

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работ №2 по курсу
«Операционные системы»**

Управление процессами в ОС

Студент: Пермяков Никита Александрович
Группа: М80 – 208Б-19
Вариант: 3
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Содержание

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Демонстрация работы программы
6. Вывод

Постановка задачи.

Написать программу на языке Си по работе с очередью. Команды считываются в родительском процессе и поступают на обработку в дочерний процесс, в котором происходят все вычисления, после чего ответ поступает в родительский процесс. 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 двух файлов: `parent.cpp` и `child.cpp`. Не используются заголовочные файлы. В программе используются следующие системные вызовы:

1. **read** – для чтения данных из файла.
2. **write** – для записи данных в файл.
3. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса. Возвращает два дескриптора файлов: для чтения и для записи.
4. **fork** – для создания дочернего процесса.
5. **close** – для закрытия файла после окончания работы с ним.

Общий метод и алгоритм решения.

Пользователь вводит команды вида: «число число число <endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int.

Файлы программы.

Файл parent.cpp

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

HANDLE g_hChildStd_IN_Rd = NULL;
HANDLE g_hChildStd_IN_Wr = NULL;
HANDLE g_hChildStd_OUT_Rd = NULL;
HANDLE g_hChildStd_OUT_Wr = NULL;

int CreateChildProcess();

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

    if (!CreatePipe(&g_hChildStd_OUT_Rd, &g_hChildStd_OUT_Wr, &saAttr, 0))
        return -1;

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

    if (!CreatePipe(&g_hChildStd_IN_Rd, &g_hChildStd_IN_Wr, &saAttr, 0))
        return -1;
```

```

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

if (!CreateChildProcess())
    return -1;

DWORD dwWritten, dwRead;

char ch;
int num = 0;
int count_request = 0;

int x, result;

while (true) {
    scanf_s("%c", &ch, 1);

    if (ch == 0x20 || ch == 0x0A) { // code space key or code enter key
        x = num;

        WriteFile(g_hChildStd_IN_Wr, &x, sizeof(int), &dwWritten, NULL);

        if (count_request != 0) {
            if (!ReadFile(g_hChildStd_OUT_Rd, &result, sizeof(int), &dwRead, NULL))
                return -1;

            printf("Result call %d: %d\n", count_request, result);
        }
        count_request++;

        if (ch == 0x20) {
            num = 0;
            continue;
        }

        if (ch == 0x0A)
            break;
    }

    num = 10 * num + (ch - '0');
}

```

```

    printf("Finish calc\n");

    return 0;
}

int CreateChildProcess()
{
    TCHAR szCmdline[] = TEXT("child.exe");
    PROCESS_INFORMATION piProcInfo;
    STARTUPINFO siStartInfo;
    BOOL bSuccess = FALSE;

    ZeroMemory(&piProcInfo, sizeof(PROCESS_INFORMATION));

    // Set up members of the STARTUPINFO structure.
    // This structure specifies the STDIN and STDOUT handles for redirection
    .

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

    // Create the child process.

    bSuccess = CreateProcess(NULL,
        szCmdline,      // command line
        NULL,           // process security attributes
        NULL,           // primary thread security attributes
        TRUE,           // handles are inherited
        0,              // creation flags
        NULL,           // use parent's environment
        NULL,           // use parent's current directory
        &siStartInfo,    // STARTUPINFO pointer
        &piProcInfo);    // receives PROCESS_INFORMATION

    // If an error occurs, exit the application.
    if (!bSuccess)
        return -1;
    else
    {

```

```

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

        CloseHandle(g_hChildStd_OUT_Wr);
        CloseHandle(g_hChildStd_IN_Rd);
    }
}

```

Файл child.cpp

```

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

int main()
{
    HANDLE readHandle = GetStdHandle(STD_INPUT_HANDLE);
    HANDLE writeHandle = GetStdHandle(STD_OUTPUT_HANDLE);
    DWORD readedBytes, writedByes;

    int x, divider = 0;
    int count_run = 0;

    int res;

    while (true) {
        if (!ReadFile(readHandle, &x, sizeof(int), &readedBytes, NULL))
            return -1;

        if (count_run == 0) {
            divider = x;
        }

        if (count_run > 0) {
            if (divider != 0)
                res = x / divider;
            else
                res = 0;
            WriteFile(writeHandle, &res, sizeof(int), &writedByes, NULL);
        }
        count_run++;
    }

    return 0;
}

```

Демонстрация работы программы.

3 9 76 343 875 2323 765 344

```
Result call 1: 3
```

Result call 2: 25

Result call 3: 114

Result call 4: 291

Result call 5: 774

Result call 6: 255

Result call 7: 114

Finish calc

45 6875 2348234 45 2 32 43 54 450 543

Result call 1: 152

Result call 2: 52182

```
Result call 3: 1
```

Result call 4: 0

Result call 5: 0

Result call 6: 0

Result call 7: 1

Result call 8: 10

Result call 9: 12

Finish calc

Создание дочернего процесса

The screenshot displays two windows from Sysinternals:

- Process Monitor - Sysinternals.com**: Shows a log of system events. The main pane lists numerous 'services.exe' processes performing operations like 'RegOpenKey', 'RegQueryValue', and 'RegCloseKey' on paths such as 'HKLM\System\CurrentControlSet\Services'. The status column indicates 'SUCCESS' for all listed events.
- Process Tree**: Provides a detailed view of a selected process, 'parent.exe (22928)'. It includes:
 - Process List**: A table listing running processes with columns for Name, Description, Image Path, Life Time, Company, Owner, Command, Start Time, and End Time. Processes include 'conhost', 'ppbtool', 'Code.exe', 'CodeHelper.exe', 'Visual Studio Code', 'WMI', 'smartscreen.exe', 'wmpirvse.exe', 'WUDFHost.exe', and 'svchost.exe'.
 - Description**: Details about the selected process, including its path ('C:\Users\permi\source\repos\university\os12_child_parent_proc\Debug\parent.exe'), command, user ('DELL\permi'), PID (22928), and start time (07.11.2020 18:17:42).

Программа позволяет считывать и обрабатывать команды от пользователя запуская дочерний процесс, для обработки вводимых данных, и возврата ответа.

Взаимодействие между процессами осуществляется через системные сигналы/события. В работе получены навыки программирования запросов — ответов, на подобии серверного взаимодействия между собой или с клиентом.