**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа № 2**

**По курсу:  
«Операционные системы»**

Студент: Патрикеева Лидия

Группа: 80-207Б-18

Преподаватель: Миронов Е.С.

Дата:

Оценка:

Москва, 2019

**Цель работы**

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Задание**

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 решения 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.

Вариант задания № 5:

На вход программе подается команда интерпретатора команд. Программа должна произвести вывод команды в верхнем регистре.

**Системные вызовы**

Для ОС Windows :

* BOOL WINAPI CreateProcess(…) - создание нового процесса
* WaitForSingleObject(…) - ожидание завершения процесса
* ExitProcess(…) - завершение выполнения процесса
* CreateFile/SetNamedPipeHandleState - создание именованного канала и установления режима его использования
* Int \_dup2(int fd1, int fd2) - переназначение файлового дескриптора
* OpenFile(…) - открытие нового файла
* CreatePipe(…) - создание безымянного канала
* CreateFile(…) - создание нового файла
* CloseHandle(…) - закрытие объекта ОС по "заголовку". Подходит для закрытия файлов.

**Исходный код**

main.c

#include <Windows.h>

#include <stdio.h>

int main() {

DWORD nbReaded;

DWORD nbWrited;

DWORD nbAvail;

STARTUPINFO si;

SECURITY\_ATTRIBUTES sa;

PROCESS\_INFORMATION pi;

HANDLE ChildStd\_IN\_Rd, ChildStd\_OUT\_Wr, ChildStd\_OUT\_Rd, ChildStd\_IN\_Wr;

sa.nLength = sizeof(SECURITY\_ATTRIBUTES);

sa.bInheritHandle = TRUE;

sa.lpSecurityDescriptor = NULL;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

if (CreatePipe(&ChildStd\_IN\_Rd, &ChildStd\_IN\_Wr, &sa, 0))

{

if (CreatePipe(&ChildStd\_OUT\_Rd, &ChildStd\_OUT\_Wr, &sa, 0))

{

GetStartupInfo(&si);

si.dwFlags = STARTF\_USESTDHANDLES;

si.hStdOutput = ChildStd\_OUT\_Wr;

si.hStdError = ChildStd\_OUT\_Wr;

si.hStdInput = ChildStd\_IN\_Rd;

if (CreateProcess("C:\\Windows\\System32\\cmd.exe", NULL, NULL, NULL, TRUE, NORMAL\_PRIORITY\_CLASS, NULL, "D:\\", &si, &pi)) {

CloseHandle(ChildStd\_IN\_Rd);

CloseHandle(ChildStd\_OUT\_Wr);

nbAvail = 1;

char buf1[2];

while (nbAvail) {

if (!(ReadFile(ChildStd\_OUT\_Rd, buf1, 1, &nbReaded, NULL))) {

printf("ReadFile: Error %ld\n", GetLastError());

}

if(!(PeekNamedPipe(ChildStd\_OUT\_Rd, NULL, 0, NULL, &nbAvail, NULL))) {

printf("PeekNamedPipe: Error %ld\n", GetLastError());

};

}

printf("D:\\>");

while (1) {

char buff[256];

unsigned char buf[2];

gets(buff);

if (!strcmp(buff, "exit")) {

break;

}

char siz[2];

char \*s = strcat(buff, "\n");

siz[0] = strlen(buff);

if (!(WriteFile(ChildStd\_IN\_Wr, buff, strlen(buff), & nbWrited, NULL))) {

printf("WriteFile: Error %ld\n", GetLastError());

}

nbAvail = 1;

while(nbAvail){

ReadFile(ChildStd\_OUT\_Rd, buf, 1, &nbReaded, NULL);

PeekNamedPipe(ChildStd\_OUT\_Rd, NULL, 0, NULL, &nbAvail, NULL);

}

nbAvail = 1;

while (nbAvail) {

if (ReadFile(ChildStd\_OUT\_Rd, buf, 2, &nbReaded, NULL)) {

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

int code = (int) buf[i];

if ( (code >= 97 && code <= 122) || (code >= 224 && code <= 255) ) {

buf[i] = (unsigned char)(code - 32);

}

printf("%c", buf[i]);

}

} else {

if(GetLastError() == 109) break;

printf("ReadFile: Error %ld\n", GetLastError());

}

if(!(PeekNamedPipe(ChildStd\_OUT\_Rd, NULL, 0, NULL, &nbAvail, NULL))){

printf("PeekNamedPipe: Error %ld\n", GetLastError());

break;

}

if(buf[nbReaded-1]=='\n') nbAvail = 1;

}

}

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

CloseHandle(ChildStd\_OUT\_Rd);

CloseHandle(ChildStd\_IN\_Wr);

} else {

printf("CreateProcess: Error %ld\n", GetLastError());

}

} else {

printf("CreatePipe\_OUT: Error %ld\n", GetLastError());

}

} else {

printf("CreatePipe\_IN: Error %ld\n", GetLastError());

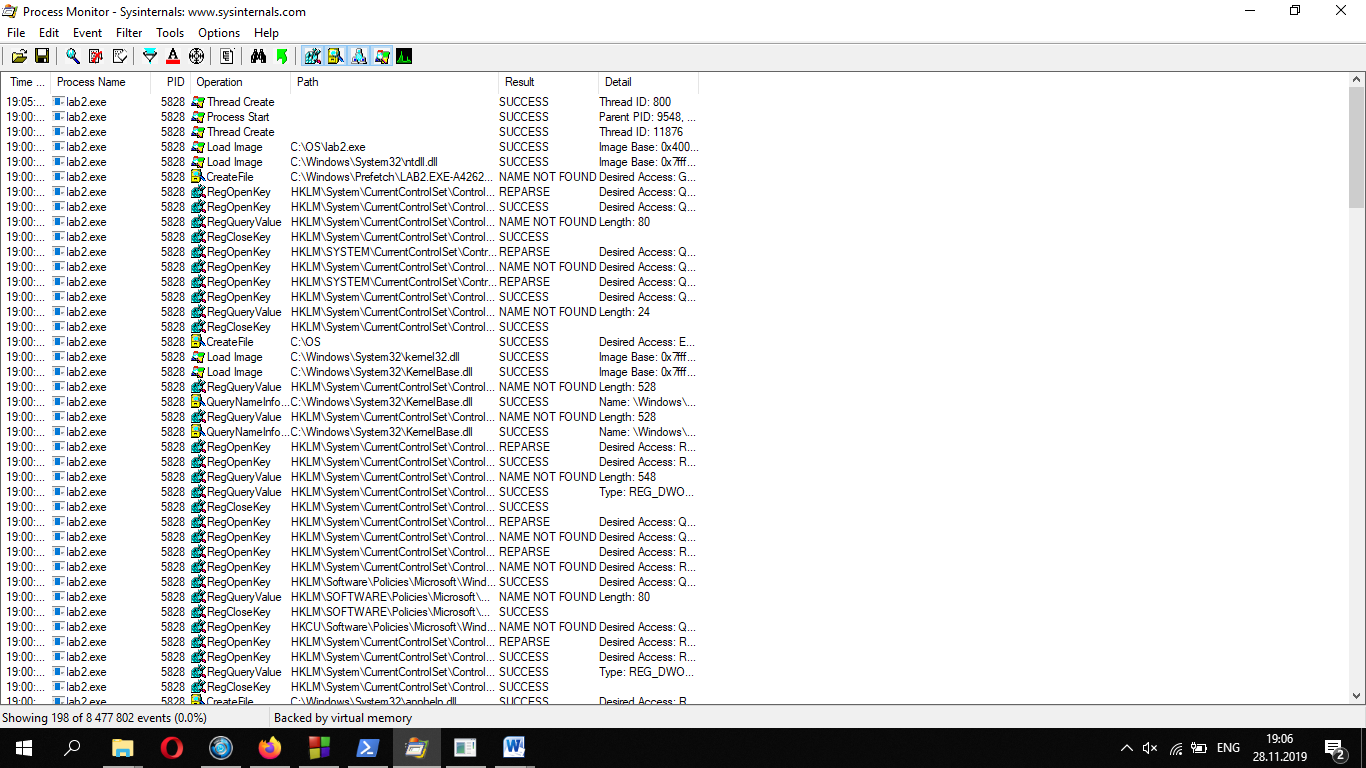
}

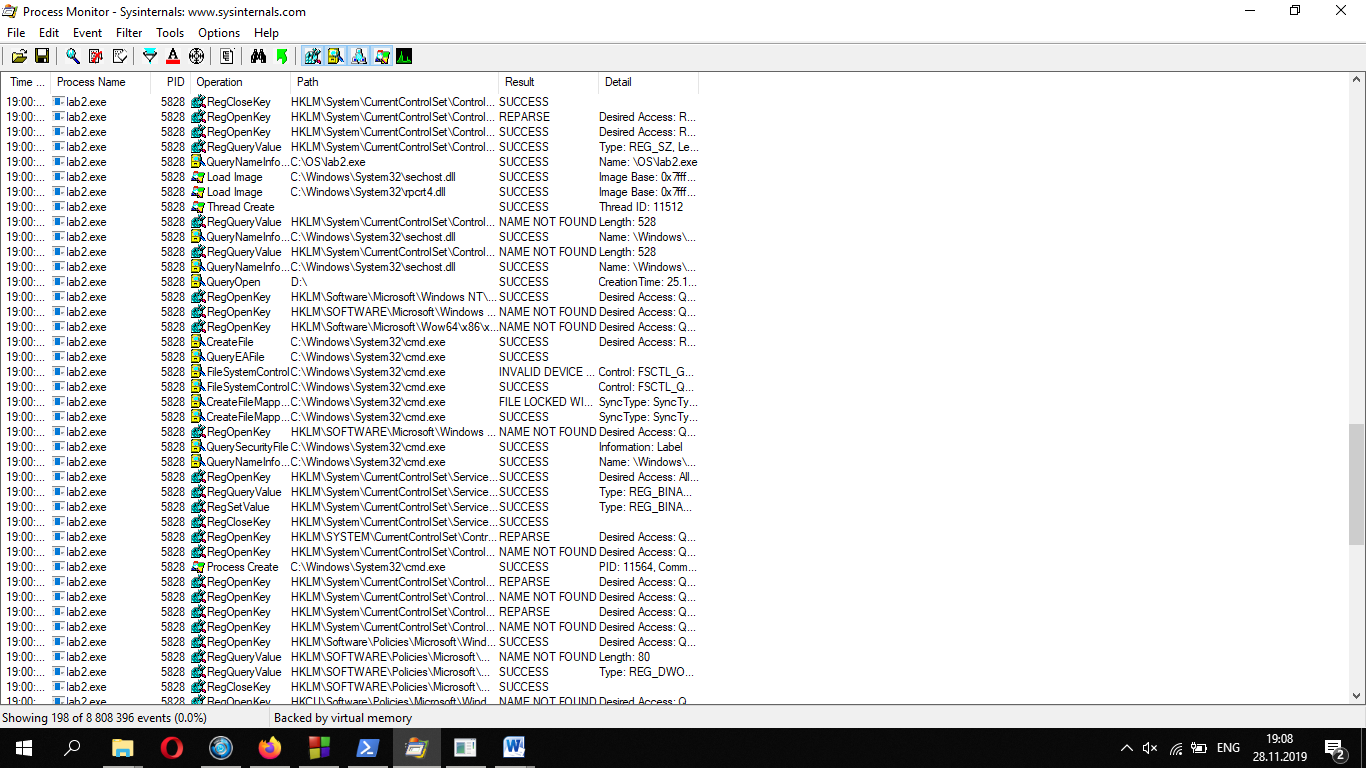
return 0;

}

**Procmon**

Ниже представлены скриншоты из программы Process Monitor, которая позволяет наблюдать в реальном масштабе времени за действиями различных процессов в среде операционной системы Windows.





**Вывод**

В процессе выполнения данной лабораторной работы я научилась осуществлять взаимодействие между двумя параллельно работающими процессами в операционной среде Windows. Для этого я научилась пользоваться таким средством передачи данных, как каналы типа Pipe, которые бывают анонимные и именованные. Они так же являются одними из наиболее простых способов передачи информации между процессами. В своей лабораторной работе я использовала анонимные каналы, потому что функция CreatePipe для их создания намного проще и имеет всего четыре аргумента, в отличие от функции CreateNamedPipe (хотя возможности именованных каналов намного богаче).

Из данной лабораторной работы могу сделать вывод, что использование дополнительных процессов в проектах бывает полезно. Каждый процесс выполняет свою задачу, а с помощью пайпов они могут легко получать нужную информацию друг от друга.