**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 <Windows.h>  #include <ctime>  using namespace std;  int mutex = 0; // для управления доступом к критической секции  DWORD WINAPI Thread();  int main() {  HANDLE hChild = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)Thread, NULL, 0, NULL);  HANDLE hChild2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)Thread, NULL, 0, NULL);  WaitForSingleObject(hChild, INFINITE);  WaitForSingleObject(hChild2, INFINITE);  CloseHandle(hChild);  CloseHandle(hChild2);  system("pause");  }  //вход в критическую функцию  void start\_critical\_section(void)  {  \_\_asm {  SpinLoop:  //lock: Эта префиксная команда указывает, что последующая операция должна быть атомарной.  lock bts mutex, 0;  jc SpinLoop //Это создает "петлю ожидания" (spinlock), где поток активно ждет, пока ресурс не станет доступным.  }  }  void exit\_critical\_section(void)  {  \_\_asm lock btr mutex, 0  }  DWORD WINAPI Thread()  {  DWORD tid = GetCurrentThreadId(); //id текущего потока  start\_critical\_section();  for (int i = 1; i < 91; i++)  {  cout << "Thread: " << tid << " " << i << endl;  }  exit\_critical\_section();  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 ChildThreadA()  {  DWORD TID = GetCurrentThreadId();  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  EnterCriticalSection(&cs); //вход в критическую секцию  }  if (i == 60)  {  LeaveCriticalSection(&cs); //выход  }  cout << "PID = " << PID << ", Child Thread A: " << "TID = " << TID << ": " << i << endl;  Sleep(100);  if (i == 60)  {  EnterCriticalSection(&cs);  LeaveCriticalSection(&cs);  }  }  return 0;  }  DWORD WINAPI ChildThreadB()  {  DWORD TID = GetCurrentThreadId();  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  EnterCriticalSection(&cs);  }  if (i == 60)  {  LeaveCriticalSection(&cs);  }  cout << "PID = " << PID << ", Child Thread B: " << "TID = " << TID << ": " << i << endl;  Sleep(100);  if (i == 60)  {  EnterCriticalSection(&cs);  LeaveCriticalSection(&cs);  }  }  return 0;  }  int main()  {  PID = GetCurrentProcessId();  DWORD TID = GetCurrentThreadId();//id главного потока  DWORD childIdA = NULL;  DWORD childIdB = NULL;  HANDLE hChildA = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThreadA, NULL, 0, &childIdA);  HANDLE hChildB = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThreadB, NULL, 0, &childIdB);  InitializeCriticalSection(&cs);  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  EnterCriticalSection(&cs);  }  if (i == 60)  {  LeaveCriticalSection(&cs);;  }  cout << "PID = " << PID << ", Main Thread: " << "TID = " << TID << ": " << i << endl;  Sleep(100);  if (i == 60)  {  EnterCriticalSection(&cs);  LeaveCriticalSection(&cs);  }  }  WaitForSingleObject(hChildA, INFINITE);  WaitForSingleObject(hChildB, INFINITE);  DeleteCriticalSection(&cs);  CloseHandle(hChildA);  CloseHandle(hChildB);  system("pause");  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.

|  |
| --- |
| OS06\_03  #include <Windows.h>  #include <iostream>  using namespace std;  int main()  {  PROCESS\_INFORMATION pi1, pi2;  DWORD PID = GetCurrentProcessId();  LPCWSTR an1 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_03A.exe";  LPCWSTR an2 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_03B.exe";  HANDLE hm = CreateMutex(NULL, false, L"smwMutex");  {  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 << "--Process OS06\_03A created\n";  }  else  {  cout << "--Process OS06\_03A 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 << "--Process OS06\_03B created\n";  }  else  {  cout << "--Process OS06\_03B not created\n";  }  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hm, INFINITE);  }  if (i == 60)  {  ReleaseMutex(hm);  }  cout << "PID = " << PID << ", Main Thread: " << i << endl;  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(hm);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  return 0;  }  } |
| OS06\_03A  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  //для хранения дескриптора мьютекса. Функция открывает существующий мьютекс с именем "smwMutex"  HANDLE hm = OpenMutex(SYNCHRONIZE, FALSE, L"smwMutex");  if (hm == NULL)  {  cout << "OS06\_03A: Open err Mutex\n";  }  else  {  cout << "OS06\_03A: Open Mutex\n";  }  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hm, INFINITE);  }  if (i == 60)  {  ReleaseMutex(hm); //Освобождает мьютекс, позволяя другим потокам или процессам продолжить выполнение  }  Sleep(100);  cout << "OS06\_03A: " << i << " PID: " << GetCurrentProcessId() << endl;  }  } |
| OS06\_03B  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  HANDLE hm = OpenMutex(SYNCHRONIZE, FALSE, L"smwMutex");  if (hm == NULL)  {  cout << "OS06\_03B: Open err Mutex\n";  }  else  {  cout << "OS06\_03B: Open Mutex\n";  }  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hm, INFINITE);  }  if (i == 60)  {  ReleaseMutex(hm);  }  Sleep(100);  cout << "OS06\_03B: " << i << " PID: " << GetCurrentProcessId() << endl;  }  } |
|  |

**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.

|  |
| --- |
| OS06\_04  #include <Windows.h>  #include <iostream>  using namespace std;  int main()  {  PROCESS\_INFORMATION pi1, pi2;  DWORD PID = GetCurrentProcessId();  LPCWSTR an1 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_04A.exe";  LPCWSTR an2 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_04B.exe";  HANDLE hs = CreateSemaphore(NULL, 2, 2, L"smwSem"); //2: Начальное значение счетчика семафора (количество разрешений, доступных сразу); 2: Максимальное значение счетчика семафора(максимальное количество разрешений).  {  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 << "--Process OS06\_04A created\n";  }  else  {  cout << "--Process OS06\_04A 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 << "--Process OS06\_04B created\n";  }  else  {  cout << "--Process OS06\_04B not created\n";  }  LONG prevcount = 0; //для хранения предыдущего значения счетчика семафора  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hs, INFINITE);  }  if (i == 60)  {  ReleaseSemaphore(hs, 1, &prevcount); //Освобождает семафор, увеличивая его счетчик на 1  cout << "OS06\_04: prevcount = " << prevcount << endl;  }  cout << "PID = " << PID << ", Main Thread: " << i << endl;  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(hs);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  return 0;  }  } |
| OS06\_04A  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  //для хранения дескриптора семафора  HANDLE hs = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"smwSem");  if (hs == NULL)  {  cout << "OS06\_04A: Open error Semaphore\n";  }  else  {  cout << "OS06\_04A: Open Semaphore\n";  }  LONG prevcount = 0; //для хранения предыдущего значения счетчика семафора  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hs, INFINITE);  }  if (i == 60)  {  ReleaseSemaphore(hs, 1, &prevcount); //Освобождает семафор, увеличивая его счетчик на 1  cout << "OS06\_04A: prevcount = " << prevcount << endl;  }  Sleep(100);  cout << "OS06\_04A: " << i << " PID: " << GetCurrentProcessId() << endl;  }  CloseHandle(hs);  system("pause");  return 0;  } |
| OS06\_04B  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  HANDLE hs = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"smwSem");  if (hs == NULL)  {  cout << "OS06\_04B: Open error Semaphore\n";  }  else  {  cout << "OS06\_04B: Open Semaphore\n";  }  LONG prevcount = 0;  for (int i = 1; i <= 90; i++)  {  if (i == 30)  {  WaitForSingleObject(hs, INFINITE);  }  if (i == 60)  {  ReleaseSemaphore(hs, 1, &prevcount);  cout << "OS06\_04B: prevcount = " << prevcount << endl;  }  Sleep(100);  cout << "OS06\_04B: " << i << " PID: " << GetCurrentProcessId() << endl;  }  CloseHandle(hs);  system("pause");  return 0;  } |
|  |

**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.

|  |
| --- |
| OS06\_05  #include <Windows.h>  #include <iostream>  using namespace std;  int main()  {  PROCESS\_INFORMATION pi1, pi2;  DWORD PID = GetCurrentProcessId();  LPCWSTR an1 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_05A.exe";  LPCWSTR an2 = L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба6\\lab6\\x64\\Debug\\OS06\_05B.exe";  HANDLE he = CreateEvent(NULL, FALSE, FALSE, L"smwEvent");  {  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 << "--Process OS06\_05A created\n";  }  else  {  cout << "--Process 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 << "--Process OS06\_05B created\n";  }  else  {  cout << "--Process OS06\_05B not created\n";  }  for (int i = 1; i <= 90; i++)  {  if (i == 15)  {  SetEvent(he);  }  cout << "PID = " << PID << ", Main Thread: " << i << endl;  Sleep(100);  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(he);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  return 0;  }  } |
| OS06\_05A  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  HANDLE he = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"smwEvent");  if (he == NULL)  {  cout << "OS06\_05A: Open error Event\n";  }  else  {  cout << "OS06\_05A: Open Event\n";  }  WaitForSingleObject(he, INFINITE);  for (int i = 1; i <= 90; i++)  {  SetEvent(he); //все потоки, ожидающие на этом событии, смогут продолжить выполнение, если они были заблокированы вызовом WaitForSingleObject  Sleep(100);  cout << "OS06\_05A: " << i << " PID: " << GetCurrentProcessId() << endl;  }  CloseHandle(he);  system("pause");  return 0;  } |
| OS06\_05B  #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  HANDLE he = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"smwEvent");  if (he == NULL)  {  cout << "OS06\_05B: Open error Event\n";  }  else  {  cout << "OS06\_05B: Open Event\n";  }  WaitForSingleObject(he, INFINITE);  for (int i = 1; i <= 90; i++)  {  SetEvent(he); //все потоки, ожидающие на этом событии, смогут продолжить выполнение, если они были заблокированы вызовом WaitForSingleObject  Sleep(100);  cout << "OS06\_05B: " << i << " PID: " << GetCurrentProcessId() << endl;  }  CloseHandle(he);  system("pause");  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.

|  |
| --- |
| 1. #include <stdio.h> #include <stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h>  pthread\_mutex\_t mx;  void\* A(void\* arg){ for(int i = 0; i <= 90; i++){ if(i == 30){ pthread\_mutex\_lock(&mx); } sleep(1); printf("%d A: %d\n",i,getpid()); if(i == 60){ pthread\_mutex\_unlock(&mx); } } printf("---> A Finished <---"); //pthread\_exit("A"); }   void\* B(void\* arg){ for(int i = 0; i <= 90; i++){ if(i == 30){ pthread\_mutex\_lock(&mx); } sleep(1); printf("%d B: %d\n",i,getpid()); if(i == 60){ pthread\_mutex\_unlock(&mx); } } printf("---> B Finished <----"); //pthread\_exit("B"); }  int main(){ pthread\_mutex\_init(&mx,NULL); pthread\_t a\_th1, a\_th2; void \*r\_th1, \*r\_th2; pid\_t pid = getpid(); printf("Main: PID= %d\n",pid); int res1 = pthread\_create(&a\_th1,NULL,A,NULL); int res2 = pthread\_create(&a\_th2,NULL,B,NULL); for(int i = 0; i <= 90; i++){ if(i == 30){ pthread\_mutex\_lock(&mx); } sleep(1); printf("%d Main: %d\n",i,pid); if(i == 60){ pthread\_mutex\_unlock(&mx); } } printf("---> Main Finished <---"); int status1 = pthread\_join(a\_th1,(void\*\*)&r\_th1); int status2 = pthread\_join(a\_th2,(void\*\*)&r\_th2); pthread\_mutex\_destroy(&mx); } |
|  |

**Задание 07.** Ответьте на следующие вопросы

1. Дайте определение понятию «синхронизация потоков».

**Синхронизация потоков** - это процесс координирования выполнения множества потоков в многозадачной среде, чтобы избежать конфликтов и обеспечить правильное взаимодействие между ними.

1. Объясните понятие «взаимная блокировка».

**Взаимная блокировка** - это ситуация, когда два или более потока или процесса блокируют друг друга, ожидая ресурсы или события, и не могут продолжить выполнение. Это приводит к зависанию программы.

1. Перечислите механизмы авторизации OS.

* **Списки контроля доступа (ACL)** — определяют права доступа для каждого пользователя к объектам системы.
* **Ролевая модель (RBAC)** — права доступа назначаются ролям, а роли присваиваются пользователям.
* **Мандатное управление доступом (MAC)** — права назначаются исходя из уровня конфиденциальности и потребностей безопасности.
* **Идентификация и аутентификация пользователей** — проверка подлинности пользователя перед тем, как дать ему доступ к ресурсам.
* **Модели доверенных доменов** — для управления доступом между разными системами или доменами.

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

**Mutex -** механизм синхронизации нескольких потоков разных процессов, является объектом ядра OS. Он позволяет только одному потоку владеть блокировкой в данный момент времени.

**Semaphore -** механизм, который позволяет управлять доступом к ресурсу для нескольких потоков, ограничивая их количество.

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

**Мьютексы, семафоры и события** создают объекты ядра ОС, так как они позволяют синхронизировать потоки не только внутри одного процесса, но и между разными процессами. Эти объекты управляются ядром, обеспечивая безопасный межпроцессорный доступ к ресурсам.

**Критическая секция** не создает объект ядра ОС, так как используется для синхронизации потоков только внутри одного процесса. Она работает быстрее, так как не требует перехода в режим ядра и позволяет управлять доступом к ресурсам в пределах одного процесса.