CS232L Operating Systems Lab
Lab 07: Process Memory Layout

CS Program
Habib University

Fall 2024

1 Introduction

In this lab you will learn how to:

1. the memory layout of a process

2 Virtual address spaces

By now we've studied in class that a process views the memory via the lens of its virtual address space, i.e., it gets the illusion that it can access addresses from 0x0 to the end of the address space.

The OS, in the background, places different parts of the process address space in different RAM locations. The addresses of these RAM locations are very different from what the process believes.

Listing 1 lists an example program from the first chapter of this book. It dynamically allocates space for an integer, prints its address and continues to increment its value. You specify the intial value as argument on command line.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
//#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
  fprintf(stderr, "usage: mem <value>\n");
  exit(1);
    }
    int *p;
    p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) addr pointed to by p: %p\n", (int) getpid(), p);
    *p = atoi(argv[1]); // assign value to addr stored in p
    while (1) {
  sleep(1);
  *p = *p + 1;
  printf("(%d) value of p: %d\n", getpid(), *p);
    }
    return 0;
}
```

Listing 1: mem.c

Compile this program and run it multiple times. Note: if the address of the pointer is different every time then your OS is using the ASLR (Address Space Location Randomization).

Disable it by following the instructions on https://stackoverflow.com/questions/5194666/disable-randomization-of-memory-addresses.

Run multiple copies of this program simultaneously from different terminal windows with different start values and observe how every program is displaying a different value for the same memory address. You can now see and observe the illusion that every process gets.

Listing 2, again from the book, tries to get rough idea of the location of the text, heap, and stack segments in the virtual address space.

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

int main(int argc, char *argv[]) {
    printf("location of code : %p\n", main);
    printf("location of heap : %p\n", malloc(100e6));
    int x = 3;
    printf("location of stack: %p\n", &x);
    return 0;
}
```

Listing 2: va.c

Try running the program in listing 2 multiple times. If the addresses are different each time, try disabling the ASLR as described above.

2.1 Exercise: address.c

Write a function which contains two variables x and y. Get the address of one variable and using that address try guessing the value of the second variable. Verify your guess by changing and displaying the changed value of the second variable using that changed address. This may require some hit and trial.

Call this function in main to verify your results.

2.2 Exercise: corruptfunc.c

Each function call stores the return address on the stack. Write a function which when called overwrites its return address thus corrupting the stack. Run call this function in main to verify that it doesn't return.

3 mmap

Read about the `mmap()` system call. It can allocate an anonymous chunk of RAM and map it to an address (aligned to a page boundary[1]). The corresponding `munmap()` system call frees the memory allocated via `mmap()`.

3.1 Exercise: 1024ints.c

Write a program that allocates memory for 1024 integers and maps it to a virtual address of your choice. Verify the memory allocation by writing and reading back 1024 integers at that address.

[1] We'll read about pages next week