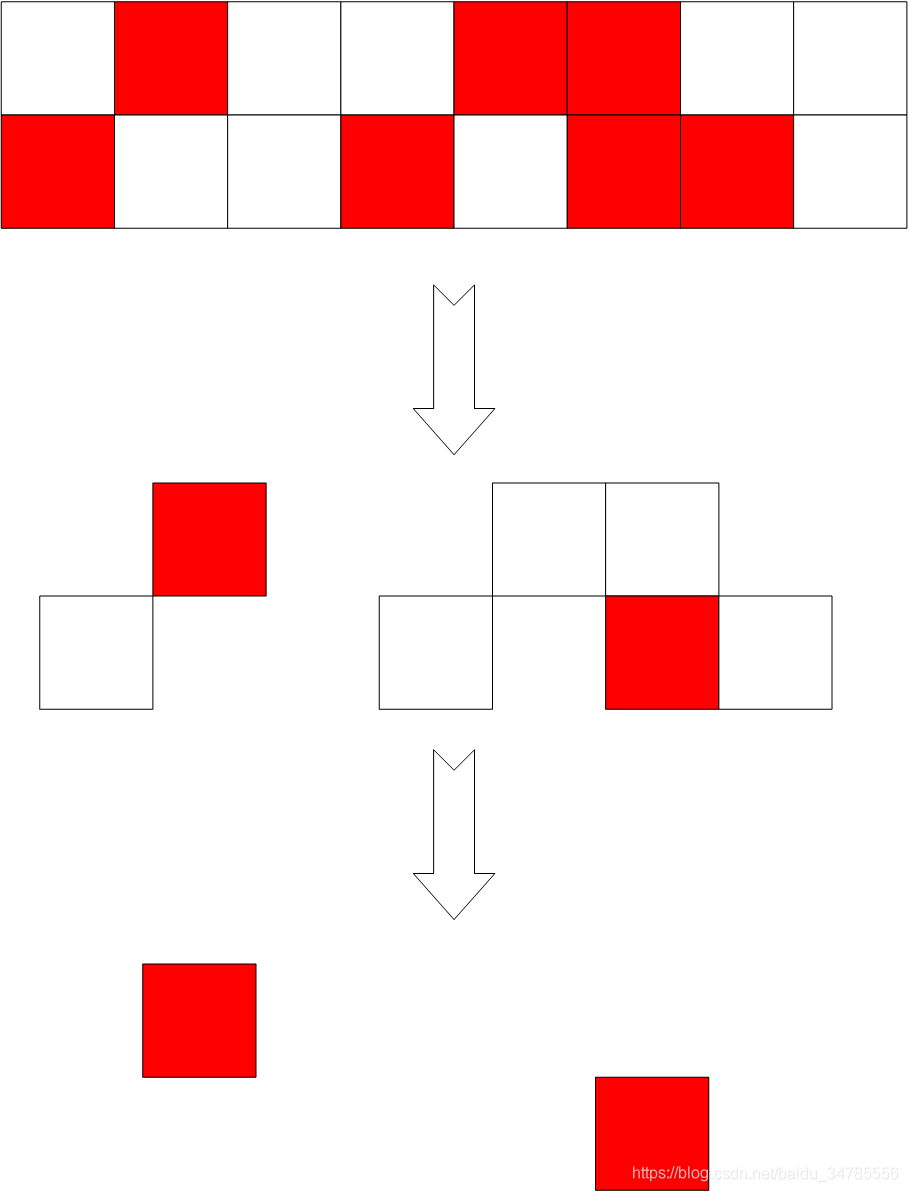
**Windows开发——内存读写API**

我们知道数据都是写在内存中的，通过一些API我们可以访问并修改内存中的数据，达到修改游戏数据的功能。

通过一个小项目，了解windos读写内存API，以及进程id获取相关API。



大体思路如下：

（1）先找到进程：（API：CreateToolhelp32Snapshot、Process32First、Process32Next）

void ShowProcessList() //功能1显示进程

{

PROCESSENTRY32 pc;//定义一个32接收变量

pc.dwSize = sizeof(pc);

int count = 0; //进程计数

HANDLE hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);//进程状态“快照”

BOOL bMore = Process32First(hProcessSnap, &pc); //“快照”中获取第一个进程

while (bMore)

{

cout << "\n---------------------------------------\n";

cout << "id: " << pc.th32ProcessID << endl; //.th32ProcessID获取进程id

wprintf(L"name:%s\n", pc.szExeFile);

bMore = Process32Next(hProcessSnap, &pc); //转向下一进程

count++;

}

CloseHandle(hProcessSnap);

cout << "目前进程数：" << count << endl;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

涉及API:

CreateToolhelp32Snapshot

Process32First

Process32Next

（2）编辑进程   （设计两轮查找，API: OpenProcess、WriteProcessMemory）

void EditProcessData() //功能2 编辑进程

{

DWORD dwId = 0;

DWORD dwSearchValue = 0;

DWORD dwAddrList[4 \* KONEK] = { 0 };

DWORD dwAddrCount = 0;

BOOL bRet = false;

scanf\_s("%d", &dwId); //输入需要编辑的进程id

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, dwId);//获取进程句柄（打开进程）

system("pause");

printf("Please input the value which you want search first round:");

scanf\_s("%u", &dwSearchValue);//输入修改的进程内数据的当前值

//首轮查找 待修改进程 查找目标值 存放数组 找到目标计数 空间

FirstRound(hProcess, dwSearchValue, dwAddrList, &dwAddrCount, 4 \* KONEK);

ShowAddrList(dwAddrList, dwAddrCount);//显示搜索结果（目标数为）：

if (dwAddrCount == 0) //无目标：返回

{

return;

}

else if (dwAddrCount == 1) //一个目标：直接修改

{

DWORD value;

printf("input the value which you want to set");

scanf\_s("%u", &value); //修改唯一值

//改写内存数据

bRet = WriteProcessMemory(hProcess, (LPVOID)dwAddrList[0], (LPVOID)&value, sizeof(DWORD), NULL);

}

else //多个目标：二轮查找

{

DWORD dwSecondRoundSearchValue = 0;

DWORD dwTargetList[KONEK] = { 0 };

DWORD dwTargetCounter = 0;

scanf\_s("%u", &dwSecondRoundSearchValue); //二次查找（需对数据进行修改）

//二轮查找 进程 二轮查找目标值 原（1轮）地址 原（1轮）目标计数 新目标地址 新目标计数

SecondRound(hProcess, dwSecondRoundSearchValue, dwAddrList, dwAddrCount, dwTargetList, &dwTargetCounter);

ShowAddrList(dwTargetList, dwTargetCounter);

DWORD value;

scanf\_s("%u", &value); //想要改成的数值

for (DWORD i = 0; i < dwTargetCounter; i++)//对二轮查找的所有值进行修改

{

bRet = WriteProcessMemory(hProcess, (LPVOID)dwTargetList[i], (LPCVOID)&value, sizeof(DWORD), NULL);

}

}

}

CloseHandle(hProcess);

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

涉及API:

OpenProcess

WriteProcessMemory

（3）结束进程  （API：OpenProcess、TerminateProcess）

void KillProcess() //功能3 结束进程

{

BOOL bRet = FALSE; //进程结束是否成功标志

DWORD dwId = 0; //需结束的进程ID

cout << "Please input process id which you want to kill...\n";

while (!scanf\_s("%d", &dwId))

{

/\*fflush(stdin);\*/

rewind(stdin);

cout << "Please input process id which you want to kill";

}

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, dwId); //打开进程

if (hProcess != NULL)

{

bRet = TerminateProcess(hProcess, 0); //结束进程

}

CloseHandle(hProcess);

if (bRet)

{

cout << "kill process success\n";

}

else

{

cout << "kill process filed\n";

}

return;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

涉及的API:

OpenProcess

TerminateProcess

ReadProcessMemory：读取进程内存

WriteProcessMemory：写入进程内存

void FirstRound(HANDLE hProcess, DWORD dwValue, DWORD \*pAddrList, DWORD \*pAddrListCounter, const DWORD addrListMax) //首轮内存查找

{

DWORD dwBaseAddr = 64 \* KONEK; //查找首地址

DWORD dwPageCount = (2 \* KONEG - 64 \* KONEK \* 2) / KPAGE; //查找计数 用以确定合法范围

| | 。。。 | | |

// |前64k |用户程序区 |64k禁区 | 2G内核区 |

// |NULL指针区 | | | |

printf("%u pages\n", dwPageCount);

printf("Start searching...\n");

DWORD dwBeginAddr = dwBaseAddr;

for (; dwBaseAddr< 2 \* KONEG - 64 \* KONEK; dwBaseAddr += KPAGE)

{

if (!CompareOnePage(hProcess, dwBaseAddr, dwValue, pAddrList, pAddrListCounter, addrListMax))

{

return;

}

//计算进度百分比

DWORD page = (dwBaseAddr - dwBeginAddr) / KPAGE + 1;

printf("current is %u page\n", page);

double temp = ((double)page / dwPageCount) \* 100;

printf("-----------%%%f---------\n", temp);

}

printf("\nSearch finished...\n");

system("pause");

}

bool CompareOnePage(HANDLE hProcess, DWORD dwBaseAddr, DWORD dwValue, DWORD \*pAddrList, DWORD \*pAddrListCounter, const DWORD addrListMax) //对单页内数据进行遍历 比对

{

BYTE byPage[KPAGE] = { 0 };

if (!ReadProcessMemory(hProcess, (LPCVOID)dwBaseAddr, (LPVOID)byPage, KPAGE, NULL))

{

printf("Read Memory error!!!\n");

return true;

}

DWORD \*pdwPointer = NULL;

pdwPointer = (DWORD\*)byPage;

for (DWORD i = 0; i < KONEK; i++)

{

if (\*pAddrListCounter >= addrListMax)

{

printf("Too many data, can not save...\n");

return false;

}

if (pdwPointer[i] == dwValue)

{

pAddrList[\*pAddrListCounter] = dwBaseAddr + i\*sizeof(DWORD);

(\*pAddrListCounter)++;

}

}

return true;

}

void ShowAddrList(DWORD \*pDwAddrList, DWORD dwAddrListCount)

{

printf("\n--------------Address list begin---------------\n");

for (DWORD i = 0; i < dwAddrListCount; i++)

{

printf("%X\n", pDwAddrList[i]);

}

printf("\n--------------Address list end---------------\n");

}

void SecondRound(HANDLE hProcess, DWORD dwValue, DWORD \*pAddrList, DWORD dwAddrListCounter, DWORD \*pTargetList, DWORD \*pTargetCounter)

{

DWORD dwTemp = 0;

for (DWORD i = 0; i < dwAddrListCounter; i++)

{

if (ReadProcessMemory(hProcess, (LPVOID)pAddrList[i], &dwTemp, sizeof(dwTemp), NULL))

{

if (dwTemp == dwValue)

{

pTargetList[\*pTargetCounter] = pAddrList[i];

(\*pTargetCounter)++;

}

}

}

}

完整代码：

<https://github.com/zjq4688/Myproject.git>