LAB Inter-local-process coms Anonymous Pipes

C++ Inter-local-process communications.

We can only use anonymous pipes between two processes, where one process has been "spawned/forked" from its parent process.

1 Creating two processes and an anonymous "pipe" in Windows OS.

The handbook shows how a new process can be "fork()ed" in unix and pipe created to communicate between them. This task is to see how WindowsOS implements that same idea. In Linux the code for both programs was held within one source file.

In Windows, the code for the child program is separate and compiled independently. The parent program initiates the child programme using the command CreateProcessA(...) "name of the child executable", ...)

We have two programs:

- Parent: creates the pipe, writes to it, and spawns the child.
- Child: inherits the pipe's read handle and reads the message.

Windows does not have fork() or pipe() in the same sense, instead, we use:

- CreatePipe() to create an anonymous unidirectional pipe with read and write handles.
- SetHandleInformation() prevents one end from being inherited by accident.
- CreateProcess() launches the child process and redirects its STDIN to the pipe's read end.
- WriteFile() and ReadFile() Parent writes to pipe, child reads from its standard input instead of write() / read() in Unix
- WaitForSingleObject() is for the child process to complete, instead of wait() in Unix.

Task:

1. Compile the following two programs (parent_pipe.cpp and child_reader.cpp).
2. Then run the parent_pipe executable, which will start up the child_reader executable.
3. Demonstrate that they communicate with each other. Then play about with them!

Hints:
To run them, in a console window, type: parent_pipe.exe
You should see the output:
[Parent] Sent message: Hello from the parent process!
[Child] Received message: Hello from the parent process

Parent Program (parent_pipe.cpp)

```
#include <windows.h>
#include <iostream>
#include <string>

int main() {
  HANDLE hReadPipe, hWritePipe;
  SECURITY_ATTRIBUTES saAttr;

  // Allow the pipe handles to be inherited by child process
  saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
  saAttr.bInheritHandle = TRUE;
  saAttr.lpSecurityDescriptor = NULL;

  // Create an anonymous pipe
  if (!CreatePipe(&hReadPipe, &hWritePipe, &saAttr, 0)) {
    std::cerr << "CreatePipe failed (" << GetLastError() << ")\n";
    return 1;
  }

  // Ensure the read handle is inherited, but not the write handle as we just want unidirectional.
  SetHandleInformation(hWritePipe, HANDLE_FLAG_INHERIT, 0);

  // Prepare to launch the child process
  STARTUPINFOA si;
  PROCESS_INFORMATION pi;
  ZeroMemory(&si, sizeof(si));
  ZeroMemory(&pi, sizeof(pi));
  si.cb = sizeof(si);

  // Redirect the child's standard input to the read end of the pipe
  si.hStdInput  = hReadPipe;
  si.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
  si.hStdError  = GetStdHandle(STD_ERROR_HANDLE);
  si.dwFlags |= STARTF_USESTDHANDLES;

  // Child program name (the child program must be compiled separately)
  const char* childProgram = "child_reader.exe";

  if (!CreateProcessA(
    NULL,                 // Application name
    (LPSTR)childProgram,      // Command line
    NULL, NULL, TRUE, 0,       // Inherit handles
    NULL, NULL, &si, &pi))
  {
    std::cerr << "CreateProcess failed (" << GetLastError() << ")\n";
    CloseHandle(hReadPipe);
    CloseHandle(hWritePipe);
    return 1;
  }
```

```
    // Write message to the pipe
    const char* message = "Hello from the parent process!";
    DWORD bytesWritten = 0;
    if (!WriteFile(hWritePipe, message, (DWORD)strlen(message), &bytesWritten, NULL)) {
        std::cerr << "WriteFile failed (" << GetLastError() << ")\n";
    } else {
        std::cout << "[Parent] Sent message: " << message << std::endl;
    }

    // Close handles
    CloseHandle(hWritePipe);
    CloseHandle(hReadPipe);

    // Wait for the child process to complete
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);

    return 0;
}
```

Child Program (child_reader.cpp)

```
#include <windows.h>
#include <iostream>

int main() {
    HANDLE hInput = GetStdHandle(STD_INPUT_HANDLE);
    if (hInput == INVALID_HANDLE_VALUE) {
        std::cerr << "[Child] Invalid handle.\n";
        return 1;
    }

    char buffer[128];
    DWORD bytesRead = 0;
    if (ReadFile(hInput, buffer, sizeof(buffer) - 1, &bytesRead, NULL) && bytesRead > 0) {
        buffer[bytesRead] = '\0';
        std::cout << "[Child] Received message: " << buffer << std::endl;
    } else {
        std::cerr << "[Child] ReadFile failed (" << GetLastError() << ")\n";
    }

    return 0;
}
```