**Лабораторная работа №2**

**Задание для выполнения**

Составить следующую программу, которая:

• принимая дескриптор, имя или полное имя модуля, возвращает другие два элемента в своих выходных параметрах (выполнить задание для своей программы и для любой известной библиотеки),

• будет выполнять последовательно по шагам следующее:

1. используя функцию GetCurrentProcessId определит идентификатор текущего процесса.

2. используя функцию GetCurrentProcess определит псевдодескриптор текущего процесса.

3. используя функцию DuplicateHandle и значение псевдодескриптора определит дескриптора текущего процесса.

4. используя функцию OpenProcess определит копию дескриптора текущего процесса

5. закроет дескриптор, полученный функцией DuplicateHandle

6. закроет дескриптор, полученный функцией OpenProcess

• выдает список перечисления всех процессов, потоков, модулей и их свойства в системе.

**Решение:**

import ctypes

import ctypes.wintypes as wintypes

import psutil

# Константы

PROCESS\_ALL\_ACCESS = 0x1F0FFF

DUPLICATE\_SAME\_ACCESS = 0x00000002

kernel32 = ctypes.windll.kernel32

# Файл для отчёта

report\_file = "report.txt"

with open(report\_file, "w", encoding="utf-8") as f:

    # 1. Получение текущего PID

    pid = kernel32.GetCurrentProcessId()

    f.write(f"[1] Текущий PID: {pid}\n")

    # 2. Получение псевдодескриптора процесса

    pseudo\_handle = kernel32.GetCurrentProcess()

    f.write(f"[2] Псевдодескриптор процесса: {pseudo\_handle}\n")

    # 3. Дублирование дескриптора

    # Сначала получаем реальный дескриптор через OpenProcess

    real\_handle = kernel32.OpenProcess(PROCESS\_ALL\_ACCESS, False, pid)

    current\_process\_handle = wintypes.HANDLE()

    if real\_handle:

        success = kernel32.DuplicateHandle(

            real\_handle,           # hSourceProcessHandle - реальный дескриптор

            real\_handle,           # hSourceHandle - тот же дескриптор

            real\_handle,           # hTargetProcessHandle - целевой процесс

            ctypes.byref(current\_process\_handle),  # lpTargetHandle

            0,                     # dwDesiredAccess

            False,                 # bInheritHandle

            DUPLICATE\_SAME\_ACCESS  # dwOptions

        )

        # Закрываем временный дескриптор

        kernel32.CloseHandle(real\_handle)

    else:

        success = False

    if success:

        f.write(f"[3] Дубликат дескриптора процесса: {current\_process\_handle.value}\n")

    else:

        error\_code = kernel32.GetLastError()

        f.write(f"[3] Ошибка при DuplicateHandle: {error\_code}\n")

    # 4. Открытие процесса по PID

    open\_handle = kernel32.OpenProcess(PROCESS\_ALL\_ACCESS, False, pid)

    if open\_handle:

        f.write(f"[4] OpenProcess дескриптор: {open\_handle}\n")

    else:

        error\_code = kernel32.GetLastError()

        f.write(f"[4] Ошибка при OpenProcess: {error\_code}\n")

    # 5. Закрытие дубликата

    if success and current\_process\_handle.value:  # Проверяем success И значение дескриптора

        kernel32.CloseHandle(current\_process\_handle)

        f.write("[5] Закрыт дубликат дескриптора.\n")

    else:

        f.write("[5] Дубликат дескриптора не был создан или уже закрыт.\n")

    # 6. Закрытие дескриптора OpenProcess

    if open\_handle:

        kernel32.CloseHandle(open\_handle)

        f.write("[6] Закрыт дескриптор OpenProcess.\n")

    else:

        f.write("[6] Дескриптор OpenProcess не был создан.\n")

    # 7. Перечисление всех процессов, потоков и модулей

    f.write("\n[7] Перечисление всех процессов, потоков и модулей:\n\n")

    for proc in psutil.process\_iter(['pid', 'name']):

        try:

            f.write(f"Процесс: {proc.info['name']} | PID: {proc.info['pid']}\n")

            p = psutil.Process(proc.info['pid'])

            # Потоки процесса

            try:

                threads = p.threads()

                for thread in threads:

                    f.write(f"  └─ Поток TID: {thread.id} | Время: {thread.system\_time:.2f}s\n")

            except (psutil.AccessDenied, psutil.NoSuchProcess):

                f.write("  └─ [!] Нет доступа к потокам процесса.\n")

            # Загруженные модули (DLL)

            f.write("  └─ Модули:\n")

            try:

                memory\_maps = p.memory\_maps()

                dll\_count = 0

                for dll in memory\_maps:

                    if '.dll' in dll.path.lower():

                        f.write(f"     - {dll.path}\n")

                        dll\_count += 1

                        if dll\_count > 10:  # Ограничиваем вывод для читаемости

                            f.write(f"     - ... и ещё {len(memory\_maps) - dll\_count} модулей\n")

                            break

                if dll\_count == 0:

                    f.write("     - Нет доступных DLL модулей\n")

            except (psutil.AccessDenied, psutil.NoSuchProcess):

                f.write("     - [!] Нет доступа к модулям процесса.\n")

        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):

            f.write("  [!] Нет доступа к процессу или он завершён.\n")

            continue

        f.write("\n")  # Пустая строка между процессами для читаемости

print(f"Отчёт сохранён в файл: {report\_file}")

**Результат:**

