**Лабораторная работа №2**

**Управление процессами I**

**Цель работы:** Получение практических навыков в работе с процессами, а также с некоторыми ресурсами доступными процессам.

**Темы для предварительного изучения:**

* Утилита Sysinternals Process Explorer ([ссылка](https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer)).

**Постановка задачи для Windows:**

**Приложение Lab-02x:**

Программа выполняет простейший цикл с заданным количеством итераций, при чём каждая итерация выполняется с задержкой 500 мс. Программа должна выводить PID на каждой итерации и перед стартом выводить количество итераций. Количество итераций передается в программу через аргументы командной строки. Если не задан аргумент командной строки приложение завершает свою работу с ошибкой. (Использовать **ExitProcess**)

**Приложение Lab-02a:**

Программа создаёт три дочерних процесса с помощью **CreateProcess**. Все три процесса должны создаваться для программы **Lab-02x**, при чём для передачи имени исполняемого файла (в данном случае используется [полное имя файла](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D1%8F_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0)) и параметра количества итераций должны быть выполнены следующие условия:

* При создании первого процесса используется только первый параметр функции **CreateProcess**. Количество итераций также требуется указать в первом параметре, так задумано.
* При создании второго процесса используется только второй параметр функции **CreateProcess**.
* При создании третьего процесса для передачи имени программы используется – первый параметр, для количества итераций – второй.

Родительский процесс должен дождаться завершения всех дочерних процессов (выполняются одновременно), перед тем как завершиться самому.

***Вопрос****: Все ли процессы созданы успешно? Если нет, почему?*

Также предусмотреть обработку ошибок и очистку ресурсов. Использовать Process Explorer для просмотра информации о процессах и проверки дескрипторов дочерних процессов.

**Модификация Lab-02х:**

Изменить Lab-02х так, чтобы количество итераций можно было получить через переменную окружения ITER\_NUM, если аргумент командной строки не задан. Если не заданы ни переменная окружения, ни аргумент командной строки приложение завершает свою работу с ошибкой.

Установить глобальную переменную окружения ITER\_NUM для вашего пользователя.

**Приложение Lab-02b:**

Создать программу идентичную программе Lab-02a, но:

* В первом вызове **CreateProcess** оставить только имя файла (т.е. убрать количество итераций которое указывалось в Lab-02a).
* В третьем вызове передать NULL вместо второго параметра, и перед этим установить локальную переменную окружения ITER\_NUM. Значение должно отличаться от глобальной переменной.

***Вопрос:*** *Все ли процессы созданы успешно? Сколько итераций выполнил каждый процесс и почему?*

**Приложение Lab-02hw:**

Создать программу, которая выводит "Hello, World!".

**Приложение Lab-02с:**

Создать программу, которая запускает 5 процессов для 5 копий приложения **Lab-02hw**. При запуске всех процессов используется только второй параметр функции **CreateProcess** и указывается только имя приложения без каких-либо путей к файлам.

В начале программы установить текущий каталог для программы **Lab-02с** в любую выбранную вами папку (в этой папке должно не должно быть файла **Lab-02с.exe**).

Перед тестированием программы создать и разместить 5 копий исполняемого файла **Lab-02hw.exe** следующим образом:

* **Lab-02hw-1.exe** в каталог, где находится файл **Lab-02с.exe** который будет запускаться при проверке работы программы;
* **Lab-02hw-2.exe** в каталог, который вы выбрали в качестве текущего для **Lab-02с**;
* **Lab-02hw-3.exe** в системный каталог Windows (можно заранее определить его с помощью функции [**GetSystemDirectory**](https://learn.microsoft.com/en-us/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemdirectorya))
* **Lab-02hw-4.exe** в основной каталог Windows (можно заранее определить его с помощью функции [**GetWindowsDirectory**](https://learn.microsoft.com/en-us/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getwindowsdirectorya));
* **Lab-02hw-5.exe** в каталог, который находится в списке переменной **PATH** (можно использовать как существующий, так и добавить новый);

Проверить успешность запуска всех процессов.

**Приложения Lab-02inf и Lab-02d:**

Написать Lab-02inf с бесконечным циклом.

Написать Lab-02d, которое запускает Lab-02inf и ждёт ввода любого символа для продолжения работы. Изучить дескрипторы процесса Lab-02d в Process Explorer, найти дескриптор дочернего процесса.

После этого в аварийном режиме завершить дочерний процесс с помощью функции **TerminateProcess** и снова перевести программу в режим ожидания ввода любого символа. Изучить, что изменилось.

После этого попытаться получить (с помощью функции **GetProcessId**) и вывести PID дочернего процесса на консоль. Программа снова ждёт ввода любого символа для продолжения работы.

***Вопрос:*** *Каковы результаты вывода на консоль и почему?*

Вызвать функцию **CloseHandle** для дочернего дескриптора и снова попытаться получить и вывести PID дочернего процесса. Изучить изменения в дескрипторах в программе Process Explorer. Программа снова ждёт ввода любого символа, после которого завершит свою работу.

**Постановка задачи для Linux:**

**Приложение Lab-02x:**

Программа является полным аналогом программы из заданий для Windows (модифицированной версии, не стартовой)

**Приложение Lab-02e:**

Программа создаёт два дочерних процесса и загружает в них исполняемый файл **Lab-02x**. При создании процессов должны быть выполнены следующие условия:

* В ходе запуска первого дочернего процесса параметр с количеством итераций должен быть передан как аргумент командной строки
* В ходе запуска второго дочернего процесса параметр с количеством итераций должен быть передан через переменную окружения ITER\_NUM. Переменная создается непосредственно в родительском процессе перед созданием второго дочернего процесса

Родительский процесс должен дождаться завершения всех дочерних процессов (выполняются одновременно).

Изучить информацию о созданных процессах через файловую систему (каталог /proc).

***Вопрос****: Все ли процессы созданы успешно? Если нет, почему?*

**Требования к выполненной работе:**

* Разработанные приложения должны быть скомпилированы и протестированы как на Windows, так и на ОС из семейства Linux в соответствии с заданиями.
* При разработке приложений на Windows требуется использовать функции WinAPI и при разработке под ОС семейства Linux – POSIX API.
* Для генерации и сборки проекта использовать средство CMake.
* Для компиляции файлов при сборке проекта использовать компилятор Clang.

**Дополнительно (совершенно не обязательно, но огромный плюс):**

1. Все операции копирования, задания переменных окружения вынести в конфигурацию CMake.
2. Создать кроссплатформенное приложение Lab-02-cross для создания единственного дочернего процесса Lab-02x с применением CMake. (например, логика и библиотеки выбираются для ОС через #ifdef и прочие директивы препроцессора). Количество итераций, как и раньше, либо через входные параметры, либо через переменные окружения, если входные параметры не указаны.

**Вопросы для контроля:**

1. Что такое процесс?
2. Какие ресурсы доступны процессу?
3. Что такое дочерний и родительский процесс?
4. Что такое системный вызов?
5. Какие системные вызовы предназначены для создания процессов в Windows? Linux?
6. Какие функции из WinAPI и POSIX связаны с ранее названными системными вызовами?
7. Сколько новых процессов появится в результате выполнения программой следующей последовательности вызовов **fork**() (учитывая, что все они выполнятся успешно)?

**fork**();

**fork**();

**fork**();

1. Для чего нужен **exec**?

1. Что такое процесс?

Процесс операционной системы - это экземпляр выполняющейся программы, представляющий собой независимую единицу работы в системе. Он включает в себя выполняемый код, данные, стек, кучу, дескрипторы ресурсов и другую системную информацию. Процесс имеет собственное адресное пространство, контекст выполнения и набор системных ресурсов, выделенных операционной системой.

2. Какие ресурсы доступны процессу?

Процессу доступны следующие ресурсы:

- Память: код, данные и стек процесса.

- Процессорное время: ресурсы ЦП, которые выделяются для выполнения процесса.

- Файлы: доступ к файловым системам и открытым файлам.

- \*\*Сетевые ресурсы\*\*: возможность взаимодействия с другими процессами через сети.

- Сигналы: механизмы для межпроцессного взаимодействия и управления.

3. Что такое дочерний и родительский процесс?

- Родительский процесс\*\* — это процесс, который создает другой процесс. Обычно он отвечает за управление дочерним процессом.

- Дочерний процесс — это процесс, созданный родительским процессом. Он наследует некоторые ресурсы и атрибуты от родительского процесса, но может иметь свое собственное состояние и ресурсы.

4. Что такое системный вызов?

Системный вызов — это механизм, который позволяет программе взаимодействовать с операционной системой. Он предоставляет интерфейс для выполнения операций, таких как создание процессов, управление файлами и сетевыми соединениями. Программы делают запросы к ядру ОС через системные вызовы.

5. Какие системные вызовы предназначены для создания процессов в Windows? Linux?

- Windows: Основной системный вызов для создания процессов — это `CreateProcess()`. Он создает новый процесс и его первичный поток.

- Linux: Для создания процессов используется системный вызов `fork()`, который создает новый процесс, копируя родительский. После этого можно использовать `exec()` для замены памяти нового процесса на другую программу.

6. Какие функции из WinAPI и POSIX связаны с ранее названными системными вызовами?

- WinAPI:

- `CreateProcess()`: создает новый процесс.

- `ExitProcess()`: завершает процесс.

- POSIX:

- `fork()`: создает дочерний процесс.

- `exec()`: заменяет текущий процесс новым.

7. Сколько новых процессов появится в результате выполнения программы следующей последовательности вызовов `fork()` (учитывая, что все они выполнятся успешно)?

Каждый вызов `fork()` создает новый процесс. Ваша последовательность вызовов:

```c

fork(); // 1

fork(); // 2

fork(); // 3

```

Рассмотрим это подробно:

- Первый `fork()` создает 1 новый процесс (всего 2).

- Второй `fork()` вызывается в обоих процессах, создавая еще 2 (всего 4).

- Третий `fork()` вызывается в каждом из 4 процессов, создавая еще 4 (всего 8).

Таким образом, после выполнения всех `fork()` будет 8 процессов (включая первоначальный).

8. Для чего нужен `exec`?

Функция `exec` (или один из её вариантов, таких как `execl()`, `execv()`, и т.д.) используется для замены текущего процесса на новый. Это позволяет загрузить и выполнить другую программу в том же процессе. При вызове `exec` текущее состояние процесса (например, память, открытые файлы) заменяется на новое состояние, связанное с новой программой. Это особенно полезно после вызова `fork()`, когда родительский процесс может создать дочерний и затем использовать `exec` для выполнения другой программы.