운영체제 보고서

빅데이터 학과

20175119 김영식

(1) 다음 코드를 동적 링킹으로 컴파일 및 정적 링킹으로 컴파일 해서 실행파일의 크기를 비교하세요.

```
hunterspin@ubuntu:~/Desktop/practice$ vi address.c
hunterspin@ubuntu:~/Desktop/practice$ gcc -o address_static address.c -static
hunterspin@ubuntu:~/Desktop/practice$ gcc -o address_dynamic address.c
```

```
hunterspin@ubuntu:~/Desktop/practice/address$ ls -alh
total 884K
drwxrwxr-x 2 hunterspin hunterspin 4.0K Apr 30 23:22 .
drwxrwxr-x 3 hunterspin hunterspin 4.0K Apr 30 23:22 ..
-rw-rw-r-- 1 hunterspin hunterspin  270 Apr 30 23:20 address.c
-rwxrwxr-x 1 hunterspin hunterspin  17K Apr 30 23:21 address_dynamic
-rwxrwxr-x 1 hunterspin hunterspin 852K Apr 30 23:21 address_static
```

동적 링킹으로 컴파일 했을 경우 17K정도의 파일크기가 생성되었고 정적링킹으로 컴파일 했을 경우는 852K정도의 파일크기가 생성 되었는데 같은 소스파일을 컴파일 한 것인데도 불구하고 엄청 큰 파일크기 차이를 나타낸다.

정적링킹을 했을 때는 모든 모듈(함수)을 다포함하고 있기 때문에 파일크기가 커진다.

Ex)printf함수등

(2) 그리고 해당 프로세스의 메모리맵을 출력해보기 바랍니다.

```
hunterspin@ubuntu:~/Desktop/practice/address$ ./address_dynamic&
[1] 2313
hunterspin@ubuntu:~/Desktop/practice/address$ num1 value is 3
num1 address is 0x55f949b6f010
num2 value is 4
num2 address is 0x55f949b6f014

hunterspin@ubuntu:~/Desktop/practice/address$ ps -ef | grep address
message+     756       1  0 23:19 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
hunters+    1394    1383  0 23:20 ?        00:00:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
hunters+    1580    1575  0 23:20 ?        00:00:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
hunters+    1684    1383  0 23:20 ?        00:00:00 /usr/libexec/evolution-addressbook-factory
hunters+    2313    1910  0 23:37 pts/0    00:00:00 ./address_dynamic
hunters+    2315    1910  0 23:37 pts/0    00:00:00 grep --color=auto address
hunterspin@ubuntu:~/Desktop/practice/address$ cat /proc/2313/maps
55f949b6b000-55f949b6c000 r--p 00000000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
55f949b6c000-55f949b6d000 r-xp 00001000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
55f949b6d000-55f949b6e000 r--p 00002000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
55f949b6e000-55f949b6f000 r--p 00002000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
55f949b6f000-55f949b70000 rw-p 00003000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
55f94a98a000-55f94a9ab000 rw-p 00000000 00:00 0                          [heap]
7f7efaf29000-7f7efaf4b000 r--p 00000000 08:05 526484                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7efaf4b000-7f7efb0c3000 r-xp 00022000 08:05 526484                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7efb0c3000-7f7efb111000 r--p 0019a000 08:05 526484                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7efb111000-7f7efb115000 r--p 001e7000 08:05 526484                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7efb115000-7f7efb117000 rw-p 001eb000 08:05 526484                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7efb117000-7f7efb11d000 rw-p 00000000 00:00 0
7f7efb12c000-7f7efb12d000 r--p 00000000 08:05 526477                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7efb12d000-7f7efb150000 r-xp 00001000 08:05 526477                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7efb150000-7f7efb158000 r--p 00024000 08:05 526477                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7efb159000-7f7efb15a000 r--p 0002c000 08:05 526477                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7efb15a000-7f7efb15b000 rw-p 0002d000 08:05 526477                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7efb15b000-7f7efb15c000 rw-p 00000000 00:00 0
7fff4d606000-7fff4d627000 rw-p 00000000 00:00 0                          [stack]
7fff4d74c000-7fff4d750000 r--p 00000000 00:00 0                          [vvar]
7fff4d750000-7fff4d752000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

```
hunterspin@ubuntu:~/Desktop/practice/address$ ./address_static&
[2] 2321
[1]   Terminated              ./address_dynamic
hunterspin@ubuntu:~/Desktop/practice/address$ num1 value is 3
num1 address is 0x4c00f0
num2 value is 4
num2 address is 0x4c00f4

hunterspin@ubuntu:~/Desktop/practice/address$ ps -ef|grep address
message+     756       1  0 23:19 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
hunters+    1394    1383  0 23:20 ?        00:00:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
hunters+    1580    1575  0 23:20 ?        00:00:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
hunters+    1684    1383  0 23:20 ?        00:00:00 /usr/libexec/evolution-addressbook-factory
hunters+    2321    1910  0 23:38 pts/0    00:00:00 ./address_static
hunters+    2323    1910  0 23:38 pts/0    00:00:00 grep --color=auto address
hunterspin@ubuntu:~/Desktop/practice/address$ cat /proc/2321/maps
00400000-00401000 r--p 00000000 08:05 1062392                            /home/hunterspin/Desktop/practice/address/address_static
00401000-00495000 r-xp 00001000 08:05 1062392                            /home/hunterspin/Desktop/practice/address/address_static
00495000-004bc000 r--p 00095000 08:05 1062392                            /home/hunterspin/Desktop/practice/address/address_static
004bd000-004c0000 r--p 000bc000 08:05 1062392                            /home/hunterspin/Desktop/practice/address/address_static
004c0000-004c3000 rw-p 000bf000 08:05 1062392                            /home/hunterspin/Desktop/practice/address/address_static
004c3000-004c4000 rw-p 00000000 00:00 0
0162e000-01651000 rw-p 00000000 00:00 0                                  [heap]
7fff22edf000-7fff22f00000 rw-p 00000000 00:00 0                          [stack]
7fff22f48000-7fff22f4c000 r--p 00000000 00:00 0                          [vvar]
7fff22f4c000-7fff22f4e000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

동적 링킹은 정적 링킹과는 다르게 라이브러리 관련 메모리 맵이 없다.

(3) 아래 코드에서 num1 과 num2 가 메모리 맵에서 어떤 곳에 위치해 있는지 찾아보세요.

메모리맵을 보면 시작주소와 끝주소가 있는데, 어떤 부분에 해당 변수가 존재하는지 확인해보면 됩니다..

```
5605b4095000-5605b4096000 rw-p 00003000 08:05 1062401                    /home/hunterspin/Desktop/practice/address/address_dynamic
```

num1 address is 0x5605b4095010

num2 address is 0x5605b4095014

주소가 저 범위에 존재하기 때문에 전역변수는 저 곳에 존재한다.

확인결과 동적링킹과 정적링킹 모두 이 부분에 전역변수가 존재한다.

(4) objdump -D 실행파일 을 수행하면 기계어 코드가 출력이 되는데 실제 printf 함수가 정적링킹과 동적링킹에서 어떤 차이가 있는지 분석해서 보고서를 작성하기 바랍니다. 실행파일이 address라고 했을때에 다음과 같이 파일로 저장한 후에 nano, vi 에디터로 내용을 확인해 볼 수 있습니다.

```
Disassembly of section .plt.sec:

0000000000001060 <printf@plt>:
    1060:       f3 0f 1e fa             endbr64
    1064:       f2 ff 25 5d 2f 00 00    bnd jmpq *0x2f5d(%rip)        # 3fc8 <printf@GLIBC_2.2.5>
    106b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000001070 <sleep@plt>:
    1070:       f3 0f 1e fa             endbr64
    1074:       f2 ff 25 55 2f 00 00    bnd jmpq *0x2f55(%rip)        # 3fd0 <sleep@GLIBC_2.2.5>
    107b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

Disassembly of section .text:

0000000000001080 <_start>:
    1080:       f3 0f 1e fa             endbr64
    1084:       31 ed                   xor    %ebp,%ebp
    1086:       49 89 d1                mov    %rdx,%r9
    1089:       5e                      pop    %rsi
    108a:       48 89 e2                mov    %rsp,%rdx
    108d:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
    1091:       50                      push   %rax
    1092:       54                      push   %rsp
    1093:       4c 8d 05 c6 01 00 00    lea    0x1c6(%rip),%r8        # 1260 <__libc_csu_fini>
    109a:       48 8d 0d 4f 01 00 00    lea    0x14f(%rip),%rcx        # 11f0 <__libc_csu_init>
    10a1:       48 8d 3d c1 00 00 00    lea    0xc1(%rip),%rdi        # 1169 <main>
    10a8:       ff 15 32 2f 00 00       callq  *0x2f32(%rip)        # 3fe0 <__libc_start_main@GLIBC_2.2.5>
    10ae:       f4                      hlt
    10af:       90                      nop
```

동적링킹에서는 간단하게 함수 같은 것들을 참조해서 가져온다.

```
0000000000401ce5 <main>:
  401ce5:       f3 0f 1e fa             endbr64
  401ce9:       55                      push   %rbp
  401cea:       48 89 e5                mov    %rsp,%rbp
  401ced:       8b 05 fd e3 0b 00       mov    0xbe3fd(%rip),%eax        # 4c00f0 <num1>
  401cf3:       89 c6                   mov    %eax,%esi
  401cf5:       48 8d 3d 08 33 09 00    lea    0x93308(%rip),%rdi        # 495004 <_IO_stdin_used+0x4>
  401cfc:       b8 00 00 00 00          mov    $0x0,%eax
  401d01:       e8 ba ec 00 00          callq  4109c0 <_IO_printf>
  401d06:       48 8d 35 e3 e3 0b 00    lea    0xbe3e3(%rip),%rsi        # 4c00f0 <num1>
  401d0d:       48 8d 3d 02 33 09 00    lea    0x93302(%rip),%rdi        # 495016 <_IO_stdin_used+0x16>
  401d14:       b8 00 00 00 00          mov    $0x0,%eax
  401d19:       e8 a2 ec 00 00          callq  4109c0 <_IO_printf>
  401d1e:       8b 05 d0 e3 0b 00       mov    0xbe3d0(%rip),%eax        # 4c00f4 <num2>
  401d24:       89 c6                   mov    %eax,%esi
  401d26:       48 8d 3d fd 32 09 00    lea    0x932fd(%rip),%rdi        # 49502a <_IO_stdin_used+0x2a>
  401d2d:       b8 00 00 00 00          mov    $0x0,%eax
  401d32:       e8 89 ec 00 00          callq  4109c0 <_IO_printf>
  401d37:       48 8d 35 b6 e3 0b 00    lea    0xbe3b6(%rip),%rsi        # 4c00f4 <num2>
  401d3e:       48 8d 3d f7 32 09 00    lea    0x932f7(%rip),%rdi        # 49503c <_IO_stdin_used+0x3c>
  401d45:       b8 00 00 00 00          mov    $0x0,%eax
  401d4a:       e8 71 ec 00 00          callq  4109c0 <_IO_printf>
  401d4f:       bf e8 03 00 00          mov    $0x3e8,%edi
  401d54:       e8 f7 e1 04 00          callq  44ff50 <__sleep>
  401d59:       b8 00 00 00 00          mov    $0x0,%eax
  401d5e:       5d                      pop    %rbp
  401d5f:       c3                      retq
```

```
00000000004109c0 <_IO_printf>:
  4109c0:       f3 0f 1e fa             endbr64
  4109c4:       48 81 ec d8 00 00 00    sub    $0xd8,%rsp
  4109cb:       49 89 fa                mov    %rdi,%r10
  4109ce:       48 89 74 24 28          mov    %rsi,0x28(%rsp)
  4109d3:       48 89 54 24 30          mov    %rdx,0x30(%rsp)
  4109d8:       48 89 4c 24 38          mov    %rcx,0x38(%rsp)
  4109dd:       4c 89 44 24 40          mov    %r8,0x40(%rsp)
  4109e2:       4c 89 4c 24 48          mov    %r9,0x48(%rsp)
  4109e7:       84 c0                   test   %al,%al
  4109e9:       74 37                   je     410a22 <_IO_printf+0x62>
  4109eb:       0f 29 44 24 50          movaps %xmm0,0x50(%rsp)
  4109f0:       0f 29 4c 24 60          movaps %xmm1,0x60(%rsp)
  4109f5:       0f 29 54 24 70          movaps %xmm2,0x70(%rsp)
  4109fa:       0f 29 9c 24 80 00 00    movaps %xmm3,0x80(%rsp)
  410a01:       00
  410a02:       0f 29 a4 24 90 00 00    movaps %xmm4,0x90(%rsp)
  410a09:       00
  410a0a:       0f 29 ac 24 a0 00 00    movaps %xmm5,0xa0(%rsp)
  410a11:       00
  410a12:       0f 29 b4 24 b0 00 00    movaps %xmm6,0xb0(%rsp)
  410a19:       00
  410a1a:       0f 29 bc 24 c0 00 00    movaps %xmm7,0xc0(%rsp)
  410a21:       00
  410a22:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  410a29:       00 00
  410a2b:       48 89 44 24 18          mov    %rax,0x18(%rsp)
  410a30:       31 c0                   xor    %eax,%eax
  410a32:       48 8b 3d 97 fc 0a 00    mov    0xafc97(%rip),%rdi        # 4c06d0 <stdout>
  410a39:       31 c9                   xor    %ecx,%ecx
  410a3b:       48 89 e2                mov    %rsp,%rdx
  410a3e:       48 8d 84 24 e0 00 00    lea    0xe0(%rsp),%rax
  410a45:       00
  410a46:       4c 89 d6                mov    %r10,%rsi
  410a49:       c7 04 24 08 00 00 00    movl   $0x8,(%rsp)
  410a50:       48 89 44 24 08          mov    %rax,0x8(%rsp)
  410a55:       48 8d 44 24 20          lea    0x20(%rsp),%rax
  410a5a:       c7 44 24 04 30 00 00    movl   $0x30,0x4(%rsp)
  410a61:       00
  410a62:       48 89 44 24 10          mov    %rax,0x10(%rsp)
  410a67:       e8 14 31 00 00          callq  413b80 <__vfprintf_internal>
  410a6c:       48 8b 4c 24 18          mov    0x18(%rsp),%rcx
  410a71:       64 48 33 0c 25 28 00    xor    %fs:0x28,%rcx
  410a78:       00 00
  410a7a:       75 08                   jne    410a84 <_IO_printf+0xc4>
  410a7c:       48 81 c4 d8 00 00 00    add    $0xd8,%rsp
  410a83:       c3                      retq
  410a84:       e8 a7 36 04 00          callq  454130 <__stack_chk_fail>
  410a89:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
```

하지만 정적링킹에서는 모듈 그 자체를 불러오기 때문에 코드가 길다.