

Process Memory Functions



what does this do?

```
#include "proc.h"
```

DWORD GetProcId(const wchar_t* procName) { // This function takes a process name as input and returns its process ID.

```
    DWORD procId = 0;
```

```
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
```

```
    if (hSnapshot != INVALID_HANDLE_VALUE) {
```

```
        PROCESSENTRY32 procEntry;
```

```
        procEntry.dwSize = sizeof(procEntry);
```

```
        if (Process32First(hSnapshot, &procEntry)) { // documentazione:
```

<https://learn.microsoft.com/en-us/windows/win32/toolhelp/taking-a-snapshot-and-viewing-processes>

```
            do {
```

```
                if (!_wcsicmp(procEntry.szExeFile, procName)) {
```

```
                    procId = procEntry.th32ProcessID;
```

```
                    break;
```

```
                }
```

```
            } while (Process32Next(hSnapshot, &procEntry));
```

```
        }
```

```
    }
```

```
    CloseHandle(hSnapshot);
```

```
    return procId;
```

```
}
```

```
uintptr_t GetModuleBaseAddress(DWORD procId, const wchar_t* modName) {
```

```
    uintptr_t modBaseAddr = 0;
```

```
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE | TH32CS_SNAPMODULE32, procId);
```

```
    if (hSnapshot != INVALID_HANDLE_VALUE) {
```

```
        MODULEENTRY32 modEntry;
```

```
        modEntry.dwSize = sizeof(modEntry);
```

```
        if (Module32First(hSnapshot, &modEntry)) {
```

```
            do {
```

```
                if (!_wcsicmp(modEntry.szModule, modName)) {
```

```
                    modBaseAddr = (uintptr_t)modEntry.modBaseAddr;
```

```
                    break;
```

```
                }
```

```
            } while (Module32Next(hSnapshot, &modEntry));
```

```
        }
```

```

    }
    CloseHandle(hSnapshot);
    return modBaseAddr;

}

uintptr_t FindDMAAddy(HANDLE hProcess, uintptr_t ptr, std::vector<unsigned int> offsets) { //
    uintptr_t addr = ptr;
    for (unsigned int i = 0; i < offsets.size(); ++i) {
        ReadProcessMemory(hProcess, (BYTE*)addr, &addr, sizeof(addr), 0);
        addr += offsets[i];
    }
    return addr;
}
}

```

This C++ code is related to **process memory manipulation** on Windows. It defines three functions commonly used in **game hacking**, **reverse engineering**, or **low-level process inspection**. Here's a breakdown of what each function does:

◆ `DWORD GetProcId(const wchar_t* procName)`

This function returns the **Process ID (PID)** of a process given its name.

What it does:

- Takes a process name (like `L"notepad.exe"`) as input.
- Uses `CreateToolhelp32Snapshot` to get a snapshot of all running processes.
- Iterates through the list using `Process32First` and `Process32Next`.
- Compares each process name with the input (`_wcsicmp` = case-insensitive comparison).
- If found, retrieves and returns the `th32ProcessID`.

 Example usage: Finding the PID of a running game or application.


◆ `uintptr_t GetModuleBaseAddress(DWORD procId, const wchar_t* modName)`

This function retrieves the **base memory address** of a module (like a DLL or EXE) loaded into a process.

What it does:

- Takes the process ID and module name as input.

- Takes a snapshot of the modules loaded by that process.
- Iterates over them to find the one that matches ``modName``.
- If found, returns its base address as a ``uintptr_t``.


 Useful for getting the starting memory address of a game or app's executable in memory.

◆ ``uintptr_t FindDMAAddy(HANDLE hProcess, uintptr_t ptr, std::vector<unsigned int> offsets)``

This is the classic **multi-level pointer dereferencing** function, often used for **Dynamic Memory Address (DMA)** calculations.

What it does:

- Takes a process handle, a base pointer, and a list of offsets.
- Iteratively:
 - Dereferences the pointer using ``ReadProcessMemory``.
 - Adds the next offset in the chain.
- Returns the final memory address reached after all dereferences and offset additions.

 Commonly used in game cheats/trainers to access variables like health or ammo that are stored at dynamic addresses.

Example Scenario

Let's say you want to read the health value of a player in a game:

1. Use ``GetProcId(L"game.exe")`` to get the PID.
2. Use ``GetModuleBaseAddress(pid, L"game.exe")`` to get the base address of the main executable.
3. Use ``FindDMAAddy`` with the base pointer and a vector of offsets to find the actual memory address of the health variable.

If you're working with this code, you're probably dabbling in reverse engineering or writing a memory trainer, yeah?