2009년 06월 18일

Memory Map과 Symbol 이야기

Linker를 잘~ 다루기 위하여 메모리에 관련한 내용을 이것 저것 다시 한번 정리해 볼 셈입니다. 뭐 여러 가지 복잡한 얘기들이 난무했었다는 걸 상기한다면 이걸 다시 체계화 한다는 것은 상당히 뭔가 복합적인 의미가 있을 것 같습니다.

아방가르드하게 접근을 해본다면, 나름대로의 Category가 있습니다.

그것은 Symbol이라는 것과 그것이 아닌 것들 입니다. 이 Symbol이라는 것은 제가 앞에서 한번 언급한 적이 있습니다. Symbol이란, Linker가 알아볼 수 있는 기본 단위인데, Link를 한 후에는 자신만의 주소를 갖게 되는 특별한 단위를 말합니다. Symbol의 이름은 그 Symbol이 갖는 메모리 영역의 시작 주소를 가리키는 Linker만의 pointer이기요. Debugging시에 이런 Symbol의 이름이 사용되기도 하지요. 이는 전역변수의 이름이나, 함수이름이 그 예이며, 아주 중요한 의미이니 꼭 새겨 두세요. Linker를 위해서 ELF object file내에는 symbol table이라는 것을 두는데, source code에 의하여 참조되는 symbol들의 이름과 위치 정보가 들어 있으며, 다른 file에서 정의된 Symbol을 가져다 쓰는 경우에는 그 해당 file에 Symbol이 없기 때문에, 그 object에서는 symbol table은 완전하지 못합니다. 다른 file에서 선언된 이런 완전치 못한 symbol들은 한 개의 object file내에서 모두 처리되지는 못하는데, 이런 불완전한 symbol들은 linker에 의해서 처리하여 다른 파일에 있는 symbol을 연결하여 사용할 수 있도록 만듭니다. 그러면 Symbol은 Memory에 실제로 적재되는 내용인가 하면? 그건 아니에요. Linker만이 이 Symbol을 참조하며, 실제로 Linker는 이른 Symbol들을 주소로 모두 변환해서 binary로 만들어 준답니다.

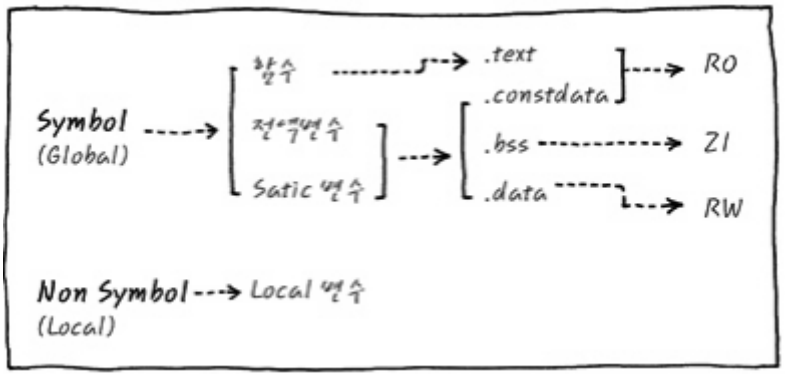
라고 했습니다. 결국 symbol이란 자기 자신만의 주소를 갖는 단위를 symbol이라고 부르며, 저는 Global이라고 명명합니다. - 뭐, 저는 나름 그렇게 구분합니다만, 아니라고 굳이 주장 하시는 분들도 있습니다. 아, 그렇습니까 정도로 하고요- 또한 symbol이 아닌 것들은 자기 자신만의 주소를 갖지 못하는 것들을 말하며, 또한 Local이라고 명명합니다. 이것 참, 곤란한 구분이지만, 저는 편리하게 사용하고 있습니다.

Global에는 함수, 전역변수, static 변수가 Global로 분류될 수 있을 것 같고요, Local에는 나머지 Local 변수입니다.

다시 말해, 함수, 전역변수, static 변수는 자기만의 주소를 가지며, Local 변수는 자기만의 주소를 갖지 못합니다. Symbol들, 즉 Global들이 자기만의 고유 주소를 갖는 다는 것은 다른 파일의 함수들에서도 직접 access하여 만질 수 있는 이유이기도 합니다.

오, 여기에서 한가지 더 나아가자면, ADS등에서는 RW, ZI, RO 세가지 형태의 구분도 있을 수 있는데, 이런 구분은 어떻게 나뉠까 한번 생각해 보시지요. RW는 read-write 로서, 초기값이 있는 전역변수를 의미하고, ZI는 Zero-initialized로서, 초기값이 0인 전역변수를 의미하고, 마지막으로 RO는 Read only로서 수정이 불가능한 const 전역변수와 text인 code를 의미합니다. 한가지만 더 comment하자면 전역변수 중에서도 RO로 분류되는 변수가 있는데, 그 녀석들은 const로 선언된 data들이에요. 후후. const는 Read Only 변수니까요.

그림으로 그려보면, 조금 더 간단할까요?



뭐 이 정도라고 할까요? 간단합니다. .text나 .bss 이런 식의 분류가 더 자세합니다만, RO, ZI, RW는 그 성질이 어떠하냐만 잘 보시면 됩니다. Read only는 읽기만 하니까, const global변수나,code를 의미하고, Read Write는 Global 변수 중 초기값이 있는 경우, 그리고 초기값이 0인 경우에는 모두 ZI로 구분하시면 됩니다. 엄청 간단간단입니다.

한가지 더, RO냐 RW, ZI만 구분하실 수 있으시면, Flash MCP를 따질 때 어디에 위치할 것이냐에 대해서도 따질 수가 있는 거지요. - Flash란 우리가 흔히 말하는 ROM을 말하는데, 전원이 꺼져도 그 내용을 계속 유지하는 메모리를 말합니다 - 예를 들어 볼까요, RO는 code하고, const data니까 우리는 항상 그 값을 유지하고 있어야 하니까, ROM, 즉 Flash에 그 내용이 들어 있어야 합니다. 또한 RW는 초기 값을 가지고 있는 전역변수니까, 언제라도 modify가능해야 하면서도 초기값을 가지고 있어야 합니다. 그러므로, ROM에 그 위치가 있어야 하며, 또한 그 값이 manipulation이 가능한 RAM에도 위치해야 합니다. 마지막으로 ZI는 그 값이 모두 0이므로 굳이 ROM에 저장될 필요도 없고, RAM에 위치하면 됩니다.

이런 것들을 자유자재로 구사하여 원하는 영역에 symbol 들을 특성에 따라 위치 시킬 수 있는 방법이 있는데, 이것이 linker script인 scatter loading file이라고 부르는 것이죠. 이 내용은 linker script 내용에서 talk talk하시고, 변수들이 메모리에 어떻게 자리 잡는지 정도만 가시지 말입니다.

이런 절대 주소를 갖는 Symbol 외에 절대 주소를 갖지 않는 그때 그때 사용하고 버리는 변수들이 있습니다. 그런 변수에는 Local 변수와 Dynamic Memory Allocation 변수들이 그렇습니다. Local 변수는 stack에 자리 잡으며, Dynamic Memory Allocation 변수들은 Heap이라는 곳에 자리 잡습니다. 이런 변수들은 Temporary로 사용하는 것들로서 굳이 절대 주소를 가질 필요가 없습니다. 이런 녀석들까지 모두 메모리에 잡으려면 엄청난 메모리 양이 필요하겠지요.

지금 symbol 얘기 한 걸로만 이용해서 아래 code에 있는 것들이 메모리에 어떻게 장착 되는지를 확인해 보시도록 하시죠.

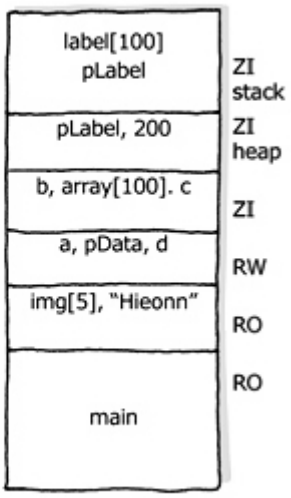
RW, ZI, RO로 구분해 보세요.

```
#include <stdio.h>

int a = 10;                          RW(.data)
int b = 0;                           ZI(.bss)
int array[10];                       ZI(.bss)
int const img[5] = {0,1,2,3,4};      RO(.constdata)
char *pData = "Hieonn"               pData → RW(.data),
                                     "Hieonn" → RO(.constdata)

main()                               RO (.text)
{
   static int c;                     ZI(.bss)
   static int d = 15;                RW(.data)
   char label [100];                 Stack
   char pLabel;                      Stack

   pLabel =                          Heap allocation
    (char *)malloc(sizeof(char)*200);
   .........                         Heap Free
   free (pLabel);
}
```



main()함수 위에 선언되어 있는 전역변수들은 RW, ZI, RO등의 영역에 자리 잡으며, main()함수자체는 RO 영역에 자리 잡습니다. main()함수 내부의 Local 변수들을 잘 보시면, static이라고 선언되어 있는 변수들은 Local이긴 하지만, 앞서 변수의 scope에서 살펴보았듯이, Global로 취급되므로 RW나 ZI에 속하게 되지요. 나머지 static이 아닌 Local 변수들은 label[100], pLabel은 satck에 pLabel은 heap에 자리 잡습니다. 뭐, 별거 아닌 것 같은 이야기 이지만, 이것만 알면 변수의 life span 따위의 원리는 모든 것이 설명되니, 잘 기억해 주세요.

이런 의미에서라면 Symbol이라 불리 우는 것들은 자신만의 주소를 가질 수 있기 때문에 모든 compile과 link 과정에서 기본 단위가 됨을 잊지 마세요.

마지막으로 한가지 더, Stack과 Heap 을 위와 같이 따로 표시했지만, 전역변수 array로 구현한 경우에는 staqck과 heap은 모두 ZI 영역에 포함 되는 것입니다. 결국 ZI = initial 되지 않은 전역변수 + initial 값이 0인 전역변수 + stack + heap 이라고 봐야 합니다. 으하하.

 : 자, 여기저기 보면 흘러나오는 용어들 중에 .data, .bss, .constdata, .text등을 다시 한번 구분해 보겠습니다. 얘네들은 RW, ZI, RO로 차례차례 Mapping될 수 있고요. Unix쪽에서 사용하는 용어에요. RW = .data, ZI = .bss, RO = .constdata + .text 로 말이죠. 이런 것들은 모두 Symbol (Global)들만이 그 구분을 가질 수 있죠.

뭐, 쉽게 말해서. .data는 초기값이 있는 전역변수, .bss는 초기값이 없는, 또는 Zero인 전역변수, .constdata는 const로 선언된 전역변수, text는 함수 등의 code가 자리잡을 수 있습니다.

컴파일러, 링커, 변수, Section, Memorymap

Linked at at 2009/06/18 20:16

... 러 option들 ⓙ 변수의 scope와 그 생애 (Memory Map) ⓚ Memory Map과 Symbol 이야기 ① ELF format Object File의 진실 ⓜ & ... more

Commented by 최경수 at 2009/07/31 22:01

좋은 글이네요. 많은 도움이 되었습니다 .

Commented by 히언 at 2009/08/02 00:05

얏호!! 감사합니다~~~