2009년 06월 23일

Memory Map과 Linker의 만남 - Locator

이제까지 Scatter Loading이니, Map file이니 잘 와닿지 않는 이야기가 주르르륵 이어져 버려서 어딘가 모자른 감이 없지 않아 있어 유감입니다. 하지만, 또 그런 걸 가지고 그냥 넘어갈 순 없는 노릇이죠. 이제까지 있었던 얘기를 총 동원해서 문제 2개를 풀어 볼 요량인데, 어때요. 재미있을까요? - 과연 - 어쨌거나 우리가 여러개의 c file을 가지고 Compiler을 하여 object file을 만들어 낸 후에 이 녀석들을 Linker를 통해서 묶어 내는 데, 내가 원하는 형태의 Memory Layout으로 묶어 내는 게 그 목적이니까 그것만 잊지 않으면 모든 것이 순탄하리라 생각해 버렸어요. 어쨌거나, 다음의 문제를 한 번 생각해 보세요.

문제1) embedded.c와 recipes.c를 32bit ARM mode로 컴파일 하여 object를 생성하고, main.s를 컴파일 하여 object를 생성하여, R0의 위치를 0x10000, RW의 위치를 0x20000에 relocate 시키고 싶어요. 어떻게 하면 좋을까요? 단, output은 elf 형식이어야 함.

문제2) embedded.c와 recipes.c를 ARM mode로 컴파일 하고요, boot.s를 컴파일 하는데, Load Region은 0x0에서 부터 시작하고, Execution Region에 boot.s의 Int_Vect를 역시나 0x0에 locate하고 싶어요. 그리고 나머지 RO를 Execution Region에 넣고 싶고요, 그리고 두 번째 Execution Region도 만들어서, 시작은은 0x4000에서부터 시작하게 하면서 embedded.c와 recipes.c의 RO를 넣고 싶고요. 마지막으로 세 번째 Execution Region을 만들어서 시작은 0x8000에서부터 시작하고 embedded.c, recipes.c, boot.s의 RW, ZI를 몽땅 집어 넣고 싶답니다 어떻게 하면 좋을까요? 단, output은 elf이어야 함.

캬아~ 어려워요 어려워. 말은 쉬운 데, 뭘 어쩌라는 건지. 그렇지요? 이런 문제는 시험문제로 나와도 아주 곤란한 문제지요. 하지만, 이제까지 잘 따라 오셨다면 그리 어렵지만은 않은 문제들이니까 너무 걱정말고 한번 생각해 보는 건 어떨까요? ..... 죄송. ㅋ
자, 답을 찾아가는 시간을 가져 볼 께요.

해답1) 이 문제는 의외로 간단한 문제에요. RO, RW, ZI가 여러군데로 나뉘어져 있지 않고 단순하게 연결되어 있지요. 이런 경우에는 어렵게 Scatter Lading을 사용하지 않고도 우리의 elf를 만들어 낼 수 있다구요. 직접 linker한테 명령을 주면 되어요.

```
armcc -c embedded.c recipes.c
armasm main.s
armlink -elf -ro-base 0x0 -rw-base 0x10000 -o target.elf embedded.o recipes.o main.o
```

오호, ro-base와 rw-base라는 optoin을 줄 수 있군요? 이럴수가.. 이런 얘기를 이제서야 꺼내다뉘... 다행이죠 머. 이제라도 꺼냈으니. ㅋ 아주 간단한 Memory Layout을 Simple Memory Layout이라고 부르고요, 이렇게 linker에게 직접적으로 parameter로 줄 수 있다구요. 에헤헤.

해답2) 자, 그러면 이 경우는 어떨까요? 아~주~ 복잡하지요. 이럴 때 바로 Scatter Loading의 위력이 발휘되는 거에요. 자, 일단은 우리 Scatter Loading을 작성해 볼까요.

```
LOAD_REGION 0x0
{
   EXE_REG_1 0x0
  {
   boot.o (+RO, Int_Vect)
  }
   EXE_REG_2 0x4000
  {
   embedded.o (+RO)
```

```
   recipes.o (+RO)
   }
  EXE_REG_3 0x8000
  {
   * (+RW, +ZI)
  }
}
```

이렇게 해보면 어떨까요? File이름은 Target.scl 이라고 이름 짓어보죠머.

```
LOAD_REGION 0x0               // Load Region이 0x0에서 시작하고요,
{
   EXE_REG_1 0x0              // 첫 번째 Execution Region이 0x0에서 시작하고요.
  {
   boot.o (+RO, Int_Vect)     // boot.o의 RO가 들어가는데, 그중에 Int_Vect Label이 젤 먼저 들어가구요.
  }
   EXE_REG_2 0x4000           // 두 번째 Execution Region이 0x4000에서 시작하고요.
  {
   embedded.o (+RO)           // 나머지 두개의 c file의 RO들이 자리 잡고요.
   recipes.o (+RO)
   }
  EXE_REG_3 0x8000            // 세 번째 Execution Region이 0x8000에서 시작하고요.
  {
   * (+RW, +ZI)               // 3개의 object에서 뽑아낸 RW, ZI가 모두 이곳에 자리 하고요.
  }
}
```

자, 어떤 가요? 일단 문제에서 요구한 내용들은 모두 적용 된 듯 보이네요. 그러면 이제 컴파일과 링크를 해야겠지요.

```
armcc -c embedded.c recipes.c
armasm main.s
armlink -elf -scatter Target.scl -o target.elf embedded.o recipes.o main.o
```

어때요!!! 간단하죠? 이런 식으로 link를 하게 되면 linker가 Relocatable Symbol들의 주소를 Scatter Loading에 description 된 것에 기초하여 주소를 할당 해 준답니다. 알고보니 별거 아니죠. 그런데. 여기서 한가지 더! 우리 Map file에 대해서 얘기 했었잖아요. 그러니까 Map file은 그냥 만들어지는 거냐 하면 그건 또 아니고요. Linker에게 map file을 만들어 내라고 알려줘야 해요. 그 option은 -map이라는 걸로 주고요, 출력하고 싶은 information은 -info라는 option을 줘서 알려줄 수 있어요. 뭐 너무 길게 쓰면 재미 없으니까, 예제 하나 읊어 볼게요.

```
armlink -elf -map -symbols -info sizes, totals, -list Target.map -scatter Target.scl -o target.elf embedded.o recipes.o main.o
```

오호, 요로코롬 해주면, Target.map이라는 Text file을 하나 만들어 주는데 map option에 의해서 Load/Execution Region에 대한 Input Section들의 정보를 출력해 주고요, -symbol에 의해서 Symbol 정보들도 출력이 되고요, -info에 의해서 각 input section들에 대한 크기 (size)와 그 합 (total) 정보를 ~~~ -list option의 Target.map이라는 file에 출력해 주는 거에요~ 냐함~
자, 이제까지 Scatter Loading, Map file을 계~속 봐온 이유를 이제 알 수 있겠나요?

linker, object, scatter, loading, memory, map, compiler, layout, 녀석들

Linked at at 2009/06/23 21:22

... ker Description Script ⓞ MAP file 분석 ⓟ Memory Map과 Linker의 만남 Locator ⓠ Makefile은 뭘하는 녀석일까~ ⓡ ... more

Commented by 감사요 at 2009/06/24 15:30

흐흐흐 . 드디어 따라잡았네요..

근데 점점 어려워지는 건.
실제 해 보면서 따라가야하는데, 눈팅만으로 따라가려니 여간 힘든게 아니네요.
그래도 담에 본다면 쉽게 접근할 수 있을 것 같네요.
쉽게 설명해 주셔서 감사합니다. 히언님.

Commented by 히언 at 2009/06/24 23:49

감사요님~ 너무 빨리 따라오시니까~ 제가 숨이 차요~