Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 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/lab4

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. Обеспечение обмена данных между процессами посредством технологии «File mapping»

**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 через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Общие сведения о программе**

Используются системные вызовы Windows

**Исходный код**

|  |
| --- |
| **Main.cpp** |
| #include "utils.h"  #include "parent.h"  int main()  {  ParentRoutine(std::cin);  return 0;  } |
| Parent.cpp |
| #include "utils.h"  #include "parent.h"  int ParentRoutine(std::istream &inFile)  {  HANDLE hMapFile; // handle for the file's memory-mapped region  HANDLE hFile;  LPVOID lpMapAddress;  char child1[512];  char child2[512];  ExpandEnvironmentStrings("%FIRST\_PROCESS%", (char \*)child1, sizeof(child1));  ExpandEnvironmentStrings("%SECOND\_PROCESS%", (char \*)child2, sizeof(child2));  TCHAR \*lpcTheFile = TEXT("fmtest.txt");  LPCSTR FileMapName = "mainfile";  hFile = CreateFile(lpcTheFile,  GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, NULL);  hMapFile = CreateFileMapping(hFile, // current file handle  NULL, // default security  PAGE\_READWRITE, // read/write permission  0, // size of mapping object, high  1024, FileMapName);  lpMapAddress = MapViewOfFile(hMapFile, // handle to  FILE\_MAP\_ALL\_ACCESS, // read/write  0, 0, 0);  //----------------------------  WriteToMapFile(inFile, lpMapAddress);  HANDLE ProcInfo[2];  ProcInfo[0] = CreateChildProcess(child1);  ProcInfo[1] = CreateChildProcess(child2);  WaitForMultipleObjects(2, ProcInfo, TRUE, INFINITE);  for (int i = 0; i < 2; i++)  CloseHandle(ProcInfo[i]);  UnmapViewOfFile(lpMapAddress);  CloseHandle(hMapFile);  CloseHandle(hFile);  remove("fmtest.txt");  return 0;  }  HANDLE CreateChildProcess(char \*child)  {  PROCESS\_INFORMATION piProcInfo = {0};  STARTUPINFO siStartInfo = {0};  BOOL bSuccess = FALSE;  bSuccess = CreateProcess(NULL,  child, // 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::cerr << "Error created process\n";  else  {  CloseHandle(piProcInfo.hThread);  }  return piProcInfo.hProcess;  } |
| Child1.cpp |
| #include <windows.h>  #include <stdio.h>  #include <algorithm>  #include <string>  #include <iostream>  int main(void)  {  LPCSTR FileMapName = "mainfile";  HANDLE hMapFile;  LPVOID lpMapAddress;  std::string name\_OutputFile;  HANDLE g\_hOutputFile1;  std::string result;  DWORD dwWritten;  hMapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, FileMapName);  if (hMapFile == NULL)  std::cerr << "PROCESS 1. UNABLE TO OPEN FILE PROJECTION OBJECT";  lpMapAddress = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 0);  if (lpMapAddress == NULL)  std::cerr << "PROCESS 1. PROJECTED FILE PRESENTATION IS NOT POSSIBLE";  TCHAR \*datafromfile = (TCHAR \*)lpMapAddress;  std::string s(datafromfile);  int countstr = 0;  int i = 0;  while (s[i] != '\n')  {  name\_OutputFile += s[i];  i++;  }  g\_hOutputFile1 = CreateFile(name\_OutputFile.c\_str(),  FILE\_SHARE\_WRITE, 0, NULL, CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, NULL);  if (g\_hOutputFile1 == INVALID\_HANDLE\_VALUE)  std::cerr << "PROCESS 1. ERROR IN CREATING THE OUTPUT FILE\n";  for (int j = i + 1; j < s.size(); j++)  {  if (s[j] != '\n')  result += s[j];  else  {  countstr += 1;  if ((result.size() % 2 == 0) && (countstr != 1))  {  reverse(result.begin(), result.end());  result += '\n';  if (!WriteFile(g\_hOutputFile1, result.c\_str(),  result.size(), &dwWritten, NULL))  std::cerr << "PROCESS 1. WRITE ERROR\n";  }  result = "";  }  }  UnmapViewOfFile(lpMapAddress);  CloseHandle(hMapFile);  return 0;  } |
| Child2.cpp |
| #include <windows.h>  #include <stdio.h>  #include <algorithm>  #include <string>  #include <iostream>  int main(void)  {  LPCSTR FileMapName = "mainfile";  HANDLE hMapFile;  LPVOID lpMapAddress;  std::string name\_OutputFile;  HANDLE g\_hOutputFile1;  std::string result;  DWORD dwWritten;  hMapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, FileMapName);  if (hMapFile == NULL)  std::cerr << "PROCESS 2. UNABLE TO OPEN FILE PROJECTION OBJECT";  lpMapAddress = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 0);  if (lpMapAddress == NULL)  std::cerr << "PROCESS 2. PROJECTED FILE PRESENTATION IS NOT POSSIBLE";  TCHAR \*datafromfile = (TCHAR \*)lpMapAddress;  std::string s(datafromfile);  int countstr = 0;  int i = 0;  while (i < s.size())  {  if (s[i] != '\n')  name\_OutputFile += s[i];  else  {  countstr++;  if (countstr == 2)  break;  else  name\_OutputFile = "";  }  i++;  }  g\_hOutputFile1 = CreateFile(name\_OutputFile.c\_str(),  FILE\_SHARE\_WRITE, 0, NULL, CREATE\_ALWAYS,  FILE\_ATTRIBUTE\_NORMAL, NULL);  if (g\_hOutputFile1 == INVALID\_HANDLE\_VALUE)  std::cerr << "PROCESS 2. ERROR IN CREATING THE OUTPUT FILE\n";  for (int j = i; j < s.size(); j++)  {  if (s[j] != '\n')  result += s[j];  else  {  if (result.size() % 2 == 1)  {  reverse(result.begin(), result.end());  result += '\n';  if (!WriteFile(g\_hOutputFile1, result.c\_str(),  result.size(), &dwWritten, NULL))  std::cerr << "PROCESS 2. WRITE ERROR\n";  }  result = "";  }  }  UnmapViewOfFile(lpMapAddress);  CloseHandle(hMapFile);  return 0;  } |
| utils.cpp |
| #include "utils.h"  void WriteToMapFile(std::istream &inFile, LPVOID lpMapAdress)  {  std::string s;  std::string resstr = "";  while (std::getline(inFile, s))  resstr += (s + '\n');  CopyMemory(lpMapAdress, resstr.c\_str(), resstr.size());  } |

**Выводы**

Составлена и отлажена программа на языке Си, осуществляющая работу и взаимодействие между процессами с использованием отображаемых файлов. Так, получены навыки в обеспечении обмена данных между процессами посредством технологии «File mapping».