# OS Project 1 Report

**Authors: ac1771 and tsc95**

## Signal Handler

**Q**: What are the contents in the stack? Feel free to describe your understanding.

The stack contains the base pointer of the current executing function as well as all the variables that have been initialized within the current executing function along with the arguments given to the function. Once the current executing function finishes running, the function looks up where in the previous called function it should return to (this is the program counter and is stored on the stack). Once it retrieves the correct line, the function goes back to the previous function which called it.

**Q**: Where is the program counter, and how did you use GDB to locate the PC?

We know that the stack starts at a high memory value and goes towards lower memory values. We also know that the program counter gets stored on the stack before the address of the argument is stored, therefore given the address of the argument we want to look at higher memory addresses since the program counter gets stored before. To view the items stored on the stack we can run the command `x/10x $sp`. Then it was a matter of trial and error until I finally found the address to which when added a value, lines were jumped. One interesting thing I noticed was that the value of register `eax` was also the same address as the one I saw in GDB, I assume that this means that the PC address gets stored in `eax` as well when switching to the new function.

**Q**: What were the changes to get the desired result?

To figure out the size to increment the program counter by, we dumped the assembly onto the terminal by using the `objdump` command. We realized that the `mov` command that we needed to skip is 5 bytes and therefore incremented the appropriate amount the skip the instruction.

## Bit Operations

### Extracting Top Order Bits

To get the top order bits (most significant ones) for a given value I followed these steps:

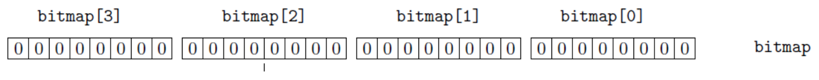
- We first need to figure out how many bits there are in the integer. To do this I first copied the value into a temp variable and repeatedly shifted the bits right until we end up with 0 for the temp value.
- Next we subtract the number of bits we want (top order bits) from the total bits in the integer, this is the number of times I want to shift the bits to the right to get the top order bits

- Lastly we just shift the value bits that many times to the right and return the resulting value

### Setting Bits at Index

Given the index of a bit, we needed to change the corresponding bit in an array of characters. Moreover, the array needed to be read from right to left (the same way we read bits), to do this I followed the following steps

- Each character has 8 bits, so first to get the index from the right of the chosen bit, we do an integer divide by 8. By doing so, if the chosen bit is 17, we know to look at the third character from the right in the array. So if the chosen bit is 17, we need to modify `bitmap[2]` since 17/8 = 2.



- To find the exact bit to change within the character, we do a modulo. If we take the same example of the bit to set being 17, we do 17%8 = 1, so therefore the bit to set here would be the second bit from the right.
- Now that we know the index of the character to set and then the bit to set within the character, we can go ahead and set the bit. We set the bit by shifting the integer 1 by the index of the bit that we want to set and doing an `OR EQUALS` operation. This way if the specific bit that we want to set is not already a 1, it will not be a 1, but if has already been set to a 1 then nothing will happen.

And that's it!

### Getting Bits at Index

Now given the index of the bit, we want to retrieve the value. To do this we follow the same first three steps as before

- Find the index of the character then get the index of the bit by doing a modulo.
- Now instead of the last step being an `OR` operation where we set the specified index, we instead use an `AND` here. More specifically, we shift the bits in the character by the index of the bit within the character to get the specified bit to be the right most bit and then do an `AND` with 1 to retrieve the stored bit. If the bit is 0, the `AND` will return 0 and if the bit is 1 the and wil return a 1.