**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 4**

Тема: Управление процессами в ОС

Студент: Калугин К. А.

Группа: М8О-207Б-19

Преподаватель: Миронов Е.С.

Дата:

Оценка:

Москва, 2021

**Постановка задачи:** 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.

7 вариант) В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float.

**Алгоритм решения задачи.**

Алгоритм решения аналогичен лабораторной №2, за исключением того, что обмен данными между процессами осуществляется при помощи технологии file mapping

**Листинг программы**

**1.с**

#include <stdio.h>

#include <windows.h>

HANDLE file = NULL;

HANDLE bufferfile = NULL;

HANDLE trigger;

SECURITY\_ATTRIBUTES sa;

PROCESS\_INFORMATION pi;

STARTUPINFO si;

DWORD err = 0;

float counter = 0;

int shift = 2;

char name[256];

int main () {

    HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    ZeroMemory (&sa, sizeof (SECURITY\_ATTRIBUTES));

    sa.nLength = sizeof (SECURITY\_ATTRIBUTES);

    sa.bInheritHandle = TRUE;

    bufferfile = CreateFileMapping (INVALID\_HANDLE\_VALUE, &sa, PAGE\_READWRITE, 0, 512, NULL);

    trigger = CreateSemaphore (&sa, 0, 1, NULL);

    void\* answer = MapViewOfFile (bufferfile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 512);

    scanf("%s", name);

    file = CreateFileA (//открытие файла на чтение

        name,

        GENERIC\_READ,

        0,

        &sa,

        OPEN\_EXISTING,

        FILE\_ATTRIBUTE\_NORMAL,

        NULL

    );

    if ((err = GetLastError ()) != 0)

        {

            SetConsoleTextAttribute(hConsole, (WORD) (0 | 4));

            printf ("ERROR\nIncorrect work of function \"CreateFileA\". Code of error: %ld.", err);

            SetConsoleTextAttribute(hConsole, (WORD) (0 | 7));

            CloseHandle (file);

            CloseHandle (bufferfile);

            CloseHandle (trigger);

            CloseHandle (hConsole);

            UnmapViewOfFile (bufferfile);

            return -1;

        }

    si.hStdError = trigger;

    si.hStdOutput = bufferfile;

    si.hStdInput = file;

    si.dwFlags |= STARTF\_USESTDHANDLES;

    if (!CreateProcess (NULL,

        "2.exe",

        NULL,

        NULL,

        TRUE,

        0,

        NULL,

        NULL,

        &si,

        &pi)

    )

        {

            SetConsoleTextAttribute(hConsole, (WORD) (0 | 4));

            printf ("ERROR\nIncorrect work of function \"CreateProcess\".");

            SetConsoleTextAttribute(hConsole, (WORD) (0 | 7));

            CloseHandle (file);

            CloseHandle (bufferfile);

            CloseHandle (trigger);

            CloseHandle (hConsole);

            UnmapViewOfFile (bufferfile);

            return -1;

        }

        else {

            WaitForSingleObject (trigger, INFINITE);

            if (\*(float\*) answer == 0) {

                counter = \*((float\*) answer + 1);

                for (int i = 0; i < (int) counter; i ++) {

                    printf ("%f\n", \*((float\*) answer + shift));

                    shift ++;

                }

            }

            else if (\*(float\*) answer == 1) {

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 4));

                printf ("ERROR\nIncorrect work of function \"ReadFile\" in child process.");

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 7));

                CloseHandle (file);

                CloseHandle (bufferfile);

                CloseHandle (trigger);

                CloseHandle (hConsole);

                UnmapViewOfFile (bufferfile);

                return -1;

            }

            else if (\*(float\*) answer == 2) {

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 4));

                printf ("ERROR\nInput file is empty.");

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 7));

                CloseHandle (file);

                CloseHandle (bufferfile);

                CloseHandle (trigger);

                CloseHandle (hConsole);

                UnmapViewOfFile (bufferfile);

                return -1;

            }

            else {

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 4));

                printf ("ERROR\nUnknown error, child process returned incorrect code.");

                SetConsoleTextAttribute(hConsole, (WORD) (0 | 7));

                CloseHandle (file);

                CloseHandle (bufferfile);

                CloseHandle (trigger);

                CloseHandle (hConsole);

                UnmapViewOfFile (bufferfile);

                return -1;

            }

        }

    CloseHandle (file);

    CloseHandle (bufferfile);

    CloseHandle (trigger);

    CloseHandle (hConsole);

    UnmapViewOfFile (bufferfile);

    return 0;

}

**2.с**

#include <stdio.h>

#include <windows.h>

int main()

{

    HANDLE in = GetStdHandle(STD\_INPUT\_HANDLE);

    HANDLE out = GetStdHandle(STD\_OUTPUT\_HANDLE);

    HANDLE err = GetStdHandle(STD\_ERROR\_HANDLE);

    DWORD D;

    void\* bufferfile = MapViewOfFile (out, FILE\_MAP\_ALL\_ACCESS, 0, 0, 512);

    float x = 0, y = 0;

    char flag;

    int shift = 2;

    float counter = 0;

    while (1) {

        if (!ReadFile (in, &y, sizeof(float), &D, NULL)) {

            \*((float\*) bufferfile) = 1;

            ReleaseSemaphore (err, 1, NULL);

            break;

        }

        x += y;

        flag = '@';

        if (!ReadFile (in, &flag, sizeof(char), &D, NULL)) {

            \*((float\*) bufferfile) = 1;

            ReleaseSemaphore (err, 1, NULL);

            break;

        }

        if (flag == '\n') {

            \*((float\*) bufferfile + shift) = x;

            shift ++;

            counter ++;

            x = 0;

        }

        else if (flag == '@') {

            if (counter == 0) {

                \*((float\*) bufferfile) = 2;

                ReleaseSemaphore (err, 1, NULL);

                break;

            }

            else {

                \*((float\*) bufferfile) = 0;

                \*((float\*) bufferfile + 1) = counter;

                ReleaseSemaphore (err, 1, NULL);

                break;

            }

        }

    }

    CloseHandle (in);

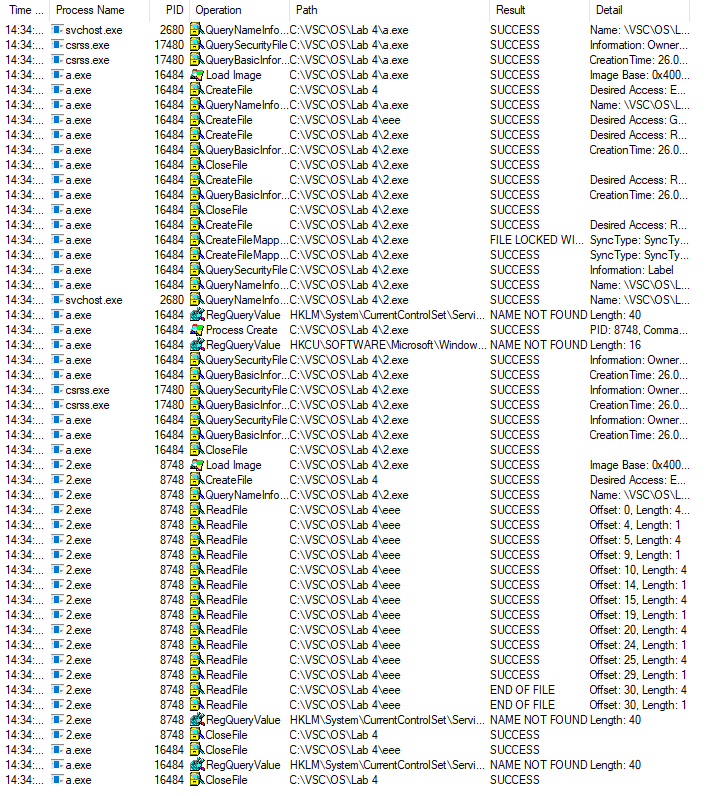
    CloseHandle (out);

    CloseHandle (err);

    UnmapViewOfFile (bufferfile);

    return 0;

}

**ProcMon**

**Выводы**

Я изучил использование file-mapping`а в Windows. Это достаточно удобный способ обмена данными, более гибкий, чем pipe`ы.

**Список литературы**

1. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. —СПб.: Издательский дом «Питер», 2018. — С. 111 - 123