Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Копылов Михаил Юрьевич

Группа: М8О-201Б-21

Вариант: 21

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Выводы

**Репозиторий**

<https://github.com/Mikhail-cWc/OS_mai/tree/main/lab2>

**Постановка задачи**

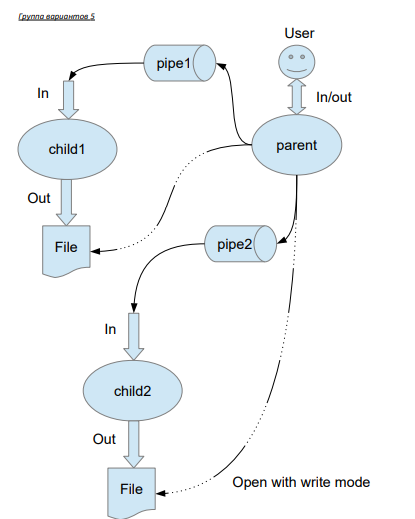
**Цель работы**

Приобретение практических навыков в:

 Управление процессами в ОС  
 Обеспечение обмена данных между процессами посредством каналов

**Задание**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.



Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.**Общие сведения о программе**

Используются системные вызовы Windows

**Исходный код**

|  |
| --- |
| **parent.cpp** |
| #include "utils.h"  #include "parent.h"  int ParentRoutine(std::istream &inFile){  // pipe1 (child1)  // Cтандартный дескриптор ввода для доч. процесса  HANDLE g\_hChildStd\_IN\_Rd = NULL;  // Дескриптор для записи во входные данные доч. процесса(не наследуется)  HANDLE g\_hChildStd\_IN\_Wr = NULL;  // Дескриптор для чтения из выходных данных доч. процесса(не наследуется)  HANDLE g\_hChildStd\_OUT\_Rd = NULL;  // Стандартный дескриптор вывода для доч. процесса  HANDLE g\_hChildStd\_OUT\_Wr = NULL;  // pipe2 (child2)  HANDLE g\_hChildStd\_IN\_Rd\_2 = NULL;  HANDLE g\_hChildStd\_IN\_Wr\_2 = NULL;  HANDLE g\_hChildStd\_OUT\_Rd\_2 = NULL;  HANDLE g\_hChildStd\_OUT\_Wr\_2 = NULL;  std::string child1 = "..\\lab2\\child1.exe";  std::string child2 = "..\\lab2\\child2.exe";  HANDLE g\_hOutputFile1 = NULL;  HANDLE g\_hOutputFile2 = NULL;  // флаг bInheritHandle, чтобы дескрипторы каналов наследовались.  SECURITY\_ATTRIBUTES saAttr;  saAttr.nLength = sizeof(SECURITY\_ATTRIBUTES);  saAttr.bInheritHandle = TRUE;  saAttr.lpSecurityDescriptor = NULL;  // Канал для стандартного выхода доч. процесса 1  PipeStdOut(g\_hChildStd\_OUT\_Rd, g\_hChildStd\_OUT\_Wr, saAttr);  // Канал для стандартного входа доч. процесса 1  PipeStdIn(g\_hChildStd\_IN\_Rd, g\_hChildStd\_IN\_Wr, saAttr);  // Канал для стандартного выхода доч. процесса 2  PipeStdOut(g\_hChildStd\_OUT\_Rd\_2, g\_hChildStd\_OUT\_Wr\_2, saAttr);  // Канал для стандартного входа доч. процесса 1  PipeStdIn(g\_hChildStd\_IN\_Rd\_2, g\_hChildStd\_IN\_Wr\_2, saAttr);  //----------------------------  CreateChildProcess(g\_hChildStd\_OUT\_Wr, g\_hChildStd\_IN\_Rd, child1);  CreateChildProcess(g\_hChildStd\_OUT\_Wr\_2, g\_hChildStd\_IN\_Rd\_2, child2);  std::string outFileName1, outFileName2;  std::getline(inFile, outFileName1);  std::getline(inFile, outFileName2);  // Доступ к записи,,,всегда создавать(перезаписывать) файл, не уст. др. атрибуты,  g\_hOutputFile1 = CreateFile((LPCSTR)outFileName1.c\_str(),  FILE\_SHARE\_WRITE, 0, NULL, CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, NULL);  g\_hOutputFile2 = CreateFile((LPCSTR)outFileName2.c\_str(),  FILE\_SHARE\_WRITE, 0, NULL, CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, NULL);  WriteToPipe(g\_hChildStd\_IN\_Wr, g\_hChildStd\_IN\_Wr\_2, inFile);  ReadFromPipe(g\_hChildStd\_OUT\_Rd, g\_hOutputFile1);  ReadFromPipe(g\_hChildStd\_OUT\_Rd\_2, g\_hOutputFile2);  return 0;  }  void CreateChildProcess(HANDLE ChildStd\_OUT\_Wr, HANDLE ChildStd\_IN\_Rd, std::string child)  {  // string -> TCHAR  TCHAR \*szCmdline = 0;  szCmdline = new TCHAR[child.size() + 1];  copy(child.begin(), child.end(), szCmdline);  szCmdline[child.size()] = 0;  PROCESS\_INFORMATION piProcInfo;  STARTUPINFO siStartInfo;  BOOL bSuccess = FALSE;  ZeroMemory(&piProcInfo, sizeof(PROCESS\_INFORMATION));  // STARTUPINFO structure.  // Определяем дескрипторы STDIN и STDOUT для перенаправления.  ZeroMemory(&siStartInfo, sizeof(STARTUPINFO));  siStartInfo.cb = sizeof(STARTUPINFO);  siStartInfo.hStdError = ChildStd\_OUT\_Wr;  siStartInfo.hStdOutput = ChildStd\_OUT\_Wr;  siStartInfo.hStdInput = ChildStd\_IN\_Rd;  siStartInfo.dwFlags |= STARTF\_USESTDHANDLES;  bSuccess = CreateProcess(NULL,  szCmdline, // command line  NULL, // process security attributes  NULL, // primary thread security attributes  TRUE, // handles are inherited  0, // creation flags  NULL, // use parent's environment  NULL, // use parent's current directory  &siStartInfo, // STARTUPINFO pointer  &piProcInfo); // receives PROCESS\_INFORMATION  if (!bSuccess)  std::cout << "Error created process\n";  else  {  CloseHandle(piProcInfo.hProcess);  CloseHandle(piProcInfo.hThread);  // Зыкрываем дескрипторы дочернего процесса и его основного потока  // Закрываем дескрипторы каналов stdin и stdout, которые больше не нужны дочернему процессу.  CloseHandle(ChildStd\_OUT\_Wr);  CloseHandle(ChildStd\_IN\_Rd);  }  }  int PipeStdOut(HANDLE &g\_hChildStd\_OUT\_Rd, HANDLE &g\_hChildStd\_OUT\_Wr, SECURITY\_ATTRIBUTES saAttr)  {  if (!CreatePipe(&g\_hChildStd\_OUT\_Rd, &g\_hChildStd\_OUT\_Wr, &saAttr, 0))  {  std::cout << "Error Pipes\n";  return -1;  }  // g\_hChildStd\_OUT\_rd не должен наследоваться  if (!SetHandleInformation(g\_hChildStd\_OUT\_Rd, HANDLE\_FLAG\_INHERIT, 0))  return -2;  }  int PipeStdIn(HANDLE &g\_hChildStd\_IN\_Rd, HANDLE &g\_hChildStd\_IN\_Wr, SECURITY\_ATTRIBUTES saAttr)  {  if (!CreatePipe(&g\_hChildStd\_IN\_Rd, &g\_hChildStd\_IN\_Wr, &saAttr, 0))  {  std::cout << "Error Pipes\n";  return -1;  }  // g\_hChildStd\_IN\_Wr не должен наследоваться  if (!SetHandleInformation(g\_hChildStd\_IN\_Wr, HANDLE\_FLAG\_INHERIT, 0))  return -2;  } |

|  |
| --- |
| **Child1.cpp** |
| #include <windows.h>  #include <stdio.h>  #include <algorithm>  #include <string>  #include <iostream>  int main(void)  {  DWORD dwWritten;  HANDLE hStdout;  BOOL bSuccess;  hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);  if (hStdout == INVALID\_HANDLE\_VALUE)  ExitProcess(1);  std::string s;  while (std::getline(std::cin, s))  {  std::reverse(s.begin(), s.end());  s += "\n";  bSuccess = WriteFile(hStdout,  s.c\_str(), s.size(), &dwWritten, NULL);  if (!bSuccess)  break;  }  return 0;  } |

|  |
| --- |
| **Child2.cpp** |
| #include <windows.h>  #include <stdio.h>  #include <algorithm>  #include <string>  #include <iostream>  int main(void)  {  DWORD dwWritten;  HANDLE hStdout;  BOOL bSuccess;  hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);  if (hStdout == INVALID\_HANDLE\_VALUE)  ExitProcess(1);  std::string s;  while (std::getline(std::cin, s))  {  std::reverse(s.begin(), s.end());  s += "\n";  bSuccess = WriteFile(hStdout,  s.c\_str(), s.size(), &dwWritten, NULL);  if (!bSuccess)  break;  }  return 0;  } |

|  |
| --- |
| **utils.cpp** |
| #include "utils.h"  void WriteToPipe(HANDLE g\_hChildStd\_IN\_Wr, HANDLE g\_hChildStd\_IN\_Wr\_2, std::istream &inFile)  // Чтение из ifstream и запись его содержимого в каналы для дочерних STDIN.  {  DWORD dwWritten;  BOOL bSuccess = FALSE;  std::string s;  while (std::getline(inFile, s))  {  // фильтрация строк по варианту  s += "\n";  if ((s.size() - 1) % 2 == 1)  {  bSuccess = WriteFile(g\_hChildStd\_IN\_Wr, s.c\_str(),  s.size(), &dwWritten, NULL);  if (!bSuccess)  break;  }  else  {  bSuccess = WriteFile(g\_hChildStd\_IN\_Wr\_2, s.c\_str(),  s.size(), &dwWritten, NULL);  if (!bSuccess)  break;  }  }  // Закрываем дескрипторы каналов, чтобы дочерние процессы прекратили чтение.  if (!CloseHandle(g\_hChildStd\_IN\_Wr\_2))  std::cout << "error closed ChildStd\_IN\_Wr\_2";  if (!CloseHandle(g\_hChildStd\_IN\_Wr))  std::cout << "error ChildStd\_IN\_Wr";  }  void ReadFromPipe(HANDLE g\_hChildStd\_OUT\_Rd, HANDLE g\_hOutputFile)  // Чтение вывода из канала дочернего процесса\_1 для STDOUT  // И записываем в файл\_1.  {  DWORD dwRead, dwWritten;  CHAR chBuf[BUFSIZE];  BOOL bSuccess = FALSE;  for (;;)  {  bSuccess = ReadFile(g\_hChildStd\_OUT\_Rd, chBuf,  BUFSIZE, &dwRead, NULL);  if (!bSuccess || dwRead == 0)  break;  bSuccess = WriteFile(g\_hOutputFile, chBuf,  dwRead, &dwWritten, NULL);  if (!bSuccess)  break;  }  if (!CloseHandle(g\_hChildStd\_OUT\_Rd))  std::cout << "error closed ChildStd\_OUT\_Rd";  if (!CloseHandle(g\_hOutputFile))  std::cout << "error closed OutputFile";  } |

**Выводы**

Составлена и отлажена программа на языке Си, осуществляющая работу с процессами. Тем самым, приобретены навыки в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов.