개발자를 위한 윈도우 후킹 테크닉

저널 훅을 사용한 매크로 제작

매크로는 귀찮은 작업을 한번에 해결해 주는 프로그램이다. WH_JOURNALRECORD, WH_JOURNALPLAYBACK 훅을 사용해서 마우스, 키보드 입력 내용을 저장하고 재생 다시 재생시키는 방법에 대해서 살펴본다.

목차

목차 1
필자 소개 1
연재 가이드 1
연재 순서 2
필자 메모 2
Intro 2
저널 훅 3
WH_JOURNALRECORD 훅 5
WH_JOURNALPLAYBACK 훅 6
저널 훅 헬퍼 클래스 7
매크로 내용 저장하기 9
매크로 내용 플레이 하기 10
메시지 처리 시간 계산 12
저널 훅의 한계 13
매크로 제작에 많이 사용되는 API 13
도전 과제 17
참고자료 17

필자 소개

신영진 pop@jiniya.net, http://www.jiniya.net

지루한 while 문을 반복하는듯한 일상에서 빠져나가고 싶다는 생각으로 머릿속이 가득 찬 요즘이다. 재미있는 책 한 권, 조용한 음악, 그리고 커피 한 잔의 여유가 그립다.

연재 가이드

운영체제: 윈도우 2000/XP
개발도구: 마이크로소프트 비주얼 스튜디오 2003

기초지식: C/C++, Win32 프로그래밍
응용분야: 매크로 프로그램

연재 순서

2006. 05 키보드 모니터링 프로그램 만들기
2006. 06 마우스 훅을 통한 화면 캡쳐 프로그램 제작
2006. 07 메시지훅 이용한 Spy++ 흉내내기
2006. 08 SendMessage 후킹하기
2006. 09 Spy++ 클론 imSpy 제작하기
2006. 10 저널 훅을 사용한 매크로 제작
2006. 11 WH_SHELL 훅을 사용해 다른 프로세스 윈도우 서브클래싱 하기
2006. 12 WH_DEBUG 훅을 이용한 훅 탐지 방법
2007. 01 OutputDebugString 의 동작 원리

필자 메모

개발자로서의 스킬을 업그레이드 하는 데에는 여러 가지 방법이 있다. 필자가 생각하는 가장 효율적인 방법은 커뮤니티 활동을 하는 것이다. 커뮤니티 활동은 자신의 실력을 알릴 수 있는 기회인 동시에 자신의 실력을 한 단계 업그레이드 할 수 있는 기회가 되기도 때문이다.

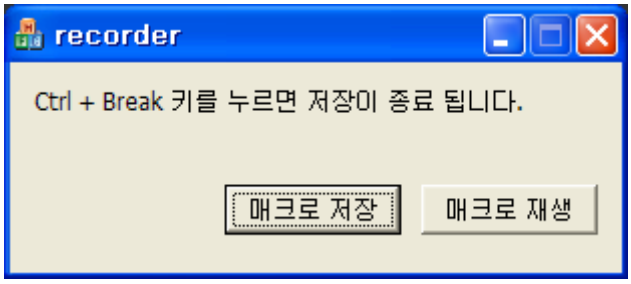
아직까지 어떤 개발자 커뮤니티 사이트가 있는지 잘 모른다면 아래 링크를 참고하면 된다. 아래 링크는 윈도우 프로그래밍과 관련된 괜찮은 커뮤니티 사이트 들이다. 특히 마지막에 있는 MS 커뮤니티 연합 링크로 들어가면 MS 공식 커뮤니티로 인정된 수많은 사이트를 찾을 수 있다.

데브피아(http://www.devpia.com)
고수닷넷(http://www.gosu.net)
디버그랩(http://www.debuglab.com)
MS VC++뉴스그룹(http://groups.google.co.kr/group/microsoft.public.kr.vc.qna)
MS 커뮤니티 연합(http://www.microsoft.com/korea/communities/related/default.mspx)

Intro

매크로란 번거로운 일련의 작업을 한번에 처리해 주는 프로그램을 말한다. 매크로만큼이나 일반인과 프로그래머 사이에 오해가 있는 단어도 없을 것 같다. 일반인들의 경우

매크로하면 가장 먼저 게임을 떠올린다. 자동 사냥이나 오토 프로그램이 그러한 것에 속한다. 프로그래머라면 아마도 C 언어나 어셈블러에서 사용되는 매크로 기능을 떠올릴 것이다. 반복되는 부분을 한번에 치환해서 처리해 주는 기능을 하는 것이다. 어쨌든 두 프로그램 모두 귀찮은 작업을 한번에 처리해 준다는 점에서는 비슷하다 할 수 있겠다.



화면 1 recorder 프로그램 시작 화면

우리가 이번 시간에 작성해 볼 프로그램은 저널 훅을 사용해서 사용자의 입력을 저장해 두었다 그대로 재생시켜 주는 매크로 프로그램이다. 시작하면 <화면 1>과 같은 대화 상자가 나타난다. 매크로 저장 버튼을 누르면 키보드 마우스의 입력 메시지를 기록하고, 매크로 재생 버튼을 누르면 저장한 입력 메시지를 재생한다. 매크로 저장을 종료하기 위해서는 Ctrl+Break 키를 누르면 된다. 한 가지 매크로만 기록하기 때문에 매크로 저장 버튼을 눌러서 새로운 매크로를 저장하면 이전 내용은 사라진다.

저널 훅

저널 훅은 매크로 제작에 널리 사용되는 훅이다. WH_JOURNALRECORD 훅은 사용자의 입력을 저장시키는 역할을 한다. WH_JOURNALPLAYBACK 훅은 이렇게 저장된 메시지 정보를 다시 재생하는 역할을 한다.

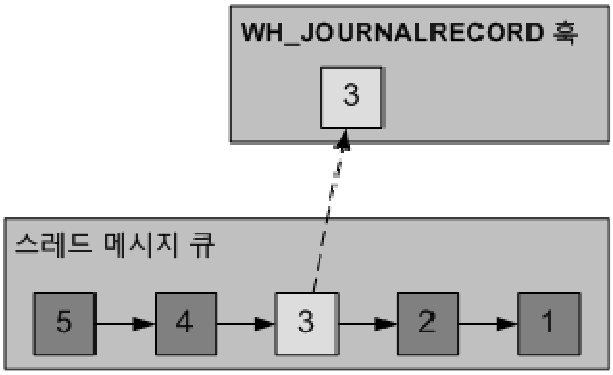


그림 1 WH_JOURNALRECORD 훅의 동작 원리

<그림 1>은 시스템에서 메시지가 1,2,3,4,5 의 순서대로 발생했을 때, 3 의 위치에서 WH_JOURNALRECORD 훅이 설치된 상황을 보여준다. 이렇게 되면 시스템은 3 번 메시지부터 WH_JOURNALRECORD 훅이 제거될 때까지 스레드 메시지 큐를 통해서 처리되는 입력 메시지를 훅 프로시저로 전달해 준다. 훅 프로시저에서는 이렇게 전달 받은 메시지를 메모리 버퍼나 파일에 보관해 두면 된다.

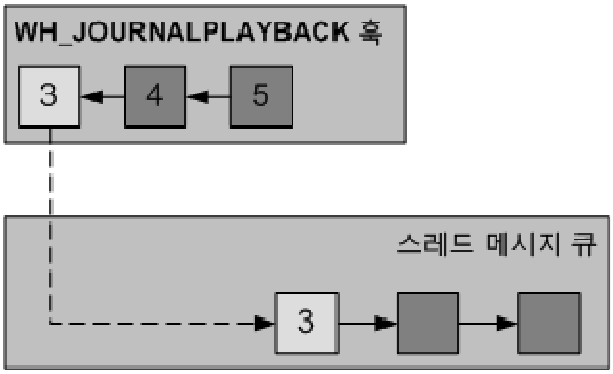


그림 2 WH_JOURNALPLAYBACK 훅의 동작 원리

<그림 2>는 시스템에서 저장된 메시지를 WH_JOURNALPLAYBACK 훅을 통해서 재생 시키는 과정을 나타내고 있다. WH_JOURNALRECORD 훅을 통해서 저장해둔 3,4,5 번 메시지를 훅 프로시저에서 스레드 메시지 큐로 차례대로 공급하면 된다.

재생할 때 한 가지 주의해야 할 점은 메시지 사이의 처리 간격이다. 시스템이 직접 처리 시간을 계산하지 않기 때문에 개발자가 메시지 사이의 간격을 계산해서 시스템에 알려주어야 한다. 이런 방식을 사용함으로써 재생 속도를 임의로 조절할 수 있다는 장점이 있다.

저널 훅의 경우 이전에 살펴보았던 WH_KEYBOARD_LL 훅과 마찬가지로 전역 훅으로만 사용할 수 있다. 한가지 특징적인 점은 전역 훅임에도 훅을 설치한 스레드 컨텍스트에서 훅 프로시저가 수행되기 때문에 별도의 DLL 에 훅 프로시저를 둘 필요가 없다는 점이다. 그래서 앞서 살펴 보았던 recorder 프로그램도 훅 DLL 없이 exe 내부에 모든 기능을 가지고 있다.

박스 1 전역 훅과 스레드 훅

메시지 훅은 동작 방식에 따라서 크게 전역 훅(global hook)과 스레드 훅으로 나눌 수 있다. 전역 훅은 시스템에 존재하는 모든 스레드에 대해서 후킹을 하는 것이고, 스레드 훅은 지정된 특정 스레드에 대해서만 후킹을 한다. 대부분의 훅이 두 가지 방식으로 모두

사용할 수 있으나, WH_KEYBOARD_LL, WH_MOUSE_LL, WH_JOURNALPLAYBACK, WH_JOURNALRECORD, WH_SYSMSGFILTER 훅은 반드시 전역 훅으로만 사용해야 한다.

WH_JOURNALRECORD 훅

WH_JOURNALRECORD 훅 프로시저의 인자와 리턴 값에 대한 내용을 간단하게 살펴보도록 하자. 아래는 WH_JOURNALRECORD 훅 프로시저의 원형이다.

```
LRESULT CALLBACK JournalRecordProc(int code, WPARAM wParam, LPARAM lParam);
```

code – [입력] 메시지를 어떻게 처리해야 하는지를 나타내는 코드 값이다(<표 1> 참고). 0 보다 작은 경우에는 훅 프로시저를 수행하지 않고 CallNextHookEx 를 호출한 후 바로 리턴 해야 한다.

표 1 code 값의 의미

코드 값	의미
HC_ACTION	lParam 에 현재 레코딩될 메시지 정보를 가지고 있는 EVENTMSG 구조체의 포인터를 가지고 있다. 훅 프로시저는 나중에 이 메시지를 재생하기 위해서 이 정보를 메모리 버퍼나 파일등에 저장해 두어야 한다.
HC_SYSMODALOFF HC_SYSMODALON	win16 과의 호환을 위해서 존재하는 플래그다. 무시해도 된다.

wParam – 사용되지 않는다.

lParam - [입력] 레코딩될 메시지 정보를 담고 있는 EVENTMSG 구조체의 포인터다(<표 2> 참고).

```
typedef struct {
    UINT message;
    UINT paramL;
    UINT paramH;
    DWORD time;
    HWND hwnd;
} EVENTMSG, *PEVENTMSG;
```

표 2 EVENTMSG 구조체 필드별 의미

필드명	의미
message	메시지 ID

paramL	메시지와 관련된 정보 (메시지에 따라 다름)
paramH	메시지와 관련된 정보 (메시지에 따라 다름)
time	메시지가 포스트된 시간
hwnd	메시지가 발생한 윈도우 핸들

리턴 값: 사용되지 않는다.

WH_JOURNALPLAYBACK 훅

WH_JOURNALPLAYBACK 훅은 훅 프로시저가 복잡하다. code 값에 따라서 해야 할 동작과 리턴 값이 다르기 때문이다. 아래는 WH_JOURNAPLAYBACK 훅 프로시저의 원형이다.

```
LRESULT CALLBACK JournalPlaybackProc(int code, WPARAM wParam, LPARAM lParam);
```

code – [입력] 메시지를 어떻게 처리해야 하는지를 나타내는 코드 값이다(<표 3>을 참고). 0 보다 작은 경우에는 훅 프로시저를 수행하지 않고 CallNextHookEx 를 호출한 후 바로 리턴 해야 한다.

표 3 code 값의 의미

코드 값	의미
HC_GETNEXT	훅 프로시저는 현재 플레이될 메시지 정보를 lParam 이 가리키는 EVENTMSG 구조체에 복사해야 한다.
HC_NOREMOVE	메시지가 메시지 큐에서 제거되지 않고 참조된 경우다(PeekMessage 를 PM_NOREMOVE 플래그를 설정해서 호출한 경우).
HC_SKIP	플레이될 메시지 정보를 다음 메시지로 이동 시킨다.
HC_SYSMODALOFF HC_SYSMODALON	win16 과의 호환을 위해서 존재하는 플래그다. 무시해도 된다.

wParam – 사용되지 않는다.
lParam – [출력] 메시지 정보를 담고 있는 EVENTMSG 구조체의 포인터다(<표 2> 참고). code 값이 HC_GETNEXT 인 경우만 유효하다.

리턴 값: 시스템이 메시지를 처리하기 위해서 대기해야 하는 시간을(클럭 틱 단위) 리턴 한다. 지금 즉시 처리되어야 한다면 0 을 리턴 한다. code 값이 HC_GETNEXT 가 아닌 경우의 리턴 값은 무시 된다.

저널 훅 헬퍼 클래스

저널 훅은 굉장히 특이한 종류의 훅 이기 때문에 대부분의 경우에 원하는 자료 구조가 한정되어 있다. 메시지를 저장할 배열과 후킹 시작 타임을 저장할 변수가 그것이다. 이러한 것들을 자동적으로 관리해 주는 헬퍼 클래스를 만들어두면 나중에 다시 사용하기도 편리하다.

<리스트 1>에는 이러한 저널 훅의 작업을 도와주는 헬퍼 클래스인 CJournalHook 의 전체 구조가 나와있다. 멤버 변수와 함수의 기능은 주석에 나와있다. 핵심 함수인 StartPlaying, StopPlaying 함수는 <리스트 2>와 <리스트 3>에 나와 있다.

WH_JOURNALPLAYBACK 훅을 설치하기 전에 m_msgOffset 과 m_playTime 을 초기화 해주는 부분에 주의해야 한다(<리스트 2> 참고). 마찬가지로 WH_JORNALRECORD 훅을 설치하기 전에는 m_msgs 벡터를 비우고, m_recordTime 을 초기화 해야 한다.

<리스트 3>에서 주의해서 봐야 할 부분은 UnhookWindowsHookEx 를 호출하는 부분이다. 소스를 살펴보면 ERROR_INVALID_HOOK_HANDLE 로 실패한 경우에는 정상적으로 함수가 동작한 것으로 처리해 주는 것을 볼 수 있다. 이렇게 하는 이유는 저널 훅의 특징 때문이다. 저널 훅의 경우 사용자가 Ctrl+Esc 나 Ctrl+Alt+Del 을 입력하는 경우에는 시스템이 강제적으로 훅을 제거한 다음 응용 프로그램에 WM_CANCELJOURNAL 메시지를 날려준다. 이 경우에 실제 훅의 상황과 내부 변수를 동기화 하기 위해서는 UnhookWindowsHookEx 를 호출하지 않는 동일한 함수를 만들거나 소스에 나타난 것과 같이 GetLastError 를 통해서 그 경우만 정상 처리해 줄 필요가 있다.

리스트 1 CJournalHook 클래스 구조

```
class CJournalHook
{
public:
   typedef std::vector<EVENTMSG> EventMsgVec;

   EventMsgVec m_msgs;        // 메시지를 저장할 벡터
   size_t      m_msgOffset;   // 현재 플레이할 메시지 번호

   HHOOK m_recordHook;        // 레코드 훅 핸들
   HHOOK m_playHook;          // 플레이 훅 핸들
   DWORD m_recordTime;        // 레코드 시작 시간
   DWORD m_playTime;          // 플레이 시작 시간

   CrecorderDlg *m_pDlg;      // UI 동기화를 위한 다이알로그 핸들
```

```
private:
    CJournalHook();

    CJournalHook(const CJournalHook &);
    CJournalHook &operator =(const CJournalHook &);

public:
    ~CJournalHook();

    BOOL StopRecording();   // 레코딩 시작
    BOOL StopPlaying();     // 플레이 시작
    BOOL StartRecording();  // 레코딩 종료
    BOOL StartPlaying();    // 플레이 종료

    friend CJournalHook &JournalHook();
};
```

리스트 2 WH_JOURNALPLAYBACK 훅을 설치하는 코드

```
BOOL CJournalHook::StartPlaying()
{
    if(m_playHook)
        return FALSE;

    m_playTime = GetTickCount();
    m_msgOffset = 0;

    m_playHook = SetWindowsHookEx(  WH_JOURNALPLAYBACK,
                                PlayProc,
                                GetModuleHandle(NULL),
                                0 );
    if(!m_playHook)
        return FALSE;

    if(m_pDlg)
        m_pDlg->StartPlaying();

    return TRUE;
}
```

리스트 3 WH_JOURNALPLAYBACK 훅을 제거하는 코드

```
BOOL CJournalHook::StopPlaying()
{
    if(m_playHook)
    {
        if(!UnhookWindowsHookEx(m_playHook))
        {
            if(GetLastError() != ERROR_INVALID_HOOK_HANDLE)
                return FALSE;
        }

        m_playHook = NULL;
```

```
        if(m_pDlg)
            m_pDlg->StopPlaying();

        return TRUE;
    }

    return FALSE;
}
```

매크로 내용 저장하기

매크로의 내용을 저장하는 WH_JOURNALRECORD 의 훅 프로시저가 <리스트 4>에 나와 있다. 코드는 전체적으로 이제껏 작성해왔던 훅 프로시저와 유사하다.
WH_JOURNALRECORD 훅 프로시저를 작성할 때에는 훅 종료 처리와 메시지 시간 처리에 주의해야 한다.

표준적으로 저널 레코딩을 종료하는 방법은론 세 가지가 있다. 첫째는 자발적으로 사용자에 의해서 종료를 나타내는 VK_CANCEL 이다. VK_CANCEL 키는 일반적으로 Ctrl+Break 키의 조합을 나타낸다. 따라서 훅 프로시저에서는 VK_CANCEL 키의 상태를 체크해서 눌려진 경우에는 훅을 해제할 필요가 있다. 다음으로 시스템에서 강제적으로 훅을 취소하는 경우다. 사용자가 Ctrl+Esc 를 누르거나 Ctrl+Alt+Del 을 누른 경우가 그에 해당한다. 이 키를 누르면 Windows 시스템은 훅을 제거한 다음 NULL 윈도우로 WM_CANCELJOURNAL 메시지를 보내준다. 이 경우는 이미 Window 시스템이 훅을 제거한 상태이므로 훅을 다시 제거할 필요는 없다. 단지 내부적으로 훅에 사용된 핸들과 상태를 갱신해서 실제 상황과 맞춰줄 필요가 있다. 이 부분에 대한 처리는 <리스트 5>에 나와있다.

LPARAM 으로 넘어온 EVENTMSG 구조체의 time 값을 계산하는 부분 또한 주의해야 한다. EVENTMSG 의 time 값은 메시지가 포스팅된 시점의 클럭 틱을 저장하고 있다. 이 값을 그대로 저장할 경우엔 나중에 플레이할 때 메시지 사이의 시간 간격을 다시 계산해야 한다. 이러한 번거로움을 없애기 위해서는 메시지를 저장할 당시에 저장 시작 시간으로부터 경과된 상대 시간을 저장하면 된다.

리스트 4 WH_JOURNALRECORD 훅 프로시저

```
LRESULT CALLBACK RecordProc(int code, WPARAM w, LPARAM l)
{
    if(code < 0)
        return CallNextHookEx(NULL, code, w, l);

    SHORT pauseState = GetAsyncKeyState(VK_CANCEL);
    if(pauseState & 0x8000)
```

```
    {
        JournalHook().StopRecording();
        return 0;
    }

    if(code == HC_ACTION)
    {
        PEVENTMSG msg = (PEVENTMSG) l;

        CJournalHook &hook = JournalHook();

        hook.m_msgs.push_back(*msg);
        hook.m_msgs.back().time = msg->time - hook.m_recordTime;
    }

    return 0;
}
```

리스트 5 WM_CANCELJOURNAL 메시지 핸들러

```
BOOL CrecorderDlg::PreTranslateMessage(MSG* pMsg)
{
    if(pMsg->message == WM_CANCELJOURNAL)
    {
        JournalHook().StopPlaying();
        JournalHook().StopRecording();
        return TRUE;
    }

    return CDialog::PreTranslateMessage(pMsg);
}
```

매크로 내용 플레이 하기

앞서 저장한 메시지 내용을 재생 시키는 WH_JOURNALPLAYBACK 훅 프로시저가 <리스트 6>에 나와 있다. code 값이 HC_GETNEXT 인 경우에는 현재 가리키고 있는 메시지를 LPARAM 이 가리키고 있는 EVENTMSG 구조체에 복사하고, HC_SKIP 인 경우는 메시지를 다음으로 이동 시킨다.

한 가지 주의해야 할 점은 code 값이 HC_GETNEXT 인 경우의 리턴 값이다. 이 경우에는 복사한 메시지가 실제로 처리되기까지 시스템이 대기해야 하는 시간을 리턴 해 주어야 한다. 만약 이 시간 계산이 정확하지 않으면 메시지 재생이 엉뚱하게 될 수 있기 때문에 주의해야 한다. 특히 이 시간은 결국에는 0 을 리턴 해야 한다. 만약 0 을 리턴 하지 않으면 시스템은 절대 HC_SKIP 을 호출하지 않는다.

<리스트 6>에서 시간을 구하는 부분은 delta 값을 계산하는 곳이다. EVENTMSG 의 time 값은 시작 시간으로부터의 상대 값이기 때문에 절대 값으로 고치기 위해서는 역으로 재생 시작 시간의 틱을 더해 주어야 한다. 그리고 이 메시지가 처리되기 까지 남은 시간은 해당 틱에서 현재 틱 값을 뺀 값이다. 이 값이 0 보다 작다면 0 으로 변경해서 바로 처리되도록 해주어야 한다.

리스트 6 WH_JOURNALPLAYBACK 훅 프로시저

```
LRESULT CALLBACK PlayProc(int code, WPARAM w, LPARAM l)
{
   if(code < 0)
      return CallNextHookEx(NULL, code, w, l);

   int delta;
   PEVENTMSG msg;
   CJournalHook &hook = JournalHook();

   switch(code)
   {
   case HC_GETNEXT:
      if(hook.m_msgOffset >= hook.m_msgs.size())
      {
         hook.StopPlaying();
         return 0;
      }

      msg = (PEVENTMSG) l;
      *msg = hook.m_msgs[hook.m_msgOffset];
      msg->time += hook.m_playTime;

      delta = msg->time - GetTickCount();
      if(delta < 0)
         delta = 0;
      return delta;

   case HC_SKIP:
      ++hook.m_msgOffset;
      break;

   default:
      break;
   }

   return 0;
}
```

메시지 처리 시간 계산

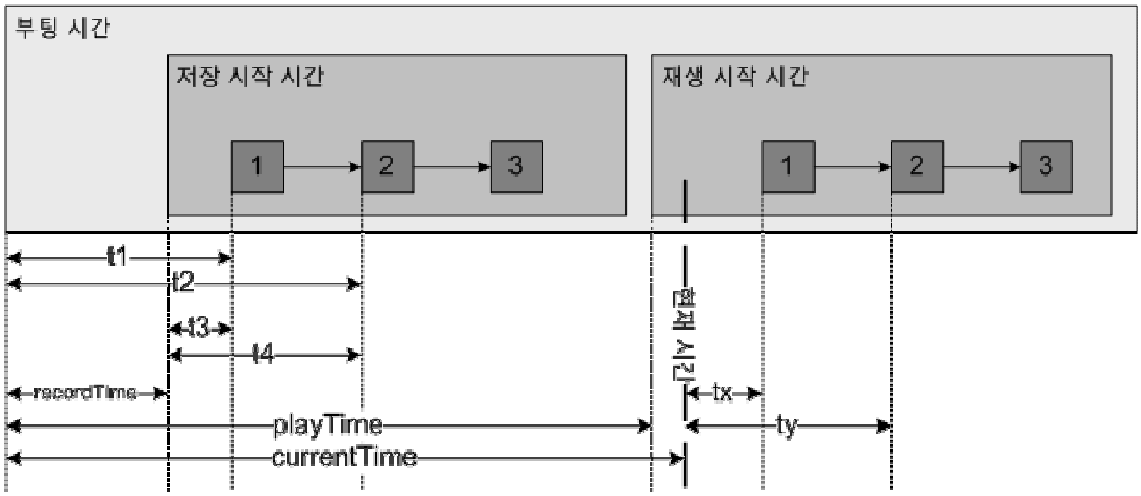


그림 3 메시지 시간 순서도

<그림 3>에 메시지에 사용되는 각종 시간의 관계가 나와있다. WH_JOURNALRECORD 훅에서 EVENTMSG 구조체의 time 값은 t1, t2 같은 값이 들어 있다. 이는 부팅 시간으로부터 경과된 클럭 값이다. 우리가 WH_JOURNALPLAYBACK 훅 프로시저에서 최종적으로 계산해야 하는 값은 tx, ty 값들이다.

상식적으로 생각했을 때 추가적인 작업 없이 단순히 t1, t2 로부터 tx, ty 를 계산해낼 방법은 없다. 그래서 그냥 저장한 경우에는 1 번 메시지는 0 을 리턴 해서 바로 처리하고 다음 메시지부터는 이전 메시지와의 시간 차이(t2 – t1)을 계산해서 리턴 하는 방식을 사용해야 한다. 또한 이 경우에는 별도로 HC_SKIP 과의 연동을 위해서도 추가적인 작업이 필요하다. 왜냐하면 0 아니면 t2 – t1 을 리턴 해야 하기 때문이다(t2-t1 만 리턴 하면 다음 메시지로 넘어가지 않는다).

우리는 이러한 문제를 해결하기 위해서 t1, t2 값을 직접 사용하지 않고 recordTime 으로부터의 상대 값인 t3, t4 를 저장했다. 이렇게 되면 나중에 재생할 때 tx 는 t3 + playtime – currentTime 이 된다. 이 경우 과정은 복잡하지만 최종 결과값을 정확하고 쉽게 계산할 수 있다는 장점이 있다. 또한 별도의 if 문을 추가하지 않아도 된다.

tx 와 ty 값을 조작함으로써 재생 속도를 조절할 수 있다. tx, ty 를 일정 비율로 늘리면 재생 속도가 느려지고, 줄이면 빨라진다.

저널 훅의 한계

저널 훅은 굉장히 강력하지만 몇 가지 단점을 가지고 있다. 일단 가장 큰 단점은 모든 메시지 훅이 그렇듯이 윈도우 프로그램에 대해서만 동작한다는 점이다. 윈도우 창이 없는 콘솔 프로그램에 대해서는 동작하지 않는다.

다른 또 하나의 단점으로는 매크로 입력 내용을 사용자가 손쉽게 편집할 수 없다는 점이다. 말 그대로 입력 메시지를 그대로 가져와서 저장한 것인데다, 시간 개념까지 포함되어 있기 때문에 편집하기가 상당히 어렵다.

그리고 큰 단점은 한 컴퓨터에서 입력한 매크로가 다른 컴퓨터에서는 동일한 작업을 보장하지 않는다는 점이다. 물론 모든 매크로가 환경에 따라 어느 정도 바뀔 수 있지만 저널 훅을 사용한 매크로는 그 영향이 더 크다. 특히 해상도가 다른 경우에는 거의 100% 엉뚱한 동작이 벌어지고 만다.

매크로 제작에 많이 사용되는 API

앞서 제시한 저널 훅의 단점으로 대부분의 상용 매크로 프로그램은 저널 훅 보다는 사용자의 입력을 발생 시키는 API 를 통해서 사용자가 지정해둔 키 입력이나 마우스 입력을 재생 시키는 방법을 사용한다. 이러한 매크로 프로그램에는 아래와 같은 API 가 많이 사용된다.

```
VOID keybd_event(BYTE bVk, BYTE bScan, DWORD dwFlags, PTR dwExtraInfo);
```

keybd_event 함수의 원형이다. 이 함수는 인자로 전달된 내용대로 키보드 메시지를 발생시켜 주는 역할을 한다. 리턴 값은 없으며 사용되는 인자의 역할은 아래와 같다.

bVk – [입력] 메시지를 발생 시킬 키에 대한 가상 키 코드
bScan – 사용되지 않음
dwFlags – [입력] 발생시킬 메시지에 대한 플래그를 나타낸다. 두 가지 값을 조합해서 사용할 수 있다(<표 4> 참고). 일반적인 키가 눌린 경우에 대한 메시지를 발생시키고 싶다면 0 을 넣으면 된다.

표 4 dwFlags 값의 의미

값	의미
KEYEVENTF_EXTENDEDKEY	메시지를 발생 시킬 키가 확장 키인 경우에 조합해 준다.
KEYEVENTF_KEYUP	키를 뗀 경우에 해당하는 메시지를 발생시킬 때 조합해 준다.

dwExtraInfo – [입력] 키보드 메시지와 관련된 추가적인 정보를 넣는 곳이다. 보통의 경우 0 을 넣으면 된다.

keybd_event 를 통해서 Numlock 키를 누르는 메시지를 발생시키는 코드가 <리스트 7>에 나와 있다. VK_NUMLOCK 의 경우 확장 키이기 때문에 KEYEVENTF_EXTENDEDKEY 를 조합해 준다. Shift+A 와 같은 키 조합을 발생 시키고자 한다면 Shift, A, A(Up), Shift(Up)과 같이 순차적으로 메시지를 발생시켜 주면 된다.

리스트 7 keybd_event 를 통해서 Numlock 키를 누르는 코드

```
keybd_event( VK_NUMLOCK,
             0,
             KEYEVENTF_EXTENDEDKEY,
             0 );

 keybd_event( VK_NUMLOCK,
             0,
             KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP,
             0);
```

```
VOID mouse_event(
    DWORD dwFlags,
    DWORD dx,
    DWORD dy,
    DWORD dwData,
    ULONG_PTR dwExtraInfo
);
```

mouse_event 함수는 마우스 메시지를 발생시켜 주는 함수다. 리턴 값은 없으며 인자의 역할은 아래와 같다.

dwFlags – [입력] 발생시킬 마우스 메시지의 플래그를 나타낸다(<표 5> 참고).

표 5 dwFlags 값의 의미

dwFlags 값	의미
MOUSEEVENTF_ABSOLUTE	dx, dy 값이 절대 좌표를 의미하는지 상대 좌표를 의미하는지를 나타낸다.
MOUSEEVENTF_MOVE	WM_MOUSEMOVE 메시지 발생.
MOUSEEVENTF_LEFTDOWN	WM_LBUTTONDOWN 메시지 발생.
MOUSEEVENTF_LEFTUP	WM_LBUTTONUP 메시지 발생.

MOUSEEVENTF_RIGHTDOWN	WM_RBUTTONDOWN 메시지 발생.
MOUSEEVENTF_RIGHTUP	WM_RBUTTONUP 메시지 발생.
MOUSEEVENTF_MIDDLEDOWN	WM_MBUTTONDOWN 메시지 발생.
MOUSEEVENTF_MIDDLEUP	WM_MBUTTONUP 메시지 발생.
MOUSEEVENTF_WHEEL	WM_MOUSEWHEEL 메시지 발생.
MOUSEEVENTF_XDOWN	WM_XBUTTONDOWN 메시지 발생.
MOUSEEVENTF_XUP	WM_XBUTTONUP 메시지 발생.

dx – [입력] 마우스 메시지가 발생한 x 축 좌표다. MOUSEEVENTF_ABSOLUTE 가 설정된 경우는 화면상의 절대 좌표 값을 가지며, 설정되지 않은 경우는 마지막 메시지 발생 위치로부터 상대 좌표 값으로 해석된다.
dy – [입력] 마우스 메시지가 발생한 y 축 좌표다. MOUSEEVENTF_ABSOLUTE 가 설정된 경우는 화면상의 절대 좌표 값을 가지며, 설정되지 않은 경우는 마지막 메시지 발생 위치로부터 상대 좌표 값으로 해석된다.
dwData – [입력] WM_WHEEL 메시지를 발생시키는 경우에는 휠의 이동량을 여기다 전달해 준다. WM_XBUTTONDOWN, WM_XBUTTONUP 메시지를 발생시키는 경우에는 어떤 버튼에 대한 메시지인지를 전달해 준다. XBUTTON1 과 XBUTTON2 가 사용될 수 있다. 그 외의 경우에는 0 을 전달한다.
dwExtraInfo – [입력] 메시지와 관련된 추가적인 정보를 넣는 곳이다. 보통의 경우 0 을 전달하면 된다.

<리스트 8>에 mouse_event 를 사용해서 마우스를 이동 시키는 코드가 나와있다. 한가지 주의해야 할 점은 MOUSEEVENTF_ABSOLUTE 를 지정한 경우에 마우스 좌표 값은 화면 해상도의 픽셀 값이 아니라, 0-65536 사이의 값으로 맵핑된 좌표계라는 것이다. 화면 좌측 끝은 (0,0)이 되고, 우측 하단은 (65536, 65536)이 된다. 또한 상대 좌표를 사용할 때에는 제어판에 설정된 마우스의 속도에 영향을 받는다는 점도 알아 두어야 한다.

리스트 8 mouse_event 를 사용해 마우스를 이동시키는 코드

```
mouse_event(MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_MOVE, 200, 200, 0, 0);
Sleep(1000);
mouse_event(MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_MOVE, 1000, 1000, 0, 0);
Sleep(1000);
mouse_event(MOUSEEVENTF_ABSOLUTE| MOUSEEVENTF_MOVE, 2000, 2000, 0, 0);
```

```
UINT SendInput(UINT nInputs, LPINPUT pInputs, int cbSize);
```

SendInput 함수는 위의 두 함수를 통합한 함수다. 키보드와 마우스의 입력 메시지를 발생시켜 주는 함수다. 위에서 살펴본 함수와 비슷한 정보를 구조체에 전달해 주면 되기 때문에 간단하게 인자의 역할만 살펴본다. 자세한 정보는 MSDN 을 참고하도록 하자.

nInputs – [입력] 발생시킬 입력 메시지의 개수를 넣어준다.
pInputs – [입력] 발생시킬 메시지 구조체의 정보를 넣어준다. 여기에 사용되는 INPUT 구조체의 형태는 아래와 같다. 구조체의 각 멤버에 대한 의미는 <표 6>에 나와있다.

```
typedef struct tagINPUT {
  DWORD type;
  union {   MOUSEINPUT mi;
            KEYBDINPUT ki;
            HARDWAREINPUT hi;
        };
}INPUT, *PINPUT;
```

표 6 INPUT 구조체 필드별 의미

필드명	의미
type	INPUT_MOUSE: 마우스 메시지를 발생시킨다(mi 사용). INPUT_KEYBOARD: 키보드 메시지를 발생시킨다(ki 사용). INPUT_HARDWARE: 마우스와 키보드를 제외한 하드웨어의 입력 메시지를 발생 시킨다(hi 사용).
mi	마우스 입력 메시지를 나타낸다.
ki	키보드 입력 메시지를 나타낸다.
hi	마우스와 키보드를 제외한 하드웨어의 입력 메시지를 나타낸다(Win95/98/ME 만 적용됨).

cbSize – [입력] INPUT 구조체의 크기를 넣어준다.

박스 2 매크로가 게임에 동작하지 않는 이유

대부분의 상용 온라인 게임은 해킹 차단 기능이 포함되어 있다. 이러한 해킹 차단 프로그램은 메모리 위 변조 해킹 및 매크로 프로그램을 검출하고 차단하는 기능을 한다. 따라서 위와 같은 API 를 통해서 게임 매크로를 작성하더라도 게임에서는 동작하지 않는 경우가 많다.

도전 과제

이번 달 도전 과제도 샘플 프로그램을 강화하는 것이다. 아래와 같은 기능들을 구현해 보도록 하자. 참고자료에 있는 winmacro 에 대부분의 기능이 구현되어 있다.

다수의 매크로 저장 및 재생 기능
매크로 내용 파일에 저장 및 불러오기 기능
매크로 재생 속도 조절 기능

참고자료

참고자료 1. Jeffrey Richter. <<Programming Applications for Microsoft Windows (4/E)>> Microsoft Press
참고자료 2. 김상형, <<Windows API 정복>> 가남사
참고자료 3. 김성우, <<해킹/파괴의 광학>> 와이미디어
참고자료 4. WinMacro 소스 http://www.geocities.com/win_macro/