3장 64비트 기반 프로그래밍

Section 01. WIN32 vs WIN64

한번에 송수신할 수 있는 데이터 크기와 **한번에 처리할 수 잇는 데이터 크기를 기준으로 32비트 컴퓨터와 64비트 컴퓨터를 구분** 짓는다.

프로그래머 입장에서 표현할 수 있는 주소 값의 범위가 넓으면 넓을수록 좋다. 범위만 충분하다면 주소 값의 범위가 넓은 만큼 더 넓은 메모리 공간을 활용할 수 있기 때문이다.

32비트 컴퓨터에는 32비트 주소표현이 최선의 선택이다. 한번에 처리할 수 있는 데이터(주소 값 포함)의 크기가 32비트이기 때문이다. 32비트로 주소 값을 표현하면 주소 값의 이동 및 연산이 한 번에 이뤄진다. 뿐만 아니라 표현할 수 있는 주소의 범위도 4G 바이트나 되므로 프로그래머 입장에서 매우 충분하다.

만약 64비트 주소표현을 사용한다면, 64비트 주소 값을 두 번에 나눠서 전송, 처리 해야 하며, 연산 또한 두 번에 걸쳐 행해진다. 32비트 시스템은 64비트 데이터를 한번에 처리하지 못하기 때문이다. 이는 상당한 속도 저하로 이어진다.

그러므로 64비트 컴퓨터 환경에서는 포인터가 64비트 주소표현으로 사용하는 것이 최선의 방법이다. 주소 값의 범위도 2의 64승이나 되므로 충분하다.

Section 02. 프로그램 구현 관전에서의 WIN32 vs WIN64

**LLP64 vs LP64**

Int, long 그리고 포인터 모두 4바이트로 표현되었으나, 이것은 32비트 환경에서 Windows가 기본 자료형과 포인터를 표현하는 방식이다. 그러나 64비트 컴퓨터에서는 다음과 같이 표현

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 운영체제 | 모델 | Char | Short | Int | Long | 포인터 |
| Windows | LLP64 | 1바이트 | 2바이트 | 4바이트 | 4바이트 | 8바이트 |
| UNIX | LP64 | 1바이트 | 2바이트 | 4바이트 | 8바이트 | 8바이트 |

Windows에서는 LLP64라는 데이터 표현 모델을 따르는데, 이 모델은 int와 Long은 그대로 4바이트로 표현하고, 포인터만 8바이트로 표현하는 방식이다. 따라서 32비트 시스템과의 호환성을 중시한 모델, 참고로 UNIX의 LP64모델은 long을 8바이트로 사용하고 있다.

**64비트와 32비트 공존의 문제점**

Ex)

#include <stdio.h>

Int main(void)

{

Int arr[10] = {0, }

Int arrVal = (int)arr;

Printf(“pointer : %d \n”, arrVal);

Return 0;

}

위 예제에서는 배열이 선언된 주소 값을 출력하기 위해 int형 변수에 그 값을 저장하고 있다. 위 코드는 32비트 환경의 시스템 에서는 전혀 문제가 되지 않는다. Int형 자료형도 포인터 주소표현도 4바이트로 표현되기 때문이다. 그러나 64비트 환경 시스템 에서는 문제가 발생한다. 64비트 환경에서는 int는 4바이트 포인터 주소표현은 8바이트로 이루어 지기 때문이다.

“64비트 시스템 에서는 포인터가 지니고 있는 주소 값을 4바이트 정수 형으로 변환하지 말 것”

**Windows 스타일 자료 형**

마이크로소프트에서 정의하고 있는 많은 양의 자료형 중에서 ANSI 기본 자료형에 대응하는 일부이다.

|  |  |  |  |
| --- | --- | --- | --- |
| Windows 자료형 | 정의형태 | Windows 자료형 | 정의형태 |
| BOOL | Typedef int BOOL | DWORD | Typedef Unsigned long DWORD |
| DWORD32 | Typedef unsigned int DWORD32 | DWORD64 | Typedef unsigned \_\_int64 DWORD64 |
| INT | Typedef int INT | INT32 | Typedef signed int INT32 |
| INT64 | Typedef signed \_\_int64 INT64 | LONG | Typedef long LONG |
| LONG32 | Typedef signed int LONG32 | LONG64 | Typedef signed \_\_int64 LONG64 |
| UINT | Typedef unsigned int INT | UINT32 | Typedef unsigned int INT32 |
| UINT64 | Typedef unsigned \_\_int64 INT64 | ULONG | Typedef unsigned int LONG |
| ULONG32 | Typedef unsigned int LONG32 | ULONG64 | Typedef unsigned \_\_int64 LONG64 |

#기본 자료형에 대한 마이크로소프트의 정의

위 표를 보면 WIN64로 넘어가면서, 새로운 자료형이 상당 수 추가되었을음알 수 있는데, ~32형태의 자료형은 시스템에 상관없이 32비트 로 표현되며, ~64 형태의 자료형은 시스템에 상관없이 64비트로 표현된다.

DWORD는 WIN32 기반 자료형 DWORD32는 WIN64기반 자료형, 이렇게 명확히 선을 그어서 생각하기 쉽지만 반드시 그런 것은 아니다. 자료형의 사용은 일종의 습관이기 때문에 어느 한 순간 쉽게 바뀌지 않는다. (64비트 환경에서도 정수형 변수로 주로 DWORD를 사용하는 것과 같은 이유이다.)

**Polymorphic 자료형**

마이크로소프트에서는 WIN64 기반으로 넘어 가면서 Polymorphic 자료형을 정의하고 있다. 보통 C++와 같은 객체지향 언어에서 처음 접하게 되는 이 Plymorphic이라는 단어는 “다양한 모습이 있는” 혹은 “다형적”이라는 뜻

자료형의 다형적이라는 뜻은 상황과 환경에 따라서 그 자료형이 의미하는 바가 유동적이라는 뜻, 마이크로소프트에서 정의하고 있는 Polymorphic 자료형의 정의형태는 다음과 같다.

#if defined (\_WIN64)

Typedef \_\_int64 LONG\_PTR;

Typedef unsigned \_\_int64 ULONG\_PTR;

#else

Typedef long LONG\_PTR;

Typedef unsigned long ULONG\_PTR;

#endif

PTR은 포인터로 생각하기 쉬우나, 정의된 자료형은 결코 포인터가 선언된 것이 아님을 알 수 있다. 이 뜻은 포인터값 기반의 산술연산을 위해 정의된 자료형 이기에 PTR이라는 이름이 붙은 것이다. 32 비트 시스템과 64비트 시스템의 포인터 정밀도(포인터의 크기)가 다르기 때문에 발생할 수 있는 문제를 해결하기 위해 등장한 자료형이다.

예제 1. PolymorphicType1.cpp

#32비트 기반 Windows 프로그래밍에 익숙한 자료 형

예제 2. PolymorphicType1.cpp

#Polymorphic 자료 형을 이용하여 예제 1을 재현, 64비트 환경에서는 64비트로, 32비트 환경에서는 32비트로 선언된다.

Polymorphic 자료 형은 32비트 시스템에서는 32비트로 64비트 시스템에서는 64비트로 선언

Section 03. 오류의 확인

**GetLastError 함수와 에러코드**

Windows시스템 함수를 호출하는 과정에서 오류가 발생하면, GetLastError 함수 호출을 통해 오류의 원인을 확인할 수 있다. 많은 수의 Windows 시스템 함수들은 오류가 발생했을 때 NULL을 반환한다. 반환되는 NULL 값으로 오류가 발생했음을 알 수 있지만, 원인은 알 수 없다.

오류가 발생했을 때, 이어서 바로 GetLastError 함수를 호츌하면 오류의 원인에 해당하는 에러코드를 얻을 수 있다.

예제 1. GetLastError.cpp

#시스템 에러코드를 얻는 방법을 확인하는 예제

예제 2. GetStateChange.cpp

위 예제2 번의 실행결과를 확인하면, Windows 시스템 함수가 호출될 때마다 GetLastError 함수가 반환하는 에러코드는 갱신된다. 위 결과에서도 오류가 발생하지 않은 시스템 함수호출후에 에러코드가 갱신되었음을 보여준다.

**요약정리**

1. 64비트 시스템과 32비트 시스템의 구조적 차이

64비트 시스템과 32비트 시스템을 구분짓는 기준 두 가지는 “한번에 송 수신할 수 있는 데이터 크기와 한번에 처리할 수 있는 데이터의 크기이다.” 64비트 환경에서는 한번에 64비트 데이터를 전송 및 처리할 수 있다.

1. 주소값 표현에 사용되는 바이트 수가 지니는 의미

프로그래머 관점에서 64비트 시스템은 주소를 표현하는데 64비트를 활용한다는 데 초점이 맞춰진다. 64비트 시스템은 한번에 처리할 수 있는 데이터의 범위가 64비트이므로, 주소값을 표현하는 데도 64비트를 활용한다. 프로그래머가 표현할 수 있는 범위를 넓혔다는데 의의가 있다.

1. Polymorphic 자료형

Polymorphic 자료형의 필요성은 예제를 통해서 익히는 것이 가장 좋다. 예제를 통해 이 자료형의 역할이 무엇인지 확인하자

1. LLP64와 LP64

LLP64와 LP64가 무엇인지 확인하자. 그리고 기존 32비트 시스템의 자료형과 비교해서 일치하는 것은 무엇인지, 또 일치하지 않는 것은 무엇인지 확인하자

1. GetLastError 함수의 사용방법

가장 많이 호출하는 함수 중에 하나이다. 윈도우 함수 호출 시 발생하는 에러의 원인을 알려주는 역할을 수행하는 함수이다.