2013년 1학기 윈도우 프로그래밍

윈도우 프로그래밍 기초

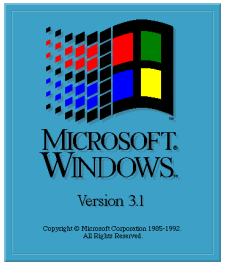
- 학습목표
  - 윈도우 프로그램의 기본 구조를 이해하고 간단한 윈도우 프로그램을 작성할 수 있다.

- 내용
  - 윈도우 역사
  - 윈도우 운영체제의 특징
  - 윈도우 프로그래밍 개념
  - 윈도우 프로그램 구조

윈도우의 역사

- MS-DOS(Disk Operating System) 시대
  - Microsoft사에서 개발
  - 1981년 IBM이 16비트 운영체제인 MS-DOS를 채택하면서 DOS 시스템이 사용됨

- Windows 1.0과 3.1
  - 1985년 11월 마이크로소프트사에서는 MS-DOS 를 기반으로 한 windows 1.0 버전 발표
    - 그래픽 유저 인터페이스 도입
    - 255KB 메모리 지원, 256 컬러 표시
  - 1992년 4월 windows 3.1 발표하면서 본격적인 윈도우 시대 도래



윈도우의 역사

- Windows 95
  - 1995년 8월 windows 95 발표되며 Windows의 전성기
  - 기존의 윈도우 버전까지는 ms-dos 기반
  - Windows 95 이후부터는 현재 사용되는 윈도우 인터페이스의 기반
- Windows 98
  - Windows 95 기반에 인터넷, 멀티미디어, 파일시스템 등의 기능을 개선한 버전
- Windows xp, windows vista, Windows 7, Windows 8 출시

윈도우 운영 체제의 특징

- DOS 운영체제기반 프로그램: 절차적 (직선적) 프로그램
  - 프로그램의 실행 흐름이 프로그래머가 기술한 코드 순서대로 순차적으로 진행
  - 사용자가 키보드로부터 문자 명령어나 파일명 등을 입력하여 프로그램을 제어
- 윈도우에서는 프로그램의 실행 흐름을 프로그래머 혼자 결정하지 않고 윈도우 OS와 상호작용하면서 처리

윈도우 운영 체제의 특징

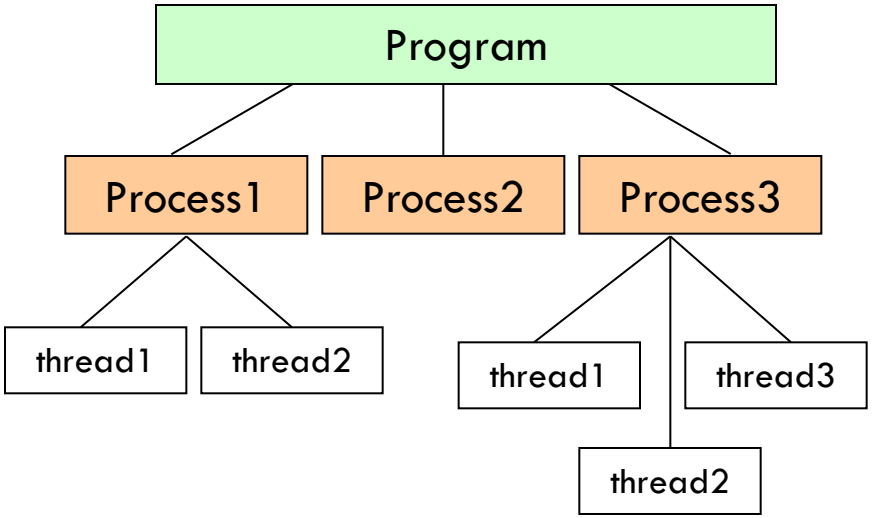
- 윈도우 운영 체제의 특징
  - 그래픽 사용자 인터페이스 (GUI) 기반의 운영체제
    - 픽셀 단위의 그래픽 기반 운영 체제
    - 아이콘 , 탐색기, 메뉴, 스크롤 등을 마우스를 이용하여 사용
  - 메시지 구동 (이벤트 구동) 시스템
    - 메시지: 윈도우가 어플리케이션에게 보내는 알림
    - 운영 체제로부터 메시지를 받아 동작한다.
      - 외부에서 발생한 일들을 윈도우 OS가 감지하여 해당 프로그램에 메시지를 전달한다.
  - 장치 독립적
    - 하드웨어 장치에 무관하게 프로그래밍할 수 있다.
    - Device Driver에 의해 주변 장치들을 제어, 관리한다.
    - 프로그래머는 어떤 하드웨어가 현재 시스템에 설치되어 있는지 신경 쓸 필요가 없음
  - 리소스 분리
    - 코드 이외의 데이터가 분리되어 있다.

윈도우 운영 체제의 특징

- Multi-tasking
  - DOS는 single-tasking: 한번에 하나의 프로그램만 실행
  - 동시에 여러 개의 프로그램을 실행시킬 수 있다.
    - 워드 프로세서로 문서를 작성하면서 스프레드 시트로 데이터를 참조할 수 있다.
  - CPU는 하나이기 때문에, 실행할 기회를 여러 개의 프로세스에게 순서대로 나누어 주어서 동시에 여러 개가 함께 실행되는 것처럼 보인다.
  - 프로그램(program), 프로세스(process)
    - 프로그램(program): 파일이 디스크에 저장된 내용과 같은 수동적인 실체 (passive entity), 실제 실행되는 파일
    - 프로세스(process): 프로그램에 의한 작업의 기본 단위, 즉 프로그램에서 지시하는 명령을 읽어 들여 실행하고, 종료 지시를 만나면 처리를 종료하는 등의 일, 프로그램을 실행하는 것

윈도우 운영 체제의 특징

- Multi-threading
  - 쓰레드(thread): 프로세스를 구성하는 프로세스보다 작은 태스크(task) 단위
  - 프로세스 하나가 쓰레드 여러 개를 병행 처리할 수 있는데 이것을 멀티 쓰레드라고 한다.
    - 앞에서는 사용자의 조작에 응답하고 백그라운드에서 인쇄 등의 시간이 걸리는 작업을 처리하는 기능
  - 각각의 쓰레드는 자신만의 코드를 수행한다.
  - 쓰레드의 컨텍스트 정보가 적기 때문에 빠르게 만들 수 있다.



윈도우 프로그래밍 개념

- API(Application Programming Interface)
  - 응용 프로그램이 OS나 데이터베이스 관리 시스템(DBMS)등과 통신할 때 사용되는 언어나 메시지 형식
  - 응용 프로그램 개발자를 위해 제공하는 함수 집합
  - 많은 종류의 API중 32비트 OS인 윈도우에서 제공하는 API를 Win32 API라고 한다.

- SDK(Software Development Kit)
  - 윈도우용 응용 프로그램 개발도구
  - 각종 편집 툴, 라이브러리, 헤더파일, 도움말, 예제 프로그램들로 구성되어 있다.
  - 그래픽 처리, 데이터베이스 접근, 동적링크 처리 등

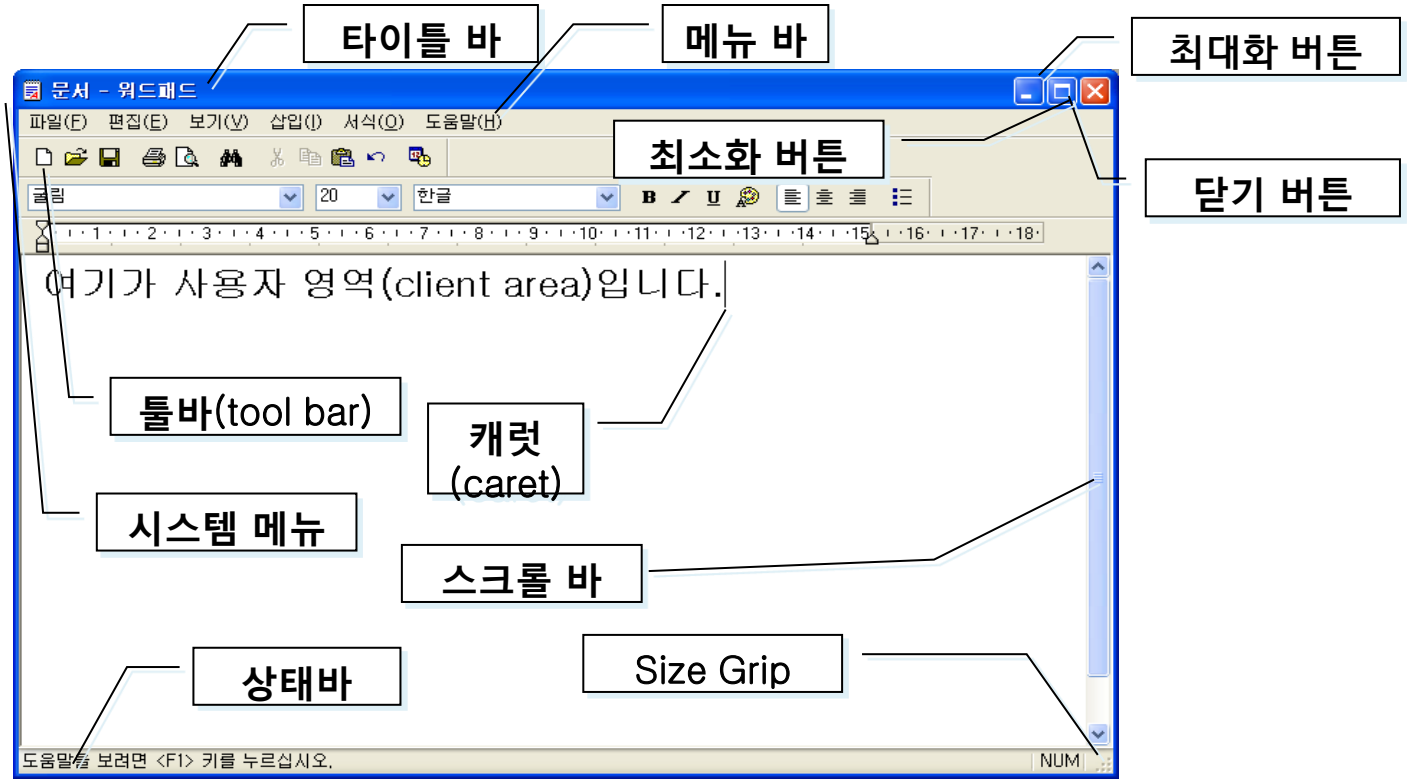
윈도우 OS를 구성하는 중요한 DLL 파일

- API는 결국 DLL (Dynamic Link Library)안에 있다.
  - 각각의 API는 C언어로 기술된 함수들
  - MFC (Microsoft Foundation Class): 목적별로 복수의 API를 모아 프로그램 코드로 간접적으로 호출하기 위한 기능 제공

윈도우 운영체제의 구성 모듈

모듈	파일명	기능
커널	KERNEL32.DLL	윈도우 OS의 핵심으로 메모리 관리, 파일 입출력, 프로그램의 로드와 실행 등 OS의 기본 기능 수행
GDI	GDI32.DLL	화면이나 프린터와 같은 출력 장치에 출력을 관리
사용자 인터페이스	USER32.DLL	윈도우, 다이얼로그, 메뉴, 커서, 아이콘 등과 같은 윈도우 기반의 사용자 인터페이스 객체들을 관리

윈도우 프로그래밍 개념



윈도우 GUI: 클라이언트 영역 & 비클라이언트 영역

윈도우 프로그래밍 개념

- 리소스(Resource)
  - 메뉴, 아이콘, 커서, 다이얼로그, 액셀러레이터, 비트맵, 문자열, 버전 정보 등 사용자 인터페이스를 구성하는 자원들로 읽기 전용 정적 데이터
  - 프로그램 실행 중 변경되지 않는 정적 데이터로 C/C++과 같은 언어로 관리하지 않고 리소스 스크립트 파일로 관리
    - 윈도우 프로그램: 소스 코드와 리소스 코드가 분리
- 이벤트(Event)와 메시지(Message)
  - 이벤트: 사용자가 키보드를 누르거나 마우스 버튼을 클릭할 때, 툴 바의 버튼을 누르거나 윈도우의 크기를 조절하는 등의 기계적인 조작에 의해 발생
  - 이벤트가 발생하면 윈도우 OS는 이를 감지하여 해당 프로그램으로 메시지를 전달
  - 마우스 누름(이벤트) –> WM_LBUTTONDOWN 메시지로 변환
    - 윈도우 메시지 유형들은 모두 “WM_” 로 시작

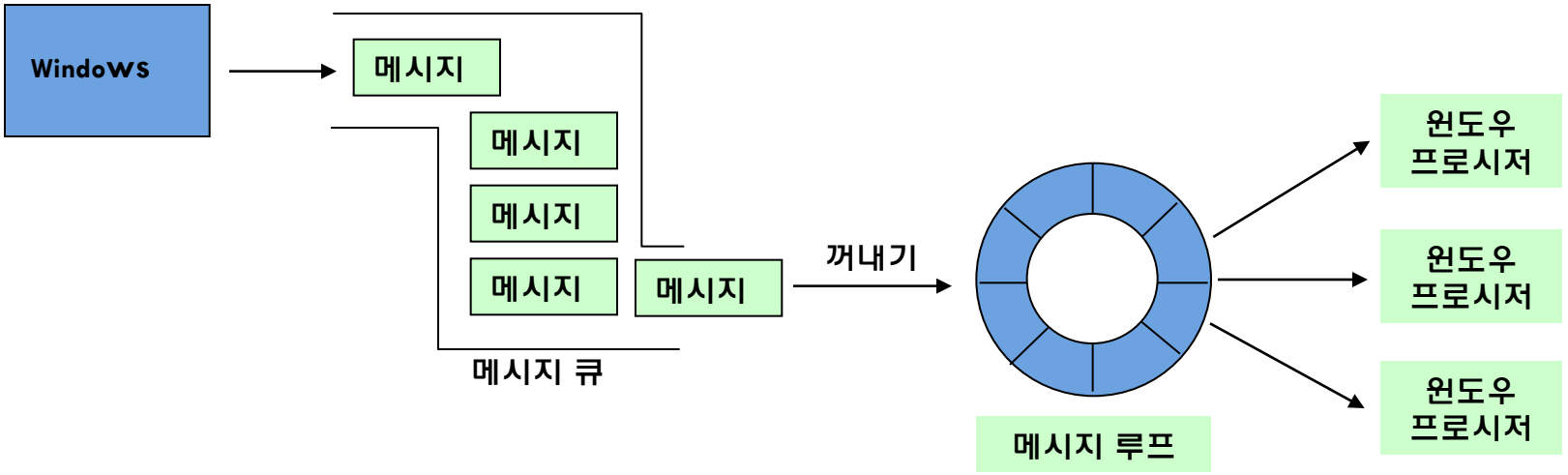
윈도우 프로그래밍 개념

윈도우 메시지 유형	발생하는 상황
WM_CREATE	윈도우가 생성될 때
WM_ACTIVATE	윈도우가 활성화되거나 비 활성화 될 때
WM_PAINT	윈도우가 다시 그려져야 할 때
WM_MOUSEMOVE	마우스 커서가 움직였을 때
WM_COMMAND	메뉴 등으로 명령을 내렸을 때
WM_LBUTTONDOWN	마우스 왼쪽 버튼이 눌렸을 때
WM_SIZE	윈도우의 크기가 변경되었을 때
WM_MOVE	윈도우가 이동되었을 때
WM_TIMER	설정된 타이머 시간이 되었을 때
WM_DESTROY	윈도우가 없어질 때

<여러 종류의 메시지들>

윈도우 프로그래밍 개념

- 메시지 큐(Message Queue)
  - FIFO (First In First Out)
  - 사용자의 컴퓨터 조작에 의해 발생한 이벤트는 메시지 형태로 만들어져 윈도우 OS가 관리하는 메시지 큐라는 곳에 모이게 됨
  - 하나의 프로그램이 실행되면 하나의 메시지 큐가 할당됨



윈도우 프로그래밍 개념

- 메시지 루프(Message Loop)
  - 윈도우 OS가 프로그램에 전달한 메시지를 받아들여 분석하는 무한 루프
  - 일반적인 메시지 루프

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

  - GetMessage() 함수가 FALSE를 리턴 (프로그램을 종료하라는 WM_QUIT 메시지일 경우)할 때까지 메시지 큐로부터 메시지를 얻어와 처리

윈도우 프로그래밍 개념

- 윈도우 프로시저(Window Procedure)
  - 메시지 루프에서 해석한 메시지를 구체적으로 처리하는 기능을 하는 소스 부분
  - OS가 호출하도록 콜백 함수(Callback function)로 만듦
    - 콜백 함수: OS로부터 호출되는 함수 (윈도우 프로시저)
  - 콜백 함수는 함수 앞에 키워드 CALLBACK을 쓰며 호출은 윈도우 OS가 함
  - 함수의 이름은 프로그래머가 마음대로 지정할 수 있다.
  - 함수 프로토타입
    - LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

윈도우 프로그래밍 개념

- 인스턴스 (Instance)
  - 어떤 대상이 메모리에 생성된 클래스의 실체

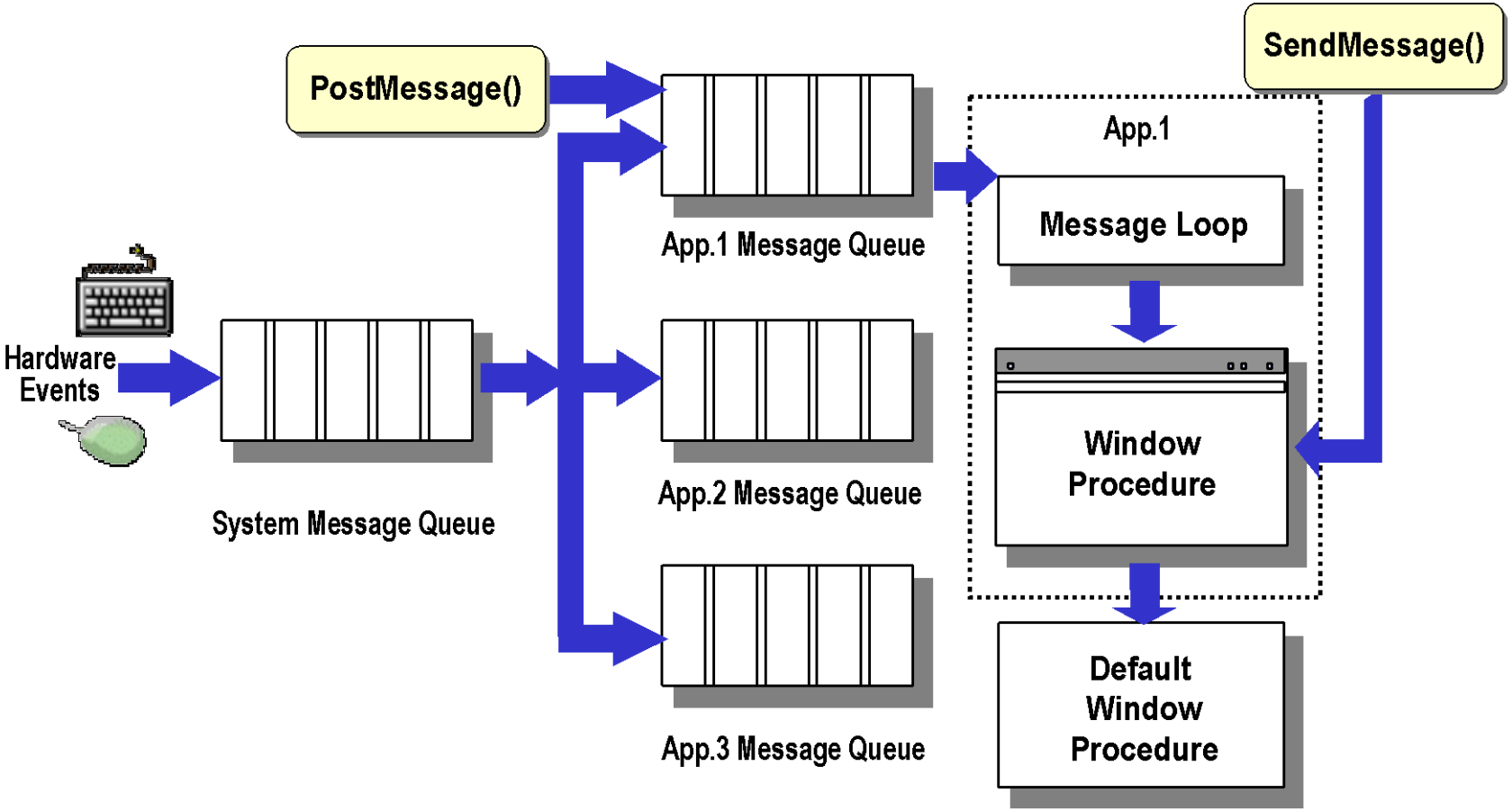
- 핸들(Handle)
  - 프로그램에서 현재 사용중인 객체 (윈도우, 커서, 아이콘, 메뉴 등)들을 구분하기 위해 윈도우 OS가 부여하는 고유 번호
    - 32비트 정수형
    - 핸들값은 접두어 h로 시작한다.
    - 핸들은 운영체제가 발급하며 사용자는 사용만 한다.
    - 같은 종류의 핸들끼리는 절대 중복된 값을 가지지 않는다.
    - 핸들은 단순한 구분자이므로 핸들에 어떤 값이 들어가 있는지 알 필요가 없다!

윈도우 프로그래밍 개념

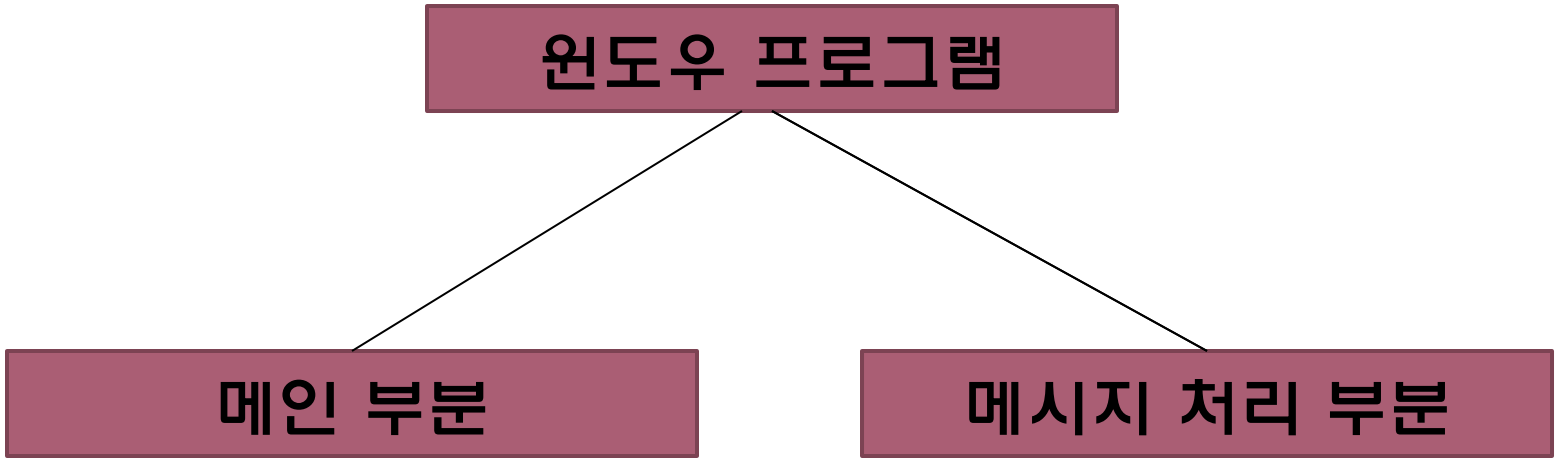
- GDI(Graphic Device Interface)
  - 윈도우 운영체제상에서 제공하는 응용 프로그램과 그래픽 장치 간의 인터페이스
  - 선 그리기, 칼라 처리 등 그래픽을 다루기 위한 함수의 모음
  - 디스플레이, 프린터, 기타 장치에 대한 그래픽 출력을 위하여 응용 프로그램이 사용할 수 있는 함수와 그에 관련된 구조를 제공
  - GDI 객체에는 펜, 브러시, 폰트, 팔레트, 비트맵, 리전

- 디바이스 컨텍스트(DC : Device Context)
  - 그래픽과 관련한 정보를 모아 놓은 구조체
  - 윈도우에서는 출력장치에 무엇인가를 출력하기 위해서는 반드시 DC가 필요
  - 보통 해당 DC 핸들을 얻은 후 출력
  - DC 핸들은 출력 대상을 나타내는 구분 번호
  - 모든 그래픽 함수들은 첫 번째 인자로 DC 핸들을 필요로 함

이벤트에 대해 **메시지** 발생, OS의 역할, 각 응용프로그램들이 동작하는 방식



3절. 윈도우 프로그램 구조



도스 기반 콘솔 프로그램

```
#include <stdio.h>

void main ()
{
      printf (  “Hello world\n”  );
}
```

윈도우 기반 프로그램

Window-based program

```
#include <windows.h>// 윈도우 헤더 파일

int WinMain (HINSTANCE hInstance,HINSTANCE
   hPrevInstance, LPSTR lpszCmdParam,int nCmdShow)
{
   HWND hWnd;
   MSG Message;
   WNDCLASS WndClass;
   g_hInst=hInstance;

   WndClass.cbClsExtra=0;
   WndClass.cbWndExtra=0;
   WndClass.hbrBackground=(HBRUSH)GetStockObje
   ct(BLACK_BRUSH);
   WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
   WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
   WndClass.hInstance=hInstance;
   WndClass.lpfnWndProc=(WNDPROC)WndProc;
   WndClass.lpszClassName=lpszClass;
   WndClass.lpszMenuName=NULL;
   WndClass.style=CS_HREDRAW | CS_VREDRAW;
   RegisterClass(&WndClass);

   hWnd=CreateWindow(lpszClass,lpszClass,WS_OVER
   LAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
   CW_USEDEFAULT, CW_USEDEFAULT,
   NULL ,(HMENU)NULL, hInstance,
         NULL);
   ShowWindow(hWnd,nCmdShow);
   UpdateWindow (hWnd);

   while(GetMessage(&Message,0,0,0)) {
         TranslateMessage(&Message);
         DispatchMessage(&Message);
      }
      return Message.wParam;
   }

   LRESULT CALLBACK WndProc (HWND hWnd,UINT
      iMessage,WPARAM wParam,LPARAM lParam)
   {  PAINTSTRUCT ps;
      HDC hDC;
      char temp[] =  "Hello world!" ;

      switch(iMessage) {
      case WM_PAINT:
         hDC = BeginPaint(hWnd, &ps);
         TextOut(hDC, x, y, temp, strlen(temp));
         EndPaint(hWnd, &ps);
         break;
      case WM_DESTROY:
         PostQuitMessage(0);
         return 0;
      }
      return(DefWindowProc(hWnd,iMessage,wParam,
                                          lParam));
   }
```

윈도우 프로그램의 형태

```
int WINAPI WinMain (.....)
{
    윈도우 생성
    메시지 전송
}

LRESULT CALLBACK WndProc (....)
{
    메시지에 따른 처리
}
```

전형적인 윈도우 프로그램의 형태

- 일반적인 윈도우즈 프로그램은 C/C++언어 소스부분과 리소스 스크립트 파일(.rc), 관련 헤더와 리소스 파일 등으로 구성되어 있다.
- 이것들을 관리하는 파일은 프로젝트작업공간(.dsw)파일이다.
  - 솔루션(.sln) –> 프로젝트(.vcproj)
- C/C++언어 소스부분은 WinMain()함수 한 개와 한 개 이상의 윈도우즈 프로시저 함수로 구성된다.

WinMain()의 처리내용

- 윈도우 클래스 만들기 : 윈도우 함수, 아이콘, 커서, 배경색
  - 윈도우 클래스를 등록하기
- 윈도우 만들기 : 윈도우 좌표, 스타일
  - 윈도우를 화면에 보이기
- 윈도우에서 발생한 이벤트에 관한 메시지 보내기

WinMain()의 형식

```
int WINAPI WinMain (HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        PSTR szCmdLine,
        int iCmdShow)
```

- WINAPI: 윈도우 프로그램이라는 의미
- hInstance: 현재 실행중인 어플리케이션의 인스턴스 핸들
- hPrevInstance: 동일한 어플리케이션이 실행중일 경우 이전에 실행된 프로그램의 인스턴스 핸들. Win32 어플리케이션의 경우 항상 NULL
- szCmdLine: 커멘드라인 상에서 프로그램 구동 시 전달된 문자열
- iCmdShow: 윈도우가 화면에 출력될 형태

윈도우 클래스

- 윈도우클래스: 생성하는 윈도우의 형태를 정의하기 위해 사용하는 구조체

```
typedef  struct  _WNDCLASSEX {
    UINT    cbSize;             //본 구조체의 크기
    UINT    style;              //출력 스타일
    WNDPROC lpfnWndProc;        //프로시저 함수
    int     cbClsExtra;         //클래스 여분 메모리
    int     cbWndExtra;         //윈도우 여분 메모리
    HANDLE  hInstance;                  //윈도우 인스턴스
    HICON   hIcon;                      //아이콘
    HCURSOR hCursor;                    //커서
    HBRUSH  hbrBackground;              //배경색
    LPCTSTR lpszMenuName;               //메뉴이름
    LPCTSTR lpszClassName;              //클래스 이름
    HICON   hIconSm;                    //작은 아이콘
} WNDCLASSEX;
```

윈도우 클래스 정의

```
WNDCLASSEX  wndclass ;                              // 구조체 정의

wndclass.cbSize  = sizeof(wndclass) ;          // 구조체 크기
wndclass.style = CS_HREDRAW | CS_VREDRAW ;
                    // 윈도우 출력 스타일 -> 수직/수평의 변화시 다시 그림
wndclass.lpfnWndProc   = WndProc ;          // 프로시저 함수명
wndclass.cbClsExtra    = 0 ;   // O/S 사용 여분 메모리(Class)
wndclass.cbWndExtra    = 0 ; // O/S 사용 여분 메모리(Window)
wndclass.hInstance   = hInstance ;// 응용 프로그램 ID
wndclass.hIcon       = LoadIcon(NULL, IDI_APPLICATION); //아이콘유형
wndclass.hCursor   = LoadCursor(NULL, IDC_ARROW);     // 커서 유형
wndclass.hbrBackground = (HBRUSH)
                  GetStockObject(WHITE_BRUSH);       // 배경색
wndclass.lpszMenuName  = NULL ;                  // 메뉴 이름
wndclass.lpszClassName =  "ClassNameEx" ;  // 클래스 이름
wndclass.hIconSm = LoadIcon(NULL,IDI_APPLICATION); // 작은 아이콘
```

윈도우 클래스 정의

- 윈도우 출력 스타일
  - 윈도우 클래스의 스타일을 나타낸다. Bitwise OR (|) 연산자를 이용하여 여러 개의 스타일을 OR로 설정할 수 있다.
  - CS_HREDRAW / CS_VREDRAW: 작업 영역의 폭/높이가 변경되면 윈도우를 다시 그린다.
  - CS_DBCLKS: 마우스 더블 클릭 메시지를 보낸다
  - CS_CLASSDC: 이 클래스로부터 만들어진 모든 윈도우가 하나의 DC를 공유한다.
  - CS_OWNDC: 각 윈도우가 하나의 DC를 독점적으로 사용한다.
  - CS_PARENTDC: 차일드 윈도우가 부모 윈도우의 DC를 사용한다.

윈도우 클래스 등록

RegisterClassEx (&wndclass);

- ATOM RegisterClassEx(CONST WNDCLASSEX *lpwcx);
  - &wndclass : 앞서 정의한 윈도우 클래스의 주소

윈도우 만들기

```
hwnd = CreateWindow // 윈도우가 생성되면 핸들(hwnd)이 반환됨
        (
                "ClassNameEx,                   // 윈도우 클래스 이름
                "Window Title Name",            // 윈도우 타이틀 이름
                WS_OVERLAPPEDWINDOW,            // 윈도우 스타일
                CW_USEDEFAULT,                  // 윈도우 위치, x좌표
                CW_USEDEFAULT,                  // 윈도우 위치, y좌표
                CW_USEDEFAULT,                  // 윈도우 폭
                CW_USEDEFAULT,                  // 윈도우 높이
                NULL,                           // 부모 윈도우 핸들
                NULL,                           // 메뉴 핸들
                hInstance,                      // 응용 프로그램 ID
                NULL) ;                         // 생성된 윈도우 정보

ShowWindow(hwnd, iCmdShow);                     // 윈도우의 화면 출력
    // nCmdShow : SW_HIDE, SW_SHOW, SW_MAXIMIZE, SW_MINIMIZE

UpdateWindow(hwnd);                 // O/S에 WM_PAINT  메시지 전송
```

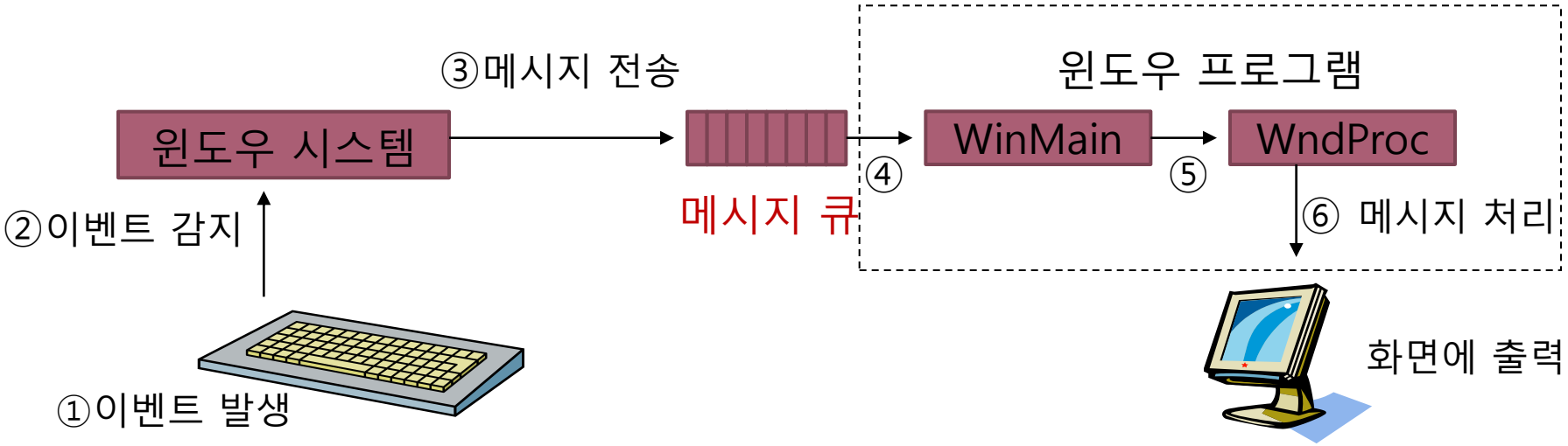
윈도우 만들기

- 윈도우 스타일
  - WS_OVERLAPPED: 디폴트 윈도우
  - WS_CAPTION: 타이틀 바를 가진 윈도우
  - WS_HSCROLL / WS_VSCROLL: 수평/수직 스크롤 바
  - WS_MAXIMIZEBOX / WS_MINIMIZEBOX: 최대화/최소화 버튼
  - WS_SYSMENU: 시스템 메뉴
  - WS_THICKFRAME: 크기 조정이 가능한 두꺼운 경계선
  - WS_BORDER: 단선으로 된 경계선, 크기 조정 불가능
  - WS_POPUP: 팝업 윈도우 (WS_CHILD와 같이 쓸 수 없다)
  - WS_CHILD: 차일드 윈도우
  - WS_VISIBLE: 윈도우를 만들자마자 화면에 출력

  - WS_OVERLAPPEDWINDOW: 가장 일반적인 윈도우 스타일 (WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX)
  - WS_POPUPWINDOW: 일반적인 팝업 윈도우 (WS_POPUP | WS_BORDER | WS_SYSMENU)

이벤트 메시지 보내기

```
while (GetMessage (&msg, NULL, 0, 0))
 {    // WinProc()에서 PostQuitMessage() 호출 때까지 처리
   TranslateMessage (&msg) ; // Shift  'a'  -> 대문자  'A'
   DispatchMessage (&msg) ; // WinMain -> WinProc
 }
```



이벤트 메시지 보내기

- 메시지 처리 루프를 만들기 위한 함수들
  - BOOL GetMessage (LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);
    - 메시지 큐로부터 메시지를 얻어오는 역할
  - BOOL TranslateMessage (CONST MSG *lpMsg);
    - 키보드 입력 이벤트 중 문자 입력을 처리하는 함수로 단축키 명령어를 기본적인 이벤트로 번역 또는 변환
  - LONG DispatchMessage (CONST MSG *lpmsg);
    - 실제적인 이벤트 처리 역할
    - GetMessage 함수로부터 전달된 메시지를 윈도우 프로시저로 보낸다.

WndProc()

- WinMain()에서 전달된 메시지를 처리

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg,
                          WPARAM wParam, LPARAM lParam)
{

    switch (iMsg) // 메시지 번호
    {
      case WM_CREATE:  // 메시지에 따라 처리
                       break;
      case WM_DESTROY:
                       PostQuitMessage (0) ;
                       break;
    } //처리할 메시지만 case문에 나열

    return DefWindowProc (hwnd, iMsg, wParam, lParam);
       // CASE에서 정의되지 않은 메시지는 커널이 처리하도록 메시지 전달
}
```

WndProc()

- WndProc 함수
  - 메시지들을 처리하는 함수
  - 함수 프로토타입:

    LRESULT CALLBACK WindowProc (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
    - hWnd: 메시지를 보내는 윈도우의 핸들
    - Msg: 처리될 윈도우 메시지의 코드나 ID
    - wParam: 메시지 부가정보 (숫자, ID, 분류 등)
    - lParam: 메시지 부가정보 (숫자, ID, 분류 등)

  - 콜백 함수: 이벤트가 발생했을 때 윈도우에 의해서 호출되는 것
    - 일반 함수 호출은 응용 프로그램이 운영체제에 내장된 함수를 호출하여 원하는 작업을 하는데, 콜백 함수는 거꾸로 운영 체제가 응용 프로그램을 부른다. (예, 타이머 함수, WndProc 함수)

WndProc()

- WndProc 함수
  - 함수명이 꼭 wndProc일 필요는 없다.
  - 윈도우 속성설정 부분에서 wc.lpfnWndProc = wndProc로 설정
  - 윈도우로 전달되는 메시지를 처리하는 메시지 처리 함수로 사용자 정의 함수

- DefWindowProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) 함수
  - WndProc 에서 처리하지 않은 나머지 메시지를 처리하도록 한다. 예를 들어, 시스템 메뉴 메시지 처리 등

윈도우 프로그램 작성하기

- Include 파일 사용
  - 윈도우 응용 프로그램에 필요한 파일을 포함시킨다.
  - windows.h / windowsx.h
- 메인 함수: 등록 부분 → WinMain ()
  - 윈도우즈 클래스 구조체에 값을 지정
  - 윈도우즈 클래스 등록
  - 윈도우즈 생성
  - 윈도우 출력
  - 이벤트 루프 처리하기
- 메시지 처리 함수: 메시지 처리 부분 → WndProc ()
  - 사용자와 시스템이 보내오는 메시지를 처리한다.
  - 윈도우에서는 WinMain과 WndProc이 모두 있어야 한다.

윈도우 프로그램 작성하기

```
#include <windowsx.h>
int WINAPI WinMain (HINSTANCE hInstance,HINSTANCE hPrevInstance, LPSTR lpszCmdParam,int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASS WndClass;
    g_hInst=hInstance;

    // 윈도우 클래스 구조체 값 설정
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hbrBackground=(HBRUSH)GetStockObject(BLACK_BRUSH);
    WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    WndClass.hInstance=hInstance;
    WndClass.lpfnWndProc=(WNDPROC)WndProc;
    WndClass.lpszClassName=lpszClass;
    WndClass.lpszMenuName=NULL;
    WndClass.style=CS_HREDRAW | CS_VREDRAW;
    // 윈도우 클래스 등록
    RegisterClass(&WndClass);

    // 윈도우 생성
    hWnd=CreateWindow(lpszClass,lpszClass,WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
                                          CW_USEDEFAULT, CW_USEDEFAULT, NULL,(HMENU)NULL, hInstance, NULL);
    // 윈도우 출력
    ShowWindow(hWnd,nCmdShow);
    UpdateWindow (hWnd);
    // 이벤트 루프 처리
    while(GetMessage(&Message,0,0,0)) {
        TranslateMessage(&Message);
        DispatchMessage(&Message);
    }
    return Message.wParam;
}
```

윈도우 프로그램 작성하기

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
        // 메시지 처리하기
        switch (uMsg) {
        case WM_CREATE:
                break;
        case WM_PAINT:
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
        break;
        }
        return DefWindowProc (hWnd, uMsg, wParam, lParam);      // 나머지는 OS로
}
```

1장에서 기억해야 할 내용들

- 윈도우 프로그램의 특징
  - GUI
  - Multi Tasking, Multi Threading
  - Device Independent
  - Message-driven system
  - Resource separated
- 윈도우 프로그래밍 개념
  - API
  - Event (Message), Message Queue
  - Window Procedure
  - Handle, Device Context, Graphic Device Interface
- 윈도우 프로그래밍
  - 윈도우 프로그램 구성
  - 윈도우 프로그램이 어떻게 실행되는지 이해!

연습문제 1-1

- 제목
  - 간단한 윈도우 프로그램 작성하여 실행하기

- 내용
  - 빈 화면을 띄우는 예제 프로그램을 컴파일하여 실행하시오.
  - 윈도우의 타이틀을 “my first window program” 으로 설정하시오.
  - 윈도우의 위치를 (0, 0)으로 설정하시오.
  - 윈도우의 크기를 800*600 으로 설정하시오.

실행 화면