Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Дмитрук И. И.

ФИТ 3 курс 5 группа

Преподаватель: Савельева М. Г.

Минск 2023

**Лабораторная работа №6**

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

|  |  |
| --- | --- |
| Код программы OS06\_01 | #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  int check;  HANDLE createThread(LPTHREAD\_START\_ROUTINE func, char\* thread\_name)  {  DWORD thread\_id = NULL;  HANDLE thread = CreateThread(NULL, 0, func, thread\_name, 0, &thread\_id);  if (thread == NULL)  throw "[ERROR] CreateThread";  return thread;  }  void EnterCriticalSectionAssembly()  {  \_asm  {  CriticalSection:  lock bts check, 0;  jc CriticalSection  }  }  void LeaveCriticalSectionAssembly()  {  \_asm lock btr check, 0  }  void WINAPI loop(char\* displayed\_name)  {  int pid = GetCurrentProcessId();  int tid = GetCurrentThreadId();  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  EnterCriticalSectionAssembly();  printf("%d.\tPID = %d\tTID = %u\tcheck: %d\tthread: %s\n", i, pid, tid, check, displayed\_name);  if (i == 60)  LeaveCriticalSectionAssembly();  Sleep(100);  }  cout << "\n========================== " << displayed\_name << " finished" << " ==========================\n\n";  }  int main()  {  const int size = 2;  HANDLE threads[size];  threads[0] = createThread((LPTHREAD\_START\_ROUTINE)loop, (char\*)"A");  threads[1] = createThread((LPTHREAD\_START\_ROUTINE)loop, (char\*)"B");  WaitForMultipleObjects(size, threads, TRUE, INFINITE);  for (int i = 0; i < size; i++)  CloseHandle(threads[i]);  return 0;  } |
| Скриншот запуска |  |

Задание 2. 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.

|  |  |
| --- | --- |
| Код программы OS06\_02 | #include <iostream>  #include <windows.h>  using namespace std;  CRITICAL\_SECTION critical\_section;  HANDLE createThread(LPTHREAD\_START\_ROUTINE func, char\* thread\_name)  {  DWORD thread\_id = NULL;  HANDLE thread = CreateThread(NULL, 0, func, thread\_name, 0, &thread\_id);  if (thread == NULL)  throw "[ERROR] CreateThread";  return thread;  }  void WINAPI loop(char\* displayed\_name)  {  int pid = GetCurrentProcessId();  int tid = GetCurrentThreadId();  for (int i = 1; i <= 90; ++i)  {  if (i == 30)  EnterCriticalSection(&critical\_section);  printf(" [%s]\t %d. PID = %d\tTID = %u\n", displayed\_name, i, pid, tid);  if (i == 60)  LeaveCriticalSection(&critical\_section);  Sleep(100);  }  cout << "\n================ " << displayed\_name << " finished" << " ================\n\n";  }  int main()  {  int pid = GetCurrentProcessId();  const int size = 2;  HANDLE threads[size];  threads[0] = createThread((LPTHREAD\_START\_ROUTINE)loop, (char\*)"A");  threads[1] = createThread((LPTHREAD\_START\_ROUTINE)loop, (char\*)"B");  InitializeCriticalSection(&critical\_section);  for (int i = 1; i <= 100; ++i)  {  if (i == 30)  EnterCriticalSection(&critical\_section);  printf("[MAIN]\t %d. PID = %d\n", i, pid);  if (i == 60)  LeaveCriticalSection(&critical\_section);  Sleep(100);  }  cout << "\n=============== MAIN finished ===============\n\n";  WaitForMultipleObjects(size, threads, TRUE, INFINITE);  for (int i = 0; i < size; ++i)  CloseHandle(threads[i]);  DeleteCriticalSection(&critical\_section);  return 0;  } |
| Скриншот запуска |  |

Задание 3. Разработайте приложение OS07\_03, запускающее два дочерних процесса OS07\_03A и OS07\_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 OS07\_03 синхронизирует выполнение процессов OS07\_03, OS07\_03A и OS07\_03B с помощью механизма mutex.

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

|  |  |
| --- | --- |
| Код программы OS06\_03 | #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 mutex;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\Labs\\ОС\\Lab6\\Debug\\OS06\_03A.exe").hProcess;  processes[1] = createProcess(L"D:\\Labs\\ОС\\Lab6\\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]\t %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);  cout << '\n';  system("pause");  return 0;  } |
| Один из дочерних процессов | #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE mutex = OpenMutex(SYNCHRONIZE, 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\_03A]\t%d. PID = %d\n", i, pid);  Sleep(100);  }  CloseHandle(mutex);  } |
| Скриншот запуска |  |

Задание 4. Разработайте приложение OS07\_04, запускающее два дочерних процесса OS07\_04A и OS07\_04B 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 OS07\_04 синхронизирует выполнение процессов OS07\_04, OS07\_04A и OS07\_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.

|  |  |
| --- | --- |
| Код программы OS06\_04 | #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 semaphore;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\Labs\\ОС\\Lab6\\Debug\\OS06\_04A.exe").hProcess;  processes[1] = createProcess(L"D:\\Labs\\ОС\\Lab6\\Debug\\OS06\_04B.exe").hProcess;  semaphore = CreateSemaphore(NULL, 2, 2, L"OS06\_04");  for (int i = 1; i <= 90; i++)  {  if (i == 30)  WaitForSingleObject(semaphore, INFINITE);  else if (i == 60)  ReleaseSemaphore(semaphore, 1, NULL);  printf("[OS06\_04]\t %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);  cout << '\n';  system("pause");  return 0;  } |
| Один из дочерних процессов | #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE semaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"OS06\_04");  for (int i = 0; i < 90; i++)  {  if (i == 30)  WaitForSingleObject(semaphore, INFINITE);  else if (i == 60)  ReleaseSemaphore(semaphore, 1, NULL);  printf("[OS06\_04A]\t%d. PID = %d\n", i, pid);  Sleep(100);  }  CloseHandle(semaphore);  } |
| Скриншот запуска |  |

Задание 5. Разработайте приложение OS07\_05, запускающее два дочерних процесса OS07\_05A и OS07\_05B 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 OS07\_05 синхронизирует выполнение процессов OS07\_05, OS07\_05A и OS07\_05B с помощью механизма event.

Синхронизация должна обеспечивать выполнение приложения в два этапа:

1. выполнение итераций с 1 по 15 процесса OS07\_05;
2. одновременное выполнение всех трех процессов: OS07\_05 – продолжает выполнение итераций; процессы OS07\_05A и OS07\_05B выполняются начиная с первой итерации.

|  |  |
| --- | --- |
| Код программы OS06\_05 | #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]\n";  return processInfo;  }  int main()  {  int pid = GetCurrentProcessId();  const int size = 2;  HANDLE event;  HANDLE processes[size];  processes[0] = createProcess(L"D:\\Labs\\ОС\\Lab6\\Debug\\OS06\_05A.exe").hProcess;  processes[1] = createProcess(L"D:\\Labs\\ОС\\Lab6\\Debug\\OS06\_03A.exe").hProcess;  event = CreateEvent(NULL, FALSE, FALSE, L"OS06\_05");  for (int i = 1; i <= 90; i++)  {  if (i == 15)  SetEvent(event);  printf("[OS06\_05]\t %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;  } |
| Один из дочерних процессов | #include <iostream>  #include <Windows.h>  int main()  {  int pid = GetCurrentProcessId();  HANDLE event = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"OS06\_05");  WaitForSingleObject(event, INFINITE);  SetEvent(event);  for (int i = 1; i <= 90; i++)  {  printf("[OS06\_05A]\t %d. PID = %d\n", i, pid);  Sleep(100);  }  CloseHandle(event);  } |
| Скриншот запуска |  |

Задание 6. Разработайте приложение OS07\_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 OS07\_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.

|  |  |
| --- | --- |
| Код программы OS06\_06 | #include <stdio.h>  #include <time.h>  #include <unistd.h>  #include <pthread.h>  #define DELAY\_MS 100  pthread\_mutex\_t mutex;  pthread\_t create\_thread(void\* function, void\* args)  {  pthread\_t thread;  pthread\_create(&thread, NULL, function, args);  return thread;  }  void\* loop(char\* name)  {  int pid = getpid();  struct timespec timespec;  timespec.tv\_sec = DELAY\_MS / 1000;  timespec.tv\_nsec = (DELAY\_MS % 1000) \* 1000000;  for (int i = 1; i <= 90; i++)  {  if (i == 30)  pthread\_mutex\_lock(&mutex);  else if (i == 60)  pthread\_mutex\_unlock(&mutex);  printf(" [%s]\t %d. PID = %d\n", name, i, pid);  nanosleep(&timespec, &timespec);  }  printf("\n======= %s finished =======\n\n", name);  pthread\_exit("done");  }  int main(int argc, char\* argv[])  {  int pid = getpid();  int size = 2;  char\* names[] = { "A", "B" };  pthread\_t threads[size];  struct timespec timespec;  timespec.tv\_sec = DELAY\_MS / 1000;  timespec.tv\_nsec = (DELAY\_MS % 1000) \* 1000000;  pthread\_mutex\_init(&mutex, NULL);  for (int i = 0; i < size; i++)  threads[i] = create\_thread(loop, (void\*)names[i]);  for (int i = 1; i <= 90; i++)  {  if (i == 30)  pthread\_mutex\_lock(&mutex);  else if (i == 60)  pthread\_mutex\_unlock(&mutex);  printf("[MAIN]\t %d. PID = %d\n", i, pid);  nanosleep(&timespec, &timespec);  }  printf("\n===== MAIN finished =====\n\n");  for (int i = 0; i < size; i++)  pthread\_join(threads[i], NULL);  pthread\_mutex\_destroy(&mutex);  return 0;  } |
| Скриншот запуска |  |