Код программы:

```
#include <windows.h>
#include <tlhelp32.h>
#include <psapi.h>
#include <iostream>

// Функция для получения идентификатора текущего процесса
DWORD getCurrentProcessId() {
    return GetCurrentProcessId();
}

// Функция для получения псевдодескриптора текущего процесса
HANDLE getCurrentProcessHandle() {
    return GetCurrentProcess();
}

// Функция для дублирования дескриптора
HANDLE duplicateProcessHandle(HANDLE hProcess) {
    HANDLE hDuplicate = NULL;
    if (DuplicateHandle(GetCurrentProcess(), hProcess,
GetCurrentProcess(), &hDuplicate, 0, FALSE, DUPLICATE_SAME_ACCESS)) {
        return hDuplicate;
    } else {
        std::cerr << "Не удалось дублировать дескриптор. Ошибка: " <<
GetLastError() << std::endl;
        return NULL;
    }
}

// Функция для открытия процесса по его идентификатору
HANDLE openProcess(DWORD processId) {
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ, FALSE, processId);
    if (hProcess == NULL) {
        std::cerr << "Не удалось открыть процесс. Ошибка: " <<
GetLastError() << std::endl;
    }
    return hProcess;
}

// Функция для закрытия дескриптора
void closeHandle(HANDLE hProcess) {
    if (!CloseHandle(hProcess)) {
```

```
        std::cerr << "Не удалось закрыть дескриптор. Ошибка: " <<
GetLastError() << std::endl;
    }
}

// Функция для создания снимка процессов
HANDLE createSnapshot() {
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hSnapshot == INVALID_HANDLE_VALUE) {
        std::cerr << "Не удалось создать снимок. Ошибка: " <<
GetLastError() << std::endl;
    }
    return hSnapshot;
}

// Функция для вывода списка процессов
void listProcesses() {
    HANDLE hSnapshot = createSnapshot();
    if (hSnapshot == INVALID_HANDLE_VALUE) {
        return;
    }

    PROCESSENTRY32 pe32;
    pe32.dwSize = sizeof(PROCESSENTRY32);

    if (!Process32First(hSnapshot, &pe32)) {
        std::cerr << "Не удалось получить первый процесс. Ошибка: " <<
GetLastError() << std::endl;
        CloseHandle(hSnapshot);
        return;
    }

    do {
        std::wcout << L"Имя процесса: " << pe32.szExeFile << L" | ID
процесса: " << pe32.th32ProcessID << std::endl;
    } while (Process32Next(hSnapshot, &pe32));

    CloseHandle(hSnapshot);
}

// Функция для получения информации о памяти процесса
void getProcessMemoryInfo(DWORD processId) {
    HANDLE hProcess = openProcess(processId);
```

```
    if (hProcess == NULL) {
        return;
    }

    PROCESS_MEMORY_COUNTERS pmc;
    if (GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc))) {
        std::cout << "ID процесса: " << processId << std::endl;
        std::cout << "Размер рабочего набора: " << pmc.WorkingSetSize
<< std::endl;
        std::cout << "Использование файла подкачки: " <<
pmc.PagefileUsage << std::endl;
    } else {
        std::cerr << "Не удалось получить информацию о памяти для
процесса с ID: " << processId << std::endl;
    }

    closeHandle(hProcess);
}

int main() {
    // Шаг 1: Получение идентификатора текущего процесса
    DWORD processId = getCurrentProcessId();
    std::cout << "ID текущего процесса: " << processId << std::endl;

    // Шаг 2: Получение псевдодескриптора текущего процесса
    HANDLE hProcess = getCurrentProcessHandle();
    std::cout << "Псевдодескриптор текущего процесса: " << hProcess <<
std::endl;

    // Шаг 3: Дублирование дескриптора
    HANDLE hDuplicate = duplicateProcessHandle(hProcess);
    if (hDuplicate != NULL) {
        std::cout << "Дублированный дескриптор процесса: " <<
hDuplicate << std::endl;

        // Шаг 4: Открытие процесса
        HANDLE hOpenProcess = openProcess(processId);
        if (hOpenProcess != NULL) {
            std::cout << "Открытый дескриптор процесса: " <<
hOpenProcess << std::endl;

            // Шаг 5: Закрытие дублированного дескриптора
            closeHandle(hDuplicate);
```

```
            std::cout << "Дублированный дескриптор закрыт" <<
std::endl;

            // Шаг 6: Закрытие дескриптора открытого процесса
            closeHandle(hOpenProcess);
            std::cout << "Открытый дескриптор процесса закрыт" <<
std::endl;
        }
    }

    // Вывод списка процессов и их информации
    listProcesses();

    // Пример получения информации о памяти текущего процесса
    getProcessMemoryInfo(processId);

    return 0;
}
```

результат:

```
[Running] cd "d:\db_go\oc\" && g++ main.cpp -o main &&
ID текущего процесса: 16576
Псевдодескриптор текущего процесса: 0xffffffffffffffff
Дублированный дескриптор процесса: 0xc4
Открытый дескриптор процесса: 0xc8
Дублированный дескриптор закрыт
Открытый дескриптор процесса закрыт
ID процесса: 16576
Размер рабочего набора: 3829760
Использование файла подкачки: 856064
```