

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу
«Операционные системы»**

МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ

Студент: Воробьева Анжелика Владимировна

Группа: М8О–203БВ–20

Вариант: 2

Преподаватель: Соколов Андрей Алексеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2025.

Постановка задачи

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

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdlib.h, string.h, unistd.h, sys/wait.h, fcntl.h. В программе используются следующие системные вызовы:

1. open() – он используется для открытия или создания файла
2. close() – используется для закрытия файла
3. fork() – создает дочерний процесс из родительского процесса, что приводит к точно такому же процессу, за исключением некоторых значений, таких как PID и родительский PID.
4. pipe() – создание неименованного канала для передачи данных между процессами
5. write() – используется для изменения файлов, создания файлов
6. read() – этот системный вызов можно использовать для чтения данных из широкого спектра типов данных, включая обычные файлы и специальные файлы, такие как каналы и сокеты
7. dup2() – переназначение файлового дескриптора
8. exit() – этот системный вызов завершает текущий процесс и возвращает в систему ресурсы, полученные этим процессом (который был недавно завершен)
9. wait() – этот системный вызов ожидает завершения дочернего

процесса, а затем предоставляет некоторую информацию о дочернем процессе. 10. `execl()` – этот системный вызов заменяет текущий образ процесса новым и используется, когда вы хотите запустить программу, отличную от текущего процесса

Основные файлы программы

main.c

```
#include "stdio.h"
#include "windows.h"

HANDLE g_ToChild = NULL;
HANDLE g_FromChild = NULL;
HANDLE g_FromParent = NULL;
HANDLE g_ToParent = NULL;

int CreateChildProcess();

int main() {
    SECURITY_ATTRIBUTES saAttr;
    saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
    saAttr.bInheritHandle = TRUE;
    saAttr.lpSecurityDescriptor = NULL;

    if (!CreatePipe(&g_FromChild, &g_ToParent, &saAttr, 0)) return -1;

    if (!SetHandleInformation(g_FromChild, HANDLE_FLAG_INHERIT, 0)) return -1;

    if (!CreatePipe(&g_FromParent, &g_ToChild, &saAttr, 0)) return -1;

    if (!SetHandleInformation(g_ToChild, HANDLE_FLAG_INHERIT, 0)) return -1;
    if (CreateChildProcess() != 0) return -1;

    HANDLE writeHandle;

    writeHandle = GetStdHandle(STD_INPUT_HANDLE);
    HANDLE readHandle = GetStdHandle(STD_OUTPUT_HANDLE);
    char
buffer[256];
    DWORD dwReadFileName;
    if (ReadFile(writeHandle, buffer, sizeof(buffer) - 1, &dwReadFileName,
NULL)) {
        buffer[dwReadFileName] = '\\0';
        for (int i = 0; i < 256; i++) {
            if (buffer[i] == '\\r' || buffer[i] == '\\n') {
buffer[i] = '\\0';
                break;
            }
        }
    }
    else {
        return -1;
    }
    DWORD realLen = dwReadFileName;
```



```

    WriteFile(g_ToChild, buffer, realLen, &dwReadFileName, NULL);
system("cls");

    DWORD dwRead;
char numStr[256];

    if (ReadFile(writeHandle, numStr, sizeof(numStr) - 1, &dwRead, NULL)
    && dwRead > 0) {        numStr[dwRead] = '\\0';

        int countToBreak = 0;
        for (int i = 0; i < 256; i++) {

            if (numStr[i] == '\\r' || numStr[i] == '\\n')
            {
                numStr[i] = '\\0';        break;
            }

        }

    }

    WriteFile(g_ToChild, numStr, dwRead, &dwRead, NULL);
float result =
0.0f;
DWORD bytesRead;
BOOL readSuccess =
    ReadFile(g_FromChild, &result, sizeof(float), &bytesRead, NULL);

    if (readSuccess != FALSE && bytesRead == sizeof(float))
{
    char header[] = "Result: ";    char newline[] =
    "\\r\\n";    char floatStr[50];
    WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), header, (DWORD)strlen(header),
    &dwRead, NULL);

    _gcvt_s(floatStr, sizeof(floatStr), result, 6);
    WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), floatStr,
    (DWORD)strlen(floatStr), &dwRead, NULL);
    WriteFile(GetStdHandle(STD_OUTPUT_HANDLE),
    newline, (DWORD)strlen(newline),
    &dwRead, NULL);
}

    CloseHandle(g_ToChild);
    CloseHandle(g_FromChild);
return 0;
}
int CreateChildProcess() {
    TCHAR szCmdline[] = TEXT("ChildProcess.exe");
    PROCESS_INFORMATION piProcInfo;
    STARTUPINFO siStartInfo;
    BOOL bSuccess = FALSE;

    ZeroMemory(&piProcInfo, sizeof(PROCESS_INFORMATION));

```

```

ZeroMemory(&siStartInfo, sizeof(STARTUPINFO));
siStartInfo.cb = sizeof(STARTUPINFO);    siStartInfo.hStdError
= g_ToParent;        siStartInfo.hStdOutput = g_ToParent;
siStartInfo.hStdInput = g_FromParent;    siStartInfo.dwFlags
|= STARTF_USESTDHANDLES;

// Создаем дочерний процесс
bSuccess = CreateProcess(NULL,
szCmdline,
NULL,
NULL,
TRUE,
0,
NULL,
NULL,
&siStartInfo,
&piProcInfo);

if
(!bSuccess)
return -1;
else {

    CloseHandle(piProcInfo.hProcess);
    CloseHandle(piProcInfo.hThread);

    CloseHandle(g_ToParent);
    CloseHandle(g_FromParent);
}
return 0;
}

```

ChildPro

cess.c

```

#include <stdlib.h>
#include <string.h>
#include <windows.h>
int main() {
HANDLE readHandle =
GetStdHandle(STD_INPUT_HANDLE);
HANDLE writeHandle =
GetStdHandle(STD_OUTPUT_HANDLE);
DWORD readedBytesNameFile,
readedBytesDigits, wroteBytes;
char bufferNameFile[256];
if (ReadFile(readHandle,
bufferNameFile, sizeof(bufferNameFile) -
1,
&readedBytesNameFile, NULL)) {
bufferNameFile[readedBytesNameFile] = '\0';
} else {
return -1;
}

HANDLE fileHandle =
CreateFileA(bufferNameFile, GENERIC_WRITE, 0,
NULL,
CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

```

```

    char bufferDigits[256];

    if (ReadFile(readHandle, bufferDigits,
        sizeof(bufferDigits) - 1,
        &readedBytesDigits, NULL)) {
        bufferDigits[readedBytesDigits] = '\\0';

        char* context = NULL;
        char* tok = strtok_s(bufferDigits, "
\\t\\r\\n", &context);
        float summa = 0.0f;

        while (tok != NULL) {
            summa += strtod(tok, NULL);
            tok = strtok_s(NULL, " \\t\\r\\n", &context);
        }

        if (fileHandle != INVALID_HANDLE_VALUE) {
            char header[] = "Sum: ";
            char newline[] = "\\r\\n";
            char floatStr[50];

            WriteFile(fileHandle, header,
                (DWORD)strlen(header), &writedBytes, NULL);

            _gcvt_s(floatStr, sizeof(floatStr), summa,
                6);
            WriteFile(fileHandle, floatStr,
                (DWORD)strlen(floatStr), &writedBytes,
                NULL);
            WriteFile(fileHandle, newline,
                (DWORD)strlen(newline), &writedBytes,
                NULL);
        }

        WriteFile(writeHandle, &summa,
            sizeof(float), &writedBytes, NULL);

    } else {
        if (fileHandle != INVALID_HANDLE_VALUE) {
            CloseHandle(fileHandle);
        }
        return -1;
    }

    if (fileHandle != INVALID_HANDLE_VALUE) {
        CloseHandle(fileHandle);
    }
    return 0;
}
}

```

CMAke

Lists.c:

```

cmake_minimum_required(VERSION 3.10)
project(ProcessCommunication C)

```

```
set(CMAKE_C_STANDARD 11) add_executable(main src/main.c)

add_executable(ChildProcess src/ChildProcess.c)

if(WIN32)
    target_compile_definitions(main PRIVATE
WIN32_LEAN_AND_MEAN)
    target_compile_definitions(ChildProcess PRIVATE
WIN32_LEAN_AND_MEAN)
endif()

set_target_properties(main ChildProcess
    PROPERTIES
    RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}
)
```

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы dlsym, dlopen, dlclose.
2. Написать библиотеку vecmd5, для работы с вектором из md5 сумм.
3. Организовать простейший командный интерфейс в файлах test1.c и test2.c.
4. В файле test1.c подключить библиотеку на этапе компиляции.
5. В файле test2.c загрузить библиотечные функции в runtime, с помощью dlsym, dlopen, dlclose.

.....

Вывод

В результате выполнения лабораторной работы была разработана и отлажена программа на языке Си, осуществляющую работу с процессами и взаимодействие между ними в операционной системе Windows.