XV6 - SCANF

DONE BY :

ADITI SRINIVAS

XV6 - SCANF

INTRODUCTION

XV6 is a modern reimplementation of Sixth Edition Unix in ANSI C for multiprocessor x86 systems. It is used for pedagogical purposes in MIT's Operating Systems Engineering (6.828) course. Unlike Linux or BSD, xv6 is simple enough to cover in a semester, yet still contains the important concepts and organisation of Unix.

PROBLEM STATEMENT AND IMPLEMENTATION

Implement scanf in xv6 by using basic file I/O.

The function scanf reads formatted input from stdin. Following are the details of the function,

- Function signature : void scanf (int fd , char * fmt , …)
- Parameters (Variable length):
  - fd : Integer argument for file descriptor of STDIN i.e 0.
  - fmt : Character array i.e the format string
  - Rest of the arguments are the addresses of variables into which the corresponding value is to be stored.

From the xv6 documentation on "I/O and File descriptors" (Pages 10 and 11), the following points were noted,

- A file descriptor is a small integer representing a kernel-managed object that a process may read from or write to. A process may obtain a file descriptor by opening a file, directory, or device, or by creating a pipe, or by duplicating an existing descriptor.
- The file descriptor interface abstracts away the differences between files, pipes, and devices, making them all look like streams of bytes.

- A process reads from file descriptor 0 (standard input), writes output to file descriptor 1 (standard output), and writes error messages to file descriptor 2 (standard error).
- The read system call reads bytes from open files named by file descriptors. The call read(fd, buf, n) reads at most n bytes from the file descriptor fd, copies them into buf, and returns the number of bytes read.

Keeping the above mentioned points in mind, scanf was implemented. The basic working of the following implementation is, input is read from STDIN one byte at a time into a buffer. This is in the form of an array of characters. The buffer is then parsed and converted to the required type and assigned to the given variable.
The formats that are supported in the implementation are :

1. %c : Character
2. %d : Integer
3. %u : Unsigned integer
4. %s : String
5. %i : Hexadecimal, Octal or signed integer based on input.
6. %x : Hexadecimal number
7. %o : Octal number

ADDING SCANF TO XV6

The following steps are to be followed to add the scanf function to xv6.

1. Check if xv6 is running properly
   - Commands
     - $ cd xv6
     - $ make qemu
   - To exit, close the qemu shell.
2. Add the files scanf.c and scanf.d to the xv6 folder and compile scanf.c as :
   - Commands
     - $ gcc -c scanf.c
     - $ ls

- Upon typing ls, the object file scanf.o along with .c and .d files should be present.

3. Changing the makefile and adding scanf to the library of user functions :
   - Command 1 : $ gedit Makefile
   - In the makefile,
     - Navigate to " EXTRA =\" and add scanf.c after printf.c
     - Navigate to " ULIB " and add scanf.o after printf.o
   - Command 2 : $ gedit user.h
   - In this file, navigate to the bottom and add the function defition of scanf after printf as : void scanf(int, char *, void *);
4. Since we dont have any way of compiling and running a program from within xv6, we add the program in the makefile and run it Via a command. Add the app.c file to the xv6 folder.
5. Changing the makefile
   - Command : $ gedit Makefile
   - Navigate to " EXTRA =\ " and add app.c after wc.c
   - Navigate to " UPROGS =\ " and add _app\ after _wc\
6. Test if the app was added succesfully
   - Commands :
     - $ make qemu
     - $ ls
   - If a command named "app" appears in the list, it was succesfully added.
7. Run the app and test scanf.
   - Command inside the qemu shell : $ app
   - This should prompt "Enter number" and so on.

IMPLEMENTATION OF SCANF

// Code - scanf implementation in xv6

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

```
// %c : character
// %d : signed integer
// %u : Unsigned integer. If sign is seen, return 0.
// %s : string
// %i : Hexadecimal if input starts with 0x , Octal if starts with 0 , default :
signed integer
// %x : Hexadecimal number
// %o : Octal number

// Funtion to reads characters from STDIN (1 byte at a time)
static void
getc(int fd, char ** buffer){
      int  i = 0;
      char buf[256];    // read input into buffer
      while(read(fd, &buf[i], 1)){
            if((buf[i] == '\n' || buf[i] == ' ' || buf[i] == '\t' || buf[i] ==
'\r' || buf[i] == '\v' || buf[i] == '\f')){
                  break;
            }
            else{
                  i++;
            }
      }
      buf[i] = '\0';
      *(buffer) = buf;
}

// atoi() function for +ve and -ve integers
int atoi_decimal(char *buf){
      int res = 0;
      int sign = 1;
      int i = 0;  // Initialize index of first digit
      if (buf[0] == '-'){     // If number is negative, then update sign
            sign = -1;
            i++;
      }
            for (; buf[i] != '\0'; i++){
            if(buf[i] - '0' < 0 || buf[i] - '0' > 9)  // If a non numeric
character is found, break
                  break ;
        res = res*10 + buf[i] - '0';
      }
      return sign*res;
}
```

```c
// Function to convert hexadecimal integer to decimal integer
int atoi_hexa(char *buf, int start){
	int res = 0;	// Stores the final result
	int i = start ;
	for(; buf[i] != '\0' ; i++){
		int temp = 0 ;
		if(buf[i] - '0' >= 0 && buf[i] - '0' <= 9)
			temp = buf[i] - '0' ;
		else{
			switch(buf[i]){
				case 'A' :
				case 'a' :  temp = 10 ;
					    break ;
				case 'B' :
				case 'b' :  temp = 11 ;
					    break ;
				case 'C' :
				case 'c' :  temp = 12 ;
					    break ;
				case 'D' :
				case 'd' :  temp = 13 ;
					    break ;
				case 'E' :
				case 'e' :  temp = 14 ;
					    break ;
				case 'F' :
				case 'f' :  temp = 15 ;
					    break ;
				default :   return res ;
			}
		}
		res = res*16 + temp ;
	}
	return res ;
}

// Function to convert Octal integer to a decimal integer
int atoi_octal(char *buf, int start){
	int res = 0 ;
	int i = start ;
	for(; buf[i] != '\0' ; i++){
		if(buf[i] - '0' >= 0 && buf[i] - '0' <= 7)
			res = res*8 + buf[i] - '0' ;
```

```
            else
                  return res ;
      }
      return res ;
}

// atoi function for decimal, hexa and octal numbers
int generalized_atoi(char *buf){
      if(buf[0] != '0' )           // String is of a decimal integer
            return atoi_decimal(buf) ;
      else if(buf[0] == '0' && (buf[1] == 'x' || buf[1] == 'X'))  // string is
of hexadecimal number
            return atoi_hexa(buf, 2) ;
      else
            return atoi_octal(buf, 1) ;   // Octal number
}
void scanf(int fd, char * fmt, ...){
      int i = 0;
      char c;
      char * buf = "";  // holds the input temporarily
      int count_args = 1;
      uint var = *((uint*)(void*)&fmt + count_args);
      count_args++;

      for(i = 0; fmt[i]; i++){
            c = fmt[i] & 0xff;
            switch(c){
                  case 'c':
                        getc(fd, &buf);
                        *(char *)(var) = buf[0];      // assign the variable
with the character read into buffer
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
                  case 'd':
                        getc(fd,&buf);
                        *(int *)(var) = atoi_decimal(buf);  // convert to an
integer and assign to the given variable
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
                  case 's':
```

```
                        getc(fd, &buf);
                        strcpy((char *)var, buf);           // copy the string
in buffer to the given variable
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        buf = "";
                        break;
                  case 'i':
                        getc(fd, &buf);
                        *(int *)(var) = generalized_atoi(buf);    // convert to
an integer and assign to the given variable
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
                  case 'u':
                        getc(fd, &buf);
                        *(int *)(var) = atoi(buf);    // convert to an unsigned
integer and assign to the given variable
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
                  case 'x':
                        getc(fd, &buf);
                        *(int *)(var) = atoi_hexa(buf, 2);  // convert to an
integer and assign to the given variable
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
                  case 'o':
                        getc(fd, &buf);
                        *(int *)(var) = atoi_octal(buf, 0); // convert to an
integer and assign to the given variable
                        buf = "";
                        var = *((uint*)(void*)&fmt + count_args);
                        count_args++;
                        break;
            }
      }
}
```