CS220 - Computer System II
Lab 3

Due: 09/21/2016, 11:59pm

1 Introduction

In this lab, you will understand different types of memory.

2 Getting Started

Download lab3.tar.gz and extract it. You should find a program named "memory". For the programs that you wirte and compile, use the flags -std=c89 and -O0. The flag c89 sets to use ansi C and -O0 turns off optimization[1].

3 Different types of memory

The precompiled program "memory" has 4 secrets. Your job is to (1) identify the secrets, (2) Identify the addresses of gfoo, lfoo, sfoo and dfoo. Identifying the value of dfoo (which is the address where one of the secrets will be stored). Finally, (3) Identify the section where each variable is stored. The source code of the program is as follows:

```
#include <stdlib.h>
int gfoo = /* secret 1 */;
int main()
{
  static int sfoo;
  int lfoo;
  int *dfoo = (int *) malloc (sizeof(int));
  sfoo = /* secret 2 */
  lfoo = /* secret 3 */
  *dfoo = /*secret 4 */
  return sfoo + lfoo + gfoo + *dfoo;
}

```

Primarily, 3 main memory stores are available to each program. The program itself (i.e., .data, .rodata, .bss etc.), the program stack, and the program heap. Any variable that is local to a function during execution is initialized each time the control enters the

[1] If we don't do this, the compiler may realize that allocated memory is not being used, and may therefore skip the allocation altogether!

function. It is stored on the stack. Any variable that persists across function invocations (e.g., global and static variables) can not be stored on the stack region. Memory allocated through malloc are acquired at runtime and are therefore stored on the heap.

Start the program on gdb and set appropriate break points. When you run the program (using gdb command "run"), the program is loaded. At that point, you can examine the memory mappings and layouts using the following commands:

```
(gdb) info proc mappings
(gdb) info files
```

In order to find the secrets, you could set a breakpoint after the move instructions that occur between the call to main and return, and look at the immediate values.

Create a lab3.txt file, and record the following:

1. Secrets 1, 2, 3 and 4.
2. Addresses of gfoo, sfoo, lfoo and dfoo.
3. Value of dfoo (which is the address that contains secret 4).
4. Start and end addresses of stack, heap and data section.

Next, we will examine the effects of each type of memory on the program.

Effect of global variables: Type in the following program (call it global_vars.c) and compile, each time replacing x with values 10, 100, 1000 and 10000 respectively.

```
#include <stdio.h>
char gfoo[x] = {0x10};
int main()
{
  printf("%p\n", &gfoo);
  return 0;
}

```

In each case, record the size of the binary in lab.txt. Because global variables are incorporated into a writable section in the binary, you will find that the size of the binary increases as the number of global variables increase.

Effect of dynamic variables: Type in the following program (call it malloc_vars.c) and compile, each time replacing x with values 10, 100, 1000 and 10000 respectively.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int *p = (int *) malloc (x * 0x1000)
return 0;
}

```

In each case, record the size of the binary in lab3.txt. Because memory is acquired at runtime, you will not find a significant change in the binary size.

Testing the limits of dynamic memory: Write a program called malloc_limits.c to allocate increasing amounts of memory using malloc. At one point, the program is will run out of memory. Record the maximum amount (in bytes) of dynamic memory you can acquire. Also, insert a break point in gdb on entering main and just before returning from main. Invoke "info proc mappings" and record the size of [heap] in each case. Unlike Java, C does not come with a garbage collector. Therefore, it is very important to free the dynamic memory that is not being used using the free() function.

Testing limits of stack memory Type in the following program (call it static_vars.c) and compile, each time replacing x with values 10, 100, 1000 and 10000 respectively.

```
#include <stdio.h>

int main()
{
int arr[x];
return sizeof(x);
}
```

Also, insert a break point in gdb on entering main and just before returning from main. Invoke "info proc mappings" and record the size of [heap] in each case.

4 Submitting the result

Create lab3_submission.tar.gz file comprising Lab3.txt and all the source files (.c files). Upload the file to Blackboard.