Rahul Issar
riss899

Compsci 340 Assignment 3

PART 1

1.1

The program header contains the segments (in the address space of a process running that ELF executable) projected in virtual memory at exec time. The final sections header defines the linkable point of view. Each section belongs to a segment and may or may not be visible – i.e. mapped into memory – at execution time). The ELF file header tells where program header table & section header table are.

1.2

```
rahul@rahul-VirtualBox:~/Desktop/A3$ gcc hello.c -o hello
rahul@rahul-VirtualBox:~/Desktop/A3$ ./hello
Hello riss899!
00400000-00401000 r-xp 00000000 08:01 550955                             /home/rahul/Desktop/A3/hello
00600000-00601000 r--p 00000000 08:01 550955                             /home/rahul/Desktop/A3/hello
00601000-00602000 rw-p 00001000 08:01 550955                             /home/rahul/Desktop/A3/hello
006d3000-006f4000 rw-p 00000000 00:00 0                                  [heap]
7fcff3436000-7fcff35f6000 r-xp 00000000 08:01 6168                       /lib/x86_64-linux-gnu/libc-2.23.so
7fcff35f6000-7fcff37f6000 ---p 001c0000 08:01 6168                       /lib/x86_64-linux-gnu/libc-2.23.so
7fcff37f6000-7fcff37fa000 r--p 001c0000 08:01 6168                       /lib/x86_64-linux-gnu/libc-2.23.so
7fcff37fa000-7fcff37fc000 rw-p 001c4000 08:01 6168                       /lib/x86_64-linux-gnu/libc-2.23.so
7fcff37fc000-7fcff3800000 rw-p 00000000 00:00 0
7fcff3800000-7fcff3826000 r-xp 00000000 08:01 6140                       /lib/x86_64-linux-gnu/ld-2.23.so
7fcff3a0a000-7fcff3a0d000 rw-p 00000000 00:00 0
7fcff3a23000-7fcff3a25000 rw-p 00000000 00:00 0
7fcff3a25000-7fcff3a26000 r--p 00025000 08:01 6140                       /lib/x86_64-linux-gnu/ld-2.23.so
7fcff3a26000-7fcff3a27000 rw-p 00026000 08:01 6140                       /lib/x86_64-linux-gnu/ld-2.23.so
7fcff3a27000-7fcff3a28000 rw-p 00000000 00:00 0
7ffdf186c000-7ffdf188d000 rw-p 00000000 00:00 0                          [stack]
7ffdf18ff000-7ffdf1901000 r--p 00000000 00:00 0                          [vvar]
7ffdf1901000-7ffdf1903000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
0 1 2 3 4 5 6 7 8 9 Goodbye world.
rahul@rahul-VirtualBox:~/Desktop/A3$
```

1.3

1) When the operating system loads the resulting file, only the allocable part is mapped into memory. The non-allocable part remains in the file, but is not visible in memory.
2) Since you have addresses bigger than 32 bits, some of the shared objects have "/lib/x86_64-linux-gnu/" in their path. Therefore, its amd64.

Rahul Issar
riss899

1.4

```
rahul@rahul-VirtualBox:~/Desktop/A3$
rahul@rahul-VirtualBox:~/Desktop/A3$ nm hello
000000000060106f B __bss_start
0000000000601060 D bye
0000000000601070 b completed.7585
0000000000601050 D __data_start
0000000000601050 W data_start
00000000004005e0 t deregister_tm_clones
0000000000400660 t __do_global_dtors_aux
0000000000600e18 t __do_global_dtors_aux_fini_array_entry
0000000000601058 D __dso_handle
0000000000600e28 d _DYNAMIC
000000000060106f D _edata
0000000000601078 B _end
00000000004007d4 T _fini
0000000000400680 t frame_dummy
0000000000600e10 t __frame_dummy_init_array_entry
0000000000400958 r __FRAME_END__
                 U getpid@@GLIBC_2.2.5
0000000000601000 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
000000000040080c r __GNU_EH_FRAME_HDR
0000000000400500 T _init
0000000000600e18 t __init_array_end
0000000000600e10 t __init_array_start
00000000004007e0 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000600e20 d __JCR_END__
0000000000600e20 d __JCR_LIST__
                 w _Jv_RegisterClasses
00000000004007d0 T __libc_csu_fini
0000000000400760 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
00000000004006d8 T main
00000000004006a6 T numbers
                 U printf@@GLIBC_2.2.5
                 U puts@@GLIBC_2.2.5
0000000000400620 t register_tm_clones
                 U sprintf@@GLIBC_2.2.5
                 U __stack_chk_fail@@GLIBC_2.4
00000000004005b0 T _start
                 U system@@GLIBC_2.2.5
0000000000601070 D __TMC_END__
0000000000601074 B unused
rahul@rahul-VirtualBox:~/Desktop/A3$
```

- Stdio.h

- sys/types

Rahul Issar
riss899

- unist.h

- stdlib.h

- bye[]

  The symbol is in the initialised data section

- int unused
  The symbol is in the uninitialized data section (known as BSS).

- void numbers()

  The symbol is in the text (code) section.

- puts("Hello riss899!");

  The symbol is undefined

- numbers();
  The symbol is in the text (code) section.

- sprintf(mem, "cat /proc/%d/maps", pid);
  The symbol is undefined

- systsem(mem);

  The symbol is undefined

- puts(bye);

  The symbol is undefined

Local variables are not accessible from object files, nm lists the symbol table of object files, symbol tables are used for linking, local variables are not needed at link time.

1.5

- A text file is a kind of computer file that is structured as a sequence of lines of electronic text. A text file exists stored as data within a computer file system.

Rahul Issar
riss899

- In computing, a data segment (often denoted .data) is a portion of an object file or the corresponding virtual address space of a program that contains initialized static variables, that is, global variables and static local variables. The data segment is read-write, since the values of variables can be altered at run time. This contrasts with the read-only data segment (rodata segment or .rodata), which contains static constants rather than variables.
- The name. bss or bss is used by many compilers and linkers for a part of the data segment containing statically-allocated variables represented solely by zero-valued bits initially. It is often referred to as the “bss section” or “bss segment”. The BSS segment contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

```
rahul@rahul-VirtualBox:~/Desktop/A3$ readelf -lW hello

Elf file type is EXEC (Executable file)
Entry point 0x4005b0
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
  INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00095c 0x00095c R E 0x200000
  LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x00025f 0x000268 RW  0x200000
  DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
  NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
  GNU_EH_FRAME   0x00080c 0x000000000040080c 0x000000000040080c 0x00003c 0x00003c R   0x4
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
  GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text
.fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
rahul@rahul-VirtualBox:~/Desktop/A3$
```

1.6

- The last digits of the offset are the same of the last digits of the Virtual address

1.7

Seg	Type	Virtual address	Size	Access(flg)	Map area (range of addresses)	File offset	Mem access
2	LOAD	0x000000400000	0x00095c	R E	00400000-00401000	0x000000	r-xp
3	LOAD	0x000000600e10	0x000268	RW	00601000-0060200	0x000e10	r-wp

Rahul Issar
riss899

1.8

.text – is executable so read / exec
.rodata – is read only so read,
.data – read and write so rw

.bss – read and write so rw

```
 2 .note.gnu.build-id 00000024  0000000000400274  0000000000400274  00000274  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 3 .gnu.hash     00000040  0000000000400298  0000000000400298  00000298  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 4 .dynsym       00000750  00000000004002d8  00000000004002d8  000002d8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 5 .dynstr       00000316  0000000000400a28  0000000000400a28  00000a28  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 6 .gnu.version  0000009c  0000000000400d3e  0000000000400d3e  00000d3e  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 7 .gnu.version_r 00000060  0000000000400de0  0000000000400de0  00000de0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 8 .rela.dyn     00000090  0000000000400e40  0000000000400e40  00000e40  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 9 .rela.plt     00000678  0000000000400ed0  0000000000400ed0  00000ed0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
10 .init         0000001a  0000000000401548  0000000000401548  00001548  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
11 .plt          00000460  0000000000401570  0000000000401570  00001570  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
12 .plt.got      00000008  00000000004019d0  00000000004019d0  000019d0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
13 .text         00007369  00000000004019e0  00000000004019e0  000019e0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
14 .fini         00000009  0000000000408d4c  0000000000408d4c  00008d4c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
15 .rodata       00001233  0000000000408d60  0000000000408d60  00008d60  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
16 .eh_frame_hdr 00000304  0000000000409f94  0000000000409f94  00009f94  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
17 .eh_frame     000010d4  000000000040a298  000000000040a298  0000a298  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
18 .init_array   00000008  000000000060be10  000000000060be10  0000be10  2**3
                  CONTENTS, ALLOC, LOAD, DATA
19 .fini_array   00000008  000000000060be18  000000000060be18  0000be18  2**3
                  CONTENTS, ALLOC, LOAD, DATA
20 .jcr          00000008  000000000060be20  000000000060be20  0000be20  2**3
                  CONTENTS, ALLOC, LOAD, DATA
21 .dynamic      000001d0  000000000060be28  000000000060be28  0000be28  2**3
                  CONTENTS, ALLOC, LOAD, DATA
22 .got          00000008  000000000060bff8  000000000060bff8  0000bff8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
23 .got.plt      00000240  000000000060c000  000000000060c000  0000c000  2**3
                  CONTENTS, ALLOC, LOAD, DATA
24 .data         000000b4  000000000060c240  000000000060c240  0000c240  2**5
                  CONTENTS, ALLOC, LOAD, DATA
25 .bss          000009a8  000000000060c300  000000000060c300  0000c2f4  2**5
                  ALLOC
26 .gnu_debuglink 00000034  0000000000000000  0000000000000000  0000c2f4  2**0
                  CONTENTS, READONLY
```

PART 2

2.1

An example of backward compatibility is the Windows 64-bit. It has software called WOW64 that provides compatibility by emulating a 32-bit system. Consequences of moving from 32bit to 64 bit are:

- You need more memory for many operations
- The effective part of processor cash is smaller
- The size of code also increases because of additional prefixes and instructions containing 8-byte operands instead of 4-byte ones.

2.2

- Since each node holds a frame number, it must also point to the next node in the free list. Therefore: 32bits * 2 = 64bits = 8 bytes
- In Kernel, it will have to store all the 2^21 bits which is calculated from 2^34 / 2^13 = 2^21. When the system is starting, it must know which

memory frames are free, so it stores all of them. This means that 2^18 bytes are stored.

- Extent is a chunk of storage space logical volume management which is used internally to provide different device mappings. Extent will have to ability to remove the metadata of the larger files.

2.3

page – 4KB
2^32 of swap space – physical space = max useful to allocate
swap is slow ram is cheap maybe reallocate zero swap space.

2.4

H = TLB hit ratio = 99% = 0.99

TLB access time = 1ns
Memory access time = 50ns
PT access = 100ns
(H)(TLB access time + mem access time) + (1-H)(TLB access + PT access + mem access) (0.99) * (1 + 50) + (1 – 0.99) *(1 + 100 + 50) = 52ns

2.5

- Logical address bit = 64 bits
- 64 - log2(2^22) = 42 bits
- Number of page will be = 2^64 / 2^22 = 2^42.
- Pages we have entry of page table = 2^6
- Number of entry in 1 page will be 2^22 / 2^6 = 2^16
- Bit to represent one entry = Log 2(2^16) = 16
- And bit for page is 42 bits
- Therefore, number of level = 42 / 16 = 3 level page table

Since there is three levels plus one page for data/program and one for the stack which makes it 5 for the minimum page number.

2.6

- Normal instruction takes 1 microsecond (10 ^ -6 sec)
- Instruction with page fault takes 2000 micro seconds

Rahul Issar
riss899

- Given the program takes 60 sec and there were 20000 p.f
- Time taken by page faults = 20000 * 2000 microseconds = 40 sec
- Rest of the 20 sec is consumed by program execution
- 2 * 10,000 = 20 seconds = the time it takes to deal with page faults with twice as much memory as execution time.
- Therefore 20 + 20 = 40 seconds.

2.7

a. FIFO

	1	**2**	**3**	**4**	**3**	**5**	**6**	**7**	**6**	**5**	**4**	**3**	**4**	**7**	**6**	**1**	**5**	**4**	**1**	**2**
0	1						6													
0		2						7												
0			3													1				
0				4																2
0						5														
PF	F	F	F	F		F	F	F								F				F

PF = 9

b. LRU

	1	**2**	**3**	**4**	**3**	**5**	**6**	**7**	**6**	**5**	**4**	**3**	**4**	**7**	**6**	**1**	**5**	**4**	**1**	**2**
0	1						6													
0		2						7												2
0			3														5			
0				4																
0						5										1				
PF	F	F	F	F		F	F	F								F	F			F

PF = 10

Rahul Issar
riss899

c. LFU

	1	2	3	4	3	5	6	7	6	5	4	3	4	7	6	1	5	4	1	2
0	1						6													
0		2						7												2
0			3													1				
0				4																
0						5														
PF	F	F	F	F		F	F	F								F				F

PF = 9

d. Optimal

	1	2	3	4	3	5	6	7	6	5	4	3	4	7	6	1	5	4	1	2
0	1							7												
0		2					6													
0			3													1				2
0				4																
0						5														
PF	F	F	F	F		F	F	F								F				F

PF = 9