Отчет по лабораторной работе №7

«Взаимодействие процессов WinAPI»

Выполнил: студент группы РИС-22-1б

Поважный Виталий Евгеньевич

Вариант 12 (4)

01.03.2025

**Задание**

Цель работы: научиться организовывать взаимодействие двух процессов в среде Windows.

Необходимо выполнить следующие задачи:

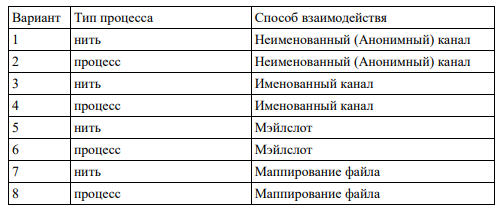
1. Создать два процесса, тип процесса определить по номеру варианта.

2. Процесс ПЕРВЫЙ должен передать ВТОРОМУ сигнал событие.

3. ВТОРОЙ процесс должен послать первому произвольную строку, используя способ взаимодействия процессов согласно варианту.

4. ПЕРВЫЙ процесс должен напечатать на экране полученные данные.

Варианты:



Мой вариант: 12 % 8 = 4;

**Ход работы**

**Описание работы:**

1. **Сервер**:
   * Создаёт именованное событие.
   * Создаёт именованный канал (Named Pipe).
   * Сигнализирует клиенту, что можно подключаться.
   * Ожидает подключения.
   * Читает данные от клиента и выводит их.
2. **Клиент**:
   * Ожидает сигнала от сервера.
   * Подключается к именованному каналу.
   * Отправляет строку серверу.

Пример работы программы ниже:

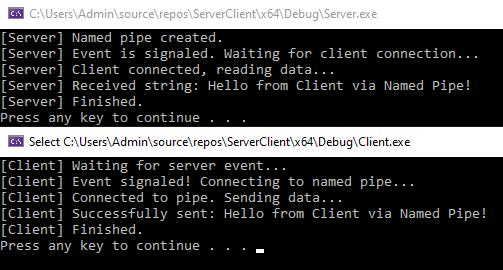


Рисунок 1 – Пример работы программы.

Сервер:

#include <windows.h>

#include <stdio.h>

int main(void)

{

// 1) Создаём (именованное) событие.

LPCWSTR eventName = L"Global\\MyEventForLab7";

HANDLE hEvent = CreateEventW(

NULL, // LPSECURITY\_ATTRIBUTES (NULL = параметры по умолчанию)

FALSE, // сбрасываемое событие (auto-reset)

FALSE, // начальное состояние nonsignaled

eventName

);

if (hEvent == NULL) {

DWORD err = GetLastError();

printf("CreateEvent failed. Error = %lu\n", err);

return 1;

}

// 2) Создаём именованный канал (Named Pipe).

LPCWSTR pipeName = L"\\\\.\\pipe\\MyNamedPipe";

HANDLE hPipe = CreateNamedPipeW(

pipeName,

PIPE\_ACCESS\_INBOUND, // сервер будет читать (вход)

PIPE\_TYPE\_BYTE | // побайтовый тип канала

PIPE\_READMODE\_BYTE |

PIPE\_WAIT, // блокирующий режим

1, // максимальное число экземпляров

0, // размер выходного буфера (0 = по умолчанию)

0, // размер входного буфера

0, // время ожидания в мс (0 = по умолчанию)

NULL // LPSECURITY\_ATTRIBUTES

);

if (hPipe == INVALID\_HANDLE\_VALUE) {

DWORD err = GetLastError();

printf("CreateNamedPipe failed. Error = %lu\n", err);

CloseHandle(hEvent);

return 1;

}

printf("[Server] Named pipe created.\n");

// 3) Сигнализируем второму процессу, что канал создан и можно подключаться.

if (!SetEvent(hEvent)) {

DWORD err = GetLastError();

printf("SetEvent failed. Error = %lu\n", err);

CloseHandle(hPipe);

CloseHandle(hEvent);

return 1;

}

printf("[Server] Event is signaled. Waiting for client connection...\n");

// 4) Ожидаем подключения клиента.

if (!ConnectNamedPipe(hPipe, NULL)) {

DWORD err = GetLastError();

// Если вернулась ошибка ERROR\_PIPE\_CONNECTED, значит клиент уже подключился.

if (err != ERROR\_PIPE\_CONNECTED) {

printf("ConnectNamedPipe failed. Error = %lu\n", err);

CloseHandle(hPipe);

CloseHandle(hEvent);

return 1;

}

}

printf("[Server] Client connected, reading data...\n");

// 5) Читаем строку, которую пришлёт клиент.

char buffer[256];

DWORD bytesRead = 0;

ZeroMemory(buffer, sizeof(buffer));

BOOL success = ReadFile(

hPipe,

buffer,

sizeof(buffer) - 1, // -1, чтобы оставить место под '\0'

&bytesRead,

NULL

);

if (!success || bytesRead == 0) {

DWORD err = GetLastError();

printf("[Server] ReadFile failed or no data. Error = %lu\n", err);

}

else {

buffer[bytesRead] = '\0';

printf("[Server] Received string: %s\n", buffer);

}

// 6) Закрываем дескрипторы.

CloseHandle(hPipe);

CloseHandle(hEvent);

printf("[Server] Finished.\n");

system("pause");

return 0;

}

Клиент:

#include <windows.h>

#include <stdio.h>

int main(void)

{

// 1) Открываем ранее созданное событие.

LPCWSTR eventName = L"Global\\MyEventForLab7";

HANDLE hEvent = OpenEventW(

SYNCHRONIZE, // права

FALSE, // флаг унаследования

eventName

);

if (hEvent == NULL) {

DWORD err = GetLastError();

printf("[Client] OpenEvent failed. Error = %lu\n", err);

return 1;

}

printf("[Client] Waiting for server event...\n");

// 2) Ждём события от сервера (бесконечно).

DWORD waitRes = WaitForSingleObject(hEvent, INFINITE);

if (waitRes != WAIT\_OBJECT\_0) {

DWORD err = GetLastError();

printf("[Client] WaitForSingleObject failed. Error = %lu\n", err);

CloseHandle(hEvent);

return 1;

}

printf("[Client] Event signaled! Connecting to named pipe...\n");

// 3) Подключаемся к именованному каналу.

LPCWSTR pipeName = L"\\\\.\\pipe\\MyNamedPipe";

HANDLE hPipe = CreateFileW(

pipeName,

GENERIC\_WRITE, // клиент будет писать

0, // без совместного доступа

NULL, // LPSECURITY\_ATTRIBUTES

OPEN\_EXISTING, // открываем существующий канал

0, // флаги/атрибуты

NULL

);

if (hPipe == INVALID\_HANDLE\_VALUE) {

DWORD err = GetLastError();

printf("[Client] CreateFile for pipe failed. Error = %lu\n", err);

CloseHandle(hEvent);

return 1;

}

printf("[Client] Connected to pipe. Sending data...\n");

// 4) Пишем произвольную строку в канал.

const char\* msg = "Hello from Client via Named Pipe!";

DWORD bytesWritten = 0;

BOOL success = WriteFile(

hPipe,

msg,

(DWORD)lstrlenA(msg), // кол-во байтов для записи

&bytesWritten,

NULL

);

if (!success) {

DWORD err = GetLastError();

printf("[Client] WriteFile failed. Error = %lu\n", err);

}

else {

printf("[Client] Successfully sent: %s\n", msg);

}

// 5) Закрываем дескрипторы.

CloseHandle(hPipe);

CloseHandle(hEvent);

printf("[Client] Finished.\n");

system("pause");

return 0;

}