

커널 디버깅 도구

1) printk() 함수

- printf()와 유사하지만 커널의 메시지를 출력하고 관리할 수 있는 특성이 있다.
- 로그 레벨은 printk() 함수에 전달되는 문자열의 선두 문자에 “<1>” 과 같이 숫자로 등급을 표현한다. linux/kernel.h 에 정의된 선언문을 이용이 바람직하다.

상수 선언문	의미
#define KERN_EMERG	<0> 시스템이 동작하지 않는다
#define KERN_ALERT	<1> 항상 출력된다.
#define KERN_CRIT	<2> 치명적인 정보
#define KERN_ERR	<3> 오류 정보
#define KERN_WARNING	<4> 경고 정보
#define KERN_NOTICE	<5> 정상적인 정보
#define KERN_INFO	<6> 시스템 정보
#define KERN_DEBUG	<7> 디버깅 정보

커널 디버깅 도구

1) printk() 함수

- 레벨에 대한 표시를 하지 않으면 KERN_WARNING와 같은 레벨이다. 다음의 처리 결과는 모두 같다.

```
printk(KERN_WARNING "system ok\n");  
printk( "<4>" "system ok\n");  
printk( "<4> system ok\n");  
printk( "system ok\n");
```

- 커널 메시지 관리 데몬
 - klogd: 커널에서 발생하는 메시지를 기록하고 관리한다.
 - syslogd: 커널에서 발생하는 메시지와 응용 프로그램에서 요청한 시스템 정보를 기록하고 관리한다.
- printk() 사용 시 주의점
 - printk를 과도하게 사용하지 않는다. → 실행 시간이 길다
 - 개행 문자가 있어야 출력을 시작한다. → '\n'을 포함 하도록 한다.

커널 디버깅 도구

1) printk() 함수

- 사용 예

```
int gpioled_init(void) {
    int result;
    result = register_chrdev(MOD_MAJOR, MOD_NAME, &gpioled_fops);
    if(result < 0) {
        printk( KERN_ERR "Can't get any major\n");
        return result;
    }
    printk( KERN_NOTICE "Init Module: Major number %d\n", MOD_MAJOR);
    printk( KERN_NOTICE "Init Module: Major number %d\n", result);
    gpio_addr = ioremap(GPIO_BASE, GPIO_RANGE);
    // GPFSELn, #18, out
    *(gpio_addr + (GPIO_LED/10)) &= ~(6 << (((GPIO_LED)%10)*3));
    return 0;
}
```

커널 디버깅 도구

2) ftrace

- 리눅스 커널에서 제공하는 가장 강력한 트레이서
- 2.6.28 버전부터 가능, **소스 코드 수정없이** 커널 내의 함수 흐름 추적
- 주요 특징
 - Static Trace Points를 이용한 Event Tracing
인터럽트, 스케줄링, 커널 타이머 등 커널 동작을 상세히 추적
 - Dynamic Kernel Function Tracing
함수 필터 지정 시 함수의 콜 스택 정보, 콜 그래프 출력
 - Latency Tracing
wakeup, interrupt latency 추적
 - 어느 프로세스가 해당 함수를 실행하는지 추적 가능
 - 실행 시각 확인
 - 거의 시스템의 부하를 주지 않음
- 선결 조건
 - ftrace 관련 코드가 포함된 리눅스 커널
 - 커널 빌드 디렉토리의 .config 파일 확인
 - mount -t debugfs nodev /sys/kernel/debug

Kernel hacking → Tracers → Kernel Function Tracer

`CONFIG_FUNCTION_TRACER`

Kernel hacking → Tracers → Kernel Function Graph Tracer

`CONFIG_FUNCTION_GRAPH_TRACER`

Kernel hacking → Tracers → enable/disable function tracing dynamically

`CONFIG_DYNAMIC_FTRACE`

...

커널 디버깅 도구

2) ftrace

- 참조 : <https://www.kernel.org/doc/html/v6.6/trace/ftrace.html#introduction>
- 추적 및 레이턴시 관련 주요 파일 - [/sys/kernel/debug/tracing](#)

주요 파일	내용
trace	ftrace의 로그가 저장되는 파일
tracing_on	ftrace on/off (1/0)
available_tracers	사용 가능한 trace 프로그램의 종류 nop : 기본 트레이서, 이벤트만 출력 function : 함수 트레이서, set_ftracer_filter에 지정된 함수를 호출하는 함수 출력 function_graph : 함수 실행시간 및 세부 호출 정보 그래프로 표시 wakeup, wakeup_dl, wakeup_rt : wakeup latency 분석 mmiotrace : 메모리맵 IO에 대한 분석 irqsoff : interrupt latency 분석
current_tracer	현재 tracer를 설정하거나 표시하는데 사용, 기본 값 : nop available_tracers에 지정된 트레이서만 설정 가능 ** 주의 set_ftrace_filter 와 함께 설정!!

커널 디버깅 도구

2) ftrace

- 함수 추적 관련 주요 파일 - /sys/kernel/debug/tracing

주요 파일	내용
set_ftrace_filter (< > set_ftrace_notrace)	트레이스하고 싶은 함수 지정 (function 또는 function_graph 설정시 이용) available_filter_functions 에 지정된 함수만 가능 ** 주의 : current_tracer 를 function 또는 function_graph 로 지정시 이 필터에 함수를 지정하지않으면 available_filter_functions에 지정된 모든 함수를 트레이스하므로 시스템 행업!!
available_filter_functions	트레이싱 가능한 함수 목록 디바이스 드라이버나 커널 소스에 새로운 함수 구현 시 확인 가능
set_ftrace_pid	추적을 원하는 특정 프로세스 또는 스레드의 ID 지정
buffer_size_kb	ftrace 로그 버퍼 크기, 로그 내용을 더 많이 저장하는 경우 설정
function_profile_enable	

커널 디버깅 도구

2) ftrace

▪ 사용 예1

```
# echo "tcp*" > set_ftrace_filter
# echo function > current_tracer
# echo 1 > tracing_on
# sleep 1
# echo 0 > tracing_on
```

```
# cat /sys/kernel/debug/tracing/trace
```

```
# tracer: function
#
# entries-in-buffer/entries-written: 2614/2614   #P:4
#
#          _-----> irqsoff
#          / _-----> need-resched
#          | / _----> hardirq/softirq
#          || / _--> preempt-depth
#          ||| /      delay
#
#          TASK-PID      CPU#  ||||   TIMESTAMP  FUNCTION
#          |   |         |   |   ||||   |         |
#          <idle>-0      [003]  ..s. 75312.281370: tcp_wfree <-skb_r
#          elease_head_state
#          <idle>-0      [003]  ..s. 75312.437466: tcp4_gro_receive
#          <-inet_gro_receive
#          <idle>-0      [003]  ..s. 75312.437467: tcp_gro_receive <
#          -tcp4_gro_receive
#          <idle>-0      [003]  ..s. 75312.437470: tcp_v4_early_demu
#          x <-ip_rcv_finish_core.isra.0
```

커널 디버깅 도구

2) ftrace

· 사용 예2

```
# echo 0 > tracing_on
# echo function_graph > current_tracer
# echo 1 > tracing_on
# sleep 1
# echo 0 > tracing_on
```

+ : 10 us 이상
! : 100 us 이상

```
# cat /sys/kernel/debug/tracing/trace
```

```
# tracer: function_graph
```

```
#
```

```
# CPU DURATION FUNCTION CALLS
```

```
# | | | | |
```

```
0) + 13.541 us | tcp_poll();
```

```
0) 7.969 us | tcp_poll();
```

```
0) | tcp_sendmsg() {
```

```
0) | tcp_sendmsg_locked() {
```

```
0) 4.167 us | tcp_rate_check_app_limited();
```

```
0) | tcp_send_mss() {
```

```
0) | tcp_current_mss() {
```

```
0) 4.115 us | tcp_established_options();
```

```
0) + 12.969 us | }
```

```
0) + 20.885 us | }
```

```
0) 3.854 us | tcp_chrono_start();
```

```
0) | tcp_cwnd_restart() {
```

```
0) 3.906 us | tcp_init_cwnd();
```


커널 디버깅 도구

2) ftrace

▪ 사용 예3

```
# echo 0 > tracing/tracing_on
# echo irqsoff > current_tracer
# echo 1 > tracing_on
# sleep 1
# echo 0 > tracing/tracing_on
```

```
# cat /sys/kernel/debug/tracing/trace
```

```
# tracer: irqsoff
#
# irqsoff latency trace v1.1.5 on 5.10.25-v7+
# -----
# latency: 3272 us, #1976/1976, CPU#0 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:4)
#
# | task: ksoftirqd/0-11 (uid:0 nice:0 policy:0 rt_prio:0)
# -----
# => started at: _nohz_idle_balance
# => ended at: _nohz_idle_balance
#
#
# _-----> CPU#
# / _-----> irqs-off
# | / _-----> need-resched
# || / _----> hardirq/softirq
# ||| / _--> preempt-depth
# |||| /      delay
#
# cmd      pid      | time | caller
# \      /      | \    | /
ksoftirq-11 0d.s. 0us : __irq_svc <_nohz_idle_balance
ksoftirq-11 0d.s. 2us : bcm2836_arm_irqchip_handle_irq <__irq_svc
ksoftirq-11 0d.s. 4us : __handle_domain_irq <bcm2836_arm_irqchip_handle_irq
ksoftirq-11 0d.s. 5us : irq_enter <__handle_domain_irq
```

커널 디버깅 도구

2) ftrace

- event 추적 관련 주요 설정 파일 - /sys/kernel/debug/tracing

주요 파일	내용
available_events	동작시 추적 가능한 이벤트 목록 형식) 서브시스템:이벤트 (kmem:kmalloc)

/sys/kernel/debug/tracing/events
/<서브시스템>/<함수(이벤트)>

- /sys/kernel/debug/tracing/events

폴더/파일	파일	의미
enable		모든 이벤트 활성화 여부(1/0)
<서브시스템>/<이벤트>/	enable	이벤트 활성화/비활성화 (1/0)
	filter	이벤트 추적을 위해 참으로 평가되어야 하는 표현식
	format	이벤트와 파라미터의 포맷
	id	식별자 (숫자)
	trigger	이벤트 발생 시 실행할 명령어(Document/trace/ftrace.txt 파일의 'Filter commands' 섹션에 문법 정의)

커널 디버깅 도구

2) ftrace

▪ 사용 예3

```
# echo nop > current_tracer
# echo 1 > events/kmem/kmalloc
# echo 1 > events/kmem/kfree
# echo 1 > tracing_on
# sleep 1
# echo 0 > tracing_on
```

또는

```
# echo nop > current_tracer
# echo "kmem:kmalloc kmem:kfree" > current_tracer
# echo 1 > tracing_on
# sleep 1
# echo 0 > tracing_on
```

```
# cat /sys/kernel/debug/tracing/trace
```

```
# tracer: nop
```

```
#
```

```
# entries-in-buffer/entries-written: 13548/13548 #P:4
```

```
#
```

```
# _-----=> irqsoff
```

```
# / _-----=> need-resched
```

```
# | / _----=> hardirq/softirq
```

```
# || / _--=> preempt-depth
```

```
# ||| / delay
```

```
# TASK-PID CPU# ||| TIMESTAMP FUNCTION
```

```
#
```

```
gnome-shell-1598 [003] .... 78359.824995: kfree: call_site=
__sys_recvmsg+0x8d/0xc0 ptr=0000000000000000
```

```
gnome-shell-1598 [003] .... 78359.825008: kfree: call_site=
skb_free_head+0x25/0x30 ptr=000000003f4a5b9c
```

```
gnome-shell-1598 [003] .... 78359.825009: kfree: call_site=
__sys_recvmsg+0x8d/0xc0 ptr=0000000000000000
```

```
gnome-shell-1598 [003] .... 78359.825014: kfree: call_site=
__sys_recvmsg+0x8d/0xc0 ptr=0000000000000000
```

커널 디버깅 도구

2) ftrace

- event 추적 관련 주요 설정 파일 - /sys/kernel/debug/events

서브시스템	이벤트(함수)	의미
irq/ HW 인터럽트 및 SW 인터럽트(시스템 콜)를 추적하는 이벤트	irq_handler_entry	인터럽트 발생 시각과 인터럽트 번호 및 이름 출력
	irq_handler_exit	인터럽트 핸들링 완료
	softirq_raise	Soft IRQ 서비스 실행 요청
	softirq_entry	Soft IRQ 실행 서비스 시작
	softirq_exit	Soft IRQ 서비스 실행 완료
sched/ 프로세스의 스케줄링 동작 및 프로파일링을 추적하는 이벤트	sched_switch	문맥 교환 동작
	sched_wakeup	프로세스를 깨우는 동작

커널 디버깅 도구

2) ftrace

- 트레이스 옵션 - /sys/kernel/debug/tracing/options/

주요 파일	내용
func_stack_trace	set_ftrace_filter 에 지정된 함수의 콜 스택 정보 on/off (1/0) 이때 current_trace는 function 으로 지정되어 있어야 함
sym-offset	함수의 콜 스택 출력 시 함수 호출할 때 주소의 오프셋 출력 on/off (1/0)

커널 디버깅 도구

■ 리눅스 커널 인터럽트 처리

1단계 : 인터럽트 발생

- ① 프로세스 실행을 중지하고 인터럽트 벡터로 이동
- ② 인터럽트 벡터에서 인터럽트 처리를 마무리한 후 다시 프로세스를 실행하기 위해 실행 중인 프로세스 레지스터 세트를 스택에 저장
- ③ 커널 내부의 인터럽트 함수를 호출



2단계 : 인터럽트 핸들러 호출

커널 내부에서 발생한 인터럽트에 대응하는 인터럽트 디스크립터를 읽어 해당 인터럽트 핸들러 호출



3단계 : 인터럽트 핸들러 수행

인터럽트 핸들러에서 하드웨어를 직접 제어하고 유저 공간으로 전달

커널 디버깅 도구

ARM 리눅스 커널 디바이스 드라이버 인터럽트 처리 흐름

ARM

1. 인터럽트 벡터 주소 실행
2. 실행 중 레지스터를 스택 공간에 푸시

Linux
Kernel

```
__irq_svc
├── bcm2836_arm_irqchip_handle_irq
│   ├── __handle_domain_irq
│   │   ├── generic_handle_irq
│   │   │   ├── bcm2836_chained_handle_irq
│   │   │   │   ├── generic_handle_irq
│   │   │   │   │   ├── handle_level_irq
│   │   │   │   │   │   ├── handle_irq_event
│   │   │   │   │   │   │   └── __handle_irq_event_percpu
```

Device
Driver

3. 인터럽트 핸들러 실행
 - 하드웨어 설정
 - 인터럽트 변화에 대한 처리(ex: 화면 업데이트)

출처 : <http://egloos.zum.com/rousalome/v/10012154>

커널 디버깅 도구

■ 리눅스 커널 인터럽트 확인

```
pi@raspberrypi:~ $ cat /proc/interrupts
```

	CPU0	CPU1	CPU2	CPU3			
50:	15378	0	0	0	ARMCTRL-level	1 Edge	3f00b880.mailbox
51:	550083	0	0	0	ARMCTRL-level	2 Edge	VCHIQ doorbell
73:	0	0	0	0	ARMCTRL-level	48 Edge	bcm2708_fb DMA
75:	223	0	0	0	ARMCTRL-level	50 Edge	DMA IRQ
77:	7279	0	0	0	ARMCTRL-level	52 Edge	DMA IRQ
89:	5148016	0	0	0	ARMCTRL-level	64 Edge	dwc_otg, dwc_otg_p
cd, dwc_otg_hcd:usb1							
113:	1347	0	0	0	ARMCTRL-level	88 Edge	mmc0
114:	4335	0	0	0	ARMCTRL-level	89 Edge	uart-pl011
119:	4786	0	0	0	ARMCTRL-level	94 Edge	mmc1
194:	0	0	0	0	bcm2836-timer	0 Edge	arch_timer
195:	125091	260585	103399	81552	bcm2836-timer	1 Edge	arch_timer
198:	0	0	0	0	bcm2836-pmu	9 Edge	arm-pmu
199:	3	0	0	0	pinctrl-bcm2835	17 Edge	SWITCH
FIQ: usb_fiq							
IPI0:	0	0	0	0	CPU wakeup interrupts		
IPI1:	0	0	0	0	Timer broadcast interrupts		
IPI2:	1302	1503	1460	1450	Rescheduling interrupts		
IPI3:	49659	1194239	446312	183189	Function call interrupts		
IPI4:	0	0	0	0	CPU stop interrupts		
IPI5:	15528	17421	11564	9763	IRQ work interrupts		
IPI6:	0	0	0	0	completion interrupts		

인터럽트
번호

인터럽트
발생 횟수

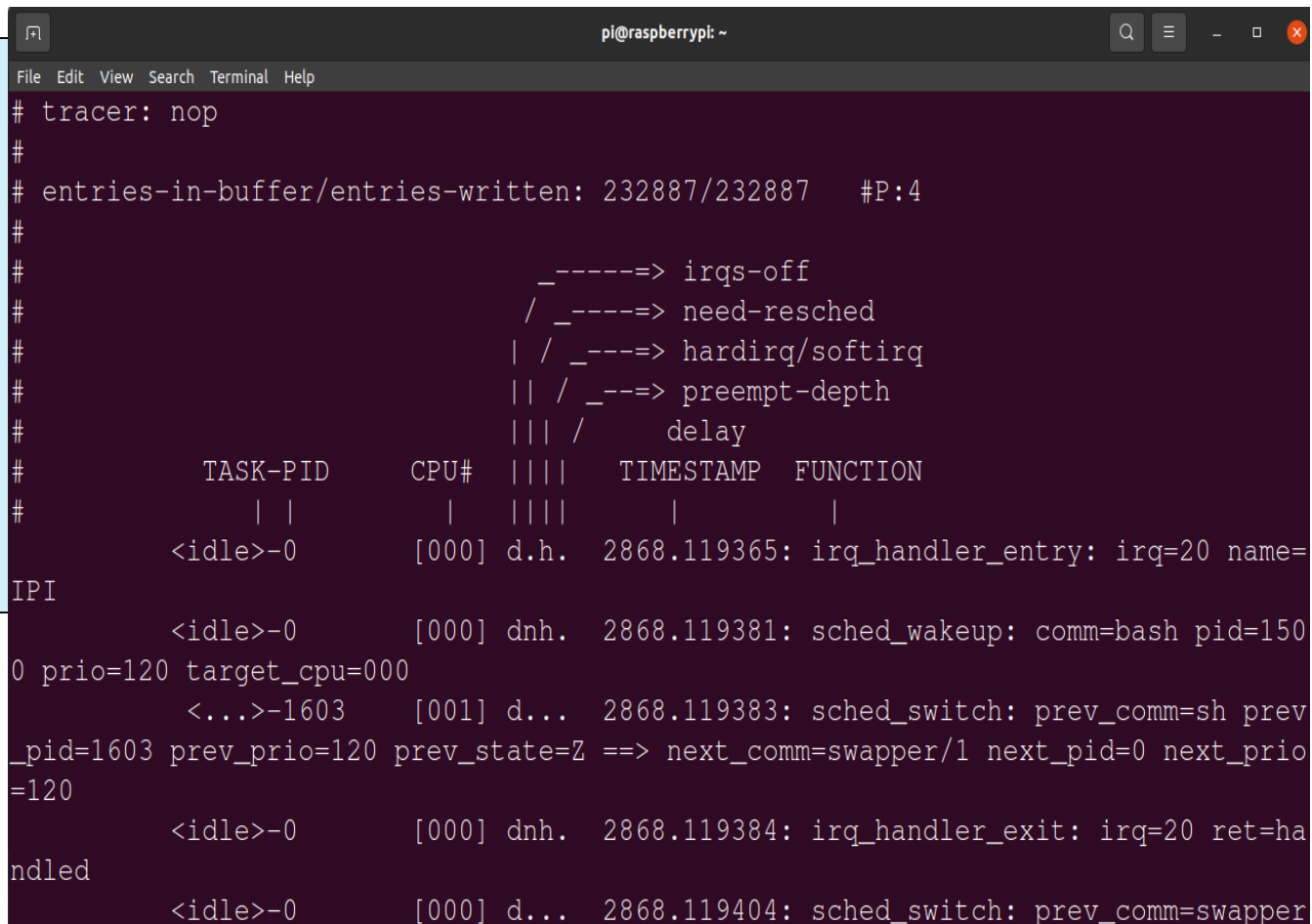
인터럽트
이름

커널 디버깅 도구

2) ftrace

· 사용 예4

```
# echo 0 > tracing_on
# echo nop > current_tracer
# echo 0 > events/enable
# echo 8096 > buffer_size_kb
# echo 1 > events/sched/sched_switch/enable
# echo 1 > events/sched/sched_wakeup/enable
# echo 1 > events/irq/irq_handler_entry/enable
# echo 1 > events/irq/irq_handler_exit/enable
# echo 1 > tracing_on
# sleep 3
# echo 0 > tracing_on
```



```
pi@raspberrypi: ~
File Edit View Search Terminal Help
# tracer: nop
#
# entries-in-buffer/entries-written: 232887/232887   #P:4
#
#          _-----=> irq=off
#          / _-----=> need-resched
#          | / _---=> hardirq/softirq
#          || / _--=> preempt-depth
#          ||| /      delay
#
#          TASK-PID      CPU#  | |||   TIMESTAMP  FUNCTION
#          | |          |   | |||   |          |
<idle>-0      [000] d.h.   2868.119365: irq_handler_entry: irq=20 name=
IPI
<idle>-0      [000] dnh.   2868.119381: sched_wakeup: comm=bash pid=150
0 prio=120 target_cpu=000
<...>-1603    [001] d...   2868.119383: sched_switch: prev_comm=sh prev
_pid=1603 prev_prio=120 prev_state=Z ==> next_comm=swapper/1 next_pid=0 next_prio
=120
<idle>-0      [000] dnh.   2868.119384: irq_handler_exit: irq=20 ret=ha
ndled
<idle>-0      [000] d...   2868.119404: sched_switch: prev_comm=swapper
```

커널 디버깅 도구

2) ftrace

· 사용 예5

```
# echo function > current_tracer
# echo 1 > events/sched/sched_switch/enable
# echo 1 > options/function_stack_trace
# echo
# echo 1 > tracing_on
# sleep 3
# echo 0 > tracing_on
```

```
# tracer: function
#
# entries-in-buffer/entries-written: 17855/17855   #P:4
#
#          _-----> irqsoft
#          / _-----> need-resched
#          | / _-----> hardirq/softirq
#          || / _-----> preempt-depth
#          ||| / _-----> delay
#
#          TASK-PID    CPU#  | |||   TIMESTAMP  FUNCTION
#          | |         |   | |||   |         |
#          bash-2478    [002] d... 1580.407069: sched_switch: prev_com
m=bash prev_pid=2478 prev_prio=120 prev_state=S ==> next_comm=kworker/u2
56:0 next_pid=2782 next_prio=120
#          kworker/u256:0-2782    [002] d... 1580.407186: sched_switch: prev_com
m=kworker/u256:0 prev_pid=2782 prev_prio=120 prev_state=I ==> next_comm=
swapper/2 next_pid=0 next_prio=120
#          <idle>-0          [000] d... 1580.407270: irq_enter_rcu <-sysvec
_call_function_single
#          <idle>-0          [000] d... 1580.407283: <stack trace>
#          => irq_enter_rcu
#          => sysvec_call_function_single
#          => asm_sysvec_call_function_single
#          => native_safe_halt
#          => acpi_idle_do_entry
#          => acpi_idle_enter
#          => cpuidle_enter_state
```

커널 디버깅 도구

2) ftrace

- 커널 소스와 ftrace의 로그 메시지

ftrace의 각 이벤트 별 추적 메시지를 출력하는 함수는 “trace_<이벤트명>” 의 형식으로 구성

ftrace의 이벤트	출력 함수
sched_switch	trace_sched_switch
irq_handler_entry	trace_irq_handler_entry
irq_handler_exit	trace_irq_handler_exit
• • •	• • •

```
++*switch_count;

psi_sched_switch(prev, next, !task_on_rq_queued(prev));

trace_sched_switch(preempt, prev, next);

/* Also unlocks the rq: */
rq = context_switch(rq, prev, next, &rf);
} else {
    rq->clock_update_flags &= ~(RQCF_ACT_SKIP|RQCF_REQ_SKIP);
    rq_unlock_irq(rq, &rf);
}
```

"kernel/sched/core.c" 8482 lines --53%-- 4529,1-8 53%

cscope 를 이용한 trace_sched_switch() 호출 위치 검색

커널 디버깅 도구

2) ftrace

로그 메시지 분석 1

프로세스 이름	PID	CPU번호	컨텍스트 정보	타임 스탬프	이벤트
kworker/u8:2 -	125	[000]	d.h.	1284.875630:	irq_handler_entry: irq=86 name=mmc1



d

.

h

.

인터럽트 활성화 여부	선점스케줄링 설정 여부	인터럽트/Soft IRQ 컨텍스트 여부	preempt_count 값
d : 해당 CPU 라인의 인터럽트 비활성화 상태	n : 선점 스케줄링될 수 있는 상태	h : 인터럽트 컨텍스트 s : Soft IRQ 컨텍스트	0~3 : 프로세스의 thread_info 구조체의 preempt_count 값

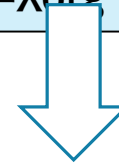
. : 해당 항목이 활성화되어 있지 않음을 의미

커널 디버깅 도구

2) ftrace

- 로그 메시지 분석 2

프로세스 이름	PID	CPU번호	컨텍스트 정보	타임 스탬프	이벤트
gnome-terminal--	1988	[001]	d...	92515.151690:	sched_switch: prev_comm=gnome-terminal- prev_pid=1988 prev_prio=120 prev_state=R ==> next_comm=Xorg next_pid=1452 next_prio=120



prev_comm=gnome-terminal- prev_pid=1988 prev_prio=120 prev_state=R ==>

스케줄링 되어야하는 프로세스 정보

next_comm=Xorg next_pid=1452 next_prio=120

다음에 수행될 프로세스 정보

FTRACE 를 이용한 커널 디버깅

■ irq setup 추적 스크립트

```
#!/bin/bash

echo 0 > /sys/kernel/debug/tracing/tracing_on
sleep 1

echo nop > /sys/kernel/debug/tracing/current_tracer
echo 0 > /sys/kernel/debug/tracing/events/enable
sleep 1

echo 8096 > /sys/kernel/debug/tracing/buffer_size_kb

echo 1 > /sys/kernel/debug/tracing/events/sched/sched_switch/enable
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_wakeup/enable
sleep 1

echo 1 > /sys/kernel/debug/tracing/events/irq/irq_handler_entry/enable
echo 1 > /sys/kernel/debug/tracing/events/irq/irq_handler_exit/enable

echo 1 > /sys/kernel/debug/tracing/tracing_on
```

FTRACE 를 이용한 커널 디버깅

로그 추출 스크립트

```
#!/bin/bash  
echo 0 > /sys/kernel/debug/tracing/tracing_on  
echo "ftrace off"  
sleep 3  
cp /sys/kernel/debug/tracing/trace .  
mv trace ftrace_log.txt
```

FTRACE 를 이용한 커널 디버깅

ftrace 관련도구

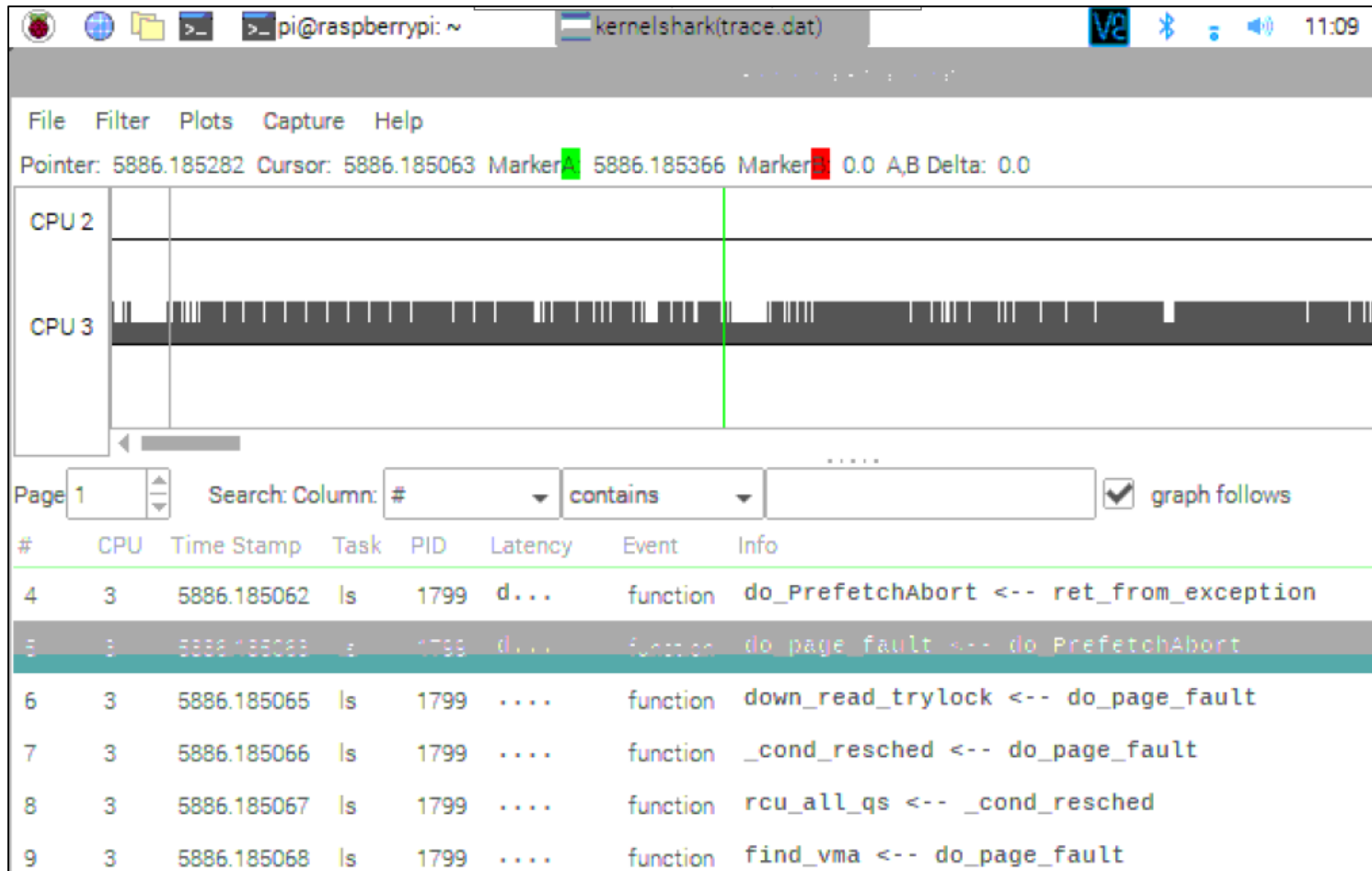
- trace-cmd : 명령어 라인에서 ftrace 사용을 위한 도구

```
root@raspberrypi:~# trace-cmd record -p function -F ls
  plugin 'function'
button_function_trace.log  irq_setup.sh  trace.dat      trace.dat.cpu1  trace.dat.cpu3
button_trace.log          irq_trace.log trace.dat.cpu0  trace.dat.cpu2
CPU0 data recorded at offset=0x45e000
    0 bytes in size
CPU1 data recorded at offset=0x45e000
    0 bytes in size
CPU2 data recorded at offset=0x45e000
    0 bytes in size
CPU3 data recorded at offset=0x45e000
    585728 bytes in size
```


FTRACE 를 이용한 커널 디버깅

ftrace 관련도구

- kernelshark : trace-cmd 도구의 수행 결과 trace.dat 파일을 GUI 형식으로 분석



디바이스 드라이버TRACE

1. 임베디드 리눅스 커널 빌드 옵션 확인 ([kernel_hacking/Trace](#))
2. debugfs 파일시스템 마운트

라즈베리파이에서는
기본 설정

```
mount -t debugfs debugfs /sys/kernel/debug #To use ftrace first need to mount this debugfs
ln -s /sys/kernel/debug /debug # Make a link during the convenience period
cd /debug
```

3. 디바이스 드라이버 관련 함수 추출

```
cat available_filter_functions > /tmp/funcs_without_mymodule
insmod mymodule.ko
cat available_filter_functions > /tmp/funcs_with_mymodule
diff /tmp/funcs_without_mymodule /tmp/funcs_with_mymodule > /tmp/mymodule_funcs
```

**** 불필요한 부분 삭제**

4. tracer 설정

```
cat /tmp/mymodule_funcs > set_ftrace_filter
echo function_graph > current_tracer
echo 1 > tracing_on #Start trace
cat trace_pipe
```

디바이스 드라이버TRACE

5. 디바이스 드라이버 테스트

```
sudo mknod /dev/gpioled c 201 0  
sudo ./led_test
```

```
pi@raspberrypi:~ $ sudo ./led_test on  
pi@raspberrypi:~ $ sudo ./led_test off  
pi@raspberrypi:~ $
```

trace_pipe 파일

```
root@raspberrypi:/sys/kernel/debug/tracing# cat trace_pipe  
led_test-2768 [003] .... 1911.618493: gpioled_open <-chrdev_open  
led_test-2768 [003] .... 1911.618549: gpioled_write <-vfs_write  
led_test-2768 [003] .... 1911.618560: gpioled_release <-__fput  
led_test-2782 [001] .... 1917.987209: gpioled_open <-chrdev_open  
led_test-2782 [001] .... 1917.987463: gpioled_write <-vfs_write  
led_test-2782 [001] .... 1917.987616: gpioled_release <-__fput
```

〈실습〉 GPIOBUTTON 드라이버 TRACE

디바이스 드라이버 소스를 빌드한 후 라즈베리파이로 드라이버 파일과 테스트 실행 파일을 전송한 후 ftrace 를 이용하여 추적해봅시다.