2013년도 1학기

8장 파일 입출력

- 학습목표
  - 표준 입출력이 아닌 API에서 제공하는 파일 입출력을 배운다.
  - 공용 대화상자를 이용하여 파일 입출력을 배운다.

- 내용
  - 파일 다루기
  - 공용 대화상자

1절. 파일 다루기

- **API 이용한 표준 입출력 및 파일 사용 방법**
  - 파일을 만들고 열어준다. 열 때는 읽기용인지 쓰기용인지 명시
  - 열린 파일에는 텍스트를 쓰거나 읽는다.
  - 작업 후에는 파일을 닫는다.

- **파일 만들기/열기 함수**

```
HANDLE CreateFile(
    LPCTSTR lpFileName,                          // (1) 파일이름 (문자열)
    DWORD dwDesiredAccess,                       // (2) 읽기/쓰기 모드
    DWORD dwShareMode,                           // (3) 파일 공유 모드
    LPSECURITY_ATTRIBUTES  lpSecurityAttributes, // (4) 보안속성 지정
    DWORD dwCreationDistribution,                // (5) 파일 생성 모드
    DWORD dwFlagsAndAttributes,                  // (6) 파일 속성
    HANDLE hTemplateFile                         // (7) NULL
);
```

속성값

- dwDesiredAccess: 읽기/쓰기 모드 (아래의 3 모드 중 1개 지정)
  - 읽기: GENERIC_READ
  - 쓰기: GENERIC_WRITE
  - 읽기 및 쓰기: GENERIC_READ| GENERIC_WRITE
- dwShareMode: 공유 모드 (파일 공유 여부 명시)
  - 읽기 공유허용: FILE_SHARE_READ
  - 쓰기 공유허용: FILE_SHARE_WRITE
- lpSecurityAttributes: 보안 속성 (자녀 프로세스에 상속 여부 설정), NULL 이면 상속 안됨
- dwCreationDistribution:파일 생성 모드
  - 만들기 모드: CREATE_NEW
  - 강제 만들기 모드: CREATE_ALWAYS (파일이 있어도 파괴하고 새로 만듦)
  - 기존파일 열기 모드: OPEN_EXISTING
  - 강제 열기 모드: OPEN_ALWAYS
- dwFlagsAndAttributes: 파일 속성 (읽기 전용 파일, 시스템 파일, 숨겨진 파일 등 지정)
  - 일반적인 파일: FILE_ATTRIBUTE_NORMAL

파일 만들기/열기(예)

```
HANDLE hFile;
hFile = CreateFile ( "test.c",
          GENERIC_READ|GENERIC_WRITE,
          FILE_SHARE_READ|FILE_SHARE_WRITE, NULL,
          OPEN_EXISTING, 0, 0);
```

- 매개변수 설명
  - 기존에 있는 “test.c” 파일을 읽기/쓰기 겸용으로 열기
  - 다른 프로그램과 읽기/쓰기 공유가 가능
  - “test.c” 파일이 없으면 hFile은 INVALID_HANDLE_VALUE임

파일 읽기

```
BOOL ReadFile(
    HANDLE hFile,                           // (1) 읽을 파일의 핸들
    LPVOID lpBuffer,                        // (2) 읽은 자료를 넣을 버퍼
    DWORD nNumberOfBytesToRead,             // (3) 읽고자 하는 바이트 크기
    LPDWORD lpNumberOfBytesRead,            // (4) 실제 읽은 바이트
    LPOVERLAPPED lpOverlapped               // (5) NULL
    );
```

- 사용 예 : 문자열 읽기

```
char pBuff[100];
int size_max = 100;
int size_read;
ReadFile(hFile, pBuff, size_max, &size_read, NULL);
```

파일 쓰기 / 파일 닫기

```
BOOL WriteFile(
    HANDLE hFile,                       // (1) 쓰기 파일의 핸들
    LPVOID lpBuffer,                    // (2) 쓸 자료의 저장버퍼
    DWORD nNumberOfBytesToWrite,        // (3) 쓸 자료의 크기
    LPDWORD lpNumberOfBytesWritten,     // (4) 실제 써진 자료의 크기
    LPOVERLAPPED lpOverlapped           // (5) NULL
    );
```

- 사용 예 : 문자열 쓰기

```
char pBuff[100];
int size_max = 100;
int size_write;
WriteFile(hFile, pBuff, size_max, &size_write, NULL);
```

- CloseHandle (hFile);

8-1 파일 입출력

```
HANDLE hFile;
char InBuff[1000];
char OutBuff[1000] =  "\nAPI 파일 입출력 테스트입니다.";
DWORD size = 1000;
hFile = CreateFile ("test.txt",
          GENERIC_READ|GENERIC_WRITE,
          FILE_SHARE_READ|FILE_SHARE_WRITE, NULL,
          OPEN_EXISTING, 0, 0);
memset (InBuf, 0, 999*sizeof(char));
ReadFile (hFile, InBuff, size, &size, NULL);      // hFile에서 size 만큼 읽어 InBuff에 저장
InBuff[size] =  '\0' ;

Hdc = GetDC(hwnd);
TextOut(hdc, 0, 0, InBuff, strlen(InBuff));       // InBuff 에 있는 내용을 화면에 출력
ReleaseDC(hwnd, hdc);

WriteFile (hFile, OutBuff, strlen(OutBuff), &size, NULL);// OutBuff의 내용을 hFile에 저장
CloseHandle(hFile);
```

파일에서 Line 읽기

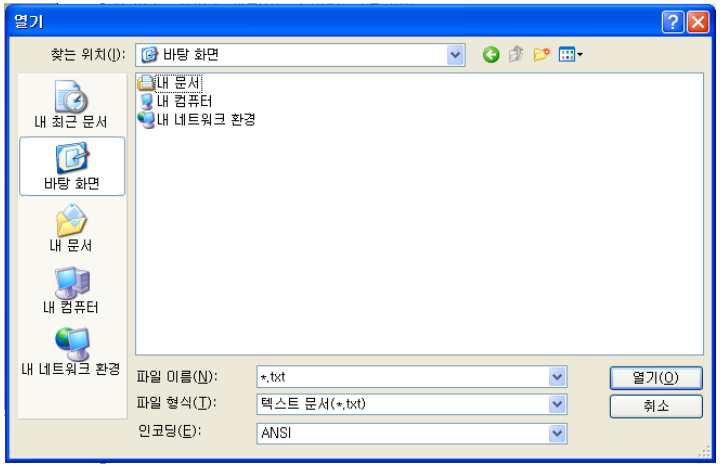
```
int inSize = ReadLine(hFile, lnBuff);

int ReadLine(HANDLE hFile, char lnBuff[])
{
    DWORD size;
    char buf[1];
    int i;

    i = 0;
    ReadFile(hFile, buf, size, &size, NULL);
    if(size == 0)
        return;  // 파일의 끝에 도달
    if(buf[0] !=  '\n' )
        lnBuff[i++] = buf[0];
    return(i);      //  '\n'  없이 반환
}
```

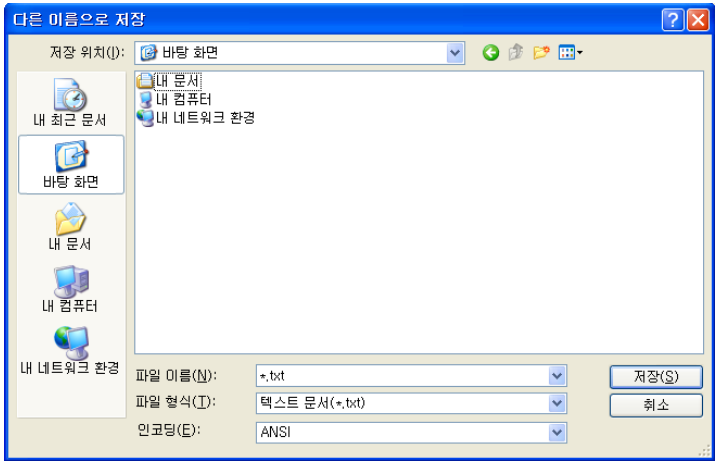
임의 접근

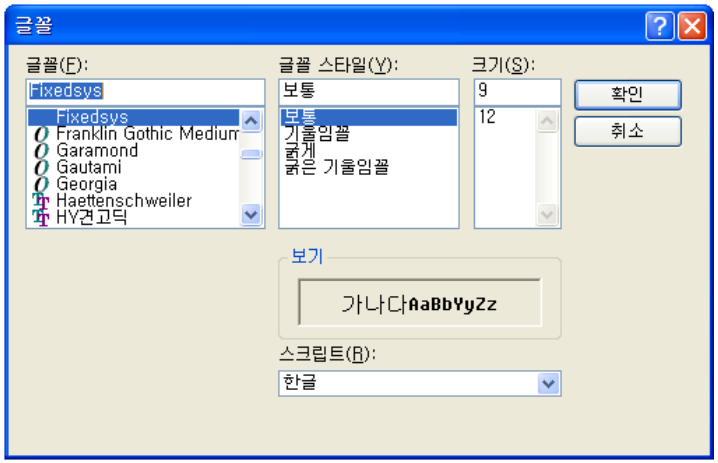
- 파일 엑세스할 때 대상 파일 위치 (File Pointer) 결정
  - 최초로 파일이 열렸을 때: FP는 파일의 선두 위치, 파일을 읽거나 쓰면 그만큼 파일 포인터가 이동 → 순차적 접근
  - 파일의 임의의 위치에서 원하는 만큼 읽는다. → 임의 접근
  - 임의 접근 함수:

```
DWORD SetFilePointer(
    HANDLE hFile,                       // 파일 핸들
    LONG lDistanceToMove,               // 파일 포인터 이동할 위치
    PLONG lpDistanceToMoveHigh,         // 파일 크기가 2GB 이상일 경우 옮길 위치
    DWORD dwMoveMethod                  // 파일 포인터의 이동 시작 위치 지정
                                        // FILE_BEGIN / FINE_CURRENT / FILE_END
);
```

2절. 공용대화상자


열기
찾는 위치(I):
바탕 화면
내 문서
내 컴퓨터
내 네트워크 환경
내 최근 문서
바탕 화면
내 문서
내 컴퓨터
내 네트워크 환경
파일 이름(N):
*.txt
파일 형식(T):
텍스트 문서(*.txt)
인코딩(E):
ANSI
열기(O)
취소


다른 이름으로 저장
저장 위치(I):
바탕 화면
내 문서
내 컴퓨터
내 네트워크 환경
내 최근 문서
바탕 화면
내 문서
내 컴퓨터
내 네트워크 환경
파일 이름(N):
*.txt
파일 형식(T):
텍스트 문서(*.txt)
인코딩(E):
ANSI
저장(S)
취소


글꼴
글꼴(F):
Fixedsys
Fixedsys
Franklin Gothic Medium
Garamond
Gautami
Georgia
Haettenschweiler
HY견고딕
글꼴 스타일(Y):
보통
보통
기울임꼴
굵게
굵은 기울임꼴
크기(S):
9
12
확인
취소
보기
가나다AaBbYyZz
스크립트(R):
한글

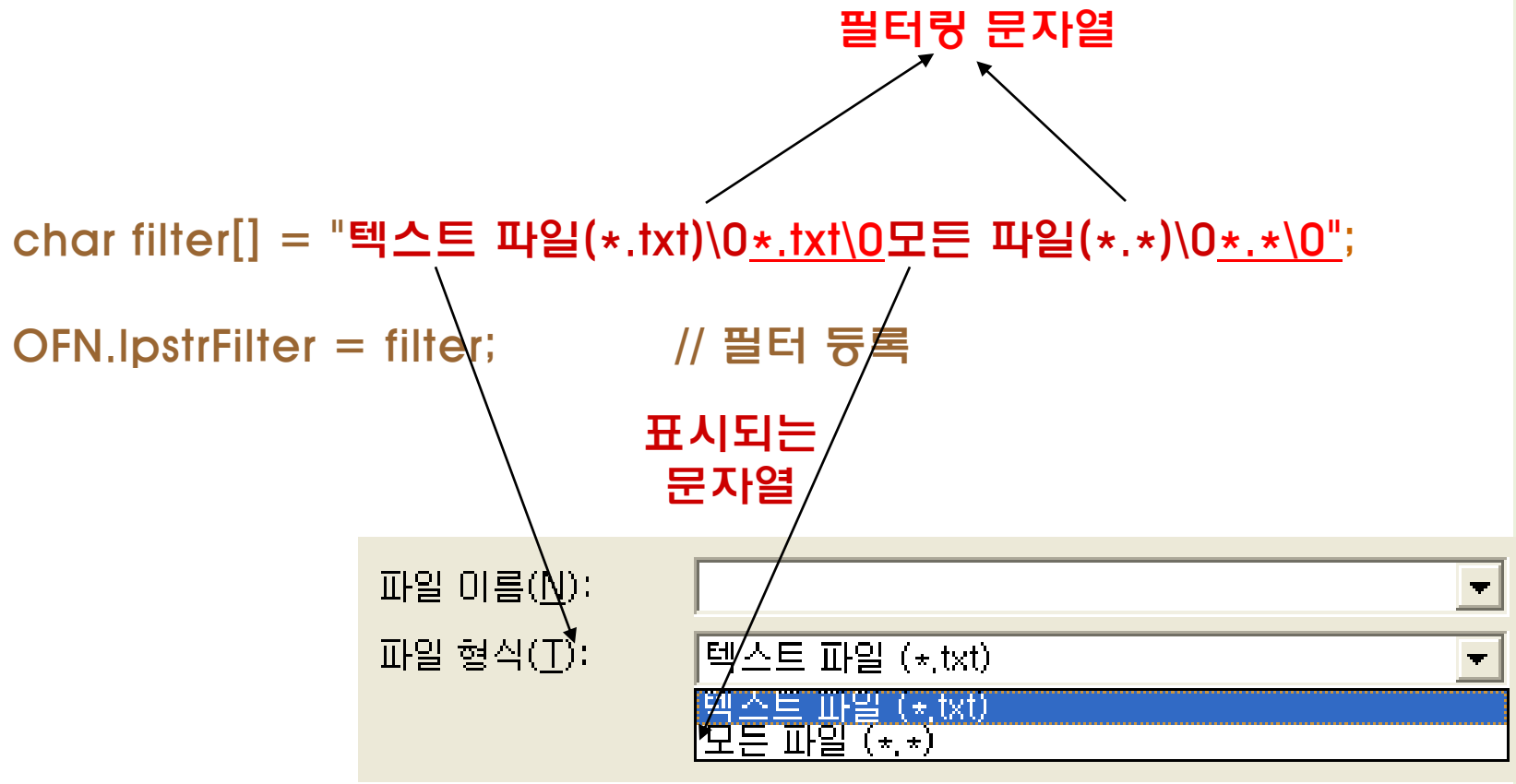
공용대화상자 – 파일 열기

- 파일열기 처리절차
  - OPENFILENAME 구조체 할당
  - 열기함수 호출 –> 파일이름 획득

```
OPENFILENAME OFN; // 구조체 할당
memset(&OFN, 0, sizeof(OPENFILENAME));     // (1) 구조체 초기화

OFN.lStructSize = sizeof(OPENFILENAME);    // (2) 구조체 크기
OFN.hwndOwner = hwnd;                      // (3) 윈도우 핸들
OFN.lpstrFile = filepath;                  // (4) 선택한 파일경로 저장
OFN.nMaxFileTitle = 100;                   // (5) 파일경로 최대길이
OFN.lpstrFileTitle = filename;             // (6) 선택한 파일이름 저장
OFN.nMaxFile = 100;                        // (7) 파일이름 최대길이

GetOpenFileName(&OFN);                     // (8) 열기할 파일이름을 획득
```

필터 지정 방법

- 필터의 용도
  - 표시되는 파일이름을 걸러 줌
  - 정의시 공문자 삽입 안 하도록



파일 열기

```
OPENFILENAME OFN;
char str[100], lpstrFile[100] = "";
switch (iMsg)
{
case WM_COMMAND:
    switch(LOWORD(wParam)) {
       case ID_FILEOPEN:
          memset(&OFN, 0, sizeof(OPENFILENAME));          // 초기화
          OFN.lStructSize = sizeof(OPENFILENAME);
          OFN.hwndOwner=hwnd;
          OFN.lpstrFilter=
                    "Every File(*.*)\0*.*\0Text File\0*.txt;*.doc\0";
          OFN.lpstrFile=lpstrFile;
          OFN.nMaxFile=256;
          OFN.lpstrInitialDir=".";    // 초기 디렉토리
          if (GetOpenFileName(&OFN)!=0) {
                    wsprintf(str,"%s 파일을 여시겠습니까 ?",OFN.lpstrFile);
                    MessageBox(hwnd,str,"열기 선택",MB_OK);
          }
          break;
```

파일 저장하기

```
OPENFILENAME OFN;       // 파일열기와 저장하기는 동일한 구조체 사용

switch (iMsg) {
case WM_COMMAND:
    switch(LOWORD(wParam)) {
       case ID_FILESAVE:  // 메뉴 선택
          memset (&OFN, 0, sizeof(OPENFILENAME));
          OFN.lStructSize = sizeof(OPENFILENAME);
          ...

          OFN.lpstrInitialDir=".";
          if (GetSaveFileName(&SFN)!=0) {
                    wsprintf(str,"%s 파일에 저장하시겠습니까 ?",
                                        OFN.lpstrFile);
                    MessageBox(hwnd,str,"저장하기 선택",MB_OK);
          }
          break;
```

8-2 공용대화상자 이용 파일 읽어오기

```
...생략...
HWND    hwndChild[100] ;            // 자식 윈도우 핸들 저장 변수
char    WinBuff[100][1000];         // 자식 윈도우에 대한 버퍼
int     WndCount;                   // 생성된 자식 윈도우의 개수
...생략...
void ReadFromFile(int WndIndex, char *filename)
        //filename에 저장된 파일이름을 이용해서 파일을 버퍼에 읽어들임
{
    HANDLE hFile;
    int size = 1000;
    hFile = CreateFile (filename, GENERIC_READ, FILE_SHARE_READ,
                                  NULL,  OPEN_EXISTING, 0, 0);
    ReadFile (hFile, WinBuff[WndIndex], size, &size,NULL);
    WinBuff[WndIndex][size] = '\0';
    CloseHandle(hFile);
}
```

8-2 공용대화상자 이용 파일 읽어오기

```
LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
    LPARAM lParam)
{
    HDC         hdc ;
    PAINTSTRUCT ps ;
    int i, SelectWnd;          // SelectWnd에는 선택된 자식 윈도우의 번호 저장

    for(i=1; i<=WndCount; i++)
        if (hwnd == hwndChild[i])
        {
                SelectWnd = i;
                break;
        }
```

8-2 공용대화상자 이용 파일 읽어오기

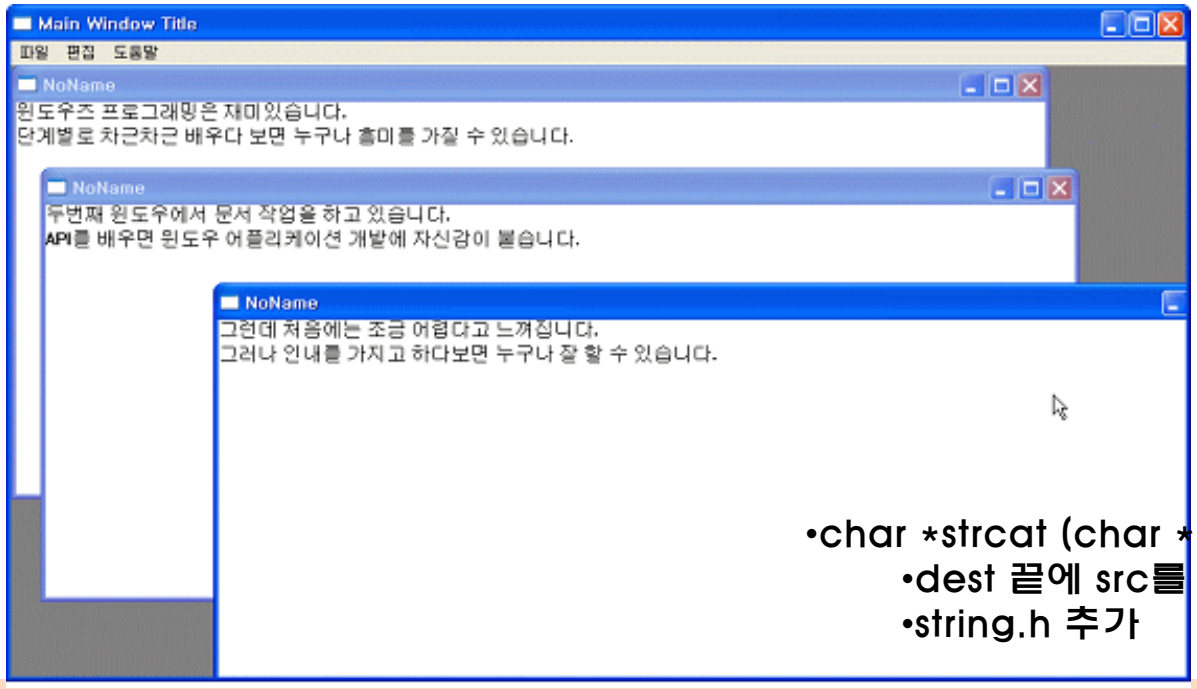
```
    switch (iMsg)
    {
    case WM_CREATE :
        return 0 ;
    case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        TextOut(hdc,0,0,WinBuff[SelectWnd],strlen(WinBuff[SelectWnd]));
        // 버퍼에 저장된 내용을 선택된 자식 윈도우에 출력
        EndPaint(hwnd, &ps);
        break;
    case WM_DESTROY :
        return 0 ;
    }
    return DefMDIChildProc (hwnd, iMsg, wParam, lParam) ;
}
```

8-2 공용대화상자 이용 파일 읽어오기

```
LRESULT CALLBACK FrameWndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
    LPARAM lParam)
{
   ...생략...
            case ID_FILEOPEN:
                ...생략... // 공용대화상자 이용 파일이름을 filepath에 저장
                 WndCount++;
                 ReadFromFile(WndCount, filepath);
                ...생략... // 자식 윈도우 생성
                 hwndChild[WndCount] = (HWND) SendMessage(
                          hwndClient, WM_MDICREATE, 0,
                          (LPARAM) (LPMDICREATESTRUCT) &mdicreate);
                return 0;
```

실습 8-1

- 제목
  - MDI기반 10라인 에디터에 입출력 기능 추가하기
- 내용
  - 각 자식 윈도우가 10라인 입력 에디터가 되도록 작성한다.
  - 공용대화상자를 이용하여 텍스트파일을 입출력 및 편집할 수 있다.



•char *strcat (char *dest, const char *src);
  •dest 끝에 src를 덧붙이는 함수
  •string.h 추가

실습 8-2

사운드 이용하기

- BOOL PlaySound (LPCSTR pszSound, HMODULE hmode, DWORD fdwSound);
  - 32비트 사운드 재생 함수
  - Wav 파일 재생
  - **mmsystem.h 파일 삽입하고, winmm.lib 링크 추가**
  - lpszSound: 재생할 파일 명
  - Hmode: 리소스의 wave 파일을 연주할 경우 리소스를 가진 실행 파일의 핸들 지정, 그 외에는 NULL로 지정
  - fdwSound: 사운드의 연주 방식과 연주할 사운드의 종류 정의
    - SND_ALIAS: pszSound가 레지스트리에 정의된 시스템 이벤트일 경우
    - SND_ASYNC: 비동기화된 연주, 연주를 시작한 직후 다른 작업을 바로 시작할 수 있다. 연주를 중지하려면 PSZsOUND를 null값으로 하여 함수를 호출
    - SND_LOOP: 지정한 함수를 반복적으로 계속 연주 (SND_ASYNC와 함께 사용)
    - SND_SYNC: 동기화된 연주, 사운드 파일의 연주가 완전히 끝나기 전에는 리턴하지 않는다.

  - 사용예) PlaySound (“Sample.wav” , NULL, SND_ASYNC);
  - 사용예) PlaySound (NULL, 0, 0); // 비동기적 연주 재생 정지
- MCI (Media Control Interface) 라이브러리 사용하기

PeekMessage () 함수

- 지금까지 우리가 사용했던 메시지루프

```
while( GetMessage( &msg, NULL, 0, 0) ){
        TranslateMessage( &msg );
        DispatchMessage( &msg );
}
```

- GetMessage()함수는 메시지 큐에 대기중인 메시지가 없을 경우 메시지가 전달될 때까지 리턴하지 않고 무한히 대기한다.

- 특별한 일을 하지 않고 대기하는 시간에 다른 일을 하려면 이 함수 대신 PeekMessage()함수를 사용하는 것이 좋다.
- 두 함수의 원형
  - BOOL GetMessage(LPMSG lpMsg, HWND hWnd,
    UINT wMsgFilterMin, UINT wMsgFilterMax);
  - BOOL PeekMessage(LPMSG lpMsg, HWND hWnd,
    UINT wMsgFilterMin, UINT wMsgFilterMax, UINT wRemoveMsg);
    - wRemoveMsg: 메시치 처리 방법 지정 플래그
      - PM_NOREMOVE: 메시지를 읽은 후 큐에서 메시지를 제거하지 않는다.
      - PM_REMOVE: 메시지를 읽은 후 큐에서 메시지를 제거한다.
      - PM_NOYIELD: 다른 스레드로 제어를 양보하지 않는다.

PeekMessage () 함수

- GetMessage()함수처럼 메시지 큐에서 메시지를 읽는다.
- 이 함수는 GetMessage()함수와 달리 읽은 메시지를 무조건 제거하지 않으며 큐가 비어 있을 경우 대기하지 않고 곧바로 리턴한다.
- 메시지를 읽지 않고 단순히 메시지가 있는지 확인만 할 수 있으며 이런 특성은 백그라운드(background) 작업에 적절하다.
- **주의할 점은 PeekMessage()함수로 메시지 루프를 구현했을 경우 WM_QUIT메시지에 대한 예외적인 처리를 반드시 해주어야 한다.**
- GetMessage()함수는 WM_QUIT메시지를 받으면 FALSE를 리턴하여 무한 메시지 루프를 빠져나올 수 있도록 하지만 PeekMessage()함수는 메시지 존재 여부만 알려주므로 무한 메시지 루프를 빠져 나올 수 없다.
- PeekMessage()함수를 사용한 메시지 루프는 다음과 같이 구현한다.

```
While (1){
    if ( PeekMessage (&msg, NULL, 0, 0, PM_REMOVE ) ){
        if(msg.message == WM_QUIT ) break;
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}
```

- 시간이 비교적 오래 걸리는 함수나 코드 부분을 실행할 때 함수나 코드내에 PeekMessage()함수로 메시지 존재 여부를 판단하는 코드를 추가하여 사용자 입력 같은 이벤트에 즉각적으로 반응하여 처리할 수 있도록 해야 한다.

GetAsyncKeyState () 함수

- SHORT GetAsyncKeyState (int vKey);
  - 현재 키 상태를 알아오는 함수, 이 함수는 키가 눌려졌을 때 (down)나 떨어졌을 때 (up) 호출됨
  - WM_KEYDOWN 메시지는 키가 눌러질 때 보내지는 메시지로, 키를 계속 누르고 있다는 것을 알려주지는 않는다.
  - 한 개의 키를 누른 상태에서 다른 키를 누르면, 두 번째 누른 키만 전달된다.
  - 키의 현재 상태, 즉 키가 눌러졌는지 아닌지를 조사해야한다.
  - GetAsyncKeyState는 메시지 처리 시점의 키 상태를 조사
  - 가상 키 코드를 인수로 전달받아, 일반 키가 눌러졌으면 최상위비트가 1로 설정되고 (0x8001), 그렇지 않으면 0 (0x8000)으로 설정된다.

GetAsyncKeyState () 함수

- SHORT GetAsyncKeyState (int vKey);
  - vKey: 가상키 코드 값, 확인하고자 하는 키를 입력
  - 리턴 값:
    - 0x0000: 이전에 누른 적이 없고 호출 시점에도 눌려있지 않은 상태
    - 0x0001: 이전에 누른 적이 있고 호출 시점에는 눌려있지 않은 상태
    - 0x8000: 이전에 누른 적이 없고 호출 시점에는 눌려있는 상태
    - 0x8001: 이전에 누른 적이 있고 호출 시점에도 눌려있는 상태
    - 사용예:

```
case WM_KEYDOWN:
    if ( GetAsyncKeyState (VK_SPACE) & 0x8000)
            // 스페이스키가 눌려짐
    Else
            // 스페이스키가 안 눌려짐
```