Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-215Б-23

Студент: Моисеев К.В.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 18.10.24

Москва, 2024

Постановка задачи

Вариант 7.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. 6 вариант) В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным. 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. Количество чисел может быть произвольным

Общий метод и алгоритм решения

В лабораторной работе используются следующие ключевые системные вызовы и функции для работы с процессами и каналами в операционной системе Windows:

1. **CreatePipe()**

Описание: Создает анонимный канал (pipe), который позволяет одному процессу передавать данные другому.
Использование: В лабораторной работе используется для создания канала между родительским и дочерним процессом, через который данные передаются от дочернего процесса к родительскому.
Аргументы: Указатели на два дескриптора — для чтения и записи, а также структура SECURITY_ATTRIBUTES для наследования дескрипторов дочерним процессом.

2. **CreateProcess()**

Описание: Создает новый процесс, а также поток, ассоциированный с ним.
Использование: Родительский процесс создает дочерний процесс с помощью этого вызова, передавая дочернему процессу имя файла, из которого он будет читать данные.
Аргументы: Указаны параметры командной строки, флаги безопасности, и дескрипторы ввода-вывода, которые перенаправлены в канал.

3. **ReadFile()**

Описание: Считывает данные из файла или канала.
Использование: Родительский процесс использует его для чтения данных, которые передаются из дочернего процесса через канал.
Аргументы: Указатель на дескриптор чтения из канала, буфер для данных, размер буфера, и переменную для хранения количества прочитанных байтов.

4. **WaitForSingleObject()**

Описание: Ожидает завершения указанного процесса или потока.
Использование: Родительский процесс использует этот вызов для ожидания завершения дочернего процесса перед завершением своей работы.

Аргументы: Дескриптор процесса и тайм-аут ожидания (в данном случае — бесконечный).

5. **CloseHandle()**

Описание: Закрывает дескрипторы объектов Windows (процессов, потоков, каналов).
Использование: Закрывает дескрипторы, такие как дескрипторы процессов и каналов, после их использования, чтобы освободить ресурсы.

Эти системные вызовы обеспечивают взаимодействие процессов (через каналы) и создание новых процессов для выполнения программ.

Родительский процесс (ParentProcess)

Запрашивает имя файла у пользователя, создает pipe для связи с дочерним процессом, запускает дочерний процесс, передавая ему имя файла как аргумент, и читает результаты суммы из дочернего процесса.

Дочерний процесс (ChildProcess)

Проверяет, что имя файла передано, открывает файл для чтения, считывает строки и делит их на токены, суммирует числа с плавающей точкой и выводит сумму на стандартный вывод (stdout), который перенаправляется в родительский процесс. Также обрабатывает ошибки, выводя сообщения о проблемах с открытием файла.

Основные моменты

Взаимодействие между процессами происходит через pipe, используется strtok() и atof() для обработки данных, обеспечивая эффективные вычисления в дочернем процессе.

Код программы

ParentProcess.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

int main() {

    char fileName[256];

    printf("Enter the name of the file: ");

    scanf("%s", fileName);

    HANDLE pipeRead, pipeWrite;

    SECURITY_ATTRIBUTES sa = { sizeof(SECURITY_ATTRIBUTES), NULL, TRUE };

    if (!CreatePipe(&pipeRead, &pipeWrite, &sa, 0)) {
```

```
        fprintf(stderr, "Pipe creation failed\n");
        return 1;
    }
    PROCESS_INFORMATION pi;
    STARTUPINFO si;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    si.hStdOutput = pipeWrite;
    si.dwFlags |= STARTF_USESTDHANDLES;
    char cmdLine[512];
    sprintf(cmdLine, "ChildProcess.exe %s", fileName);
    if (!CreateProcess(NULL, cmdLine, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi))
{
        fprintf(stderr, "Process creation failed\n");
        return 1;
    }
    CloseHandle(pipeWrite);
    char buffer[128];
    DWORD bytesRead;
    while (ReadFile(pipeRead, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {
        buffer[bytesRead] = '\0';
        printf("%s", buffer);
    }
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    CloseHandle(pipeRead);
    return 0;
}
```

ChildProceess.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char* argv[]) {

    if (argc < 2) {

        fprintf(stderr, "File name not provided\n");

        return 1;

    }

    FILE *file = fopen(argv[1], "r");

    if (!file) {

        fprintf(stderr, "Failed to open the file\n");

        return 1;

    }

    char line[256];

    while (fgets(line, sizeof(line), file)) {

        float sum = 0;

        char *token = strtok(line, " ");

        while (token != NULL) {

            sum += atof(token);

            token = strtok(NULL, " ");

        }

        printf("Sum: %.2f\n", sum);

    }

    fclose(file);

    return 0;

}
```

test_example.cpp

```c
#include <gtest/gtest.h>
```

```
TEST(SumTest, BasicAssertions) {

    float a = 1.2, b = 2.3;

    EXPECT_FLOAT_EQ(a + b, 3.5);

}
```

test_input.txt

1.5 2.5 3.0
4.0 5.1
10.5 0.5 0.0
7.2 -2.1 3.0
8.5 9.5

Протокол работы программы

PS C:\Users\klosh\Desktop\OS\build>
.\ParentProcess.exe ..\test\test_input.txt
Enter the name of the file: ..\test\test_input.txt
Sum: 7.00
Sum: 9.10
Sum: 11.00
Sum: 8.10
Sum: 18.00
PS C:\Users\klosh\Desktop\OS\build> ctest
>>
Test project C:/Users/klosh/Desktop/OS/build
Start 1: MyTests
1/1 Test #1: MyTests ..... Passed 0.12 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) = 0.17 sec

Вывод

Выполнив данную лабораторную работу, я понял принцип работы с процессами в ОС и обеспечения обмена данными между процессами посредствам каналов. Я познакомился с понятием pipe, fork, узнал о существовании и принципах работы таких функций Windows API, обеспечивающих работу с процессами, как: CreatePipe, CreateProcess, WriteFile, ReadFile, CloseHandle и WaitForSingleObject. Кроме того, благодаря заданию моего варианта я узнал, как можно написать генератор случайных чисел на языке C. Данная работа не только развила мои навыки работы с процессами ОС, но и улучшила моё понимание языка C