CSE 3211 : Operating Systems Lab

Lab Assignment 01

Implement kprintf and kscanf functions for the kernel

Submitted by

Alif Ruslan
Roll: CSE-FH-014
Reg: 2018-025-309

Kazi Shadman Sakib
Roll: CSE-FH-097
Reg: 2017-614-973

Submitted to

Dr. Mosaddek Hossain Kamal
Professor
Department of Computer Science & Engineering
University of Dhaka

Submission Date

19.08.2022

1 Introduction

In this Assignment 01 we had to implement kprintf and kscanf functions for the kernel, using the Micro-controller STM32F446RE.

2 Implementation of kprintf

The implemnetation of kprintf was done for four variable types, such as %c for charater, %s for string, %d for integer and %x hexadecimal.

2.1 For Character Type

We implemented the character type in the kprintf function which resides in the file "kstdio.h",

```
    case 'c':{
    UART_SendChar(USART2, outvar[0]);
    break;
}
```

From the above implemented code, we can see that, we have implemented the character type simply using the UART_SendChar(USART2, outvar[0]) built in function, which simply takes the 0th index of the array outvar to print the character type variable using USART2.

2.2 For String Type

We implemented the string type in the kprintf function which resides in the file "kstdio.h",

```
    case 's':{
    _USART_WRITE(USART2, outvar);
    break;
}
```

From the above implemented code, we can see that, we have implemented the string type simply using the _USART_WRITE(USART2, outvar) built in function, which simply takes the outvar to print the string type variable using USART2.

2.3 For Integer Type

We implemented the integer type in the kprintf function which resides in the file "kstdio.h",

```
    case 'd':{
        uint32_t num = *(outvar);
```

```
    int rev = 0;
    int len = 0;
    while (num > 0){
        rev = rev*10 + num%10;
        num /= 10;
        len++;
    }
    char str[10];
    for(int i = 0; i < len; i++) {
        str[i] = rev%10 + '0';
        rev /= 10;
    }
    str[len] = 0;
_USART_WRITE(USART2, str);
break;
}
```

From the above implemented code, we can see that, we have implemented the integer type by taking the address of outvar to an uint32_t variable named 'num'. And then we reversed this 'num' variable contents to 'rev' variable. And then we had to store this 'rev' variable reversed as characters to print it to the command line. Thus we took a character array names str[10] and stored the 'rev' variable contents in reverse as characters to print using _USART_WRITE(USART2, str);

2.4 For Hexadecimal Type

We implemented the hexadecimal type in the kprintf function which resides in the file "kstdio.h",

```
case 'x': {
    uint32_t num = *(outvar);
    char str[10];
    int i=0;
    while (num > 0) {
        int rem = num%16;
        if (rem < 10)
            str[i++] = rem + '0';
        else
            str[i++] = (rem-10) + 'a';
        num /= 16;
    }
    int len = i;
    for (int j = 0; j < len/2; j++) {
        char temp = str[j];
        str[j] = str[len-1-j];
        str[len-1-j] = temp;
    }
```

```
        str[len] = 0;
        _USART_WRITE(USART2, str);
        break;
    }
```

From the above implemented code, we can see that, we have implemented the hexadecimal type by taking the address of outvar to an uint32_t variable named 'num'. In the same way as decimal type, we have reversed the actual 'num' type, but this time we have taken the base 16 to mod it with 'num', saving it in 'rem' (remainder) variable until the 'num' variable is 0. We have checked that if the 'rem' (remainder) is less than 10 then in the str variable will collect 'rem' + '0'. If the remainder is greater then 10 then it stores str[i++] = (rem-10) + 'a', because it falls in the hexadecimal cases. Thus after performing this, we simply reverse the str array to have the actual result of the hexadecimal value. Thus, we print the value using _USART_WRITE(USART2, str).

3 Implementation of kscanf

The implemnetation of kscanf was done for four variable types, such as %c for charater, %s for string, %d for integer and %x hexadecimal.

3.1 For Character Type

We implemented the character type in the kscanf function which resides in the file "kstdio.h",

```
    case 'c':{
        invar[0] = UART_GetChar(USART2);
        break;
    }
```

From the above implemented code, we can see that, we have implemented the character type simply using the UART_GetChar(USART2) built in function, which simply takes the input value from the usart-¿DR, thus reading the variable from the terminal.

3.2 For String Type

We implemented the string type in the kscanf function which resides in the file "kstdio.h",

```
    case 's':{
        _USART_READ_WORD(USART2, invar, 60);
        break;
    }
```

From the above implemented code, we can see that, we have implemented the character type simply using the _USART_READ_WORD(USART2, invar, 60). This function is built by us, to read the input string as an array of characters using the UART_GetChar(USART2) function. To terminate a String input we have to add a 't' character with no spaces with the string input.

3.3 For Integer Type

We implemented the integer type in the kscanf function which resides in the file "kstdio.h",

```
case 'd': {
    uint32_t num = 0;
    char str[50];
    _USART_READ(USART2, str, 50);
    int len;
    for(len = 0; str[len]; len++);
    for (int j = 0; j < len; j++) {
        num *= 10;
        num += (str[j]-'0');
    }
    *invar = num;
    break;
}
```

From the above implemented code, we can see that, we have implemented the integer type simply by reading the input that comes from the _USART_READ(USART2, str, 50) to str[50] and convert it to an integer 'num' using a for loop. And then assigning the 'num' variable to our *invar.

3.4 For Hexadecimal Type

We implemented the hexadecimal type in the kscanf function which resides in the file "kstdio.h",

```
case 'x': {
    uint32_t num = 0;
    char str[50];
    _USART_READ(USART2, str, 50);
    int len;
    for(len = 0; str[len]; len++);
    int prod = 1;
    for (int j = len-1; j >= 0; j--) {
        char c = str[j];
        if (c >= '0' && c <= '9') {
            num += (c-'0')*prod;
        }
```

```
            else {
                num += (c-'A'+10)*prod;
            }
            prod *= 16;
        }
        *invar = num;
        break;
    }
```

From the above implemented code, we can see that, we have implemented the hexadecimal type simply by reading the input that comes from the _USART_READ(USART2, str, 50) to str[50]. This str[50] is then used to convert the input into hexadecimal number, 'num' by using a for loop with some conditions. Then, we assign this 'num' variable to *invar, which holds our result of hexadecimal number.