БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

***Факультет информационных технологий и робототехники***

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

ОТЧЕТ

о лабораторной работе №7

По дисциплине:

«Операционные системы и системное программирование»

Тема:

«ПРОЦЕССЫ»

Выполнил: ст. гр.

Принял: Разорёнов Н.А.

Минск 2020

**1. Цель работы:** ознакомление с основами создания, управления и атрибутами процессов в ОС WINDOWS.

**2. Изучаемые вопросы:**

1. Создание процесса, атрибуты объектов: процесс, главный поток.

2. Структуры STARTUPINFO, PROCESS\_INFORMATION.

3. Наследование дочерними процессами ресурсов родительского процесса.

4. Переменные окружения процесса: структура, значения.

5. Время выполнения процесса.

6. Список процессов.

7. Окончание процесса.

**3. Постановка задачи**

1. Разработать приложение, состоящее из трех процессов (головного и двух дочерних). В головном процессе создаются/открываются два файла: текстовый (hFile) и исполняемый файл (hExe). Для исполняемого файла создается объект отображения файла (hMapFile). Дескрипторы объектов hFile и hMapFile передаются в дочерние процессы на этапе их создания соответственно через командную строку и переменные окружения – среду процесса.

2. Первый процесс дописывает строки в текстовый файл.

3. Второй процесс обрабатывает исходный файл (вывод таблицы импорта).

4. После окончания дочерних процессов, главный процесс отображает:

– содержимое модифицированных файлов и закрывает их;

– системную информацию о процессе (GetProcessInformation).

**4. Ход выполнения работы**

4.1. Создание процесса.

В программе для создания процесса используют функцию CreateProcess:

CreateProcess(NULL, //имя исполняемого модуля

cmdproc2, //командная строка

NULL, //защита процесса

NULL, //защита потока

TRUE, //признак наследования дескриптора

0, //флаги создания процесса

NULL, //блок новой среды окружения

NULL, //текущий каталог

&si, //вид главного окна

&pi); //информация о процессе

При неудачной работе функции её результатом будет значение FALSE.

4.2. Структуры STARTUPINFO, PROCESS\_INFORMATION.

В создании и работе процесса участвуют две структуры: STARTUPINFO и PROCESS\_INFORMATION. Структура STARTUPINFO отвечает за параметры окна при создании процесса. Важно: все поля структуры STARTUPINFO (кроме поля, указывающего её непосредственный размер) должны быть обнулены:

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(STARTUPINFO);

PROCESS\_INFORMATION – это структура, которую возвращает функция создания процесса. Данная функция содержит 4 поля: дескрипторы созданного процесса и его главного потока и их идентификаторы. Пример использования дескриптора дочернего процесса:

CreateProcess(NULL, cmdproc2, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi);

WaitForSingleObject(pi.hProcess, INFINITE);

4.3. Наследование дочерними процессами ресурсов родительского процесса.

Для наличия возможности наследования ресурсов родительского процесса дочерним процессом необходимо в родительском процессе объявить структуру SECURITY\_ATTRIBUTES и установить права по совместному доступу к созданному файлу:

sa.nLength = sizeof(SECURITY\_ATTRIBUTES);

sa.lpSecurityDescriptor = NULL;

sa.bInheritHandle = TRUE;

Затем при в функции создания файла и его адресного пространства необходимо передать указатель на структуру SECURITY\_ATTRIBUTES:

hFile = CreateFile(L"file.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, &sa, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

hMapFile = CreateFileMapping(hBin, &sa, PAGE\_READONLY, 0, 0, NULL);

Далее, для того, чтобы дочерний процесс смог использовать данные дескрипторы, необходимо передать через параметры командной строки. Кроме этого, пятый параметр функции создания процесса должен быть установлен в TRUE, что дает возможность созданному процессу наследовать все доступные дескрипторы объектов ядра, принадлежащие родительскому процессу. Например:

j = swprintf\_s(cmdproc1, 500, L"%s", L"\"E:\\Для учёбы\\Инфоматика\\2 курс\\Системное программирование\\лабы\\Lab 7-2\\Process 2\\Debug\\Process 2.exe\" ");

swprintf\_s(cmdproc1 + j, 500 - j, L"%d", (int)hFile);

CreateProcess(NULL, cmdproc1, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi);

В дочернем же процессе необходимо взять командную строку и извлечь оттуда дескриптор объекта:

pArgv = CommandLineToArgvW(GetCommandLine(), &Num);

hFile = (HANDLE)\_wtoi(pArgv[1]);

Результат работы с текстовым файлом (запись в файл), использую командную строку, приведен на Рисунке 1:

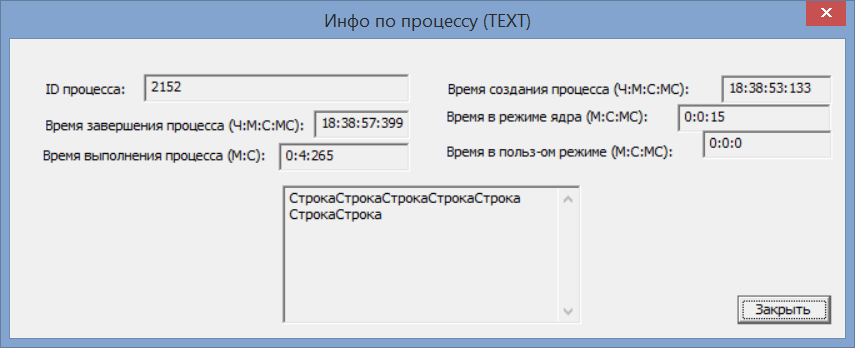


Рисунок 1. Взаимодействие процессов через командную строку (работа с текстом).

4.4. Переменные окружения процесса: структура, значения.

Каждый процесс имеет область памяти, которая выделенна в адресном пространстве процесса. Это – блок переменных окружения. Каждый блок содержит группу строк следующего вида:

Имя переменной 1 = значение 1\0

Имя переменной 2 = значение 1\0

Имя переменной 3 = значение 1\0

. . .

Имя переменной n = значение n\0

*\0*

Для получения переменных окружения используется функция GetEnvironmentStrings(). Для вывода используем цикл while с условием выхода “если строка переменной окружения равна ‘\0’ ”:

lpvEnv = GetEnvironmentStrings();

lpszVariable = (LPSTR)lpvEnv;

while (lpszVariable[0] != '\0')

{

SendMessage(listbox, LB\_ADDSTRING, 0, (LPARAM)lpszVariable);

lpszVariable = lpszVariable + strlen(lpszVariable) + 1;

}

FreeEnvironmentStrings((LPTSTR)lpvEnv);

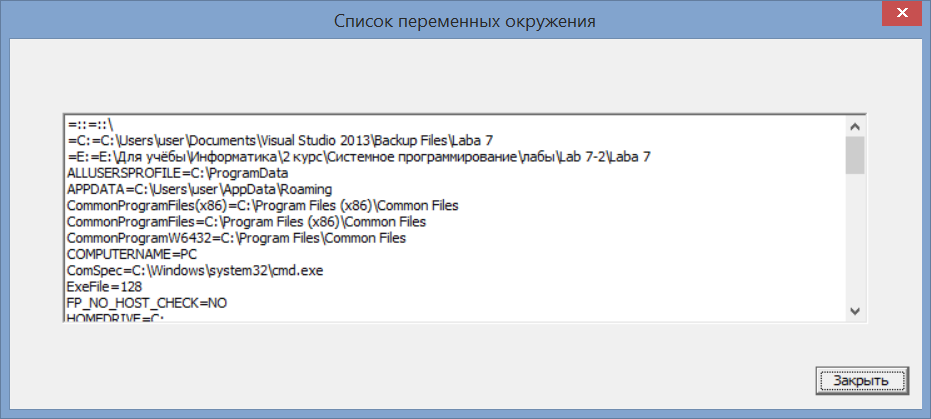


Рисунок 2. Список переменных окружения.

Переменные окружения служат для передачи информации между процессами. Для этого в родительском процессе происходит создание и инициализация переменной среды:

\_itoa((int)hMapFile, envVal, 10);

SetEnvironmentVariableA("ExeFile", envVal);

Далее в дочернем процессе отыскиваем переменную и считываем значение:

if (!GetEnvironmentVariableA("ExeFile", envVal, 10))

{

MessageBox(hDlg, L"Переменная не найдена!", L"ERROR", MB\_OK | MB\_ICONERROR);

}

hMapFile = (HANDLE)atoi(envVal);

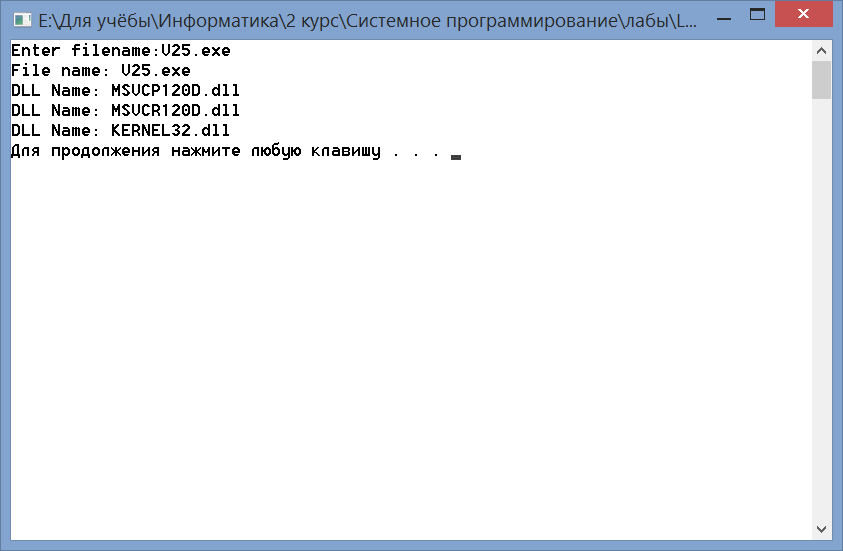


Рисунок 3. Взаимодействие процессов через переменную окружения (таблица импорта).

4.5. Время выполнения процесса.

Существует несколько функций для отображения информации о работе процесса (время создания процесса, время завершения, время работы в режиме ядра, время работы в пользовательском режиме). В данной задаче использовалась функция GetProcessTimes(pi.hProcess, // дескриптор процесса

&timeCreation, // время создания процесса

&timeExit, // время выхода из работы процесса

&timeKernel, // время работы процесса в режиме ядра

&timeUsed); // время работы процесса в режиме кользователя

)

Пример использования функции GetTimeProcess:

FILETIME timeCreation, timeExit, Time, timeKernel, timeUser;

SYSTEMTIME timeSys;

GetProcessTimes(pi.hProcess, &timeCreation, &timeExit, &timeKernel, &timeUser);

Time.dwHighDateTime = timeExit.dwHighDateTime - timeCreation.dwHighDateTime;

Time.dwLowDateTime = timeExit.dwLowDateTime - timeCreation.dwLowDateTime;

FileTimeToLocalFileTime(&timeCreation, &timeCreation);

FileTimeToSystemTime(&timeCreation, &timeSys); //время создания

sprintf(date, "%d:%d:%d:%d", timeSys.wHour, timeSys.wMinute, timeSys.wSecond, timeSys.wMilliseconds);

SetDlgItemTextA(hDlg, IDC\_EDIT3, date);

FileTimeToLocalFileTime(&timeExit, &timeExit);

FileTimeToSystemTime(&timeExit, &timeSys); //время завершения

sprintf(date, "%d:%d:%d:%d", timeSys.wHour, timeSys.wMinute, timeSys.wSecond, timeSys.wMilliseconds);

SetDlgItemTextA(hDlg, IDC\_EDIT5, date);

FileTimeToLocalFileTime(&Time, &Time);

FileTimeToSystemTime(&Time, &timeSys);

sprintf(date, "%d:%d:%d", timeSys.wMinute, timeSys.wSecond, timeSys.wMilliseconds);

SetDlgItemTextA(hDlg, IDC\_EDIT8, date);

FileTimeToSystemTime(&timeUser, &timeSys); //время в режиме пользователя

sprintf(date, "%d:%d:%d", timeSys.wMinute, timeSys.wSecond, timeSys.wMilliseconds);

SetDlgItemTextA(hDlg, IDC\_EDIT7, date);

FileTimeToSystemTime(&timeKernel, &timeSys); //время в режиме ядра

sprintf(date, "%d:%d:%d", timeSys.wMinute, timeSys.wSecond, timeSys.wMilliseconds);

SetDlgItemTextA(hDlg, IDC\_EDIT6, date);

SetDlgItemInt(hDlg, IDC\_EDIT1, pi.dwProcessId, FALSE);

SetDlgItemInt(hDlg, IDC\_EDIT2, pi.dwThreadId, FALSE);

Скриншот результата получения параметров работы процесса представлен на скриншоте 1 (работа с текстом)

4.6. Список процессов.

Для получения списка актуальных процессов используются функции Process32First, Process32Next, а так же структура PROCESSENTRY32. Чтобы вывести список процессов необходимо сделать снимок процессов (функция CreateToolhelp32Snapshot), получить первый кадр из этого снимка (функция Process32First), в цикле do-while инициализировать элементы структуры PROCESSENTRY32 следующих процессов (в условии выхода из цикла используем функцию Process32Next). Параметрами для функций Process32Next и Process32First являются: handle функции CreateToolhelp32Snapshot (вызванной с  dwFlags = TH32CS\_SNAPPROCESS), а также указатель на структуру PROCESSENTRY32. Пример использования этих функций приведен ниже.

hs = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

PROCESSENTRY32 P;

P.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(hs, &P))

{

do

{

j = swprintf\_s(name, 500, L"Имя: %s", P.szExeFile);

j += swprintf\_s(name + j, 500 - j, L" ID процесса: %d", P.th32ProcessID);

swprintf\_s(name + j, 500 - j, L" ID родителя: %d",P.th32ParentProcessID);

SendMessage(listbox, LB\_ADDSTRING, 0, (LPARAM)name);

}

while (Process32Next(hs, &P));

}

CloseHandle(hs);

На Рисунке 4 приведен скриншот списка процессов:

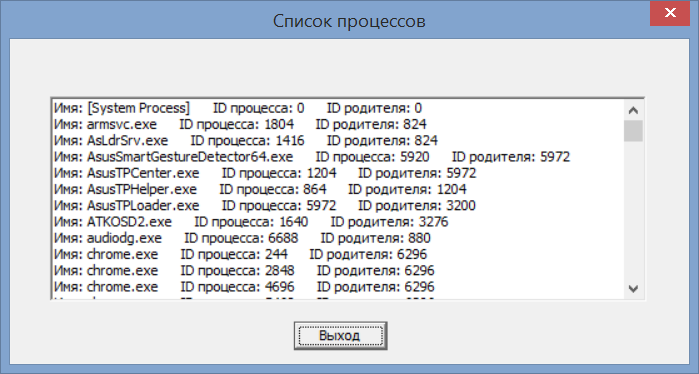


Рисунок 4. Список процессов.

4.7. Окончание процесса.

Процесс может быть завершен несколькими способами:

WaitForSingleObject(pi.hProcess, INFINITE);

ExitProcess (exitCode)

TerminateProcess (pi.hProcess, exitCode)

Параметр exitCode определяет код выхода для процесса и для всех потоков, которые завершают работу в результате вызова функции ExitProcess (или TerminateProcess). Параметр pi.hProcess передает в функцию Handle процесса, который будет остановлен через некоторое время (указывается вторым параметром при вызове функции). Если вместо времени передать значение INFINITE, родительский процесс будет ждать завершения дочернего процесса до тех пор, пока он сам не остановится.

Самой безопасной функцией завершения считается WaitForSingleObject, так как она ждем пока сам дочерний процесс себя остановит. Категорически не рекомендуется использовать TerminateProcess, так как он останавливает процесс безоговорочно, что может сказаться на данных, с какими работал этот процесс.

**Выводы.**

1. Для создания процесса используется функция CreateProcess.
2. Дочерний процесс может наследовать объекты ядра родительского процесса. Для этого заполняется структура SECURITY\_ATTRIBUTES.
3. Структура SECURITY\_ATTRIBUTES передается в функции создания объектов.
4. Чтобы дочерний процесс мог использовать ресурсы родительского процесса, необходимо пятый параметр установить в TRUE.
5. Взаимодействие процессов может осуществляться через переменную окружения.
6. Переменная окружения инициализируется функцией SetEnvironmentVariableA("ExeFile", envVal);

