

NOTE (etl 공지사항 ‘실습 관련 링크’)

- 본 실습 자료: **Development Environment and Debugging Tips-0**
 - <https://github.com/hyojeonglee/osfall2019/tree/master/presentations>
- 수령한 기기 번호 및 제본 명단 → 공유문서에 각자 업데이트
- 실습실 컴퓨터 사용을 위한 컴퓨터공학부 통합계정 생성 및 구성원 신청 (id.snucse.org)
 - 관련해서 계정 생성 혹은 구성원 신청을 한 적이 없는 경우
 - 기존 id.snucse.org 계정으로 컴퓨터에 로그인이 안되는 경우
- 라즈베리파이 키트 확인
 - SD카드 꽂혀있는 USB 리더
 - 전원 케이블
 - 시리얼 통신 케이블
 - 보드



Linux Kernel Exploration

Development Environment & Debugging Tips

Sep 18, 2019

SNU Operating Systems

NOTE

- 교재/기기 수령 및 테스트
- 과제 #0
 - 기한: ~ 9/24 화 13시
 - 방법: 개별 메일 제출
- 과제 #1
 - 기한: ~ 10/8 화 13시
 - 방법: 팀별 repository에 proj1 branch 생성 / 발표자료는 대표 한 명이 메일 제출
 - 당일 수업시간에 랜덤 2팀 발표 (5분 이내. 개발 내용, 트러블 슈팅, ...)

Index

- **Github repository**
- **#0 Compiling the Kernel & Flashing the Device**
- **#1 Overview**
- Booting test
- Development Environment
 - (1) with Serial communication or HDMI cable
 - (2) Ctags and Cscope
- Debugging Tips

Github repository

[main] <https://github.com/hyojeonglee/osfall2019>

[per-team] <https://github.com/hyojeonglee/osfall2019-teamX>

Notification Settings

1

A screenshot of a GitHub repository page for 'hyojeonglee / osfall2019'. The top navigation bar shows 'Code' is selected. Below it, there are links for Issues (0), Pull requests (0), Projects (0), Wiki, and Security. The repository name is 'Undergraduate Operating Systems course (2019 fall)'. A 'Manage topics' link is present. The main stats area shows 1 commit, 1 branch, and 0 releases. A dropdown menu under 'Create new pull request' has options: 'Branch: master', 'New pull request', and 'Create new topic'. At the bottom, it says 'hjlee and hjlee first commit' with a timestamp 'ago'. On the right, there's a 'Notifications' sidebar with four options: 'Not watching' (Be notified only when participating or @mentioned), 'Releases only' (Be notified of new releases, and when participating or @mentioned), 'Watching' (Be notified of all conversations, checked), and 'Ignoring' (Never be notified). A red box highlights the 'Watching' option.

2

A screenshot of the GitHub 'Emails' settings page. It lists three email addresses: 'hyon2224@naver.com' (Backup, Visible in emails), 'hyojeong5663@gmail.com' (Primary, Visible in emails, Receives notifications), and '1826027@snu.ac.kr' (Visible in emails). Each entry has a trash can icon at the end.

3

Email notification preferences

Default notification email

hyojeong5663@gmail.com

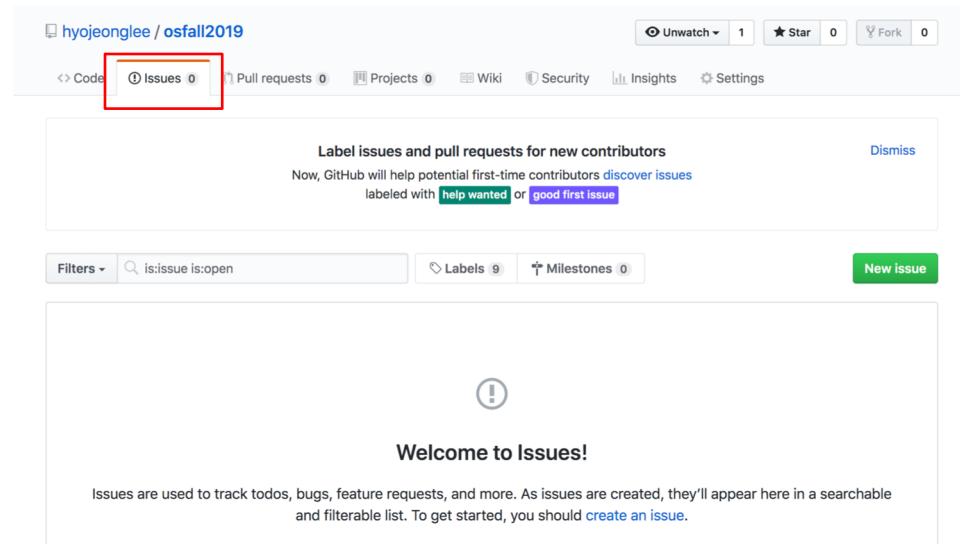
Choose which email updates you receive on conversations you're participating in or watching

- Comments on Issues and Pull Requests
- Pull Request reviews
- Pull Request pushes
- Include your own updates

1. <https://github.com/hyojeonglee/osfall2019>
2. <https://github.com/settings/emails>
3. <https://github.com/settings/notifications>

Issue Board

- Please use the GitHub issue board for all questions that can be shared with other students
- Active participation highly encouraged: help out others!
- Do NOT close your issues so that other students can find them easily



#0 Compiling the Kernel & Flashing the Device

- **Information**

- OS (kernel): Tizen 5.0
- Device: Raspberry Pi 3 Model B

- **Environment**

- **Native Ubuntu : Strongly recommended-!!!**
- Native OS or Virtual Machines
 - Building process is typically much faster when using more than one core
- List of possible hypervisors
 - VirtualBox
 - VMware (not tested)
 - Parallels (not tested)
 - ...

#0 Compiling the Kernel & Flashing the Device

- Step (detail docs at <https://github.com/hyojeonglee/osfall2019/blob/master/doc/Project0.md>)
 1. Download the Tizen kernel source code.
 2. Build the kernel and make the booting image.
 3. Flash the SD card.

#1 Overview

- **Write a system call**

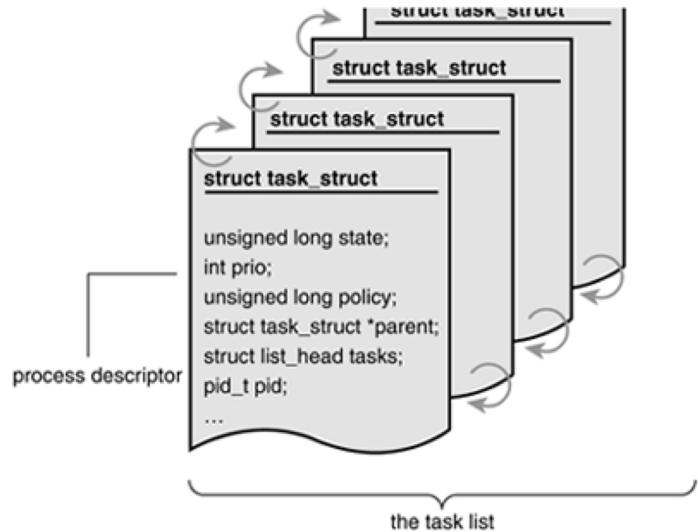
- `int ptree(struct prinfo *buf, int *nr)`
- System call number **398**
- You can name your function `sys_ptree`; doesn't matter as long as it works

- **Test your system call**

- Print the entire process tree in pre-order

- **Reference**

- Detail docs at <https://github.com/hyojeonglee/osfall2019/blob/master/doc/Project1.md>
- Project 1 Help Document at <https://github.com/hyojeonglee/osfall2019/tree/master/presentations>



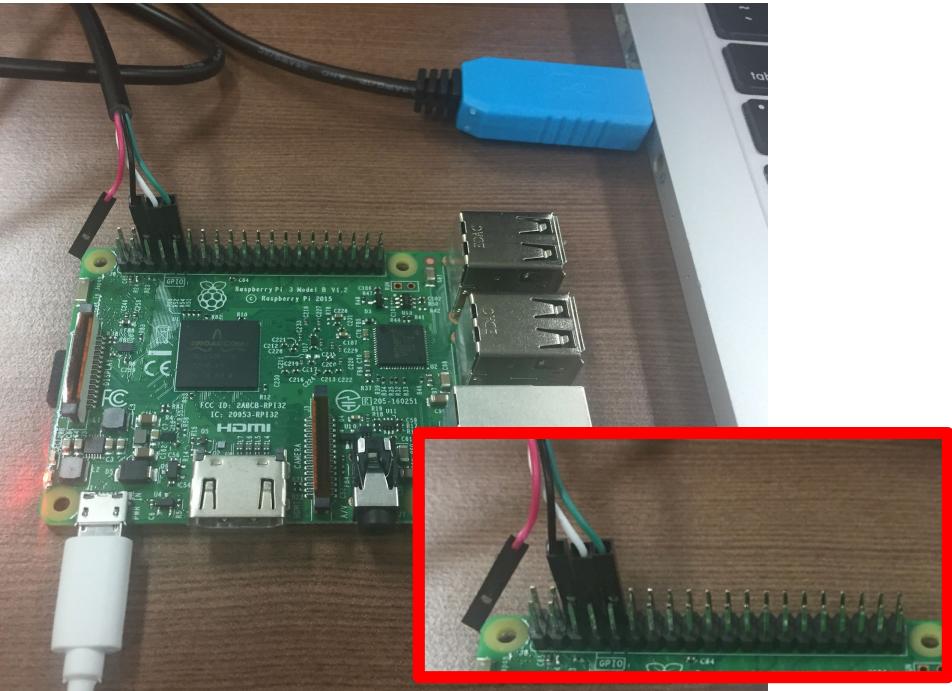
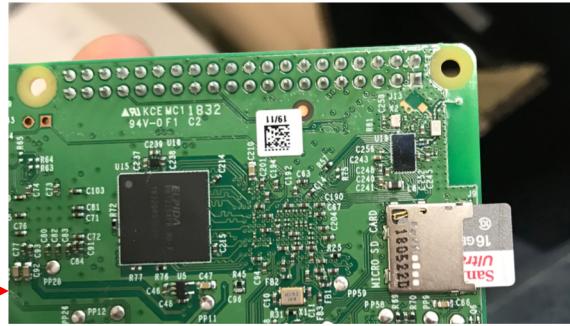
Booting Test

Index

- Connect your Raspberry Pi 3 with your computer using USB-to-UART cable.
- Step
 1. Complete #0 and unmount the SD card from Ubuntu desktop.
 2. Insert the card to Tizen board. (with USB reader or directly)
 3. Connect the device to desktop.
 4. Communicate with screen.
 5. Log in to your device as `root`, with password `tizen`.

Step

1. Unmount the SD card from Ubuntu desktop.
2. Insert the card to Tizen board. (with USB reader
or directly)
3. Connect the device to desktop.
4. Communicate with screen,
 - sudo apt install screen
 - ls /dev or dmesg | grep tty | tail
 - Find the device. (ex) /dev/ttyUSB0
 - sudo screen /dev/ttyUSB0 115200
5. Log in to your device as root, with password tizen.



Development Environment (1)

Preparation

- Download the kernel source code
 - git clone <https://github.com/hyojeonglee/tizen-5.0-rpi3>
- sudo apt install ctags cscope
- cd tizen-5.0-rpi3
 - make tags
 - make cscope

System Settings

- **Native Ubuntu : Strongly recommended-!!!**
- Native OS or Virtual Machines
 - Building process is typically much faster when using more than one core
- List of possible hypervisors
 - VirtualBox
 - VMware (not tested)
 - Parallels (not tested)
 - ...
- You don't have to follow the mounting guide if your hypervisor can handle it automatically

(1) Serial Communication

- PuTTY
- screen
- minicom



(1) Serial Communication

- For doing project #1~4,
you may test transferring files from your local Linux machine into Tizen.
- Follow this
<https://github.com/hyojeonglee/osfall2019/blob/master/doc/MoveFilesToTizen.md>

(1) Serial Communication

- Trouble shooting: 시리얼 통신이 잘 안될 때 추가로 해볼 수 있는 작업들

(1) HDMI 케이블을 이용하여 모니터에 연결하는 방법 (다음 장부터 자세히 설명)

- 이를 통해서 부팅이 잘 되는지 먼저 확인해 볼 수 있습니다.
- 부팅이 다 되고 나면 다음 장 캡처 화면처럼 라즈베리 4개가 화면에 보이는 상태로 정지되는데 이렇게 되면 부팅에 성공한 것입니다 (초기 부팅에 시간이 조금 걸릴 수도 있습니다).
- 되지 않는다면 부팅 과정에서 문제가 있을 가능성이 크기 때문에 앞의 과정(#0)을 천천히 다시 해보시고,
- 부팅이 된다면 Raspberry Pi 3의 USB 포트에 키보드를 연결하시고 Ctrl + Alt + F2 등의 키를 이용하셔서 터미널로 화면을 전환하실 수 있고, 거기서 login 과정을 진행하실 수 있습니다.

(2) 위의 과정으로 부팅이 되는지 확인되었으면 시리얼 통신을 확인해 볼 차례인데,

현재 조교들이 파악하기로는 몇몇 시리얼 케이블들이 꽂나 불안정하다는 것을 확인하였습니다.

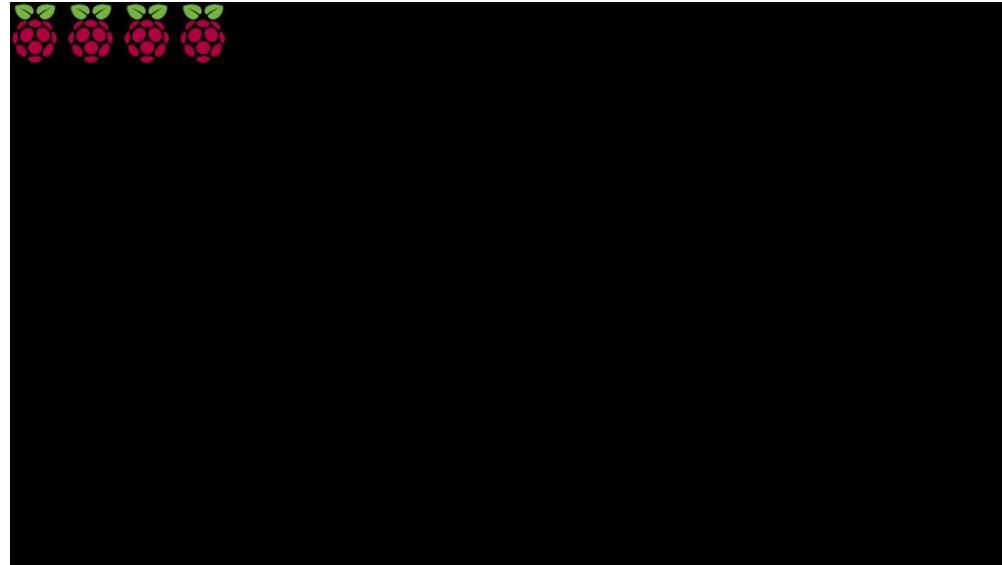
따라서 케이블을 연결한 뒤 putty 혹은 screen 과 같은 방법을 통해 시리얼 통신을 하실 때 잘 안되시면

- 1) 시리얼 케이블을 연결한 상태로 raspberry pi 3의 전원을 뽑았다 끌어보시거나
- 2) USB를 재연결 해 보시는 방법을 시도해 보실 수 있습니다.

이후에도 계속 안된다면, 관련하여 조교들에게 연락주시면 UART-to-USB 케이블을 바꿔드리도록 하겠습니다.

(2) Working Without Serial Communication: HDMI

- Simply connect your Raspberry Pi to a monitor using an HDMI cable
- The booting process will stop here:



(2) Working Without Serial Communication: HDMI

- Press Ctrl + Alt + F2 and log in
- Username: root
- Password: tizen

```
localhost login: root  
Password:
```

(ref) Configuring Internet Access

- The default wired network interface of your Raspberry Pi is `eth0`
 - `ifconfig eth0 IP netmask NETMASK`
 - `route add default gw GATEWAY`
- If you don't want to connect your Raspberry Pi to the Internet, but want to set up a communication channel, assign your computer and your rpi private IP addresses and connect them directly with a (modern) LAN cable
- Example configuration (no need to set up the default gateway):
 - Raspberry Pi:
 - IP 192.168.76.1
 - Netmask 255.255.255.0
 - Computer:
 - IP 192.168.76.2
 - Netmask 255.255.255.0

(2) Working Without Serial Communication: sdb

- Download Tizen Studio for Ubuntu from
<https://developer.tizen.org/development/tizen-studio/download>
 - It suffices to use the CLI installer
- Add `(tizen-studio-path)/tools` to your PATH environment variable
 - `export PATH=(tizen-studio-path)/tools:$PATH`
 - Add this to `.bashrc`, `/etc/profile` or such to have this handled automatically every time
- Enable `sdb` on your Raspberry Pi
 - `direct_set_debug.sh --sdb-set`
- Run `sdb` on your computer
 - `sdb connect (IP of your Raspberry Pi)`

(2) Working Without Serial Communication: sdb (cont)

- Switch to root
 - sdb root on
 - sdb root off to switch back to normal user
- Open a shell
 - sdb shell
- Send a file
 - sdb push local_path remote_path
- Receive a file
 - sdb pull remote_path optional_local_path
- More about sdb:
<https://developer.tizen.org/development/tizen-studio/web-tools/running-and-testing-your-app/sdb>

Development Environment (2)

Text Editor

- vim + ctags + Cscope + ...
 - vim
 - ctags: generates index from source files
 - Cscope: find declarations, definitions, and usages easily
 - <http://vimawesome.com/>
- Atom
 - GUI Text Editor by GitHub
 - <https://atom.io/>

ctags and Cscope

- Dealing with the kernel code only with vim is *extremely* painful
- ctags and Cscope help users to navigate the code more easily
- ctags
 - For searching variables, function, macros, and struct definitions
- Cscope
 - For searching symbols, global definitions, functions called in a specific function, functions calling a specific function, strings, files, ...

Configuring ctags

- Install ctags
 - `sudo apt install ctags`

- Create tags
 - `cd (your_linux_source_directory)`
 - `make tags`

- Edit `~/.vimrc`
 - `set tags+= (your_linux_source_directory)/tags`

ctags wontfix

```
hskim@hskim-desktop:~/tizen-5.0-rpi3$ make tags
  GEN      tags
ctags: Warning: drivers/clocksource/arm_arch_timer.c:302: null expansion of name pattern "\1"
```

- ctags works fine even with this warning
- If this annoys you, there is a fix for this:

```
diff --git a/drivers/clocksource/arm_arch_timer.c b/drivers/clocksource/arm_arch_timer.c
index 4bed671..21669ec 100644
--- a/drivers/clocksource/arm_arch_timer.c
+++ b/drivers/clocksource/arm_arch_timer.c

@@ -299,8 +299,7 @@ static u64 notrace arm64_858921_read_cntvct_el0(void)
#endif

#ifndef CONFIG_ARM_ARCH_TIMER_OOL_WORKAROUND
-DEFINE_PER_CPU(const struct arch_timer_erratum_workaround *,
-              timer_unstable_counter_workaround);
+DEFINE_PER_CPU(const struct arch_timer_erratum_workaround *, timer_unstable_counter_workaround);
EXPORT_SYMBOL_GPL(timer_unstable_counter_workaround);
--
```

Example: finding spinlock_t

- spinlock_t is used in kernel/sched/core.c

```
4860 int __cond_resched_lock(spinlock_t *lock)
4861 {
4862     int resched = should_resched(PREEMPT_LOCK_OFFSET);
4863     int ret = 0;
4864
4865     lockdep_assert_held(lock);
4866
4867     if (spin_needbreak(lock) || resched) {
4868         spin_unlock(lock);
4869         if (resched)
4870             preempt_schedule_common();
4871         else
4872             cpu_relax();
4873         ret = 1;
4874         spin_lock(lock);
4875     }
4876     return ret;
4877 }
4878 EXPORT_SYMBOL( cond_resched_lock);
```

Example: finding spinlock_t (cont'd)

- Move the cursor to `spinlock_t`, and press `Ctrl +]` to find its definition
 - vim automatically moves to `include/linux/spinlock_types.h`
- Press `Ctrl + T` to get back to `kernel/sched/core.c`

```
64 typedef struct spinlock {
65     union {
66         struct raw_spinlock rlock;
67
68 #ifdef CONFIG_DEBUG_LOCK_ALLOC
69 # define LOCK_PADSIZE (offsetof(struct raw_spinlock, dep_map))
70         struct {
71             u8 __padding[LOCK_PADSIZE];
72             struct lockdep_map dep_map;
73         };
74 #endif
75     };
76 } spinlock_t;
```

Example: finding struct task_struct

- Type :tj task_struct in vim command line to jump the definition of task_struct
 - Use tj or ts command to search for a specific tag

```
559 struct task_struct{  
560 #ifdef CONFIG_THREAD_INFO_IN_TASK  
561     /*  
562      * For reasons of header soup (see current_thread_info()), this  
563      * must be the first element of task_struct.  
564      */  
565     struct thread_info           thread_info;  
566 #endif  
567     /* -1 unrunnable, 0 runnable, >0 stopped: */  
568     volatile long                state;  
569  
570     /*  
571      * This begins the randomizable portion of task_struct. Only  
572      * scheduling-critical items should be added above here.  
573      */  
574     randomized_struct_fields_start  
575  
576     void                         *stack;  
577     atomic_t                      usage;  
578     /* Per task flags (PF_*), defined further below: */  
579     unsigned int                  flags;  
580     unsigned int                  ptrace;
```

Available ctags commands

Command	Action
Ctrl +]	Jump to the tag for the current cursor
:tj <tag>	Search for a particular tag and directly moves it if there is only one definition
:ts <tag>	Search for a particular tag, and search for the last tag if <tag> is omitted
:tn	Go to the next definition of the last tag
:tp	Go to the previous definition of the previous tag
Ctrl + T	Go back to the previous location

Configuring Cscope

- Install Cscope
 - `sudo apt install cscope`
- Generate Cscope database
 - `make cscope`
- Add the database path to `~/.vimrc`
 - `cs add (your_linux_source_directory)/cscope.out`

Example: finding a symbol

- :cs find s start_kernel
 - Use `s` to find a symbol

```
Cscope tag: start_kernel
#  line  filename / context / line
1    11  include/linux/start_kernel.h <<GLOBAL>>
      extern asmlinkage void __init start_kernel(void );
2    56  arch/x86/kernel/head32.c <<i386_start_kernel>>
      start_kernel();
3   378  arch/x86/kernel/head64.c <<x86_64_start_reservations>>
      start_kernel();
4   513  init/main.c <<start_kernel>>
      asmlinkage __visible void __init start_kernel(void )
Type number and <Enter> (empty cancels): █
```

Example: finding where a function is used

- :cs find c spin_lock
 - Use `c` to find where a function is used

```
Cscope tag: spin_lock
#  line  filename / context / line
1   466 arch/x86/crypto/sha1-mb/sha1_mb.c <<sha_complete_job>>
    spin_lock(&cstate->work_lock);
2   484 arch/x86/crypto/sha1-mb/sha1_mb.c <<sha_complete_job>>
    spin_lock(&cstate->work_lock);
3   515 arch/x86/crypto/sha1-mb/sha1_mb.c <<sha1_mb_add_list>>
    spin_lock(&cstate->work_lock);
4   464 arch/x86/crypto/sha256-mb/sha256_mb.c <<sha_complete_job>>
    spin_lock(&cstate->work_lock);
5   482 arch/x86/crypto/sha256-mb/sha256_mb.c <<sha_complete_job>>
    spin_lock(&cstate->work_lock);
6   513 arch/x86/crypto/sha256-mb/sha256_mb.c <<sha256_mb_add_list>>
    spin_lock(&cstate->work_lock);
7   852 arch/x86/include/asm/uv/uv_bau.h <<atomic_inc_unless_ge>>
    spin_lock(lock);
```

Available :cs find options

Option	Action
s	Search for C symbol names
g	Search for global variables
d	Search for functions called by a specific function
c	Search for functions calling a specific function
t	Search for text strings
e	Search for strings using regex
f	Search for files by file name
i	Search for files including a specific file

Debugging Tips

Use `printf` for kernel messaging

- `printf` does not work inside the linux kernel
- `printf` syntax
 - `printf(log level "message", <arguments>);`
- You can use `printf` instead for feeding kernel messages
 - Declared in `include/linux/kernel.h`
 - Mostly compatible with ANSI C `printf` function
 - Can give priority by concatenating pre-defined string constants
 - e.g. `printf(KERN_INFO "this syscall is called!\n");`
 - `KERN_INFO` is what you want for most cases
- You can read kernel messages using `dmesg`
 - `dmesg -w`: wait for new messages

Use `printf` for kernel messaging

- You should be aware that...
 - The internal buffer size of `printf` is limited to 1kB
 - Too many kernel messages will make the system slow or even hang completely
 - Make sure to add a line separator at the end of each message
- Log level

```
#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*/
```

- Use `KERN_ALERT` to print a message directly to the console
 - `printf(KERN_ALERT "hello world\n");`

Find where the print function is called

- `printf("%s: %s (%d)\n",
 __FILE__, __FUNCTION__,
 __LINE__);`
- Can also be used in kernel; change `printf` to `printk`

```
1 #include <stdio.h>  
2  
3 int main(void)  
4 {  
5     printf("%s: %s (%d)\n",  
6             __FILE__, __FUNCTION__, __LINE__);  
7     return 0;  
8 }  
hskim@hskim-desktop:~$ ./test  
test.c: main (6)
```

Linux Cross Reference (LXR)

- <https://elixir.bootlin.com/linux/v4.14.67/source>
- Find functions/symbols in Linux kernel

The screenshot shows a search interface for the Linux kernel source code. At the top, there is a navigation bar with a '/' icon, a search input field containing 'start_kernel', and a magnifying glass icon. Below the search bar, the page title is 'Defined in 6 files:' followed by a list of file locations where the symbol is defined. Underneath, another section titled 'Referenced in 16 files:' lists the files where the symbol is used, along with specific line numbers.

/ start_kernel

Defined in 6 files:

- arch/alpha/boot/bootp.c, line 135 (as a function)
- arch/alpha/boot/bootpz.c, line 263 (as a function)
- arch/alpha/boot/main.c, line 152 (as a function)
- arch/um/kernel/skas/process.c, line 16 (as a prototype)
- include/linux/start_kernel.h, line 11 (as a prototype)
- init/main.c, line 513 (as a function)

Referenced in 16 files:

- arch/alpha/boot/bootp.c, line 135
- arch/alpha/boot/bootpz.c, line 263
- arch/alpha/boot/main.c, line 152
- arch/frv/kernel/debug-stub.c, line 123
- arch/metag/kernel/setup.c, line 553
- arch/mips/kernel/relocate.c
 - └── line 305
 - └── line 399

Q & A

Back-up Slides