**Министерство цифрового развития, связи и массовых**

**коммуникаций Российской Федерации**

**Ордена Трудового Красного Знамени**

**федеральное государственное бюджетное**

**учреждение высшего образования**

**«Московский Технический Университет Связи и Информатики»**

Кафедра «Системное программирование»

**ЛАБОРАТОРНАЯ РАБОТА №5**

по дисциплине «Операционные системы»

по теме «Реализация обмена данными между процессами»

Выполнил: студент группы БСТ2001

Савкин Д. И.

Проверил: асс. каф. СП

Алексанян Д. А.

Москва 2022

# Цели и задачи

Целями работы являются изучение набора средств коммуникации процессов в Windows NT, а также получение практических навыков использования Win32 API для программирования механизмов межпроцессорного взаимодействия.

Задачей является разработка программного продукта, который реализует перемножение двух матриц с помощью нескольких процессов. Программный продукт должен использовать следующие механизмы межпроцессорного взаимодействия: отображение файлов, почтовые ящики и каналы.

# Ход выполнения работы

Исходный код конечного продукта представлен ниже.

## Клиентское приложение

#include <math.h>

#include <stdio.h>

#include <stdint.h>

#include <assert.h>

#include <stdarg.h>

#include <stdbool.h>

#include <windows.h>

#pragma warning(disable: 4305 4018)

#define fatal\_error(fmt, ...) do { error(fmt, \_\_VA\_ARGS\_\_); exit(1); } while (0)

#define For(x) for (size\_t i = 0; i < (x); ++i)

#define BUFFER\_SIZE 4096

static void error(const char\* const fmt, ...) {

va\_list args;

va\_start(args, fmt);

printf("ERROR: ");

vprintf(fmt, args);

printf("\n");

va\_end(args);

}

static bool check\_matrix\_size(uint32\_t n) {

uint32\_t x = (uint32\_t)sqrt(n);

return n ? ((x \* x) == n) : false;

}

static void\* xmalloc(size\_t size) {

void\* ptr = malloc(size);

if (!ptr) {

fatal\_error("unable to allocate memory");

}

return ptr;

}

static char\* strf(const char\* const fmt, ...) {

va\_list args;

va\_start(args, fmt);

size\_t n = 1 + vsnprintf(NULL, 0, fmt, args);

va\_end(args);

char\* buffer = xmalloc(n);

va\_start(args, fmt);

vsnprintf(buffer, n, fmt, args);

va\_end(args);

return buffer;

}

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

argc--;

argv++;

if (argc != 8) {

fatal\_error("Usage: la05.exe <v1> <v2> <v3> <v4> <v5> <v6> <v7> <v8>");

}

DWORD written;

const char\* mailslot\_name = "\\\\.\\mailslot\\sample\_mailslot";

HANDLE mailslot = CreateFileA(mailslot\_name, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (mailslot == INVALID\_HANDLE\_VALUE) {

fatal\_error("unable to open the mailslot (code %lu)", GetLastError());

}

char\* short\_pipe\_name = strf("sample\_%lu", GetCurrentProcessId());

char\* pipe\_name = strf("\\\\.\\pipe\\%s", short\_pipe\_name);

if (!WriteFile(mailslot, short\_pipe\_name, strlen(pipe\_name) + 1, &written, NULL)) {

fatal\_error("unable to write to the mailslot (code %lu)", GetLastError());

}

HANDLE pipe;

for (;;) {

// NOTE(wkns37): Well, it is not the best solution

Sleep(200);

pipe = CreateFileA(pipe\_name, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

if (pipe != INVALID\_HANDLE\_VALUE) {

break;

} else if (GetLastError() != ERROR\_PIPE\_BUSY) {

fatal\_error("unable to open the pipe (code %lu)", GetLastError());

}

WaitNamedPipeA(pipe\_name, NMPWAIT\_WAIT\_FOREVER);

}

free(pipe\_name);

free(short\_pipe\_name);

DWORD mode = PIPE\_READMODE\_MESSAGE;

if (!SetNamedPipeHandleState(pipe, &mode, NULL, NULL)) {

fatal\_error("unable to set the pipe state (code %lu)", GetLastError());

}

uint32\_t values[8] = { 0 };

for (uint32\_t\* ptr = values; \*argv; ptr++, argv++) {

uint32\_t value = strtoul(\*argv, NULL, 10);

if (value != ULONG\_MAX) {

\*ptr = value;

} else {

error("incorrect matrix value %s, using 0 instead", \*argv);

}

}

char filename[] = "MySampleFileMappingObject";

HANDLE file = CreateFileMappingA(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, sizeof(values), filename);

if (file == NULL) {

fatal\_error("unable to create a file mapping object (code %lu)", GetLastError());

}

uint32\_t\* view = MapViewOfFile(file, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(values));

if (view == NULL) {

fatal\_error("unable to map view of the file (code %lu)", GetLastError());

}

memcpy(view, values, sizeof(values));

if (!WriteFile(pipe, filename, strlen(filename) + 1, &written, NULL)) {

fatal\_error("unable to write to the pipe (code %lu)", GetLastError());

}

DWORD read;

char msg[BUFFER\_SIZE];

if (!ReadFile(pipe, msg, sizeof(msg), &read, NULL) || !read) {

if (GetLastError() == ERROR\_BROKEN\_PIPE) {

fatal\_error("the pipe was disconnected");

} else {

fatal\_error("unable to read from the pipe (code %lu)", GetLastError());

}

}

printf("%s\n", msg);

printf("(%-3d %3d)(%-3d %3d) = (%-3d %3d)\n", values[0], values[1], values[4], values[5], view[0], view[1]);

printf("(%-3d %3d)(%-3d %3d) = (%-3d %3d)\n", values[2], values[3], values[6], values[7], view[2], view[3]);

CloseHandle(mailslot);

CloseHandle(pipe);

CloseHandle(file);

return 0;

}

## Серверное приложение

#include <math.h>

#include <stdio.h>

#include <stdint.h>

#include <assert.h>

#include <stdarg.h>

#include <stdbool.h>

#include <windows.h>

#define fatal\_error(fmt, ...) do { error(fmt, \_\_VA\_ARGS\_\_); exit(1); } while (0)

#define BUFFER\_SIZE 4096

static void error(const char\* const fmt, ...) {

va\_list args;

va\_start(args, fmt);

printf("ERROR: ");

vprintf(fmt, args);

printf("\n");

va\_end(args);

}

static bool check\_matrix\_size(uint32\_t n) {

uint32\_t x = (uint32\_t)sqrt(n);

return n ? ((x \* x) == n) : false;

}

static void\* xmalloc(size\_t size) {

void\* ptr = malloc(size);

if (!ptr) {

fatal\_error("unable to allocate memory");

}

return ptr;

}

static char\* strf(const char\* const fmt, ...) {

va\_list args;

va\_start(args, fmt);

size\_t n = 1 + vsnprintf(NULL, 0, fmt, args);

va\_end(args);

char\* buffer = xmalloc(n);

va\_start(args, fmt);

vsnprintf(buffer, n, fmt, args);

va\_end(args);

return buffer;

}

static DWORD WINAPI process\_pipe(LPVOID param) {

HANDLE pipe = (HANDLE)param;

DWORD read;

char filename[BUFFER\_SIZE];

BOOL success = ReadFile(pipe, filename, sizeof(filename), &read, NULL);

if (!success || !read) {

if (GetLastError() == ERROR\_BROKEN\_PIPE) {

error("the pipe was disconnected");

goto end;

} else {

error("unable to read from the pipe (code %lu)", GetLastError());

goto end;

}

}

HANDLE file = OpenFileMappingA(FILE\_MAP\_ALL\_ACCESS, FALSE, filename);

if (file == NULL) {

error("unable to open a file mapping object (code %lu)", GetLastError());

goto end;

}

uint32\_t\* view = MapViewOfFile(file, FILE\_MAP\_ALL\_ACCESS, 0, 0, 8 \* sizeof(uint32\_t));

if (view == NULL) {

error("unable to map view of the file (code %lu)", GetLastError());

goto end;

}

uint32\_t result[8] = { 0 };

result[0] = view[0] \* view[4] + view[1] \* view[6];

result[1] = view[0] \* view[5] + view[1] \* view[7];

result[2] = view[2] \* view[4] + view[3] \* view[6];

result[3] = view[2] \* view[5] + view[3] \* view[7];

memcpy(view, result, sizeof(result));

UnmapViewOfFile(view);

CloseHandle(file);

printf("Job done for the %p file mapping object\n", file);

DWORD written;

char msg[] = "ok";

if (!WriteFile(pipe, msg, sizeof(result) + 1, &written, NULL)) {

error("unable to write to the pipe (code %lu)", GetLastError());

}

end:

return 1;

}

int main() {

const char\* mailslot\_name = "\\\\.\\mailslot\\sample\_mailslot";

HANDLE mailslot = CreateMailslotA(mailslot\_name, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (mailslot == INVALID\_HANDLE\_VALUE) {

fatal\_error("unable to create mailslot (code %lu)", GetLastError());

}

printf("Waiting for a message...\n");

for (;;) {

DWORD msg\_size = 0, msg\_count = 0, msg\_read = 0;

if (!GetMailslotInfo(mailslot, NULL, &msg\_size, &msg\_count, NULL)) {

error("unable to read from the mailslot (code %lu)", GetLastError());

goto end;

}

if (msg\_count == MAILSLOT\_NO\_MESSAGE) {

goto end;

}

while (msg\_count) {

char\* short\_pipe\_name = xmalloc(msg\_size);

if (!ReadFile(mailslot, short\_pipe\_name, msg\_size, &msg\_read, NULL)) {

error("unable to read from the mailslot (code %lu)", GetLastError());

goto end;

}

char\* pipe\_name = strf("\\\\.\\pipe\\%s", short\_pipe\_name);

HANDLE pipe = CreateNamedPipeA(

pipe\_name,

PIPE\_ACCESS\_DUPLEX,

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

1, BUFFER\_SIZE, BUFFER\_SIZE, 5000, NULL

);

free(pipe\_name);

free(short\_pipe\_name);

if (pipe == INVALID\_HANDLE\_VALUE) {

error("unable to create a pipe '%s' (code %lu)", short\_pipe\_name, GetLastError());

continue;

}

BOOL connected = ConnectNamedPipe(pipe, NULL) ? TRUE : (GetLastError() == ERROR\_PIPE\_CONNECTED);

if (connected) {

DWORD tid;

HANDLE thread = CreateThread(NULL, 0, process\_pipe, (LPVOID)pipe, 0, &tid);

if (thread == NULL) {

error("unable to create a thread (code %lu)", GetLastError());

continue;

}

CloseHandle(thread);

} else {

CloseHandle(pipe);

}

if (!GetMailslotInfo(mailslot, NULL, &msg\_size, &msg\_count, NULL)) {

error("unable to read the mailslot (code %lu)", GetLastError());

goto end;

}

}

end : 0;

}

}

# Заключение

Результат работы программного продукта представлен на рисунках ниже.

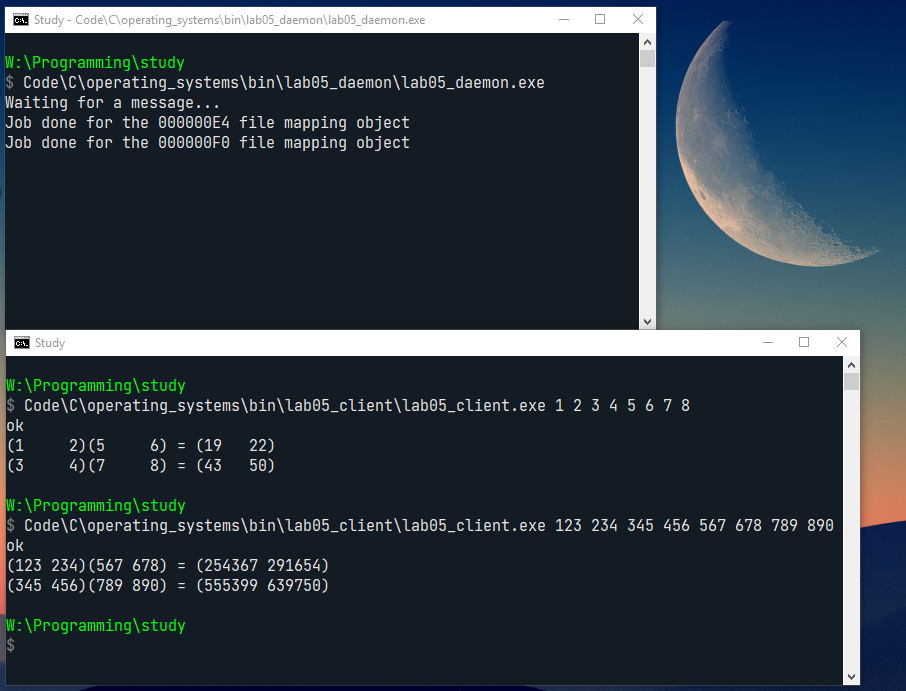


Рисунок 1 – Результат работы программы