %u\n", thread\_names[2], i, pid, (unsigned int)tid);  if (i == 60) {  pthread\_mutex\_lock(&mx);  pthread\_mutex\_unlock(&mx);  }  }  pthread\_exit(NULL);  }  int main() {  pthread\_mutex\_init(&mx, NULL);  pthread\_t thread1, thread2;  pid\_t main\_pid = getpid();  pid\_t main\_tid = (pid\_t)syscall(SYS\_gettid);  printf("[%s] 1) pid = %d tid = %u\n", thread\_names[0], main\_pid, (unsigned int)main\_tid);  pthread\_create(&thread1, NULL, threadA, NULL);  pthread\_create(&thread2, NULL, threadB, NULL);  for (int i = 2; i <= 90; i++) {  if (i == 30) {  pthread\_mutex\_lock(&mx);  }  if (i == 60) {  pthread\_mutex\_unlock(&mx);  }  usleep(100000);  printf("[%s] %d) pid = %d tid = %u\n", thread\_names[0], i, main\_pid, (unsigned int)main\_tid);  if (i == 60) {  pthread\_mutex\_lock(&mx);  pthread\_mutex\_unlock(&mx);  }  }  pthread\_join(thread1, NULL);  pthread\_join(thread2, NULL);  pthread\_mutex\_destroy(&mx);  return 0;  } |

