Минский государственный колледж цифровых технологий

**Отчет**

по лабораторной работе № 9

«**Исследование приемов работы в пользовательском режиме**»

по учебному предмету «Операционные системы»

Выполнил:

Студент группы 81ТП Житкевич М.Д.

Проверила: Варицкая А.В.

Минск 2024

Цель работы: ознакомиться с основами работы процессов в пользовательском режиме.

Задание 1

#include <windows.h>

#include <iostream>

int main() {

setlocale(LC\_ALL, "ru");

// Команда для выполнения (например, запуска консольного приложения)

const char\* command = "cmd.exe /C echo Hello from child process!";

wchar\_t wcommand[256];

MultiByteToWideChar(CP\_ACP, 0, command, -1, wcommand, 256);

// Структура для создания процесса

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

// Создание дочернего процесса

if (!CreateProcessW(NULL, // Не используем имя исполняемого файла

wcommand, // Команда для выполнения

NULL, // Атрибуты процесса

NULL, // Атрибуты потока

FALSE, // Наследование дескрипторов

0, // Используем стандартные флаги

NULL, // Используем стандартную среду

NULL, // Текущий каталог

&si, // Структура STARTUPINFO

&pi) // Структура PROCESS\_INFORMATION

) {

std::cerr << "Ошибка создания процесса: " << GetLastError() << std::endl;

return 1;

}

// Ожидание завершения дочернего процесса

WaitForSingleObject(pi.hProcess, INFINITE);

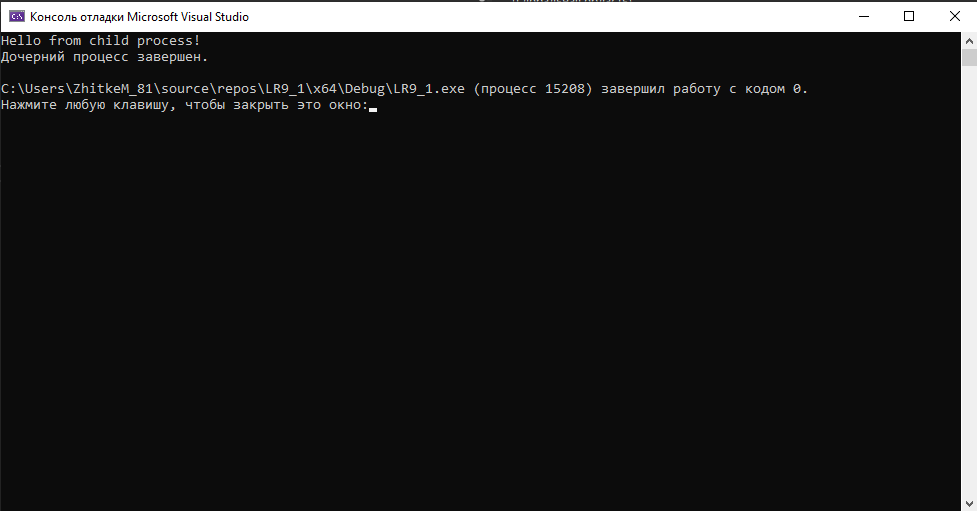
// Закрытие дескрипторов

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

std::cout << "Дочерний процесс завершен." << std::endl;

return 0;

}

Задание 2

#include <iostream>

#include <windows.h>

#include <string>

int main() {

const int num\_processes = 5; // Количество дочерних процессов

HANDLE processes[num\_processes]; // Массив дескрипторов процессов

for (int i = 0; i < num\_processes; i++) {

// Команда для выполнения (здесь просто запускаем cmd с echo)

std::wstring command = L"cmd.exe /C echo Дочерний процесс: PID = " + std::to\_wstring(GetCurrentProcessId()) + L", PPID = " + std::to\_wstring(GetCurrentProcessId());

// Структуры для создания процесса

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

// Создание дочернего процесса

if (!CreateProcessW(

NULL, // Не используем имя исполняемого файла

&command[0], // Команда для выполнения

NULL, // Атрибуты процесса

NULL, // Атрибуты потока

FALSE, // Наследование дескрипторов

0, // Используем стандартные флаги

NULL, // Используем стандартную среду

NULL, // Текущий каталог

&si, // Структура STARTUPINFO

&pi) // Структура PROCESS\_INFORMATION

) {

std::cerr << "Ошибка создания процесса: " << GetLastError() << std::endl;

return 1;

}

processes[i] = pi.hProcess; // Сохраняем дескриптор процесса

CloseHandle(pi.hThread); // Закрываем дескриптор потока, он нам не нужен

}

// Родительский процесс ждет завершения всех дочерних процессов

for (int i = 0; i < num\_processes; i++) {

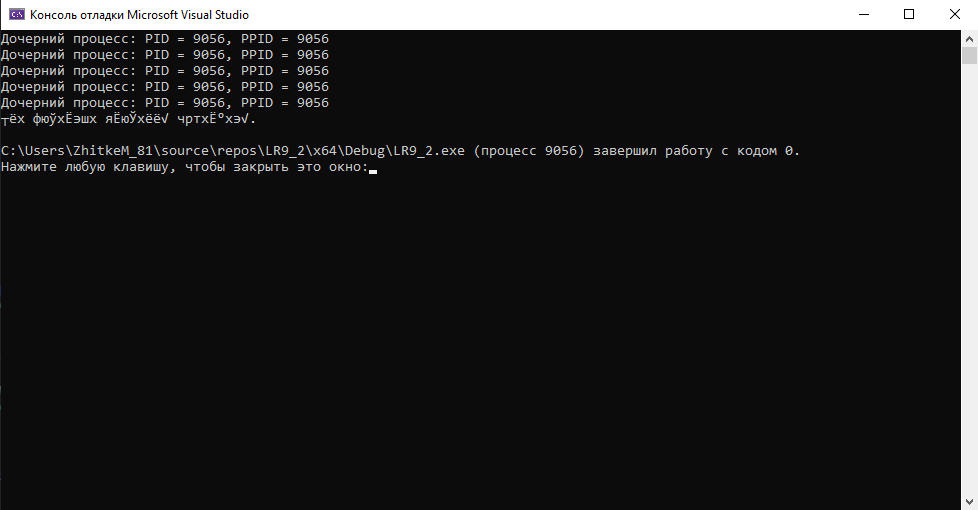
WaitForSingleObject(processes[i], INFINITE);

CloseHandle(processes[i]); // Закрываем дескриптор процесса после завершения

}

std::cout << "Все дочерние процессы завершены." << std::endl;

return 0;

}

Задание 3

#include <iostream>

#include <windows.h>

int main() {

setlocale(LC\_ALL, "ru");

HANDLE hReadPipe, hWritePipe;

char buffer[20];

const char\* msg = "Привет от дочернего процесса!";

// Создаем анонимный канал

if (!CreatePipe(&hReadPipe, &hWritePipe, NULL, 0)) {

std::cerr << "Ошибка создания канала: " << GetLastError() << std::endl;

return 1;

}

// Создаем процесс

PROCESS\_INFORMATION pi;

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

si.hStdOutput = hWritePipe; // Перенаправляем стандартный вывод

si.dwFlags |= STARTF\_USESTDHANDLES;

std::wstring command = L"cmd.exe /C echo Привет от дочернего процесса!";

if (!CreateProcess(

NULL, // Имя исполняемого файла

(LPWSTR)command.c\_str(),// Команда для выполнения

NULL, // Атрибуты процесса

NULL, // Атрибуты потока

TRUE, // Наследование дескрипторов

0, // Стандартные флаги

NULL, // Стандартная среда

NULL, // Текущий каталог

&si, // Структура STARTUPINFO

&pi) // Структура PROCESS\_INFORMATION

) {

std::cerr << "Ошибка создания процесса: " << GetLastError() << std::endl;

return 1;

}

// Закрываем конец записи в родительском процессе

CloseHandle(hWritePipe);

// Читаем из канала в родительском процессе

DWORD bytesRead;

if (ReadFile(hReadPipe, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {

buffer[bytesRead] = '\0'; // Завершаем строку нулевым символом

std::cout << "Родительский процесс прочитал: " << buffer << std::endl;

}

else {

std::cerr << "Ошибка чтения из канала: " << GetLastError() << std::endl;

}

// Закрываем дескрипторы

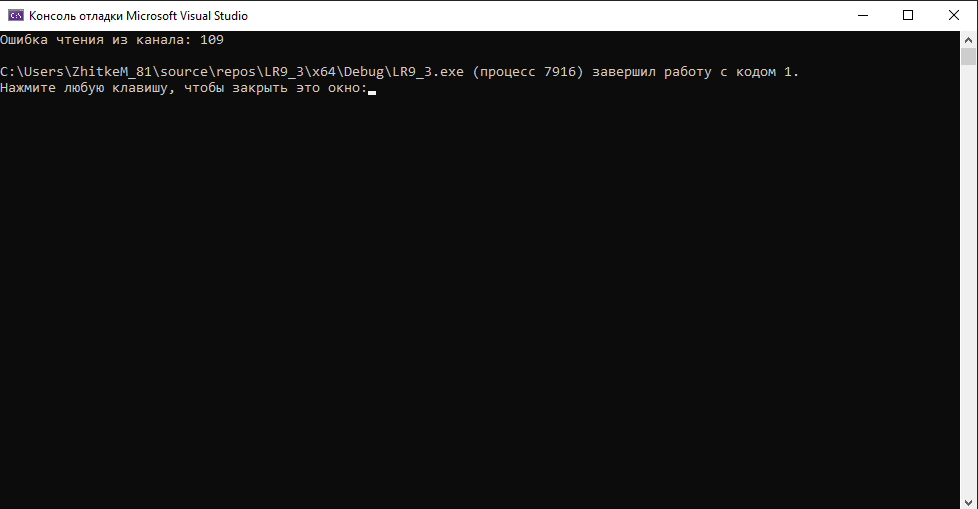
CloseHandle(hReadPipe);

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return 0;

}



Задание 4

#include <iostream>

#include <windows.h>

int main() {

setlocale(LC\_ALL, "ru");

// Структуры для хранения информации о процессе и запуске

PROCESS\_INFORMATION pi;

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

si.dwFlags |= STARTF\_USESHOWWINDOW;

si.wShowWindow = SW\_SHOW;

// Команда для выполнения дочернего процесса

const char\* command = "child\_process.exe"; // Убедитесь, что этот исполняемый файл существует

std::wstring wCommand = std::wstring(command, command + strlen(command));

// Создаем новый процесс

if (!CreateProcess(

NULL, // Имя исполняемого файла

(LPWSTR)wCommand.c\_str(), // Команда для выполнения

NULL, // Атрибуты процесса

NULL, // Атрибуты потока

FALSE, // Наследование дескрипторов

0, // Стандартные флаги

NULL, // Стандартная среда

NULL, // Текущий каталог

&si, // Структура STARTUPINFO

&pi) // Структура PROCESS\_INFORMATION

) {

std::cerr << "Ошибка создания процесса: " << GetLastError() << std::endl;

return 1;

}

// Ожидаем завершения дочернего процесса

WaitForSingleObject(pi.hProcess, INFINITE);

std::cout << "Дочерний процесс завершен." << std::endl;

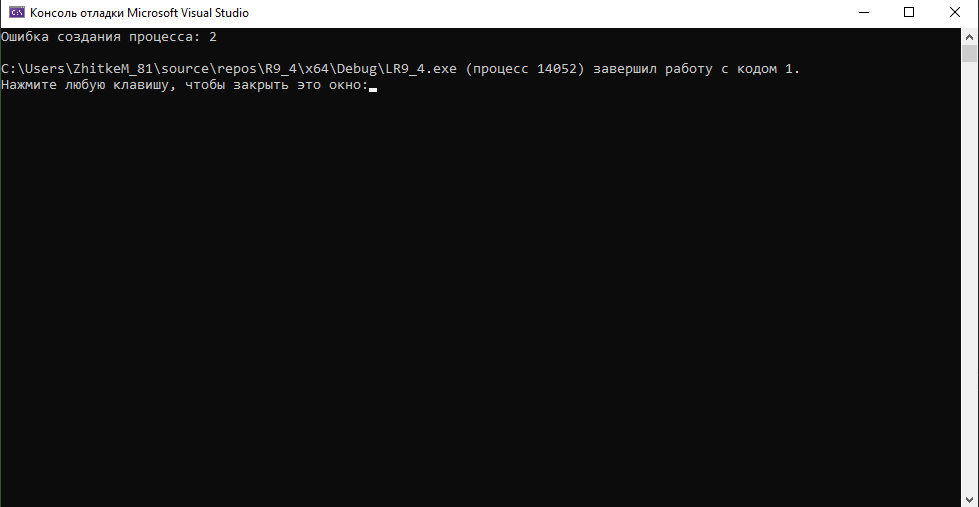
// Закрываем дескрипторы процесса и потока

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return 0;

}



Ответы на контрольные вопросы:

1. Пользовательский режим – в этом режиме работают приложения, и он ограничен в доступе к ресурсам системы для предотвращения вредоносных действий или сбоев. Системный режим – в этом режиме выполняется код операционной системы с полным доступом ко всем ресурсам. Это необходимо для управления аппаратными и системными ресурсами. Важно ограничить доступ в пользовательском режиме для предотвращения сбоев и атак. Пользовательские программы могут только ограниченно взаимодействовать с системой, обращаясь к ней через системные вызовы.
2. Системные вызовы – это интерфейс между приложениями и операционной системой. Они позволяют программам выполнять привилегированные операции, такие как:

open() – открывает файл.

read() – читает данные из файла.

write() – записывает данные в файл.

fork() – создает новый процесс.

exec() – выполняет программу в контексте процесса.

exit() – завершает процесс.

1. Процесс – независимая единица выполнения, содержащая свое собственное адресное пространство. Поток – более легковесная единица выполнения, которая делит адресное пространство с другими потоками того же процесса. Потоки более эффективно используют ресурсы по сравнению с другими процессами.
2. Прерывания – сигналы от аппаратного или программного обеспечения, которые прерывают текущие операции процессора для немедленного выполнения другой задачи. В пользовательском режиме они могут приостановить выполнение программы, чтобы обработать важные события, такие как ввод-вывод.
3. Разделение привилегий – пользовательские программы не имеют прямого доступа к критическим ресурсам. Контроль доступа – программы взаимодействуют с системой через системные вызовы, которые контролируют доступ и разрешения.
4. fork() – копирует текущий процесс.

exec() – заменяет содержимое процесса новой программой.

write() – ожидает завершения дочернего процесса.

1. Семафоры – используются для управления доступом к общим ресурсам. Мьютексы – обеспечивают взаимное исключение, предотвращая одновременный доступ к ресурсу. Мониторы – высокоуровневый механизм синхронизации, объединяющий мьютексы и условные переменные. Барьеры – синхронизируют группы потоков.
2. - Программа инициирует системный вызов.

- Процессор переключается в системный режим.

- ОС обрабатывает запрос, проверяя разрешения и параметры.

- После выполнения задачи возвращается управление приложению.

Вывод: я ознакомился с основами работы процессов в пользовательском режиме.