asn89 Aaron Newland
tjr151 Tyler Radziemski

stack.c

1
1.1 In the process for stack.c, there are up to three stack frames (at the time of the segfault). These stack frames are for the main function, the signal_handle function, and a third kernel operation that
will return the process back to the main function after signal_handle finishes. Within signal_handle's stack,
there is the address of the parameter- signalno -and a return address to the kernel function. In the kernel
function, the most significant thing in its stack is a return address to the main function.

1.2 Using a combination of the GDB commands "info frame" and "backtrace" we found that there are multiple
stack frames for each function, and both signal_handle and the kernel function held return adresses to separate
functions (namely, that signal_handle would return to the kernel operation and from there back to main). So,
with these commands we found the address for the saved eip register in the kernel function that would return to
main.

1.3 Using the address found for the saved eip in the kernel operation, we recognized the offset from the address
of signalno (the parameter we could access using a local variable in signal_handle), which we found to be 15 * 4
bytes. Now that our local variable pointed to the saved eip that pointed to the program counter for main, we
incremented that by 2 so that when signal_handle finished, main returned two bytes after the original instruction
that created the segfault, allowing the process to continue.

bitops.c
2
To implement the get_top_bits() function first a bit mask is created to match the number
of desired bits. This is done by assigning the bit mask with the value of 1 and using a
left bit shift to the number of bits that are to be retrieved, that value is then subtracted
from 1 to set all bits 1. Then a right shift is used on the argument 'value' by 32 - num_bits
(which highlights only the desired bits). This is then bit wise anded with the bit mask
that was created earlier. To handle the edge case of all wanting to return all 32 bits, the
value is just returned.

To implement the set_bit_at_index() function 2 values are needed to set the bit: the block (set of 4 bits) that the desired bit resides in, and the index of that individual bit in the block. To get the block, take the argument 'index' and integer divide by 8; similarly, to get the index inside the block take the same argument 'index' and modulus divide by 8 and then subtract from 1 to get the true index from 0. Next a bit mask is needed so we create one and left shift by the new value we set at 'index'. Now we use a bit wise or with the block of our bitmap and our bit mask to set the desired bit but leave all other bits as they were. Finally, we return our val to the desired block of our bitmap.

Finally, to implement get_bit_at_index() we use the same steps in set_bit_at_index() to get the desired block and index of the bit we wish to read. Next, we take the desired block of our bitmap and bit shift it right by the new value we set at 'index'. This gives us the bit we need to read, so we return val.