## NAME

**printf**, **uprintf**, **tprintf**, **log** - formatted output conversion

## SYNOPSIS

**#include <sys/types.h>**
**#include <sys/systm.h>**

*int*
**printf**(*const char *fmt, ...*);

*void*
**tprintf**(*struct proc *p, int pri, const char *fmt, ...*);

*int*
**uprintf**(*const char *fmt, ...*);

*int*
**vprintf**(*const char *fmt, va_list ap*);

**#include <sys/syslog.h>**

*void*
**log**(*int pri, const char *fmt, ...*);

*void*
**vlog**(*int pri, const char *fmt, va_list ap*);

## DESCRIPTION

The **printf** family of functions are similar to the printf(3) family of functions. The different functions each use a different output stream. The **uprintf**() function outputs to the current process' controlling tty, while **printf**() writes to the console as well as to the logging facility. The **tprintf**() function outputs to the tty associated with the process *p* and the logging facility if *pri* is not -1. The **log**() function sends the message to the kernel logging facility, using the log level as indicated by *pri*, and to the console if no process is yet reading the log.

Each of these related functions use the *fmt* parameter in the same manner as printf(3). However, **printf** adds two other conversion specifiers and omits one.

The **%b** identifier expects two arguments: an *int* and a *char *. These are used as a register value and a print mask for decoding bitmasks. The print mask is made up of two parts: the base and the arguments.

The base value is the output base (radix) expressed as an octal value; for example, \10 gives octal and \20 gives hexadecimal. The arguments are made up of a sequence of bit identifiers. Each bit identifier begins with an *octal* value which is the number of the bit (starting from 1) this identifier describes. The rest of the identifier is a string of characters containing the name of the bit. The string is terminated by either the bit number at the start of the next bit identifier or NUL for the last bit identifier.

The **%D** identifier is meant to assist in hexdumps. It requires two arguments: a *u_char* * pointer and a *char* * string. The memory pointed to by the pointer is output in hexadecimal one byte at a time. The string is used as a delimiter between individual bytes. If present, a width directive will specify the number of bytes to display. By default, 16 bytes of data are output.

The **%n** conversion specifier is not supported.

The **log**() function uses syslog(3) level values LOG_DEBUG through LOG_EMERG for its *pri* parameter (mistakenly called 'priority' here). Alternatively, if a *pri* of -1 is given, the message will be appended to the last log message started by a previous call to **log**(). As these messages are generated by the kernel itself, the facility will always be LOG_KERN.

## RETURN VALUES

The **printf**() and the **uprintf**() functions return the number of characters displayed.

## EXAMPLES

This example demonstrates the use of the **%b** and **%D** conversion specifiers. The function

```
void
printf_test(void)
{

        printf("reg=%b\n", 3, "\10\2BITTWO\1BITONE");
        printf("out: %4D\n", "AAZZ", ":");
}
```

will produce the following output:

```
reg=3<BITTWO,BITONE>
out: 41:41:5a:5a
```

The call

```
log(LOG_DEBUG, "%s%d: been there.\n", sc->sc_name, sc->sc_unit);
```

will add the appropriate debug message at priority "kern.debug" to the system log.

**SEE ALSO**

printf(3), syslog(3)