Scott Skibin (ss3793), Kevin Schwarz (kjs309)

CS 416 Operating Systems

Due: 02/15/2023

Project 1: Stacks and Basics(Report)

Part 1:

1. **What are the contents in the stack?**
   a. The contents in the stack are any elements that can be pushed and popped from the stack. This includes numbers (i.e. signalno) and pointers (i.e ptr = &signalno) as it relates to this project. In general, the stack also stores local variables of any data type. Also included on the stack frame are arguments, a return address, saved registers, frame pointer and stack pointer.
2. **Where is the program counter, and how did you use GDB to locate the PC?**
   a. The program counter, otherwise known as the instruction pointer, is located on the main stack frame. When signal_handle() handle is called, it is pushed onto the stack frame of the subroutine before it is invoked. Specifically, I used GDB to locate the program counter at different breakpoints in my program. First, I set a breakpoint at the bad instruction in main() and used a combination of ‘gbd disassemble main’ and ‘p $pc’ to inspect the program counter. I cross referenced these two to confirm. Once I found my program counter, I set another breakpoint inside of signal_handle() and inspected the stack to locate it within the stack of signal_handle(). Once found, I was able to find my saved return address.
3. **What were the changes to get the desired result?**
   a. Once I had the address of signalno, the saved return address (where the PC was located on signal_handle() stack), I was able to calculate the offset between the two, which happened to be 156. Since I defined pc as int type pointer, I divided this number by 4, to account for the type. I added this calculated value (39) to the address of signalno, to point PC at the program counter on the stack. I then disassembled the main again to find the length of the bad instruction (in this case it was 2). I incremented the PC by 2, to move it on to the next instruction to get the desired result.

Part 2:

- **get_top_bits()**
  For this function I first calculated the total bits in the value passed. I used the sizeof() function on the value parameter passed to the function and then multiplied the returned value(which in this case, on Linux, was 4) by 8 to get the total bits of the value space. Then I calculated the amount of bits that needed to be shifted by subtracting the total bit value by the num_bits parameter passed into the function. I then used the right shift operator to shift the bits of value by the calculated shift amount and returned the outcome. The top bits are the left-most bits in any binary value, using the right shift

operator in C maintains the top order bits of the original value while shedding a given number of bits stored in the shift variable.

- **set_bit_at_index()**
  I implemented the set_bit_at_index() and get_bit_at_index() functions under the assumptions that the index value passed started from 0 and referenced the bit position from left to right in the entire bitmap. In order to achieve this I first had to figure out which byte index was being referenced in the bitmap. To do so, I used integer division to divide the given index by 8 (as each byte contains 8 bits). Then I needed to figure out at which index the bit to be set fell under within its byte position in the entire bitmap. This was simple algebra again, by just subtracting the max index value in an 8-bit representation(8-1 = 7) by the (index -(byte_index * 8)). Then it is just a matter of using the bitwise "or" operator and the left shift operator to set the bit in the given byte of the bitmap at the correct index position in its 8-bit representation.
- **get_bit_at_index()**
  The process for finding the bit-index position corresponding to the overall index given is the same in this function as set_bit_at_index(). Then the bitwise "and" operator in combination with the left shift operator are used to isolate the bit at the specified index in the bitmap which is compared to 0 and a boolean integer value of 1 (if the bit specified is set) or 0(if the specified bit is not set) is returned.