Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «МОП ЭВМ»

ЛАБОРАТОРНАЯ РАБОТА 5

по дисциплине «Операционные системы и оболочки»

## Синхронизация потоков в операционной системе Windows

Студент группы 6ИСб-1 О. Хусейнов

Преподаватель В.А Тихомиров

2018

**Цель работы**: Изучить теоретические вопросы управления синхронизацией программных потоков в ОС Windows и освоить приемы практической реализации этого управления с использованием системных функций.

**Задание:** Имеется файл F1, в котором записано 20 слов (можно больше). Три потока генерируют случайные числа в диапазоне 0 – 100. Если сгенерированное число больше 90, поток генерирует случайное число n от 1 до 20, открывает файл F1, берет из него слово под номером n, открывает файл F2 и вписывает взятое слово в него файлы F1 и F2 закрываются. Каждый поток должен записать в файл F2 по три слова. Полученное предложение приложите в отчет.

**Выполнение:**

Функция **ShellExecute** может не только выполнять заданное приложение, но и открывать документ и печатать его. Под термином «открыть файл документа» понимается выполнение связанного с ним приложения и загрузка в него этого документа. Например, обычно с документами, имеющими расширение .doc, связан Word. В этом случае открыть файл, например, с именем file.doc означает запус­тить Word и передать ему в качестве параметра имя файла file.doc. Кроме описан­ных возможностей функция ShellExecute позволяет открыть указанную папку. Это означает, что будет запущена программа «Проводник» с открытой указанной папкой.

Для использования функции ShellExecute в операторе uses надо добавить мо­дуль ShellAPI. Автоматически Delphi не включает этот модуль в программу.

Функция ShellExecute определена следующим образом:

function ShellExecute(Wnd: HWnd; Operation, FileName, Parameters,

Directory: PChar; ShowCmd: Integer): THandle;

Параметр Wnd является дескриптором родительского окна, в котором отобра­жаются сообщения запускаемого приложения. Обычно в качестве него можно про­сто указать Handle.

Параметр Operation указывает на строку с нулевым символом в конце, кото­рая определяет выполняемую операцию. Эта строка может содержать текст «open» (открыть) или «print» (напечатать). Для Windows 95/98 и NT определено еще одно значение: «explore» (исследовать) — открыть папку. Если параметр Operation ра­вен nil, то по умолчанию выполняется операция «open».

Параметр FileName указывает на строку с нулевым символом в конце, кото­рая определяет имя открываемого файла или имя открываемой папки.

Параметр Parameters указывает на строку *с*нулевым символом в конце, кото­рая определяет передаваемые в приложение параметры, если FileName определяет выполняемый файл. Если FileName указывает на строку, определяющую откры­ваемый документ или папку, то этот параметр задается равным null.

Параметр Directory указывает на строку с нулевым символом в конце, которая определяет каталог по умолчанию.

Параметр ShowCmd определяет режим открытия указанного файла. Этот па­раметр может иметь множество различных значений. Обычно, как и для функции WinExec, используется значение SW\_RESTORE, при котором окно запускаемого приложения активизируется и отображается на экране.

Текст программы по индивидуальному заданию приведен в листингах 1,2.

Листинг 1 – Текст файла SyncThread.cpp

|  |
| --- |
| #include <tchar.h>  #include <windows.h>  #include <process.h>  // Идентификаторы кнопок  #define EXIT 404  #define RUN 405  /\* Declare Windows procedure \*/  LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);  /\* Make the class name into a global variable \*/  TCHAR szClassName[] = \_T("CodeBlocksWindowsApp");  size\_t i = 0;  SHELLEXECUTEINFO ShExecInfo;  unsigned \_\_stdcall thread(void \*)  {  ShExecInfo = { 0 };  ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);  ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;  ShExecInfo.hwnd = NULL;  ShExecInfo.lpVerb = NULL;  ShExecInfo.lpFile = L"C:\\Users\\miste\\Source\\WinAPI\\Debug\\Lab5(SyncThread)Child.exe";  ShExecInfo.lpParameters = L"";  ShExecInfo.lpDirectory = NULL;  ShExecInfo.nShow = SW\_SHOW;  ShExecInfo.hInstApp = NULL;  ShellExecuteEx(&ShExecInfo);  WaitForSingleObject(ShExecInfo.hProcess, INFINITE);  --i;  return 0;  }  unsigned \_\_stdcall myWhile(void \*)  {  while (true)  {  while (i < 5)  {  int rnd = rand() % 1000 + 1;  if (rnd>900 && rnd < 1000)  {  \_beginthreadex(NULL, 0, &thread, NULL, 0, NULL);  ++i;  }  else Sleep(200);  }  Sleep(200);  }  return 0;  }  int WINAPI WinMain(HINSTANCE hThisInstance,  HINSTANCE hPrevInstance,  LPSTR lpszArgument,  int nCmdShow)  {  HWND hwnd; /\* This is the handle for our window \*/  MSG messages; /\* Here messages to the application are saved \*/  WNDCLASSEX wincl; /\* Data structure for the windowclass \*/  /\* The Window structure \*/  wincl.hInstance = hThisInstance;  wincl.lpszClassName = szClassName;  wincl.lpfnWndProc = WindowProcedure; /\* This function is called by windows \*/  wincl.style = CS\_DBLCLKS; /\* Catch double-clicks \*/  wincl.cbSize = sizeof(WNDCLASSEX);  /\* Use default icon and mouse-pointer \*/  wincl.hIcon = LoadIcon(NULL, IDI\_APPLICATION);  wincl.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);  wincl.hCursor = LoadCursor(NULL, IDC\_ARROW);  wincl.lpszMenuName = NULL; /\* No menu \*/  wincl.cbClsExtra = 0; /\* No extra bytes after the window class \*/  wincl.cbWndExtra = 0; /\* structure or the window instance \*/  /\* Use Windows's default colour as the background of the window \*/  wincl.hbrBackground = (HBRUSH)COLOR\_BACKGROUND;  /\* Register the window class, and if it fails quit the program \*/  if (!RegisterClassEx(&wincl))  return 0;  /\* The class is registered, let's create the program\*/  hwnd = CreateWindowEx(  0, /\* Extended possibilites for variation \*/  szClassName, /\* Classname \*/  \_T("Code::Blocks Template Windows App"), /\* Title Text \*/  WS\_OVERLAPPEDWINDOW, /\* default window \*/  CW\_USEDEFAULT, /\* Windows decides the position \*/  CW\_USEDEFAULT, /\* where the window ends up on the screen \*/  544, /\* The programs width \*/  375, /\* and height in pixels \*/  HWND\_DESKTOP, /\* The window is a child-window to desktop \*/  NULL, /\* No menu \*/  hThisInstance, /\* Program Instance handler \*/  NULL /\* No Window Creation data \*/  );  /\* Make the window visible on the screen \*/  ShowWindow(hwnd, SW\_HIDE);  // ShowWindow (hwnd, nCmdShow);  /\* Run the message loop. It will run until GetMessage() returns 0 \*/  while (GetMessage(&messages, NULL, 0, 0))  {  /\* Translate virtual-key messages into character messages \*/  TranslateMessage(&messages);  /\* Send message to WindowProcedure \*/  DispatchMessage(&messages);  }  /\* The program return-value is 0 - The value that PostQuitMessage() gave \*/  return messages.wParam;  }  /\* This function is called by the Windows function DispatchMessage() \*/  LRESULT CALLBACK WindowProcedure(HWND hWnd, // дескриптор окошка  UINT message, // сообщение, посылаемое ОС  WPARAM wParam, // параметры  LPARAM lParam) // сообщений, для последующего обращения  {  static bool bCreated = false;  if (lParam == WM\_RBUTTONDOWN)  {  SetForegroundWindow(hWnd);  POINT mouse;  GetCursorPos(&mouse);  HMENU hMenu = CreatePopupMenu();  AppendMenu(hMenu, MF\_ENABLED, RUN, L"Run");  AppendMenu(hMenu, MF\_ENABLED, EXIT, L"Exit");  TrackPopupMenu(hMenu, TPM\_RIGHTBUTTON, mouse.x, mouse.y, 0, hWnd, NULL);  }  switch (message)  {  // Сообщение WM\_COMMAND посылается когда:  // - производится выбор пункта меню  // - элемент управления посылает уведомительное сообщение родительскому окну  // - происходит нажатие клавиши акселератора.  case WM\_COMMAND:  {  // Wparam - определяет источник сообщения: элемент управления или акселератор.  // Lparam - идентификатор элемента, если это не акселератор.  switch (LOWORD(wParam))  {  case RUN:  {  \_beginthreadex(NULL, 0, &myWhile, NULL, 0, NULL);  }  break;  case EXIT:  {  ShellExecute(NULL, NULL, L"taskkill", L"/F /IM Lab5(SyncThread)Child.exe", NULL, SW\_HIDE);  exit(1);  }  break;  }  }  break;  case WM\_CREATE:  {  NOTIFYICONDATA nid;  memset(&nid, 0, sizeof(NOTIFYICONDATA));  nid.cbSize = sizeof(NOTIFYICONDATA);  nid.hWnd = hWnd;  nid.uID = 1;  nid.uFlags = NIF\_ICON | NIF\_MESSAGE | NIF\_TIP;  nid.uCallbackMessage = WM\_USER + 200;  nid.hIcon = LoadIcon(NULL, IDI\_INFORMATION);  lstrcpy(nid.szTip, L"SyncThread");  Shell\_NotifyIcon(NIM\_ADD, &nid);  // ShowWindow(hWnd, SW\_HIDE);  bCreated = true;  }  break;  case WM\_DESTROY:  {  if (bCreated)  {  NOTIFYICONDATA nid;  memset(&nid, 0, sizeof(NOTIFYICONDATA));  nid.cbSize = sizeof(NOTIFYICONDATA);  nid.hWnd = hWnd;  nid.uID = 1;  Shell\_NotifyIcon(NIM\_DELETE, &nid);  }  PostQuitMessage(0);  }  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  } |

Листинг 1 – Текст файла SyncThread.cpp

|  |
| --- |
| /\*  Имеется файл F1, в котором записано 20 слов (можно больше).  Три потока генерируют случайные числа в диапазоне 0 – 100.  Если сгенерированное число больше 90, поток генерирует случайное число n от 1 до 20,  открывает файл F1, берет из него слово под номером n, открывает файл F2 и вписывает взятое слово в него файлы F1 и F2 закрываются.  Каждый поток должен записать в файл F2 по три слова. Полученное предложение приложите в отчет.  \*/  #include <iostream>  #include <windows.h>  #include <process.h>  #include <string>  #include <fstream>  #include <mutex>  #include <random>  using namespace std;  mutex writeLock;  unsigned \_\_stdcall readWrite(void \*);  int getRandomInt(int start, int end);  int main()  {  setlocale(LC\_ALL, "rus");  system("color 0A ");  HANDLE hThread[3];  for (size\_t i = 0; i < 3; i++)  hThread[i] = (HANDLE)\_beginthreadex(NULL, 0, &readWrite, NULL, 0, NULL);    cout << "Все потоки запущены...\n";  WaitForMultipleObjects(3, hThread, TRUE, INFINITE);  cout << "Все потоки завершили работу\n";  system("pause");  return 0;  }  unsigned \_\_stdcall readWrite(void \*)  {  size\_t count = 0;  while (count<3)  {  int rndNum = 1 + rand() % 100;  if (rndNum > 90)  {  int rndWord = getRandomInt(1,20);  ifstream file1("text1.txt");  if (!file1)  {  cout << "Файл не найден\nError\n";  system("pause");  exit(0);  }  string word;  for (size\_t i = 0; i < rndWord; i++)  getline(file1, word, ' ');  file1.close();  writeLock.lock();  ofstream file2;  file2.open("text2.txt", ios::app);  file2 << word << ' ';  file2.close();  writeLock.unlock();  count++;  }  Sleep(200);  }  return 0;  }  int getRandomInt(int start, int end)  {  default\_random\_engine g(chrono::system\_clock::now().time\_since\_epoch().count());  uniform\_int\_distribution<int> distribution(start, end);  return distribution(g);  } |

Результат работы программы показан на рисунках 1,2.

Текстовый файл из которого будут браться слова (рис. 1).

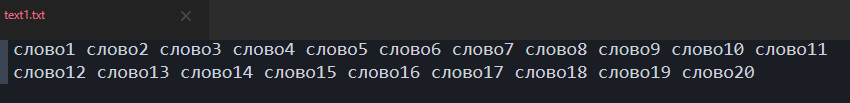


Рисунок 1 – Исходный текстовый файл

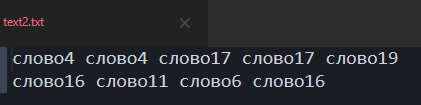


Рисунок 2 - Полученный текст

**Список использованных источников**

1. Саймон, Р. Windows 2000 API Энциклопедия программиста / Р. Саймон. - М: DiaSoft, 2002