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

Репозиторий

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

Постановка задачи

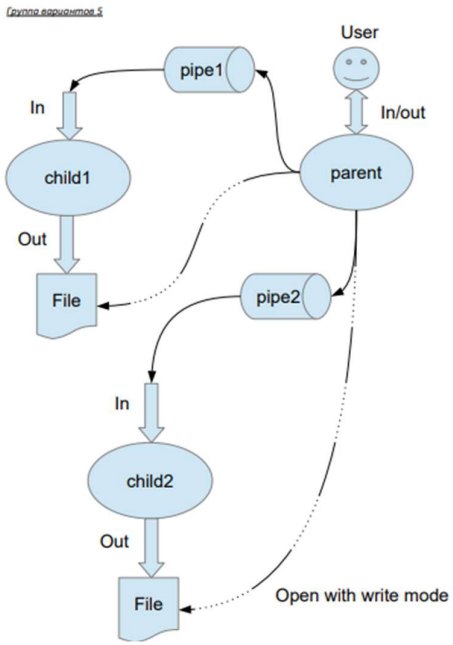
Цель работы

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)
	// Стандартный 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. атри-
буты,
	g_hOutputFile1 = CreateFile((LPCSTR)outFileName1.c_str(),
                                                                    FILE_SHARE_WRITE,
0, NULL, CREATE_ALWAYS,
                                                                    FILE_ATTRIB-
UTE_NORMAL, NULL);
	g_hOutputFile2 = CreateFile((LPCSTR)outFileName2.c_str(),
                                                                    FILE_SHARE_WRITE,
0, NULL, CREATE_ALWAYS,
                                                                    FILE_ATTRIB-
UTE_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_ATTRIB-
UTES 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(), &dwWrit-
ten, NULL);
			if (!bSuccess)
				break;
		}
		else
		{
			bSuccess = WriteFile(g_hChildStd_IN_Wr_2, s.c_str(),
                                                                s.size(), &dwWrit-
ten, 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.