

运行时内存修改器 说明文档

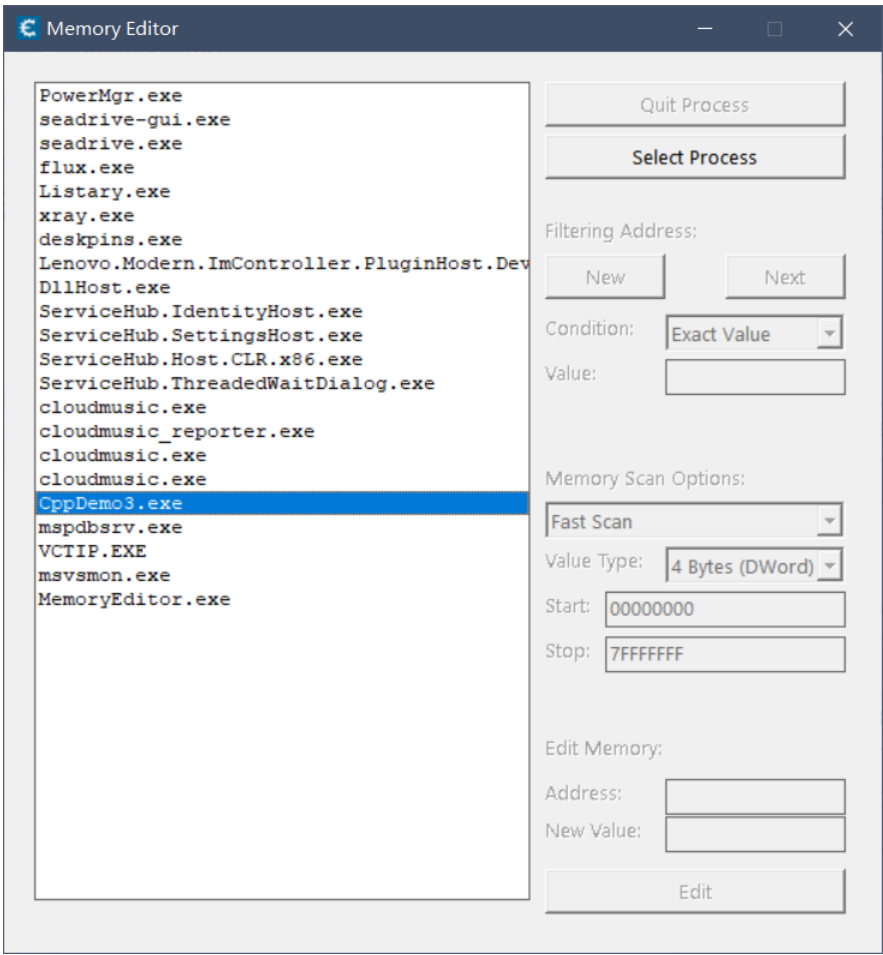
开发环境

Visual Studio 2022 + MASM32 SDK Version 11.0，按照文档中提供的方案配置编程环境、设置项目的链接器和汇编器。

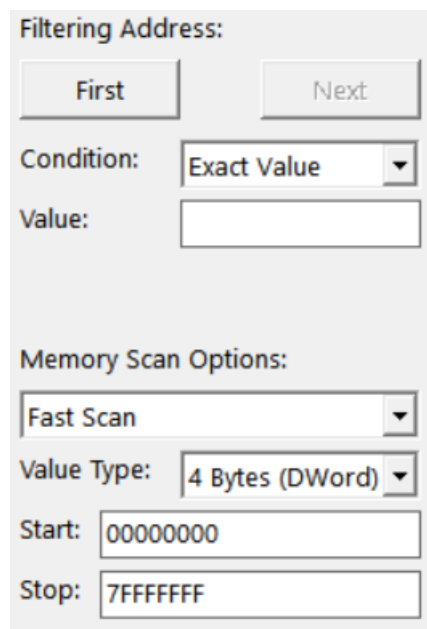
使用指南

打开程序后，进入进程选择界面。左侧的列表展示了当前正在运行的、支持修改的进程名称（注：仅支持32位用户级进程）。鼠标左键点击要选择的进程名称，点击右侧操作栏上方的“Select Process”，即可选中进程，进入修改界面。

在修改界面也随时可以点击右上角的“Quit Process”退回进程选择界面。



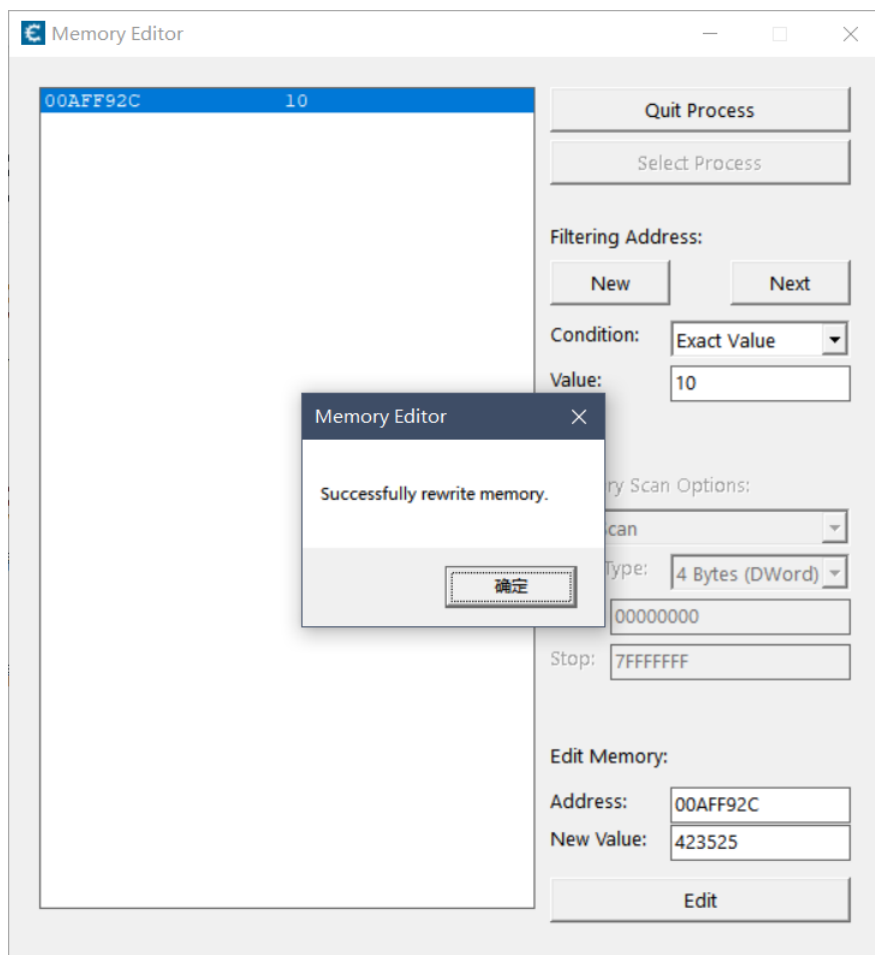
修改界面右侧的操作栏主要包括三个部分：**搜索操作**（Filtering Address）、**搜索选项**（Memory Scan Options）、**修改操作**（Edit Memory）。点击搜索操作栏的“New”按钮开启一次新的搜索任务。此时可设置的选项如下所示。



- **搜索操作栏可以在每次搜索后重新设置，调整不同的搜索条件：**
 - “First”按钮开启第一次搜索，之后的搜索通过“Next”按钮启动。
 - 在“Condition”一栏选择搜索的条件（大于、大于等于、等于、小于等于、小于）；
 - 在“Value”一栏输入十进制的要搜索的数值。
 - 在第一次搜索完成后，“First”按钮会变为“New”按钮，点击该按钮可以重新从头开启一次新的搜索任务。
- **搜索选项栏需要在首次搜索前设置好，无法在后几次搜索中调整：**
 - 首个下拉选项对应搜索的步长，“Fast Scan”表示步长默认设为数据类型的大小（最大为4），“x-Byte Alignment”可以根据不同的数据对齐方式，选择不同的搜索步长；
 - “Value Type”表示数据的类型，支持各种长度的无符号整数和浮点数；
 - “Start”和“Stop”表示要搜索的地址范围（十六进制格式），默认为 00000000 到 7FFFFFFF。

每次执行搜索时，左侧的列表栏会更新符合条件的地址，第二次及以后的搜索会在上一次搜索的结果中进一步查找。用户可以通过多次搜索来确定数据的地址。

鼠标左键点击列表栏中的某一行，程序会自动将其地址填写到修改操作栏的地址“Address”一栏中。用户也可以在该栏中手动输入地址的数值（十六进制格式）。确定地址后，在修改操作栏中的“New Value”中填写新的十进制数值，点击下方的“Edit”按钮，即可完成修改。修改成功会弹窗提示“Successfully rewrite memory.”。



实现原理

主程序逻辑

主程序代码见 `winmain.asm`，在创建窗口后，利用事件循环和处理来进行状态间的转移，以此来实现完整的程序逻辑。其中涉及到的和内存修改相关的功能都实现在单独的模块中，借助 `memeditor.inc` 引入。

事件的处理包含在WinProc过程中，其中维护了一个有限状态自动机（当前状态为 `state`），对于初始状态、选择进程后、准备开始扫描、进行扫描后的几种状态分别进行处理。当在界面中按下按钮之后，通常涉及到界面组件状态的改变、实际功能的调用等。

在中期提交时，我们还提供了控制台版本的程序，主程序代码见 `main.asm`。由于之后又添加了许多内存扫描参数，这在控制台中修改并不容易，不便于使用，因此程序的最终版本不再支持控制台的运行方式。

GUI界面

在创建主窗口之后，通过添加子窗口的方式，使用 `api: CreateWindow` 根据需求添加组件。在创建组件的过程中，需要查阅Win32中的窗口风格、窗口类型等常量，通过参数值的不同来区分组件的类型。

此外，在添加组件元素、修改内容时，还需要在查阅信息类型之后，使用 `api: SendMessage` 向窗口传递不同的信息。读取组件内容也需要使用特定的 `api: GetDlgItemText` 来处理，并根据需要将字符串转换为特定的数据类型。

选取进程

这部分的代码见 `process.asm` 文件。

要让用户选择程序来修改，首先就需要给用户所有可供修改内存的进程。由于 32 位程序中使用的 `EnumProcesses` 等 Windows API 只能列举出 32 位进程，因此这里支持的程序**仅限于 32 位**。该步骤思路如下：

1. 使用 `api: EnumProcesses` 将所有的进程 PID 存储在数组中。
2. 迭代读取进程的 PID，使用 `api: OpenProcess` 和 `api: EnumProcessModules` 打开并获取进程的依赖项和模块。由于 `api: OpenProcess` 也会在修改内存时使用，因此如果该操作打开失败，那么就不支持内存的修改。
3. 将支持的进程 PID 存储在数组中，在之后选择进程时使用。

目标地址搜索

中期进度

这部分的代码见 `filter.asm` 文件。

需要实现内存修改功能，首先应能定位到相应的内存。因此，这部分的基本思路模仿了 [Cheat Engine](#)，即：

1. 根据用户选择的进程 PID，通过 `api: OpenProcess` 打开相应进程并获得其句柄（首个参数设置为 `PROCESS_AL_ACCESS` 以便读取与更改）。之后进行第一遍搜索。
2. 从地址 `0` 开始，向 `7FFFFFFFH`，利用获得的句柄，使用 `api: VirtualQueryEx` 逐内存页面读取信息。
3. 若 `api: VirtualQueryEx` 读取的内存页面信息中页面状态为 `MEM_COMMIT` 才逐 `DWORD` 读取此页面中的内存地址。
4. 使用 `api: ReadProcessMemory` 逐 `DWORD` 读取内存地址中的值，并与用户输入的数值进行比较。
5. 若二者相同，则这个内存地址记录在 `lastsearch` 数组(`lastsearch DWORD 10240 DUP(?)`)中，并显示出来。
6. 第一遍搜索完成后，用户在游戏（想更改的进程对象）中使欲改动的值发生一定变化，并记录这个值。让用户输入记录的变化后的值，之后进行第二遍搜索。
7. 逐个读取 `lastsearch` 中每个单元存储的内存地址中的值，并与用户此次输入的值比较，若相同则输出其内存地址。

这样，通过若干次搜索，就基本可定位用户想更改的数值所在的内存地址，这样就能通过下一步的 `Edit` 更改它了。

增加内容

在上述思路的基础上，增加（或修改）了以下内容：

1. 查询了32位机器内存中用户空间及内核空间的分配，一度将搜索地址的最大范围修改为用户空间 `0H~BFFFFFFFH`。但之后发现实际上用户程序使用的空间集中于 `0H~7FFFFFFFH`，`7FFFFFFFH` 以上的内存部分被 `PCI Memory Address Range` 所占据，因此将搜索地址的最大范围修改回了 `0H~7FFFFFFFH`。
2. 增加了搜索选项：**判断条件**，即大于等于、大于、小于等于、小于、等于。这些选项分别对应 `>=`、`>`、`<=`、`<`、`=`。用户可以选择其中一个选项，然后输入一个数值，之后进行搜索。搜索结果中的内存地址对应的值与用户输入的数值进行比较，若满足用户选择的条件，则输出该内存地址。
3. 增加了搜索选项：**地址步长**，现在可以按1个字节、2个字节和4个字节的步长进行搜索。这考虑到了进程内存中数据的对齐方式，可以灵活处理未按字长对齐的数据。在GUI界面中，这可以通过选择"Memory Scan Options"来选择，默认的Fast Scan会根据数据类型自动选择步长。

- 增加了搜索选项：**目标值数据类型**，现在可以按 byte、2 bytes (word)、4 bytes (dword)、8 bytes (qword)、32-bit float (real4)、64-bit double (real8) 的数据类型进行搜索。其中 qword 在比较的过程中还需要分别提取高位低位并分别比较；float 和 double 数据类型则需要借助FPU的相关指令实现大小比较，并结合实际的二进制数位来判断浮点数类型，解决异常值的问题。
- 在内存中**搜索的地址范围**不再为固定的值，而是可以由用户自己指定。如用户此前搜索到某个数值的地址为 0x12345678，则在搜索与该数值联系紧密的地址时，可以将搜索范围设置为 0x12345600~0x12345700，因为联系紧密的数值通常位于内存中地址相接近的区域，提供自定地址的搜索可以大大减少搜索的时间。

内存修改

这部分的代码见 memedit.asm 文件。

内存修改的实现比较简单，在使用 api: OpenProcess 打开相应进程并获取句柄后，借助 api: ReadProcessMemory 和 api: WriteProcessMemory 即可完成读写操作。需要根据不同的数据类型写入特定长度的内容。

难点

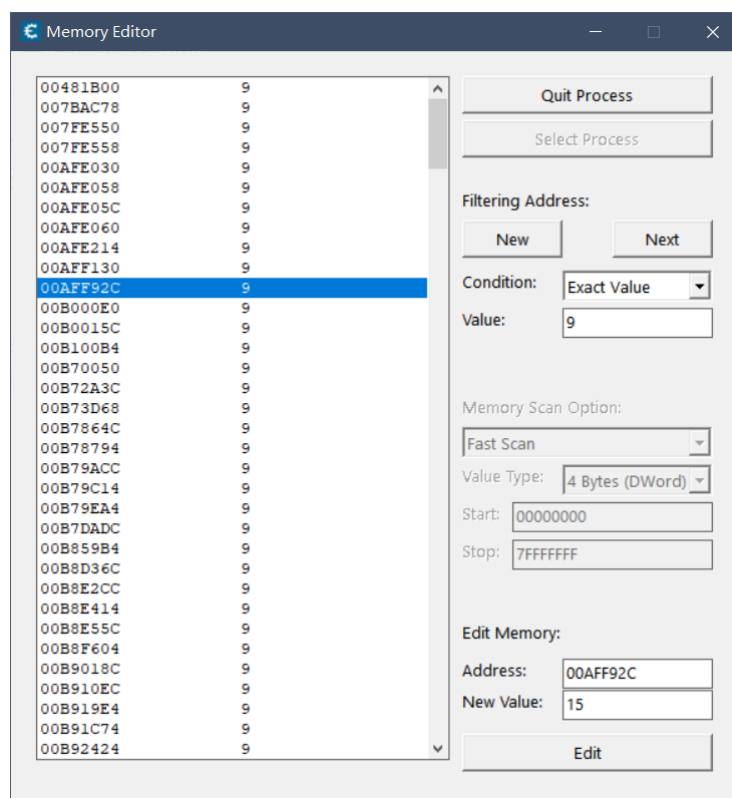
- 对 Win32 API 不熟悉，此前完全没有用过，这次使用的 OpenProcess, EnumProcessModules, VirtualQueryEx, ReadProcessMemory 等 API 都是临时参考了 [Win32 API 编程参考 - Win32 apps | Microsoft Learn](#) 学习使用的。
- 在使用 API 时，尤其是执行处理打开进程、访问内存等可能不成功的操作时，需要**充分考虑异常情况**，设置过程的出口，防止出现意料之外的运行问题。
- 此前并不是很了解使用汇编语言编程，不熟悉汇编语言程序中出现的 bug，如过程的定义方式、多模块程序设计、指针变量的使用等，导致 debug 难度更高、花费的时间更长。
- 寄存器中的值会因为某些操作（如输入输出）而改变**，因此最初未注意保存寄存器值时，出现了一些令人摸不着头脑的 bug。同时我们组的项目涉及到对内存的大量读取和检测，寄存器的值更新次数多、速度快，影响了找到问题所在，最后是通过加很多个断点并单步执行多个循环才发现问题。
- GUI 界面的上手和基本使用并不容易**。由于没有现有的绘图库和组件库，我们需要参考提供的 WinApp 窗口程序，借助 Win32 API 从建立窗口开始，为每个按钮、文本框等组件创建各自子窗口，并设置正确的格式和位置。创建和布局 GUI 的所有控件就需要一定的工作量，且在使用 CreateWindowEx 等 API 创建不同组件时，或处理捕捉到的事件类型时，参数均需要一个个查询，比较耗时。
- 对**宏编译的条件**掌握不够充分，原本搜索选项想采用宏的条件编译伪代码(IF & ENDIF)实现以节省代码，但是由于 IF 伪代码之后 expression 只能使用立即数进行 EQ 比较，而不能传入变量，导致无法利用宏完成，最后只能采用 .IF & .ENDIF 伪代码的形式实现。
- 在实现浮点数类型搜索的过程中，需要使用到**浮点处理硬件FPU**，并使用x86浮点数指令集中的指令来实现所有的比较操作。首先需要学习FPU寄存器栈的原理和相关的使用方式。其次，由于内存扫描的过程中会遇到许多的特殊类型浮点数（无穷、NaN），而使用浮点数指令进行比较时，并不会提供显式的异常信息，因此需要在进行比较前，基于位操作单独实现浮点数类型的判断。
- 内存的读取和修改涉及到对另一程序的存储方式的分析，因此有时需要使用OllyDbg等**反汇编工具**来分析程序的内存结构和机器代码。比如在测试一个C++程序时，MSVC编译器在编译时进行了优化，没有将代码中声明的变量都分别存储在内存中，对不同的变量建立了数值关系，这可能导致内存修改的失败或异常行为。

创新点

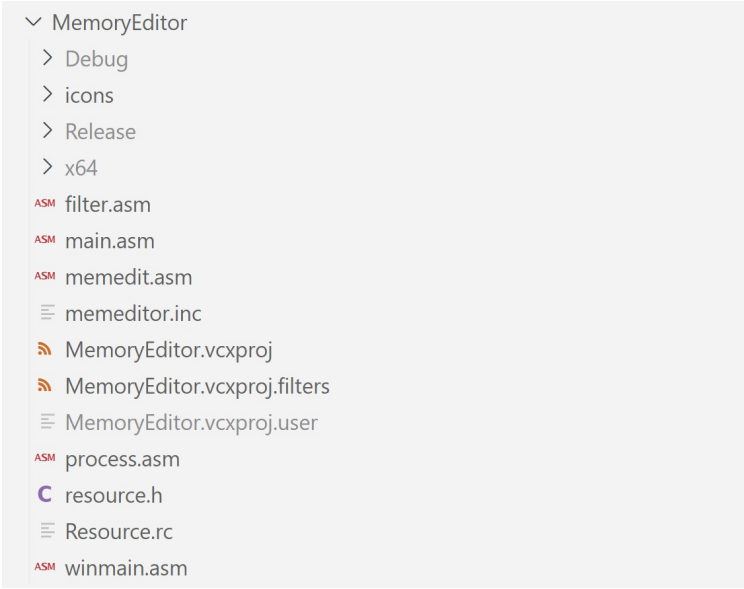
- 最初的内存搜索是暴力的全局搜索（00000000H-7FFFFFFFH），速度较慢，而之后使用了 `VirtualQueryEx` 这个 API，使用了上学期所学的操作系统的知识，**以页表为单位跳过当前句柄没有访问权限的内存区域**，节省了大量不必要的内存访问与查询，极大地加快了运行速度。

```
invoke    VirtualQueryEx, ebx, edi, ADDR mbi, SIZEOF mbi
test      eax, eax
jz        fail_RET
mov       edx, maxAddr
add       edx, mbi[12]
mov       maxAddr, edx
mov       eax, mbi[16]
cmp       eax, MEM_COMMIT
je        PIECE
```

- 加入了丰富多样的**搜索选项**，包括搜索步长、搜索地址、被搜索数数据类型、搜索判断条件等，适用范围广泛，拓展了可用性。



- 相比参考作品“Cheat Engine”有更简洁、直接的操作界面，在保留常用的基本功能的同时，程序运行更轻便，还支持直接输入地址进行修改。
- 对工程文件进行了分模块管理，使得代码更加清晰，易于维护。



```

MemoryEditor
├── Debug
├── icons
├── Release
├── x64
├── filter.asm
├── main.asm
├── memedit.asm
├── memeditor.inc
├── MemoryEditor.vcxproj
├── MemoryEditor.vcxproj.filters
├── MemoryEditor.vcxproj.user
├── process.asm
├── resource.h
├── Resource.rc
└── winmain.asm

```

与中期相比的进展

中期提交时已经实现了内存修改的基本功能，包括进程选取、简单的地址扫描、内存修改等，并实现了简单的GUI界面，同时也提供了控制台版本的程序。

之后，我们主要对修改器的功能进行了扩展，完善了GUI界面，并解决了不少的bug。我们依次完成了更多数据类型的支持、搜索地址步长选择、搜索条件选择（如大于小于）、搜索范围选择等功能（如上文所述“目标地址搜索”部分所述），并针对这些功能重新排布了GUI界面，设计了图标、组件字体等元素，并根据用户的使用步骤、功能的类别来设计界面，更易于使用。

小组分工

- 顾洋丞：进程选取、内存修改，主事件逻辑，长整数和浮点数的搜索
- 王麒杰：目标地址的搜索和存储，搜索选项的功能扩展
- 王子扬：用户 GUI 界面设计和全部组件的布局