Systems Programming

Spring 2023

Q&A Session 진행 계획 (수정)

- 1주차 (3/9 목)
  - 01-linking
- **2주차 (3/16 목)**
  - 02-relocation
  - 03-exceptions
- **3주차 (3/23 목)**
  - 04-signals
- **4주차 (3/30 목)**
  - 05-io

- **Tentative (4/4 화)**
- **중간고사1 (4/6 목)**

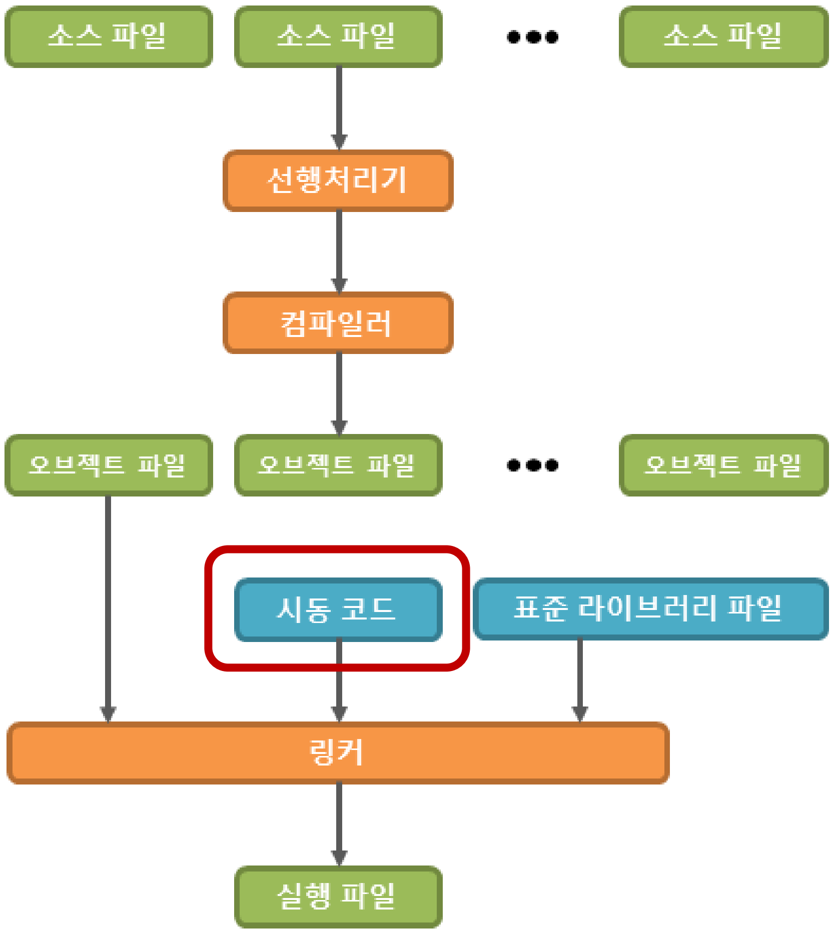
2 주차

- Review & Summary
  - 01-linking
  - 02-relocation
  - 03-exceptions
- Practices
  - 02-relocation
  - 03-exceptions; fork()
- Q&A

01-linking

Questions

하나의 object file로 구성되어 있으며 어떠한 라이브러리도 참조하지 않는 모듈의 경우, relocatable object file인 동시에 executable object file인가요? 만약 아니라면 이런 경우 executable은 object file과 무엇이 다른지 궁금합니다.



- Startup code
  - Small block of assembly code
  - prepares the way for the execution of software written in a high-level language
- Startup code for C/C++ programs
  1. Disable all interrupts
  2. Copy any initialized data from ROM to RAM
  3. Zero the uninitialized data area
  4. Allocate space for and initialize the stack
  5. Initialize the processor's stack pointer
  6. Create and Initialize the heap
  7. Execute the constructors and initializers for all global variables (C++ only)
  8. Enable interrupts
  9. Call main

http://www.tcpschool.com/c/c_intro_programming *https://t.ly/xQYx*

02-relocation

Questions

수업 자료의 relocation algorithm은 교재 727페이지의 relocation algorithm과 다른데, 왜 수업자료에서는 r.addend 대신에 *refptr가 들어가게 되는지 궁금합니다.

2nd Edition

Relocation Algorithm – x86(Elf32_rel)

```
typedef struct {
    int offset;
    int symbol:24,
        type:8;
} Elf32_rel;
```

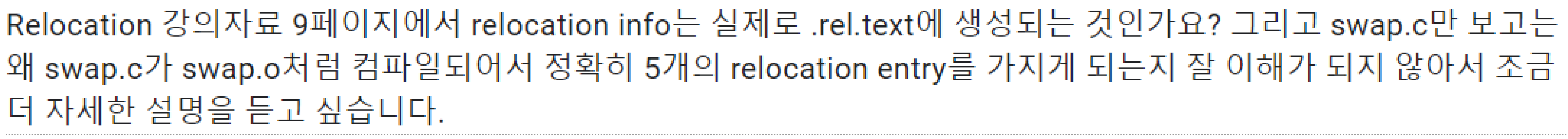
```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }
}
```

Relocation Algorithm – x64(Elf64_Rela)

```
/* $begin elfrelo */
typedef struct {
    long offset;    /* Offset of the reference to relocate */
    long type:32,   /* Relocation type */
         symbol:32; /* Symbol table index */
    long addend;    /* Constant part of relocation expression */
} Elf64_Rela;
/* $end elfrelo */
```

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_X86_64_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_X86_64_32)
            *refptr = (unsigned) (ADDR(r.symbol) + r.addend);
    }
}
```

Questions



Relocation 강의자료 9페이지에서 relocation info는 실제로 .rel.text에 생성되는 것인가요? 그리고 swap.c만 보고는 왜 swap.c가 swap.o처럼 컴파일되어서 정확히 5개의 relocation entry를 가지게 되는지 잘 이해가 되지 않아서 조금 더 자세한 설명을 듣고 싶습니다.

- Relocation entries for code are placed in .rel.text

- Relocation entries for initialized data are placed in .rel.data

Questions

relocation 시에 R_386_32 type인 symbol은 absolute address를 가진다고 했는데, 왜 relocation할 때 array offset을 더해주어야 하나요?

Questions

relocation에서 ppt 17~22쪽에 있는 예제들을 ppt7쪽에 있는 relocation algorithm에 맞춰 설명해 주셨으면 좋겠습니다. 예를 들어서, ppt 17쪽의 main.o의 relocation을 보면 refaddr = ADDR(s) + r.offset 이라는 식에서 ADDR(s)= 0x4004ed, r.offset =a 이므로 refaddr 0x4004f7 되고, *refptr = (unsigned) (ADDR(r.symbol) + * refptr - refaddr 식에서는 ADDR(r.symbol)=0x400502이 된다고 이해했습니다. 그러면 *refptr의 값이 relocation entry에서의 value 값과 같은데 refptr의 정의(refptr= s+ r.offset;)로부터 이게 어떻게 value 값과 같아지는지 잘 이해가 안 됩니다. (value 값이 무엇인지 제대로 이해하지 못한 것 같기도 합니다.)

Relocation entry에서 value에 대해 좀 더 자세히 설명해주셨으면 좋겠습니다. 제가 이해한 바로는 value = symbol reference의 (run time)주소와 PC 값의 차이라고 이해했는데 이게 왜 차이가 나는지를 잘 모르겠습니다.

Questions

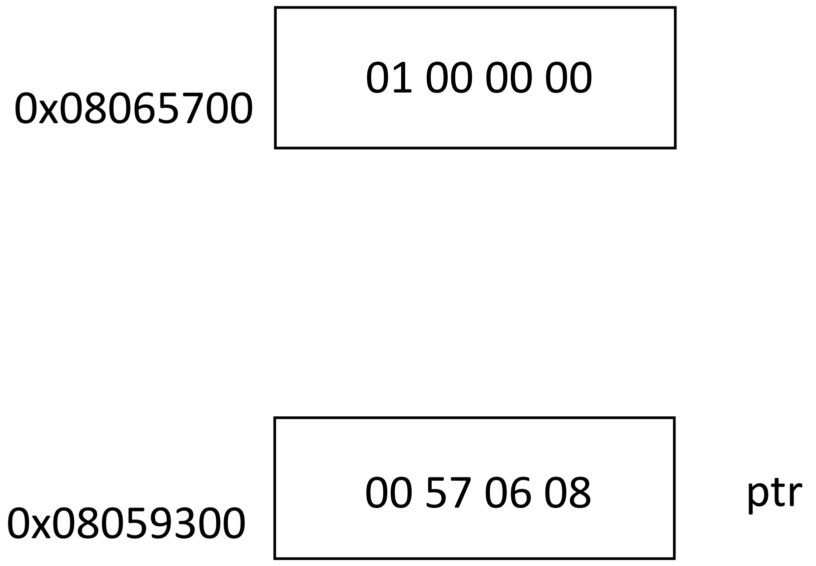
2. 02- relocation 8p 9p
오른쪽 위에 relocation entry를 보면 앞서 배운 entry 내용 말고도 value항목이 추가되어있는데 이 또한 relocation entry에 포함되어야하는 정보인가요? 아니면 .o파일에서 relocation이 필요한 위치에 저장되어있으면 되는 값인가요?(fc ff ff ff가 12번지에 있는것처럼)

3. 02- relocation 14p 15p
2번 질문이랑 연동되는 질문인데 이쪽 코드에서는 수행시 pc보정값을 .o 파일의 text값 내부에 포함시키지 않고 따로 오른쪽에 -0x8, -0x4이런식으로 표현했는데 이는 앞에서 나온 8p 9p의 표현법과 달라서 질문드립니다.

Relocation Practices #0 – Warming up

- Pointer basics

```
int *ptr = 1;
```



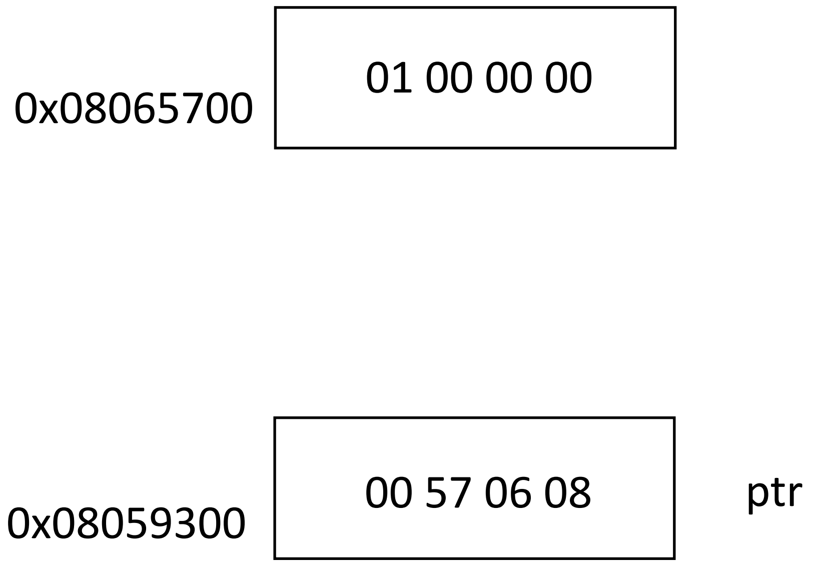
ptr = ?

*ptr = ?

Relocation Practices #0 – Warming up

- Pointer basics

```
int *ptr = 1;
```



ptr = 0x08065700
*ptr = 0x1

Relocation Algorithm

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }
}
```

Relocation Algorithm

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }
}
```

Relocation Revisit #1 [02-relocation.pdf (p.11)]

- PC-Relative References

```
Disassembly of section .text:

0000000000000000 <main>:
   11:  e8 fc ff ff ff          callq  12 <main+0x12>
                        a: R_386_PC32   <swap>
```

```
r.offset = 0x12
r.symbol = swap
r.type = R_386_PC32
(value = -4 ?????)
```

```
ADDR(s)                 = ADDR(.text)   = 0x08048380
ADDR(r.symbol)          = ADDR(swap)    = 0x080483b0
                              refptr  = ?????
                              *refptr = ?????
```

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
                  = 0x08048380 + 0x12 = 0x08048392
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }        = (unsigned) (0x080483b0 + *refptr - 0x08048392)
```

Relocation Revisit #1 [02-relocation.pdf (p.11)]

- PC-Relative References

```
Disassembly of section .text:

0000000000000000 <main>:
  11:  e8 fc ff ff ff          callq  12 <main+0x12>
                       a: R_386_PC32   <swap>
```

```
r.offset = 0x12
r.symbol = swap
r.type = R_386_PC32
(value = -4 !!!!!!)
```

```
ADDR(s)              = ADDR(.text)   = 0x08048380
ADDR(r.symbol)       = ADDR(swap)    = 0x080483b0
                         refptr  = 0x08048392 !!!!!
                         *refptr = fc ff ff ff (= -4) !!!!!
  foreach section s {    → changed to 0x1a after relocation
      foreach relocation entry r {
          refptr = s + r.offset;  /* ptr to reference to be relocated */

          /* Relocate a PC-relative reference */
          if (r.type == R_386_PC32) {
              refaddr = ADDR(s) + r.offset; /* ref's run-time address */
                    = 0x08048380 + 0x12 = 0x08048392
              *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
          }         = (unsigned) (0x080483b0 + (-4) - 0x08048392)
                    = (unsigned) (0x1a)
```

Relocation Revisit #1 [02-relocation.pdf (p.11)]

- PC-Relative References

```
Disassembly of section .text:

0000000000000000 <main>:
   11:  e8 fc ff ff ff          callq  12 <main+0x12>
                        a: R_386_PC32   <swap>
```

```
r.offset = 0x12
r.symbol = swap
r.type = R_386_PC32
(value = -4)
```

```
ADDR(s)                 = ADDR(.text)   = 0x08048380
ADDR(r.symbol)          = ADDR(swap)    = 0x080483b0
```

- After relocation

```
Disassembly of section .text:

0000000008048380 <main>:
 8048391:      e8 1a 00 00 00          callq  80483b0 <swap>
 8048396:
```

Relocation Practices #1

- PC-Relative References

```
r.offset = 0x7
r.symbol = swap
r.type = R_386_PC32
```

```
Disassembly of section .text:

0000000000000000 <main>:
   6:    e8 fc ff ff ff          callq  7 <main+0x7>
                       7:    R_386_PC32 <swap>
```

```
ADDR(s)               = ADDR(.text)   = 0x080483b4
ADDR(r.symbol)        = ADDR(swap)    = 0x080483c8
```

```
Disassembly of section .text:

00000000080483b4 <main>:
 80483ba:       e8 ?? ?? ?? ??         callq  ?? ?? ?? ?? <swap>
```

Relocation Practices #1 – Check

- PC-Relative References

```
Disassembly of section .text:

0000000000000000 <main>:
   6:   e8 fc ff ff ff          callq  7 <main+0x7>
                        7:    R_386_PC32 <swap>
```

```
r.offset = 0x7
r.symbol = swap
r.type = R_386_PC32
(value = -4)
```

```
ADDR(s)                 = ADDR(.text)   = 0x080483b4
ADDR(r.symbol)          = ADDR(swap)    = 0x080483c8
                              refptr  = 0x080483bb
                              *refptr = -4

  foreach section s {
      foreach relocation entry r {
          refptr = s + r.offset;  /* ptr to reference to be relocated */

          /* Relocate a PC-relative reference */
          if (r.type == R_386_PC32) {
              refaddr = ADDR(s) + r.offset; /* ref's run-time address */
                    = 0x080483b4 + 0x7 = 0x080483bb
              *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
          }       = (unsigned) (0x080483c8 + (-4) - 0x080483bb)
                  = (unsigned) (0x9)
```

Relocation Practices #1 – Done!

- PC-Relative References

```
Disassembly of section .text:

0000000000000000 <main>:
   6:    e8 fc ff ff ff          callq  7 <main+0x7>
                       7:    R_386_PC32 <swap>
```

```
r.offset = 0x7
r.symbol = swap
r.type = R_386_PC32
(value = -4)
```

```
ADDR(s)              = ADDR(.text)   = 0x080483b4
ADDR(r.symbol)       = ADDR(swap)    = 0x080483c8
```

- After relocation

```
Disassembly of section .text:

00000000080483b4 <main>:
 80483ba:       e8 09 00 00 00          callq  80483c8 <swap>
```

Relocation Revisit #2 [02-relocation.pdf (p.12)]

- Absolute References

```
r.offset = 0x2
r.symbol = buf
r.type = R_386_32
(value = 0)
```

```
Disassembly of section .text:

0000000000000000 <swap>:
   0:    8b 15 00 00 00 00              mov    0x0, %edx
                        2: R_386_32     buf
```

```
ADDR(r.symbol)          = ADDR(buf)     = 0x08049620
```

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }                  = (unsigned) (0x08049620 + 0)
}                      = (unsigned) (0x08049620)
```

Relocation Revisit #2 [02-relocation.pdf (p.12)]

```
r.offset = 0x2
r.symbol = buf
r.type = R_386_32
(value = 0)
```

- Absolute References

```
Disassembly of section .text:

0000000000000000 <swap>:
   0:    8b 15 00 00 00 00                        mov     0x0, %edx
                         2: R_386_32              buf
```

```
ADDR(r.symbol)            = ADDR(buf)      = 0x08049620
```

- After relocation

```
Disassembly of section .text:

00000000080483b0 <swap>:
 80483b0: 8b 15 20 96 04 08                       mov     0x08049620, %edx
```

Relocation Entries

```
typedef struct {
   int offset;
   int symbol:24,
       type:8;
   int value;
} Elf32_rel;
```

- Offset : section offset of the references to relocate
- Symbol : identifies the symbol that the modified reference should point to.
- Type : tells the linker how to modify the new reference
- Value : PC adjustment / array offset (sometimes in the .o file)
- ELF defines 11 relocation types.
- two most widely used :
  - R_386_PC32 : 32 bit PC_relative address
    - Add the 32 bit value to the current PC value (address of the next instruction)
  - R_386_32 : 32 bit absolute address

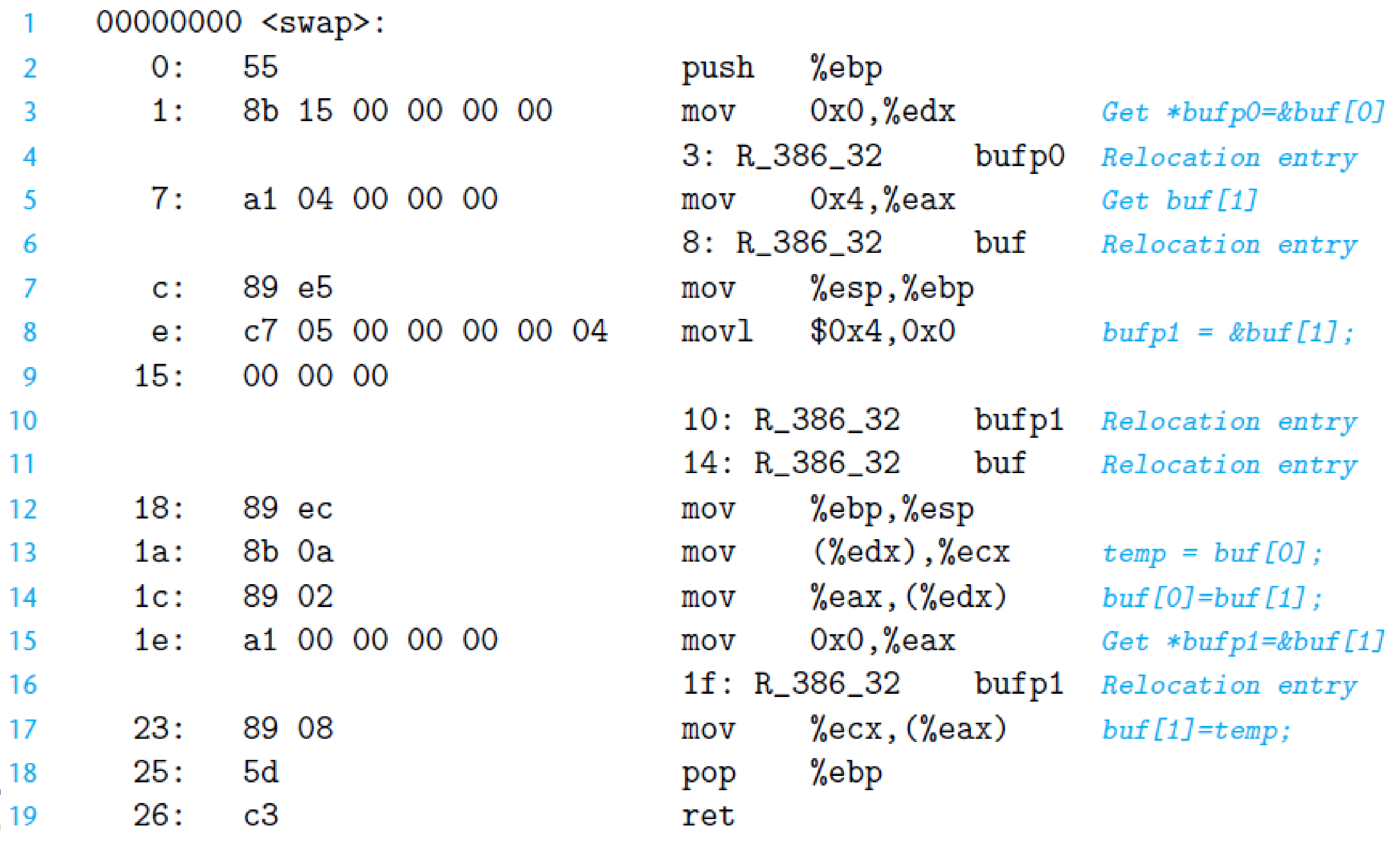
Relocation Algorithm

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_386_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
        }
                                                  ↖
                                             PC adjustment

        /* Relocate an absolute reference */
        if (r.type == R_386_32)
            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
    }                                             ↖
}                                              Array offset
```

Some compiler/linker store the value separately in relocation entry

Questions

1. 02- relocation 8p 9p
혹시 relocation부분 말고도 나머지 assembly코드가 완성되는 과정을 간단하게 설명해주실 수 있으실까요? 특히 9p에서 swap이 진행될 때 buf[0]의 위치에 있는 값과 buf[1]의 위치에 있는 값이 불리는 횟수는 동일해야 할텐데 swap.o를 보면 buf[0]는 relocation entry가 하나밖에 없는 반면 buf[1]은 relocation entry가 3개나 들어가는 등 .c 파일이 .o파일로 바뀌는 과정에 이해가 안가는 부분이 많아 질문드립니다.


```
00000000 <swap>:
   0:   55                      push   %ebp
   1:   8b 15 00 00 00 00       mov    0x0,%edx           Get *bufp0=&buf[0]
                                3: R_386_32      bufp0    Relocation entry
   7:   a1 04 00 00 00          mov    0x4,%eax           Get buf[1]
                                8: R_386_32      buf      Relocation entry
   c:   89 e5                   mov    %esp,%ebp
   e:   c7 05 00 00 00 00 04    movl   $0x4,0x0           bufp1 = &buf[1];
  15:   00 00 00
                                10: R_386_32     bufp1    Relocation entry
                                14: R_386_32     buf      Relocation entry
  18:   89 ec                   mov    %ebp,%esp
  1a:   8b 0a                   mov    (%edx),%ecx        temp = buf[0];
  1c:   89 02                   mov    %eax,(%edx)        buf[0]=buf[1];
  1e:   a1 00 00 00 00          mov    0x0,%eax           Get *bufp1=&buf[1]
                                1f: R_386_32     bufp1    Relocation entry
  23:   89 08                   mov    %ecx,(%eax)        buf[1]=temp;
  25:   5d                      pop    %ebp
  26:   c3                      ret
```

03-exceptions

Questions

Exception 8쪽에서 interrupts는 handler returns to "next" instruction라고 나와있는데, 프로세스를 종료하는 ctrl-c나 soft/hard reset에서도 유효한 이야기인가요?

Questions

질문2 : (단원 : 3. exceptions) child의 child(grandchild?)도 child 취급을 하나요? (자동 reaping등등)

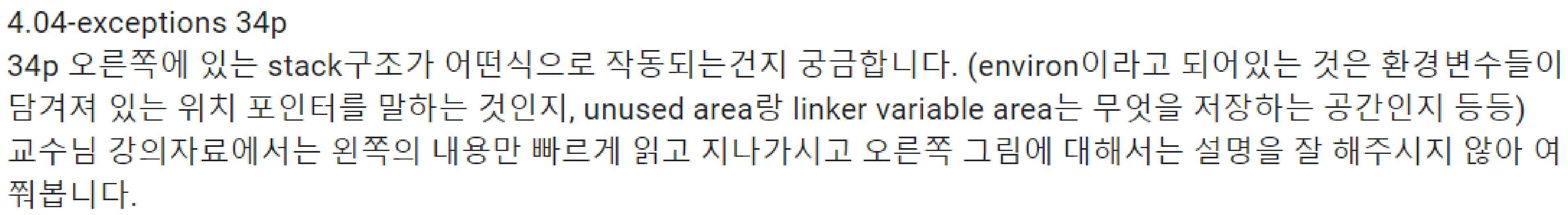
- Orphan process
  - A process that is still executing, but whose parent has died
  - When the parent dies, the orphaned child process is adopted by `init` (pid 1)

- When orphan processes die,
  - they do not remain as zombie processes
  - they are `wait`ed on by `init`

Questions

질문1 : (단원 : 3. exceptions) parent가 terminate될 시 init에 의해서 child process들이 reap되는데, reap되는 child 들의 순서가 정해져 있나요?

Questions


4.04-exceptions 34p

34p 오른쪽에 있는 stack구조가 어떤식으로 작동되는건지 궁금합니다. (environ이라고 되어있는 것은 환경변수들이 담겨져 있는 위치 포인터를 말하는 것인지, unused area랑 linker variable area는 무엇을 저장하는 공간인지 등등) 교수님 강의자료에서는 왼쪽의 내용만 빠르게 읽고 지나가시고 오른쪽 그림에 대해서는 설명을 잘 해주시지 않아 여쭤봅니다.

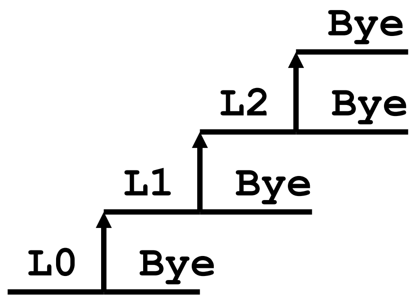
- The global variable environ points to the first of these pointers, envp[].
- unused area between the ‘envp[]’ areas and the ‘null-terminated command line argument strings’
  - future expansion of the environment variable area (e.g., modified at runtime)
  - prevent buffer overflow vulnerabilities
- example of ‘dynamic linker variables’
  - Environment variables
  - Function arguments
  - …

fork() Revisit

```
void fork5()
{
    printf("L0\n");
    if (fork() == 0) {
        printf("L1\n");
        if (fork() == 0) {
            printf("L2\n");
            fork();
        }
    }
    printf("Bye\n");
}
```

fork() Revisit [03-exceptions.pdf (p.25)]

```
void fork5()
{
    printf("L0\n");
    if (fork() == 0) {
        printf("L1\n");
        if (fork() == 0) {
            printf("L2\n");
            fork();
        }
    }
    printf("Bye\n");
}
```



fork() Practice

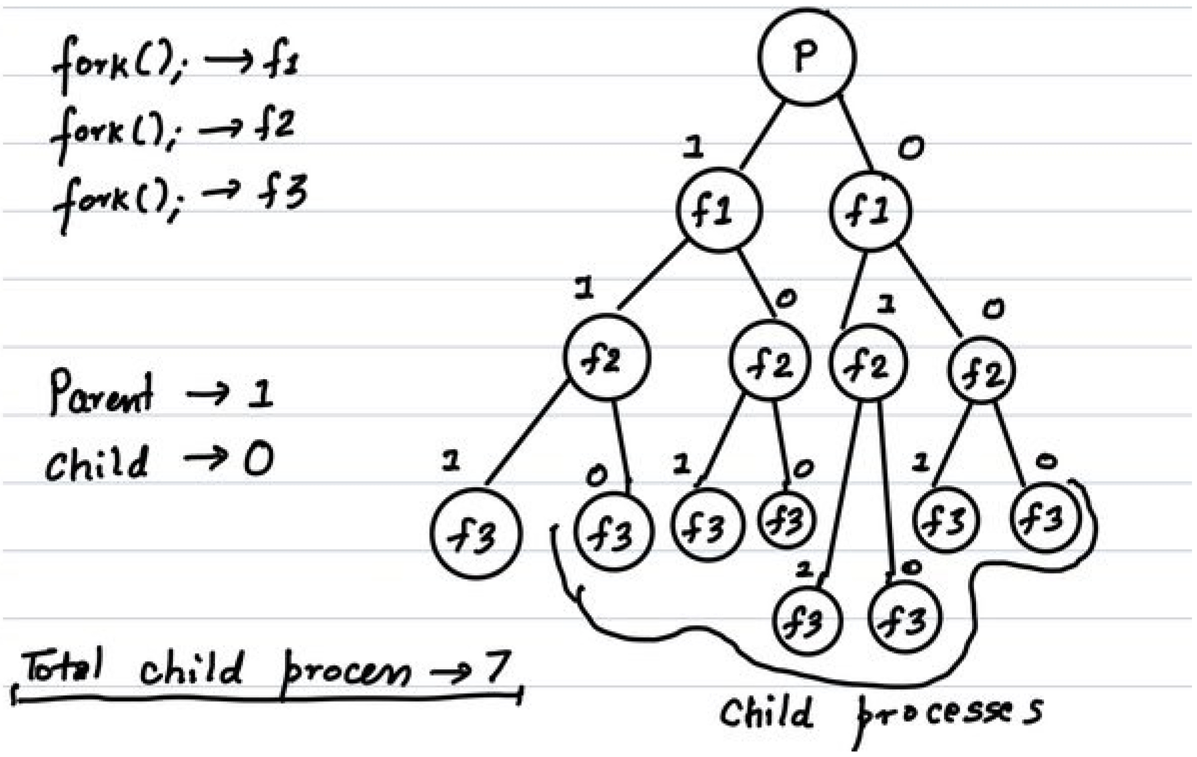
- How many child processes?

```
fork();
fork();
fork();
```

fork() Practice – Check

- How many child processes?

```
fork();
fork();
fork();
```



숙제

- 제출방법
  - pdf 파일
  - eTL 에 과제 (2주차 모듈) 생성
    - relocation-숙제

- 기한
  - 3/19 (이번주 일요일) 11:59 PM

Executable Before/After Relocation

-4(수행시의 PC 보정값)

```
0000000 <main>:
   . . .
   e:   83 ec 04            sub     $0x4,%esp
  11:   e8 fc ff ff ff      call    12 <main+0x12>
                  12: R_386_PC32 swap
  16:   83 c4 04            add     $0x4,%esp
  . . .
```

```
08048280 <main>:
 .   .   .
804828e:       83 ec 04                    sub    $0x4,%esp
8048291:       e8 ?? ?? ?? ??              call   80502ca <swap>
8048296:       83 c4 04                    add    $0x4,%esp
 .   .   .
```

(1) Call swap()을 수행할때의 PC의 값은 ?
(2) swap() 의 주소가 080502ca 라고 하면 relocation이 끝나서 ?? ?? ?? ?? 에 들어갈 값을 계산하시오.

Before Relocation (.text) swap.o

(수행시의 PC 보정값)

```
0000000000000000 <swap>:
   0:    55                         push    %rbp
   1:    48 89 e5                   mov     %rsp,%rbp
   4:    48 c7 05 00 00 00 00       movq    $0x0,0x0(%rip)          # f <swap+0xf>
                       7: R_386_PC32         bufp1   -0x8
   b:    00 00 00 00
                        b: R_386_32 buf+0x4
   f:    48 8b 05 00 00 00 00       mov     0x0(%rip),%rax          # 16 <swap+0x16>
                        12: R_386_PC32        bufp0   -0x4
  16:    8b 00                      mov     (%rax),%eax
  18:    89 45 fc                   mov     %eax,-0x4(%rbp)
  1b:    48 8b 05 00 00 00 00       mov     0x0(%rip),%rax          # 22 <swap+0x22>
                        1e: R_386_PC32        bufp0   -0x4
  22:    48 8b 15 00 00 00 00       mov     0x0(%rip),%rdx          # 29 <swap+0x29>
                        25: R_386_PC32        bufp1   -0x4
  29:    8b 12                      mov     (%rdx),%edx
  2b:    89 10                      mov     %edx,(%rax)
  2d:    48 8b 05 00 00 00 00       mov     0x0(%rip),%rax          # 34 <swap+0x34>
                        30: R_386_PC32        bufp1   -0x4
  34:    8b 55 fc                   mov     -0x4(%rbp),%edx
  37:    89 10                      mov     %edx,(%rax)
  39:    5d                         pop     %rbp
  3a:    c3                         retq
```

Swap – Before Relocation

```
0000000000000000 <swap>:
   0:    55                           push    %rbp
   1:    48 89 e5                     mov     %rsp,%rbp
   4:    48 c7 05 00 00 00 00         movq    $0x0,0x0(%rip)        # f <swap+0xf>
                        7: R_386_PC32          bufp1   -0x8
   b:    00 00 00 00
                        b: R_386_32S buf+0x4
   f:    48 8b 05 00 00 00 00         mov     0x0(%rip),%rax        # 16 <swap+0x16>
```

Swap – After Relocation

```
0000000000400500 <swap>:
  400500:   55                          push    %rbp
  400501:   48 89 e5                    mov     %rsp,%rbp
  400504:   48 c7 05 ?? ?? ?? ??        movq    $0x60103c,????????(%rip) #600000 <bufp1>
  40050b:   3c 10 60 00
  40050f:   …                                (next instruction)
```

```
Disassembly of section .bss:
0000000000600000 <bufp1>:
```

PC-relative Address:

(3) movq를 수행할 시점의 PC는 ? (4) ?? ?? ?? ?? 에 들어갈 주소값을 구하시오.