CSCI 4434 - Program 3

Yung Brinney

Introduction

This program expands on the original uio.s module by introducing numeric validation, digit limits, decimal-to-binary conversion, and buffer protection. The program runs without interruption and uses Linux system calls to read input from STDIN and print output to STDOUT. No C library functions were utilized. All functionality was implemented directly in the ARM assembly.

System Calls Used

All input and output operations are performed via Linux system calls. The program uses SYS_READ (3) to read input from STDIN and SYS_WRITE (4) to print output to STDOUT. Each character is read one byte at a time to ensure safe validation and control of the input buffer.

Input Validation

The program reads input one character at a time and determines whether it is within the ASCII range of digits '0' through '9'. If a character outside that range is entered, the program clears the remaining characters on the line, prints an error message, and prompts the user again.

Figure 1 shows the behavior of non-numeric validation. In this test, characters like a, 12a, and @ correctly display the invalid input message and the prompt.

```
brooza@raspberrypi:~/CSCI_4434/Program_3_Brinney $ ./program3
Enter 1-3 digits: a
Invalid input. Try again.
Enter 1-3 digits: 12a
Invalid input. Try again.
Enter 1-3 digits: @
Invalid input. Try again.
```

Figure 1: Non-numeric input validation

Limiting Input to 1–3 Digits

To enforce the three-digit limit, the program uses register r6 as a counter to keep track of the number of valid digits entered. Digitbuf stores each valid digit. If the counter reaches three and another digit is entered, the program rejects the input, flushes the remaining characters, displays a "Too many digits" message, and prompts the user again.

Figure 2 shows how the three-digit limit is enforced, with inputs like 1234 and 9999 producing the correct "Too many digits" message.

```
brooza@raspberrypi:~/CSCI_4434/Program_3_Brinney $ ./program3
Enter 1-3 digits: 1234
Too many digits. Try again.
Enter 1-3 digits: 9999
Too many digits. Try again.
```

Figure 2: Handling of inputs exceeding three digits

Decimal Conversion

After valid input is received, the digits_to_decimal routine converts the ASCII digits to a decimal integer. The conversion employs the formula: “value = value × 10 + digit”

The result is stored in register r4. For example, if the user enters 123, the decimal value stored in r4 will be 123.

Figure 3 displays how valid decimal inputs such as 5, 12, and 123 are accepted and printed correctly.

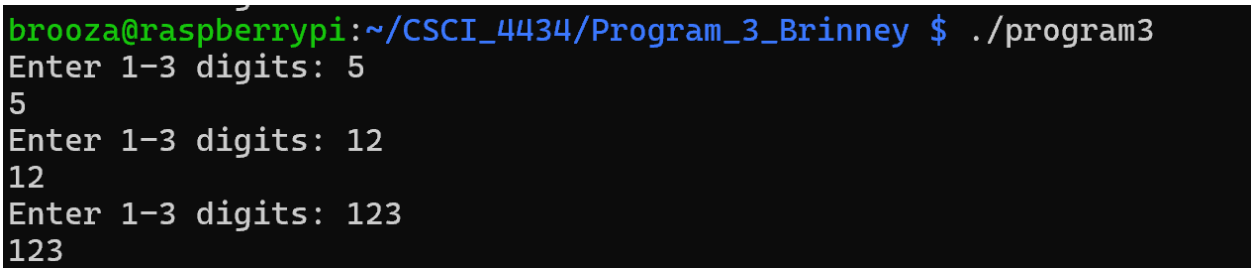

Figure 3: Valid 1–3-digit decimal input

Binary Representation Stored in Register

The function dec_to_binary converts a decimal number to a binary digit representation stored in register r5. This is accomplished by repeatedly checking the least significant bit using an AND operation with 1 and shifting the value to right. The binary digits are created with increasing powers of ten.

For example, if the user enters 13, the value in r5 will be 1101. This meets the requirement for creating a binary representation that can be stored in a register.

Converting Binary Back to Decimal

After generating the binary digit representation, the binary_to_decimal routine returns it to base 10. Each binary digit is processed and multiplied by the related powers of two. The final decimal result is printed using the write system call.

Figure 4 shows the correct conversion of test values with known binary patterns such as 1, 2, 3, 4, 7, 8, 15, 16, 31, 63, 127, and 255.

```
brooza@raspberrypi:~/CSCI_4434/Program_3_Brinney $ ./program3
Enter 1-3 digits: 1
1
Enter 1-3 digits: 2
2
Enter 1-3 digits: 3
3
Enter 1-3 digits: 4
4
Enter 1-3 digits: 7
7
Enter 1-3 digits: 8
8
Enter 1-3 digits: 15
15
Enter 1-3 digits: 16
16
Enter 1-3 digits: 31
31
Enter 1-3 digits: 63
63
Enter 1-3 digits: 127
127
Enter 1-3 digits: 255
255
```

Figure 4 Shows the return verification from decimal to binary to decimal.

Leading Zero Handling

Along with this, the program correctly handles leading zeros. Inputs such as 00 and 007 are correctly converted to the decimal values 0 and 7.

Figure 5 displays the proper processing of leading zeros.

```
brooza@raspberrypi:~/CSCI_4434/Program_3_Brinney $ ./program3
Enter 1-3 digits: 0
0
Enter 1-3 digits: 00
0
Enter 1-3 digits: 007
7
```

Figure 5: Leading zero test cases

Buffer Overflow Protection

The buffer overflow prevention was carefully set up. The program reads one byte at a time and stores up to three digits in a fixed-size buffer of four bytes. If more digits are entered than the limit, the input is flushed before restarting. Because the program validates both character type and length before storing values, it prevents writing beyond available memory and buffer overflow.

Challenges and Debugging Process

A few problems happened during testing, needing debugging efforts. One issue involved the incorrect use of the mul instruction. ARM requires that the destination register (Rd) not be identical to the

first source register (Rm). Initially, instructions like mul r6, r6, r4 resulted in assembler errors. This was fixed by rearranging the operands so that the destination register was different from the first source register.

Another issue was incorrect string lengths when using the SYS_WRITE system call. If the byte count passed to write exceeded the actual string length, additional characters from adjacent memory were printed. This resulted in unexpected characters like "I" appearing before valid input. The problem was solved by exactly counting the number of characters in each string and adjusting the length values accordingly.

Conclusion

This program successfully expands on the original uio.s module by introducing numeric validation, digit limits, decimal-to-binary conversion, and buffer protection. All input and output operations are carried out using Linux system calls. Testing confirmed proper handling of valid input, invalid characters, excessive digits, leading zeros, and conversion.