

Assignment 3

Kalpesh Krishna (140070017)
Kumar Ayush (140260016)

1 Question 1

1.1 Part a) and b)

We obtain the following output on running `memory.c`,

```
1 kalpesh@kalpesh-Inspiron-3542:assign3$ ./a.out
2 bss 1 - 0x601054
3 bss 2 - 0x601058
4 data segment 1 - 0x601048
5 data segment 2 - 0x60104c
6 stack 1 - 0x7ffd3dd72458
7 stack 2 - 0x7ffd3dd7245c
8 heap 1 - 0xf57010
9 heap 2 - 0xf57030
10 heap 1 (pointer) - 0x7ffd3dd72460
11 heap 2 (pointer) - 0x7ffd3dd72468
```

This is consistent with the output of `objdump -h a.out`,

```
1 12 .text          00000262  0000000000400490  0000000000400490  00000490
   2**4
2
3      CONTENTS, ALLOC, LOAD, READONLY, CODE
4
5 ...
6 23 .data          00000018  0000000000601038  0000000000601038  00001038
   2**3
7
8      CONTENTS, ALLOC, LOAD, DATA
9
10 24 .bss          00000010  0000000000601050  0000000000601050  00001050
   2**2
11
12      ALLOC
```

1.2 Part c)

- **Stack** - As expected, the variables have been allocated on contiguous memory locations on the stack (they are of type `int` and each have size 4 bytes)
- **Heap** - The two integer variables are not on contiguous memory locations since we used two separate `malloc()` calls. Notice that the addresses are much smaller when compared to the stack addresses, indicating that the heap generally occupies the lower address spaces.
- **bss** - The addresses we obtain are contiguous and consistent within the `bss` segment range from `0x601050` to `0x601060` (exclusive).

- **data segment** - The addresses we obtain are contiguous and consistent within the data segment range from 0x601038 to 0x601050 (exclusive).

1.3 Part d)

In this part we allocate exactly 32 bytes (eight integers) in each of the segments. The first size call is before the allocation and the second size call is after allocation.

```
1 kalpesh@kalpesh-Inspiron-3542:assign3$ gcc memory2.c
2 kalpesh@kalpesh-Inspiron-3542:assign3$ size ./a.out
3   text    data    bss    dec    hex filename
4   1115    552      8   1675   68b ./a.out
5 kalpesh@kalpesh-Inspiron-3542:assign3$ gcc memory2.c
6 kalpesh@kalpesh-Inspiron-3542:assign3$ size ./a.out
7   text    data    bss    dec    hex filename
8   1115    584     40   1739   6cb ./a.out
9 kalpesh@kalpesh-Inspiron-3542:assign3$
```

Notice both the data and bss segments have increased by 32 bytes. Uninitialized global or static variables are allocated in the bss segment whereas initialized global or static variables are allocated in the data segment.

1.4 Part e)

We obtain the virtual address section of the text segment using the `objdump -h` command.

```
1 kalpesh@kalpesh-Inspiron-3542:assign3$ objdump -h memory | grep 'text'
2 12 .text          00000272 0000000000400490 0000000000400490 00000490
3     2**4
4     CONTENTS, ALLOC, LOAD, READONLY, CODE
5 13 .fini          00000009 0000000000400704 0000000000400704 00000704
6     2**2
7     CONTENTS, ALLOC, LOAD, READONLY, CODE
8 kalpesh@kalpesh-Inspiron-3542:assign3$ objdump -h memory2 | grep 'text'
9 12 .text          00000172 0000000000400400 0000000000400400 00000400
10    2**4
11    CONTENTS, ALLOC, LOAD, READONLY, CODE
12 13 .fini          00000009 0000000000400574 0000000000400574 00000574
13    2**2
14    CONTENTS, ALLOC, LOAD, READONLY, CODE
15 kalpesh@kalpesh-Inspiron-3542:assign3$
```

For the first program, the size is 0x272 bytes, or 626 bytes. The text segment begins at 0x400490 and ends at 0x400704 (exclusive). 2 extra bytes are needed due to alignment ([reference](#)).

Similarly, for the second program, the size is 0x172, or 370 bytes. The text segment begins at 0x400400 and ends at 0x400574 (exclusive). 2 extra bytes are needed due to alignment.

2 Question 2

The `/proc` filesystem contains a map between virtual memory pages and physical memory pages in the `/proc/$ (PID) /pagemap` and `/proc/$ (PID) /maps` file. This map can be used to obtain a mapping from virtual addresses to physical addresses.