printk and console log level.

printk is used in kernel programming to print messages into the kernel logs.

The syntax of printk is

```
printk ("log level" "message", <arguments>);
```

The log levels decide the importance of the message being printed, kernel defines 8 log levels in the file printk.h

```
#define KERN_EMERG "<0>" /* system is unusable*/
#define KERN_ALERT "<1>" /* action must be taken immediately*/
#define KERN_CRIT "<2>" /* critical conditions*/
#define KERN_ERR "<3>" /* error conditions*/
#define KERN_WARNING "<4>" /* warning conditions*/
#define KERN_NOTICE "<5>" /* normal but significant condition*/
#define KERN_INFO "<6>" /* informational*/
#define KERN_DEBUG "<7>" /* debug-level messages*/
```

We can see each log level corresponds to a number and the lower the number higher the importance of the message.
The levels are useful in deciding what should be displayed to the user on the console and what should not be.

Every console has log level called as the the console log level and any message with a log level number lesser than the console log level gets displayed on the console, and other messages which have a log level number higher or equal to the console log level are logged in the kernel log which can be looked into using the command "dmesg".

The console loglevel can be found by looking into the file /proc/sys/kernel/printk

```
$ cat /proc/sys/kernel/printk
4 4 1 7
```

The first number in the output is the console log level, the second is the default log level, third is the

minimum log level and fourth is the maximum log level.

Log level 4 corresponds to KERN_WARNING. Thus all the messages with log levels 3,2,1 and 0 will get displayed on the screen as well as logged and the messages with log level 4,5,6,7 only get logged and can be viewed using "dmesg".

The console log level can be changed by writing into the proc entry

```
$ echo "6" > /proc/sys/kernel/printk
$ cat /proc/sys/kernel/printk
6 4 1 7
```

Now the console log level is set to 6, which is KERN_INFO.

We can test logging by using the following module

hello.c:

```
#include<linux/kernel.h>
#include<linux/module.h>
#include<linux/init.h>

static int hello_init(void)
{
 printk(KERN_WARNING "Hello, world \n ");
 return 0;
}

static void hello_exit(void)
{
 printk(KERN_INFO "Goodbye, world \n");
}

module_init(hello_init);
module_exit(hello_exit);
```

The printk called in the init function uses KERN_WARNING which is log level and lesser than 6 which is the console log level and hence should be seen on the screen.

The printk used in the exit function is KERN_INFO which is log level 6,same as the console log level, and hence should not be visible on the screen.

Note: We can test the operation of the code only by logging into a text mode as none of the messages are displayed on a terminal of GUI.

Makefile:

```
ifneq ($(KERNELRELEASE),)
   obj-m := hello.o
else

KERNELDIR ?= /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

default:
 make -C $(KERNELDIR) M=$(PWD) modules
clean:
 $(MAKE) -C $(KERNELDIR) M=$(PWD) clean
endif
```

Compile and insert

```
$ make
$ insmod hello.ko
[5377.966743] Hello world
```

We can see the hello world being printed on the screen.

```
$ rmmmod hello
$ dmesg| tail -2
[5424.190552] Good bye world
```

The good bye world message gets logged but is not printed on the screen but can be see in the logs.

Thus using printk and the console log levels we can control the kernel messages visible to the user.

Newer Post Older Post Home
Subscribe to: Post Comments (Atom)

Follow by Email

Email address... Submit

Subscribe Now

Subscribe in a reader

Blog Archive

- 2018 (3)
  - May (3)
- 2017 (12)
  - December (3)
  - August (1)
  - July (1)
  - June (2)
  - May (2)
  - March (1)
  - January (2)
- 2016 (6)
  - December (3)
  - May (1)
  - January (2)
- 2015 (7)
  - November (2)
  - July (1)
  - May (1)
  - March (2)
  - January (1)

- 2014 (36)
  - December (3)
  - November (1)
  - October (3)
  - September (3)
  - August (5)
  - July (5)
  - June (3)
  - May (4)
  - March (3)
  - February (2)
  - January (4)
- 2013 (81)
  - December (6)
  - November (8)
  - October (5)
  - September (3)
  - August (4)
  - July (8)
  - June (9)
  - May (3)
  - April (6)
  - March (10)
  - February (8)
  - January (11)
- 2012 (174)
  - December (17)
  - November (4)
  - October (7)
  - September (5)
  - August (7)
  - July (18)
    - offsetof: Using the offsetof funcion in c
    - Answers to linux history quiz
    - Linux history quiz
    - expand: To convert a tab space to space characters...
    - tr: To translate characters
    - gzip: Compressing files using gzip
    - bzip2: Compressing files using bzip2
    - tar: Using the tar command to create an archive.
    - using printk_ratelimit
    - paste: Join multiple files into one
    - printk and console log level.
    - cut: To pick parts of lines from a file.
    - Unknown symbol find_task_by_pid_ns
    - Module to find a task from its pid
    - csplit : Split a file based on patterns
    - Finding the kernel version (release)
    - split: To split a file based on line or byte size
    - readonly arguments in a script
  - June (38)
  - May (31)
  - April (20)
  - March (7)
  - February (10)
  - January (10)
- 2011 (69)
  - December (5)
  - November (1)
  - October (4)
  - September (13)
  - August (11)
  - July (12)
  - June (3)
  - May (3)
  - April (4)
  - March (5)
  - February (4)
  - January (4)
- 2010 (85)
  - December (2)
  - November (6)
  - October (4)
  - September (11)
  - August (10)
  - July (9)
  - June (8)
  - May (15)
  - April (18)
  - March (2)
- 2009 (1)
  - December (1)

Followers

About Me

Tux Think

View my complete profile