

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

## ЛАБОРАТОРНАЯ РАБОТА №2

по курсу операционные системы  
I семестр, 2019/20 уч. год

Студент Попов Данила Андреевич, группа 08-208Б-18

Преподаватель Миронов Евгений Сергеевич

## Условие

Написать программу, которая рекурсивно вычисляет  $n$ -ое число Фибоначи.

## Описание программы

Код программы состоит из 3-х файлов:

1. `main.c`: файл, содержащий точку входа приложения и верхний уровень абстракции реализации рекурсии.
2. `childprocess.h`: файл, содержащий объявление функций, необходимых для создания дочерних процессов.
3. `childprocess.c`: файл, содержащий реализацию функций, необходимых для создания дочерних процессов.

## Ход выполнения программы

1. Чтение индекса  $n$  вычисляемого числа Фибоначи.
2. Если индекс достаточно велик, то создаётся два дочерних процесса с перенаправленными потоками ввода и вывода:
  - (a) Передача в дочерние процессы через перенаправленные потоки ввода  $n - 1$  и  $n - 2$  соответственно.
  - (b) Чтение результатов через перенаправленные потоки вывода дочерних процессов.
  - (c) Суммирование результатов.

Иначе выбирается заранее вычисленное число по индексу.

3. Вывод конечного результата в стандартный поток вывода.
4. Завершение работы программы.

## Недочёты

Примерно 65000 активных процессов при вычислении 16-го числа.





## Выводы

Рекурсивное вычисление чисел Фибоначи через создание дочерних процессов — не лучший способ решения данной задачи.

# Вывод программы

4  
2

## procmon log

Time of Day	Process Name	PID	Operation	Path	Result	Detail
12:44:56.2165708 PM	Lab 1.exe	4348	 Process Create	C:\Users\deadblasoul\OneDrive\Dev\S...	SUCCESS	PID: 13460, Comm...
12:44:56.2356095 PM	Lab 1.exe	4348	 Process Create	C:\Users\deadblasoul\OneDrive\Dev\S...	SUCCESS	PID: 2692, Comma...
12:44:56.2433342 PM	Lab 1.exe	13460	 Process Create	C:\Users\deadblasoul\OneDrive\Dev\S...	SUCCESS	PID: 7868, Comma...
12:44:56.2504064 PM	Lab 1.exe	13460	 Process Create	C:\Users\deadblasoul\OneDrive\Dev\S...	SUCCESS	PID: 8336, Comma...

## Исходный код

### main.c

```
#define _CRT_SECURE_NO_WARNINGS

#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <inttypes.h>

#include "childprocess.h"

#define PRECALCULATED 2
#define PRECALCULATED_RANGE_ERROR UINT64_MAX
#define MAX_FIB_N 16

inline uint64_t precalculated(const size_t n) {
    switch (n) {
        case 1:
            return 0;
        case 2:
            return 1;
    }

    return PRECALCULATED_RANGE_ERROR;
}

int fib(FILE* const inp, FILE* const out, const char* const filename) {
    // Read index of number from fibonacci sequence
    size_t n;
    if (fscanf(inp, "%zu", &n) != 1) {
        return 1;
    }

    // Check that n is not too high
    if (n > MAX_FIB_N) {
        return EXIT_FAILURE;
    }

    // If n too low return precalculated
```

```

if (n <= PRECALCULATED) {
    uint64_t result = precalculated(n);
    if (result == PRECALCULATED_RANGE_ERROR) {
        return EXIT_FAILURE;
    }

    fprintf(out, "%" PRIu64 "\n", result);
    return EXIT_SUCCESS;
}

// Initiate child processes
child_process process_first;
child_process process_second;
create_child_process(filename, &process_first);
create_child_process(filename, &process_second);

// Write data to the first process
fprintf(process_first.inp, "%zu\n", n - 1);
fflush(process_first.inp);
// Write data to the second process
fprintf(process_second.inp, "%zu\n", n - 2);
fflush(process_second.inp);

// Read output
uint64_t first_result;
uint64_t second_result;
int first_err = fscanf(process_first.out, "%" PRIu64, &first_result);
int second_err = fscanf(process_second.out, "%" PRIu64, &second_result);

// Check that we read successful
if (first_err != 1 && second_err != 1) {
    return EXIT_FAILURE;
}

// Return result
uint64_t result = first_result + second_result;
fprintf(out, "%" PRIu64 "\n", result);

return EXIT_SUCCESS;
}

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

```

```
    const char* const filename = argv[0];  
    int err = fib(stdin, stdout, filename);  
  
    return err;  
}
```

# childprocess.h

```
#pragma once

#include <stdio.h>

typedef struct system_handle system_handle;

typedef struct child_process {
    FILE* inp;    // stdin
    FILE* out;    // stdout

    system_handle* process_handle;
#ifdef _WIN32
    system_handle* thread_handle;
#endif // _WIN32
} child_process;

int create_child_process(const char* const program, child_process* const cp);
```

# childprocess.c

```
#include "childprocess.h"

#include <Windows.h>
#include <io.h>
#include <fcntl.h>

#ifdef _WIN32
/*
    Here is a WINAPI stuff
*/

typedef struct pipe_pair {
    HANDLE hPipeStream_Rd;
    HANDLE hPipeStream_Wr;
} pipe_pair;

int open_rw_pipes(pipe_pair* const in, pipe_pair* const out);
int create_win32_piped_process(const char* const cmd_line, child_process* const cp, pipe_pair* const in_out);

int create_child_process(const char* const program, child_process* const cp) {
    int err;

    pipe_pair in;
    pipe_pair out;
    if (err = open_rw_pipes(&in, &out), err != 0) {
        return err;
    }
    if (err = create_win32_piped_process(program, cp, &in, &out), err != 0) {
        return err;
    }

    return 0;
}

int open_rw_pipes(pipe_pair* const in, pipe_pair* const out) {
    HANDLE hChildStd_IN_Rd = NULL;
    HANDLE hChildStd_IN_Wr = NULL;
    HANDLE hChildStd_OUT_Rd = NULL;
    HANDLE hChildStd_OUT_Wr = NULL;
    SECURITY_ATTRIBUTES saAttr;
```



```

saAttr.nLength = sizeof(saAttr);
saAttr.bInheritHandle = TRUE;
saAttr.lpSecurityDescriptor = NULL;

// Create a pipe for the child process STDOUT.
if (!CreatePipe(&hChildStd_OUT_Rd, &hChildStd_OUT_Wr, &saAttr, 0)) {
    return 1;
}

// Ensure the read handle to the pipe for STDOUT is not inherited.
if (!SetHandleInformation(hChildStd_OUT_Rd, HANDLE_FLAG_INHERIT, 0)) {
    return 1;
}

// Create a pipe for the child process STDIN.
if (!CreatePipe(&hChildStd_IN_Rd, &hChildStd_IN_Wr, &saAttr, 0)) {
    return 1;
}

// Ensure the write handle to the pipe for STDIN is not inherited.
if (!SetHandleInformation(hChildStd_IN_Wr, HANDLE_FLAG_INHERIT, 0)) {
    return 1;
}

*in = (pipe_pair){ hChildStd_IN_Rd, hChildStd_IN_Wr };
*out = (pipe_pair){ hChildStd_OUT_Rd, hChildStd_OUT_Wr };

return 0;
}

int create_win32_piped_process(const char* const cmd_line, child_process* const cp, pipe
PROCESS_INFORMATION piProcInfo;
STARTUPINFO siStartInfo;
BOOL bSuccess = FALSE;

// Set up members of the PROCESS_INFORMATION structure.
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 = out->hPipeStream_Wr;
siStartInfo.hStdOutput = out->hPipeStream_Wr;
siStartInfo.hStdInput = in->hPipeStream_Rd;
siStartInfo.dwFlags |= STARTF_USESTDHANDLES;

// Create the child process.
bSuccess = CreateProcess(NULL,
    cmd_line,          // command line
    NULL,              // process security attributes
    NULL,              // primary thread security attributes
    TRUE,              // handles are inherited
    0,                 // creation flags
    NULL,              // use environment of parent
    NULL,              // use current directory
    &siStartInfo,      // STARTUPINFO pointer
    &piProcInfo);      // receives PROCESS_INFORMATION

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

cp->inp = _fdopen(_open_osfhandle(in->hPipeStream_Wr, _O_TEXT), "w");
cp->out = _fdopen(_open_osfhandle(out->hPipeStream_Rd, _O_TEXT), "r");
cp->process_handle = (system_handle*)piProcInfo.hProcess;
cp->thread_handle = (system_handle*)piProcInfo.hThread;
return 0;
}

#endif // _WIN32

```