

Linux System 성능 프로파일링

“커널에서 네트워크까지 제대로 성능을 프로파일링 해보자”

2017 한국 리눅스 사용자 그룹 KLUG

이호성 (Tommy Lee)

한국 리눅스 사용자 그룹 리더

한국 오라클 Linux & VM senior Engineer

Email : sprdding@gmail.com



이호성 (Tommy Lee)

- 한국 리눅스 사용자 그룹 리더로 활동
- 한국 오라클 *Linux & VM Senior Engineer*
- 난공불락 오픈소스 인프라 세미나 기획자
- 주요 관심사 : *Linux performance arch, / OpenSource container, Cloud,*
- 좋은 아빠되기 (육아), 오픈소스 관련 참여 오지랖

Today talk is..

1. 리눅스 시스템 성능 프로파일링에 대한 고민 ?

- *Linux kernel*에 포함된 *system* 영역부터 *User-Level*까지 한번에 성능에 대한 *Trouble Shooting*이 가능한가?

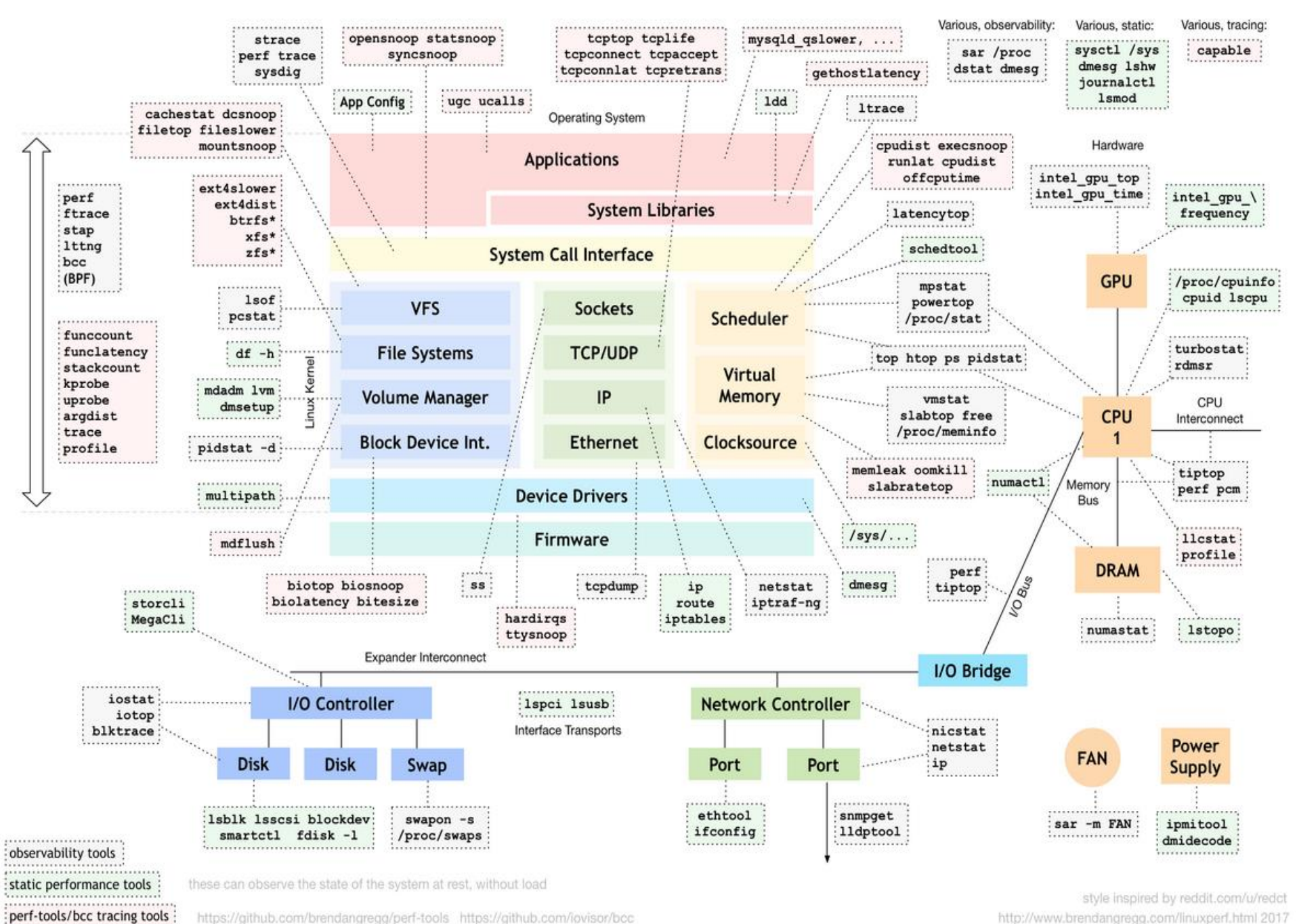
2. 개발자 입장에서 *Linux system* 전반에 대한 진단이 가능?

- 성능의 문제를 일으키는 곳이 어디인가? (*Kernel vs User space*)
- 시스템에서 제공하는 기본 툴을 가지고 성능에 대한 *Trouble Shooting*?

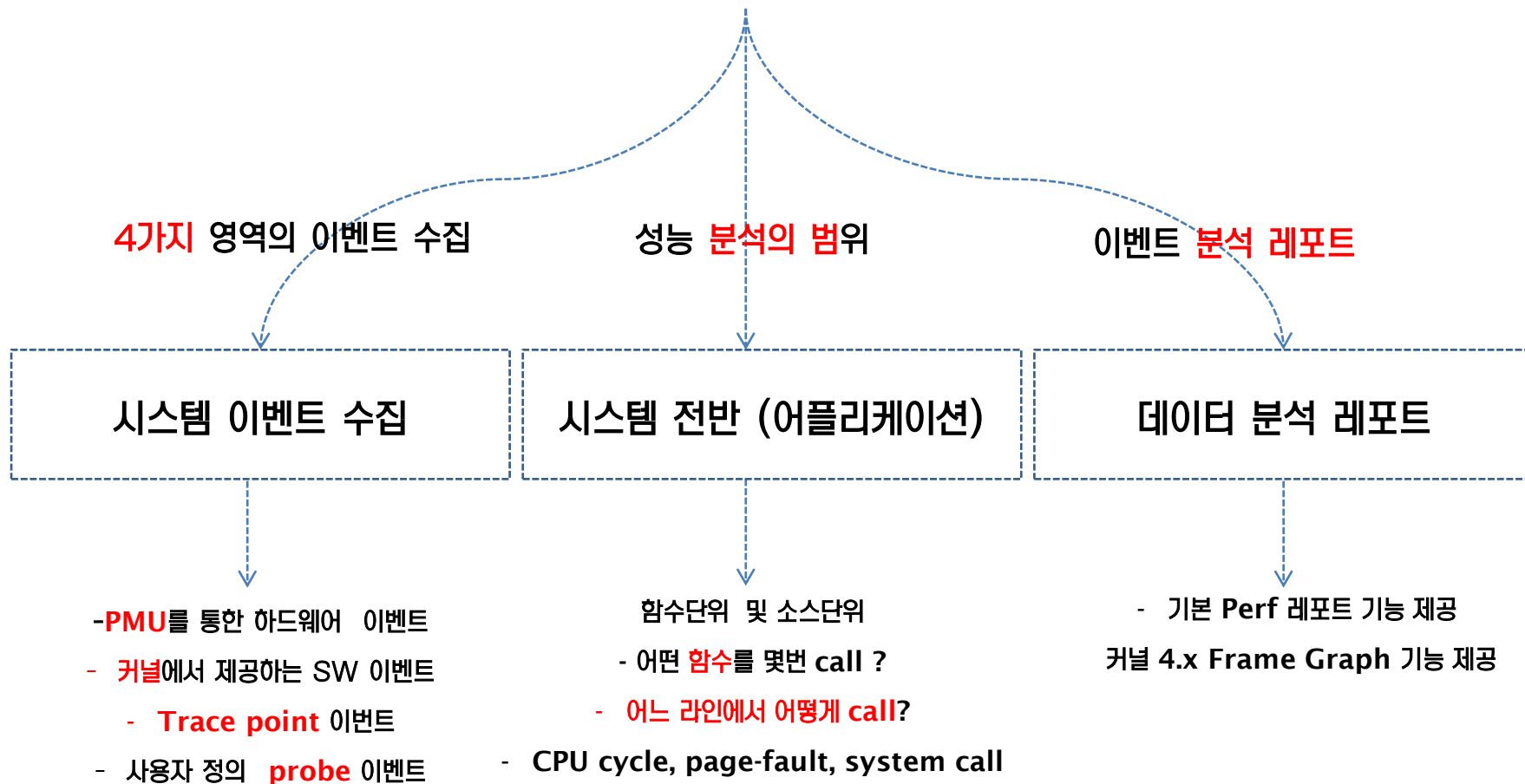
3. 리눅스 성능 분석 도구인 *perf*에 대한 소개와 시스템 영역별 성능에 대한 *trace* 가능한 몇가지 툴 소개

Contents

1. 리눅스 성능 분석 도구인 **perf tool** 에대한 소개와 **Profiling** 방법 소개
2. **perf** 를 이용해서 **커널과 관련된 성능 문제를 trace** 해보자.
3. **Linux system 영역별 성능 프로파일링** 도구 소개



perf : Performance Counter for Linux



perf tool의 Event 정의

Linux 성능 관련 **Event**의 정의 / Perf에서 가장 중요한 개념



Kernel/App 이 실행될때 발생하는 **Action**



- 시스템 전반의 **Event Sampling**

4가지 영역의 시스템 **Event** 영역



- 하드웨어/소프트웨어, Trace point, Probe Point

perf tool의 Event 정의

- Perf에서 **event sampling**을 하기 위한 **Event** 정의 종류는 아래와 같다

Event 종류	설명	예시
하드웨어 이벤트	<ul style="list-style-type: none">- PMU & PMC 를 통해서 수집될수 있는 측정될수 있는 이벤트- 데이터는 콘텐츠를 통해서 Overhead 없이 데이터 수집 가능- CPU의 유형에 따라 다르다. (cpu가 지원해야 측정 가능)	<ul style="list-style-type: none">- CPU Cycle,- Cache-misses- Cache-reference
하드웨어 캐쉬 이벤트		<ul style="list-style-type: none">- L1-cache-misses and- Branch-load 등
소프트웨어	<ul style="list-style-type: none">- 리눅스 커널에서 측정할수 있는 소프트웨어 이벤트	<ul style="list-style-type: none">- page-fault, cpu-clock
Trace Point	<ul style="list-style-type: none">- 커널 코드에 내장되어 있는 코드의 위치들의 추적 이벤트	<ul style="list-style-type: none">- syscall:sys_enter- net, ext4
Probe Point	<ul style="list-style-type: none">- 커널에 동적으로 삽입된 사용자 정의 코드 및 이벤트들	

perf tool의 profiling 목적

Perf에서 Profiling의 목적

↓

Sampling을 통한 **Event** 정보 수집 및 분석하는 행위 (언제/어디서/얼마나)

- ↓
- 시스템 전반의 **Event Sampling**
 - 명령어 : **perf list**

↓

Profiling : **병목 구간** 을 찾아 내기 위함

↓

어떤 **함수**가 CPU를 많이 사용하는지, 어느 **소스 코드**가 CPU를 많이 사용하는지

perf tool의 Event sampling 항목 root/> perf list

```
root@hslee-15ZD960-GX5BK:~/perf/disk#
root@hslee-15ZD960-GX5BK:~/perf/disk#
root@hslee-15ZD960-GX5BK:~/perf/disk# perf list

List of pre-defined events (to be used in -e):

  branch-instructions OR branches      [Hardware event]
  branch-misses                        [Hardware event]
  bus-cycles                           [Hardware event]
  cache-misses                         [Hardware event]
  cache-references                     [Hardware event]
  cpu-cycles OR cycles                 [Hardware event]
  instructions                         [Hardware event]
  ref-cycles                           [Hardware event]

  alignment-faults                    [Software event]
  bpf-output                           [Software event]
  context-switches OR cs               [Software event]
  cpu-clock                            [Software event]
  cpu-migrations OR migrations         [Software event]
  dummy                                [Software event]
  emulation-faults                    [Software event]
  major-faults                        [Software event]
  minor-faults                        [Software event]
  page-faults OR faults                [Software event]
  task-clock                           [Software event]

  L1-dcache-load-misses                [Hardware cache event]
  L1-dcache-loads                      [Hardware cache event]
  L1-dcache-stores                     [Hardware cache event]
  L1-icache-load-misses                [Hardware cache event]
  LLC-load-misses                      [Hardware cache event]
  LLC-loads                            [Hardware cache event]
  LLC-store-misses                     [Hardware cache event]
  LLC-stores                           [Hardware cache event]
  branch-load-misses                   [Hardware cache event]
  branch-loads                         [Hardware cache event]
  dTLB-load-misses                     [Hardware cache event]
  dTLB-loads                           [Hardware cache event]
  dTLB-store-misses                    [Hardware cache event]
  dTLB-stores                          [Hardware cache event]
  iTLB-load-misses                     [Hardware cache event]
  iTLB-loads                           [Hardware cache event]
  node-load-misses                     [Hardware cache event]
  node-loads                           [Hardware cache event]
  node-store-misses                    [Hardware cache event]
  node-stores                          [Hardware cache event]

  branch-instructions OR cpu/branch-instructions/ [Kernel PMU event]
```

perf tool의 trace 목적

Perf에서 trace 목적

↓

각각의 Event 들에 대한 흔적 trace 함으로써 실행 과정과 문제상황 분석

- ↓
- 추적 가능한 Event
 - Block Device 레이어에서 발생하는 i/o
 - Ext4, 또는 스케줄러 이벤트

↓

특정 커널 함수는 왜 호출이 되었는가? / 그 커널 함수가 호출된 과정은 어떤가?

- Mapping된 event 발생 지점

perf tool의 Event sampling 항목

```
root@hslee-15ZD960-GX5BK:~/perf/disk#  
root@hslee-15ZD960-GX5BK:~/perf/disk#  
root@hslee-15ZD960-GX5BK:~/perf/disk# perf list sw tracepoint  
  
List of pre-defined events (to be used in -e):  
  
alignment-faults [Software event]  
bpf-output [Software event]  
context-switches OR cs [Software event]  
cpu-clock [Software event]  
cpu-migrations OR migrations [Software event]  
dummy [Software event]  
emulation-faults [Software event]  
major-faults [Software event]  
minor-faults [Software event]  
page-faults OR faults [Software event]  
task-clock [Software event]  
  
asoc:snd_soc_bias_level_done [Tracepoint event]  
asoc:snd_soc_bias_level_start [Tracepoint event]  
asoc:snd_soc_dapm_connected [Tracepoint event]  
asoc:snd_soc_dapm_done [Tracepoint event]  
asoc:snd_soc_dapm_path [Tracepoint event]  
asoc:snd_soc_dapm_start [Tracepoint event]  
asoc:snd_soc_dapm_walk_done [Tracepoint event]  
asoc:snd_soc_dapm_widget_event_done [Tracepoint event]  
asoc:snd_soc_dapm_widget_event_start [Tracepoint event]  
asoc:snd_soc_dapm_widget_power [Tracepoint event]  
asoc:snd_soc_jack_irq [Tracepoint event]  
asoc:snd_soc_jack_notify [Tracepoint event]  
asoc:snd_soc_jack_report [Tracepoint event]  
ath10k:ath10k_htt_pktlog [Tracepoint event]  
ath10k:ath10k_htt_rx_desc [Tracepoint event]  
ath10k:ath10k_htt_stats [Tracepoint event]  
ath10k:ath10k_htt_tx [Tracepoint event]  
ath10k:ath10k_log_dbg [Tracepoint event]  
ath10k:ath10k_log_dbg_dump [Tracepoint event]  
ath10k:ath10k_log_err [Tracepoint event]  
ath10k:ath10k_log_info [Tracepoint event]
```

perf tool의 probe point

사용자 정의 Event로 특정 함수명을 정의

- Kerne debug info를 가지고 있는 상태 여야 한다.

Perf probe를 이용하기 위해서는 kernel debug info를 Enable 후 빌드

- Kprobe : kernel 영역
- uprobe : 사용자 영역

특정 함수의 argument 정보, 변수정보, 리턴값, 소스 몇 라인에서 어떤 값?

perf_event에 대한 정리

1. 리눅스 자체 성능 profiling tool.

- ‘**perf**’ 명령어를 통해서 각각의 Event 들에 대한 profiling을 진행

2. Perf를 설치하기 위해서는 **linux-tools-common** 패키지 설치

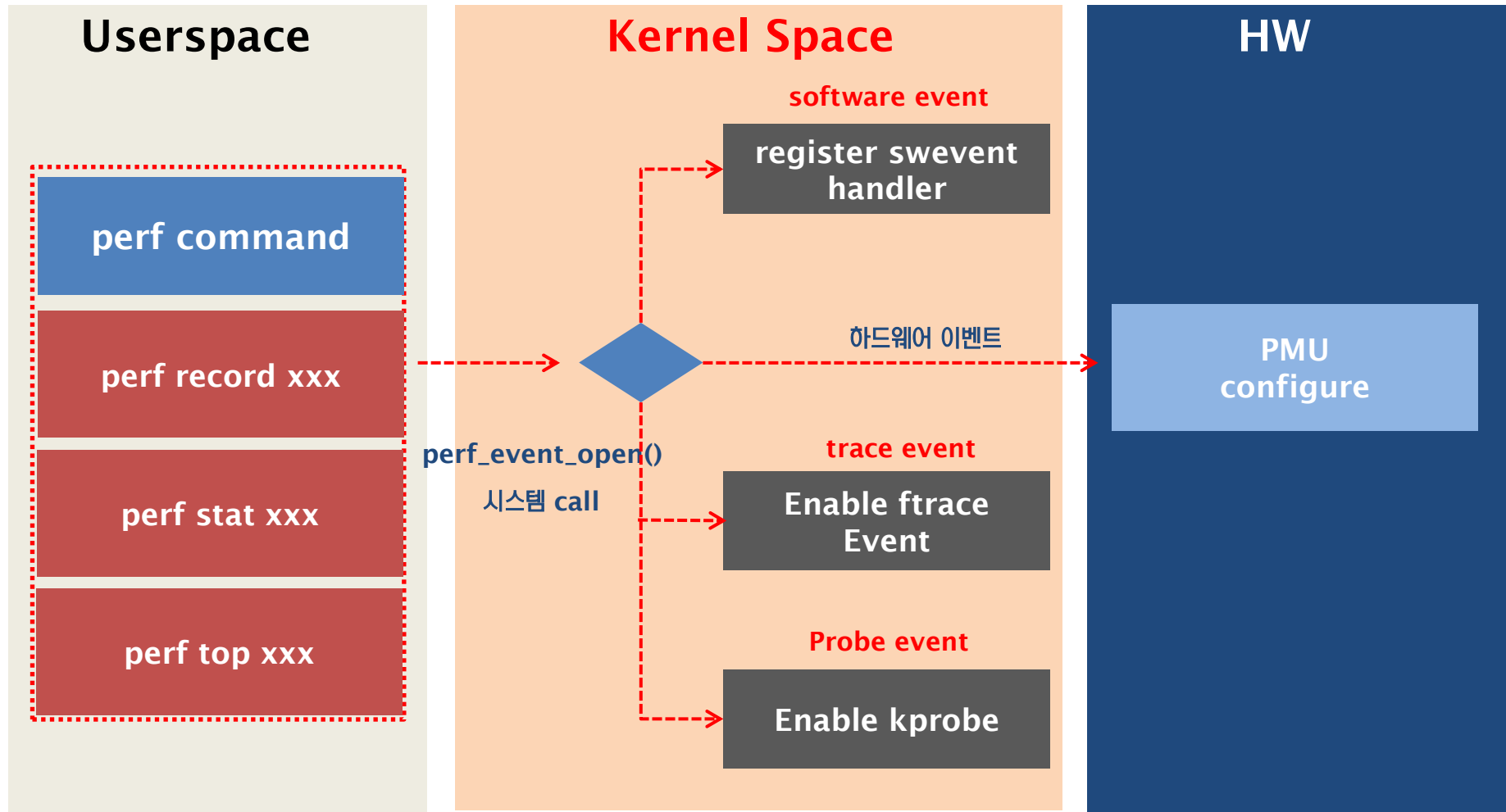
- **apt-get install linux-tools-common, linux-tools-generic**
linux-tool-`uname -r`

3. 다양한 Event 영역에 대한 Profiling 수집이 가능하다.

- User 및 Kernel 영역에 대한 trace point를 지정하여 수집
- PMC 를 통한 CPU 모니터링 Event 수집
- Trace point를 정적을 지정하다 (perf probe)

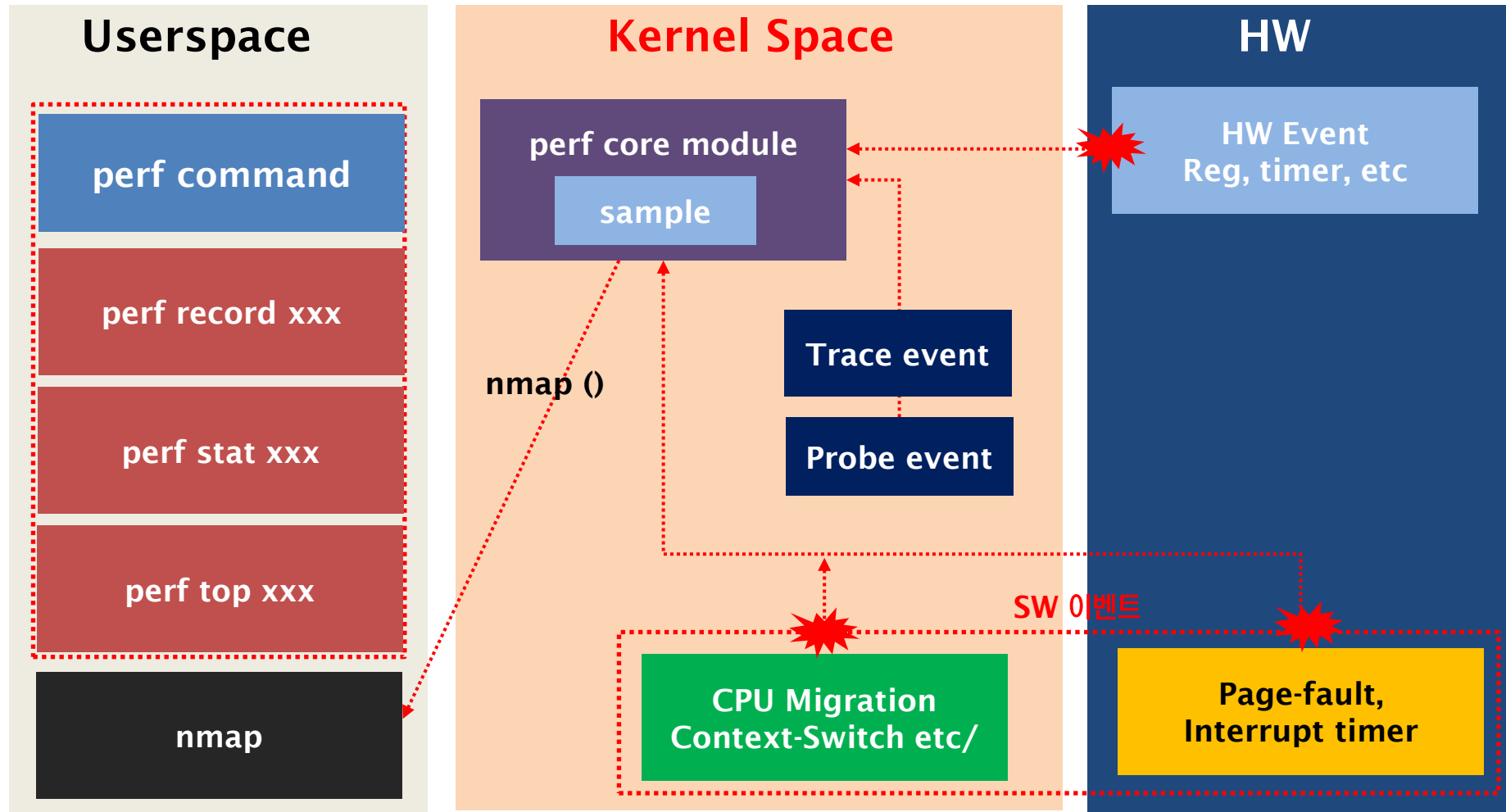
perf_event Event sampling 과정 -1

- 각각의 이벤트에 대한 sampling을 하기 위해서 perf 명령어를 **perf_event_open()** 함수를 호출.
- hw/sw에 따라서 return 받은 **file descriptor**를 통해서 각각의 성능 정보에 대한 정보를 수집하게 된다.



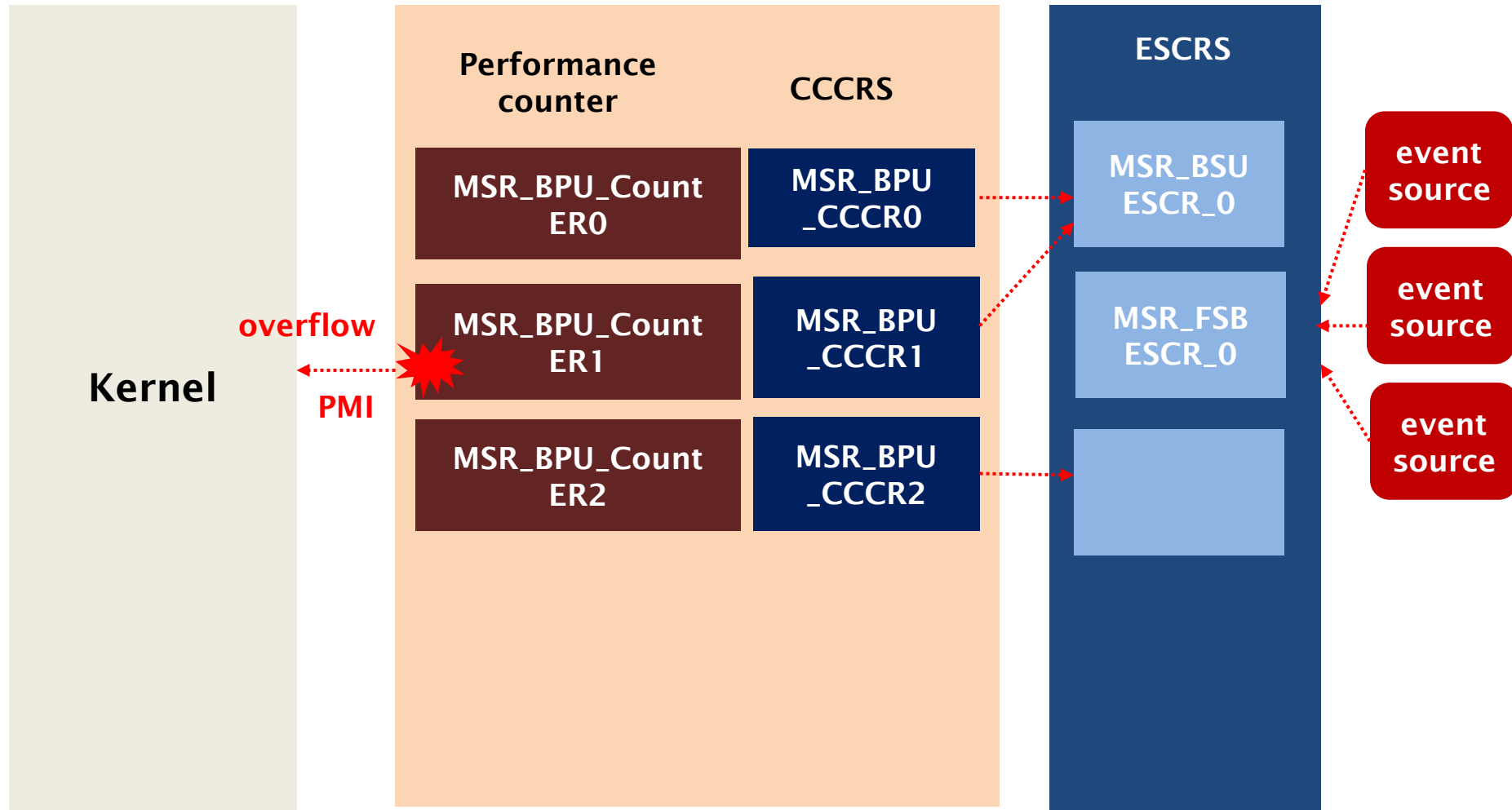
perf_event Event sampling 과정 -2

- Perf가 Event에 대한 샘플링을 진행하기 위해서 **nmap()** 함수를 호출하게 되고 특정 **sampling** 데이터를 저장하기 위한 **메모리 영역**을 확보하게 된다,

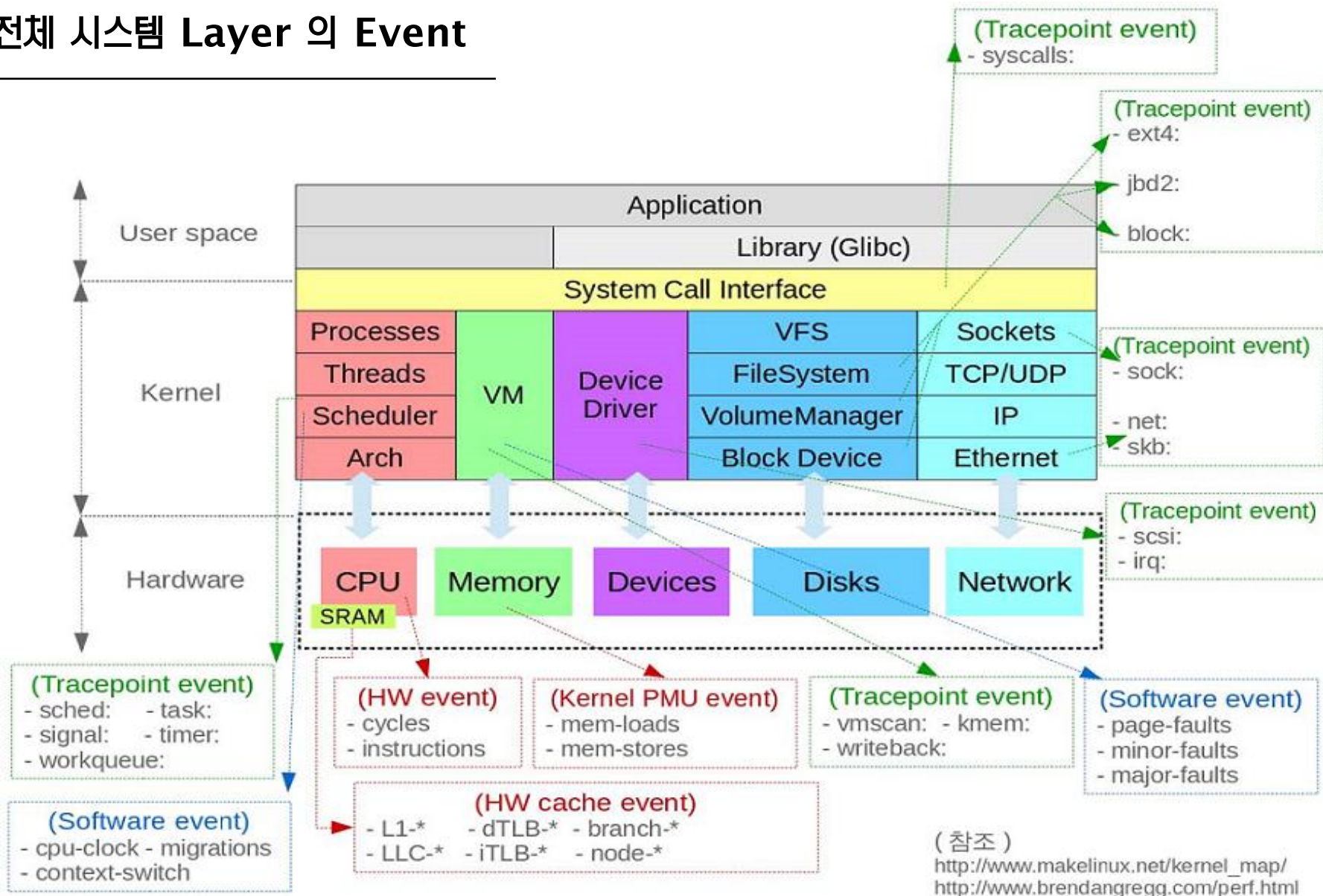


Hardware Event - Overview

- 하드웨어 이벤트 샘플 데이터 수집은 대부분 하드웨어에 의해 수행됩니다.
- 한 쌍의 성능 카운터와 **CCCR**은 **ESCR**에 의해 선택된 이벤트에서 데이터를 기록합니다.
- 커널이 PMI 인터럽트를 수신하고 레지스터에서 정보를 복사 할 때만 성능 카운터가 오버 플로우됩니다.



전체 시스템 Layer 의 Event



이미지 출처 :

<http://www.oss.kr/files/attach/images/654916/575/667/a240bb06f6b30ebb4bd6fee015a1b8ec.png>

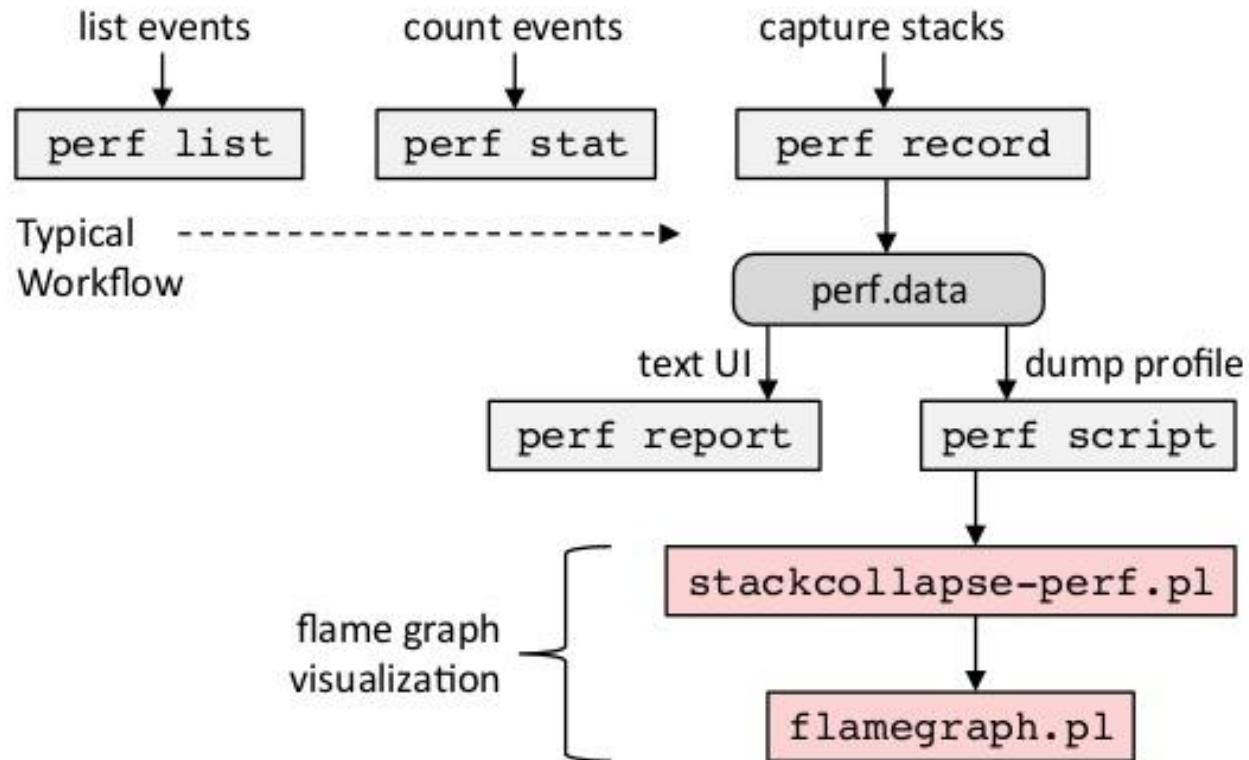
기본적인 **perf** 툴의 **Layout**을 봅시다

perf usage – 명령어 Layout

- perf 명령어에서는 프로파일링의 통계와 기록을 위해서 **stat**와 **record** 사용
- **stat** (count event)
 - kernel 의 event 카운터에 대한 통계 수치를 보여준다.
- **record**
 - **perf.data** 덤프 파일에 모든 event 기록
 - Timestamp, CPU, PID, 명령어 또는 호출된 라이브러리등
 - 이벤트 수집시에 overhead 가 발생할수 있다. (ex: -g를 이용하여 call graph를 생성할 경우)
- 그밖의 **Action** 들..
 - perf list , perf report, perf script, perf top

perf usage – Action flow

perf Actions: Workflow



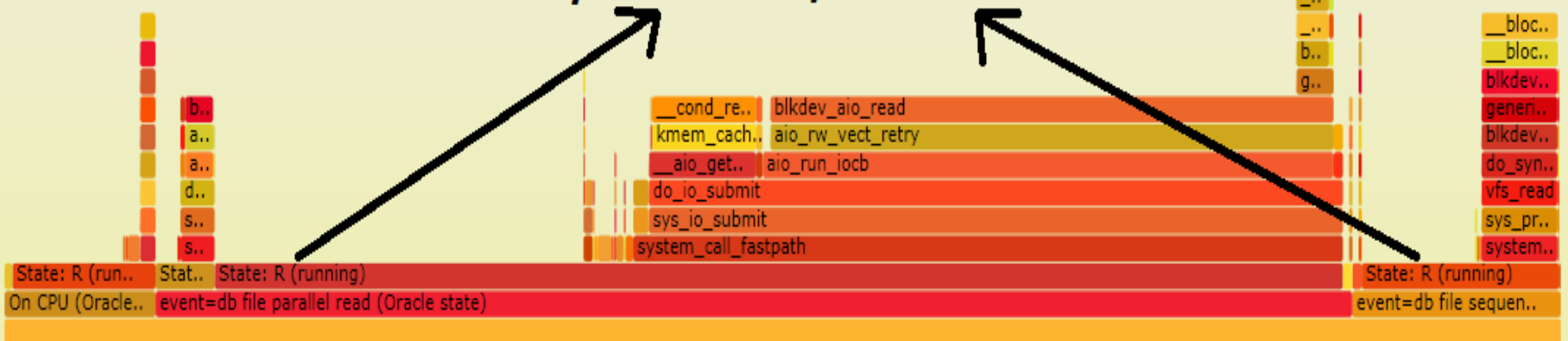
perf usage – FramGraph

- 하드웨어 뿐만 아니라 **Software**에 대한 **Event profiling** 된 결과에 대한 **Visualized** 해주는 **FrameGraph**를 **SVG** 형태로 제공하며, 별도의 **Type**을 지정하여 **H/W** 또는 **S/W**를 **Profiling** 한 결과를 **Gathering** 해서 볼수 있다

Kernel stack, OS State and Wait event profiling of Oracle random I/O (SLOB workload, low latency I/O)

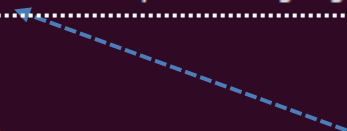
Search

CPU cycles inside I/O wait events



perf usage – FramGraph

```
root@hslee-15ZD960-GX5BK:~/perf# git clone --depth 1 https://github.com/brendangregg/FlameGraph
FlameGraph에 복제합니다...
remote: Counting objects: 112, done.
remote: Compressing objects: 100% (80/80), done.
remote: Total 112 (delta 37), reused 76 (delta 31), pack-reused 0
오브젝트를 받는 중: 100% (112/112), 975.52 KiB | 5.00 KiB/s, 완료.
델타를 알아내는 중: 100% (37/37), 완료.
연결을 확인하는 중입니다... 완료.
root@hslee-15ZD960-GX5BK:~/perf#
root@hslee-15ZD960-GX5BK:~/perf# ls
FlameGraph  cpu  disk  memory  sched
root@hslee-15ZD960-GX5BK:~/perf#
```



Github에서 복제 하여 설치

```
root@hslee-15ZD960-GX5BK:~/perf/FlameGraph# perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
Filtering for events of type: cycles:k
Failed to open /tmp/perf-3248.map, continuing without symbols
Failed to open /tmp/perf-6519.map, continuing without symbols
root@hslee-15ZD960-GX5BK:~/perf/FlameGraph# ls
README.md      example-dtrace-stacks.txt  jmaps              stackcollapse-aix.pl    stackcollapse-ljp.awk    stackcollapse-stap.pl
aix-perf.pl    example-dtrace.svg         perf_data          stackcollapse-elfutils.pl  stackcollapse-perf-sched.awk  stackcollapse-vtune.pl
demos          example-perf-stacks.txt.gz  perf.svg           stackcollapse-gdb.pl     stackcollapse-perf.pl     stackcollapse.pl
dev            example-perf.svg           pkgsplit-perf.pl   stackcollapse-go.pl      stackcollapse-pmc.pl      test
difffolded.pl  files.pl                  range-perf.pl      stackcollapse-instruments.pl  stackcollapse-recursive.pl  test.sh
docs           flamegraph.pl             record-test.sh     stackcollapse-jstack.pl   stackcollapse-sample.awk
root@hslee-15ZD960-GX5BK:~/perf/FlameGraph#
```

perf usage – scope

- 모든 CPU에 대한 Profiling 데이터 수집 : **-a**
- 특정 PID : **-p PID**
- 실행하고자 하는 명령어에 대한 데이터 수집
- 특정 CPU : **-c**
- User-level 영역에 대한 데이터 수집 : (**<event>:u**)
- Kernel-Level 영역에 대한 데이터 수집 : (**<event>:k**)
- 특정 변수값에 대한 matching을 통한 데이터 수집 : **--filter**

모든 scope에 대한 정의 및 profiling 데이터 수집을 위한 Action 은 on-liner 형태로 가능하다.

perf usage – 명령어 Layout

```
root@hslee-15ZD960-GX5BK:~/perf/cpu# perf
```

```
usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]
```

The most commonly used perf commands are:

annotate	Read perf.data (created by perf record) and display annotated code
archive	Create archive with object files with build-ids found in perf.data file
bench	General framework for benchmark suites
buildid-cache	Manage build-id cache.
buildid-list	List the buildids in a perf.data file
data	Data file related processing
diff	Read perf.data files and display the differential profile
evlist	List the event names in a perf.data file
inject	Filter to augment the events stream with additional information
kmem	Tool to trace/measure kernel memory properties
kvm	Tool to trace/measure kvm guest os
list	List all symbolic event types
lock	Analyze lock events
mem	Profile memory accesses
record	Run a command and record its profile into perf.data
report	Read perf.data (created by perf record) and display the profile
sched	Tool to trace/measure scheduler properties (latencies)
script	Read perf.data (created by perf record) and display trace output
stat	Run a command and gather performance counter statistics
test	Runs sanity tests.
timechart	Tool to visualize total system behavior during a workload
top	System profiling tool.
trace	strace inspired tool
probe	Define new dynamic tracepoints

See 'perf help COMMAND' for more information on a specific command.

perf usage – perf list

```
root@hslee-15ZD960-GX5BK:~/perf/cpu# perf list
```

```
List of pre-defined events (to be used in -e):
```

branch-instructions OR branches	[Hardware event]
branch-misses	[Hardware event]
bus-cycles	[Hardware event]
cache-misses	[Hardware event]
cache-references	[Hardware event]
cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
ref-cycles	[Hardware event]
alignment-faults	[Software event]
bpf-output	[Software event]
context-switches OR cs	[Software event]
cpu-clock	[Software event]
cpu-migrations OR migrations	[Software event]
dummy	[Software event]
emulation-faults	[Software event]
major-faults	[Software event]
minor-faults	[Software event]
page-faults OR faults	[Software event]
task-clock	[Software event]
L1-dcache-load-misses	[Hardware cache event]
L1-dcache-loads	[Hardware cache event]
L1-dcache-stores	[Hardware cache event]
L1-icache-load-misses	[Hardware cache event]
LLC-load-misses	[Hardware cache event]
LLC-loads	[Hardware cache event]
LLC-store-misses	[Hardware cache event]
LLC-stores	[Hardware cache event]
branch-load-misses	[Hardware cache event]
branch-loads	[Hardware cache event]
dTLB-load-misses	[Hardware cache event]
dTLB-loads	[Hardware cache event]
dTLB-store-misses	[Hardware cache event]
dTLB-stores	[Hardware cache event]
iTLB-load-misses	[Hardware cache event]
iTLB-loads	[Hardware cache event]
node-load-misses	[Hardware cache event]
node-loads	[Hardware cache event]
node-store-misses	[Hardware cache event]
node-stores	[Hardware cache event]

perf usage – 명령어 Layout

Action: **stat** count Scope : all CPUs

perf **stat** **-a** -g -- sleep 10

```
root@hslee-15ZD960-GX5BK:~/perf/cpu# perf stat -a -g -- sleep 10
```

Performance counter stats for 'system wide':

40102.006649	task-clock (msec)	#	4.009 CPUs utilized	(100.00%)
20,810	context-switches	#	0.519 K/sec	(100.00%)
1,613	cpu-migrations	#	0.040 K/sec	(100.00%)
4,982	page-faults	#	0.124 K/sec	(100.00%)
2,194,939,322	cycles	#	0.055 GHz	(100.00%)
<not supported>	stalled-cycles-frontend			
<not supported>	stalled-cycles-backend			
1,794,024,108	instructions	#	0.82 insns per cycle	(100.00%)
294,699,976	branches	#	7.349 M/sec	(100.00%)
9,449,328	branch-misses	#	3.21% of all branches	

10.003560676 seconds time elapsed

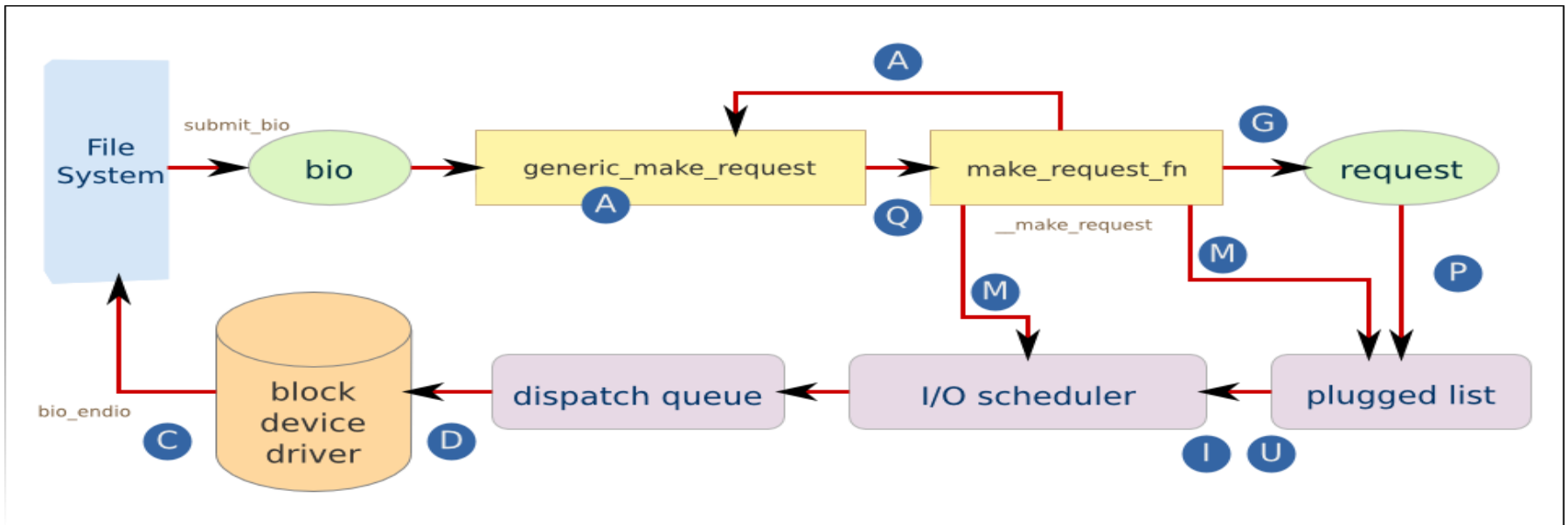
perf usage – perf record & report

perf record -e **block:block_rq_issue**,**block:block_rq_complete** -ag -- sleep 10

D

C

```
root@hslee-15ZD960-GX5BK:~/perf/disk# perf record -e block:block_rq_issue,block:block_rq_complete -ag -- sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.865 MB perf.data (13 samples) ]
root@hslee-15ZD960-GX5BK:~/perf/disk# ls
perf-disk.sh perf.data perf.data.old
root@hslee-15ZD960-GX5BK:~/perf/disk#
```



perf usage – perf record & report

- Perf report를 수행할 때 -s 옵션을 이용하여 sort 된 결과의 레포트를 출력 할수 있다

perf **report** -s pid, dso

Available samples

6 block:block_rq_issue

7 block:block_rq_complete

Samples: 6 of event 'block:block_rq_issue', Event count (approx.): 6					
Children	Self	Command	Shared Object	Symbol	
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	scsi_request_fn
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	__blk_run_queue
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	jbd2_journal_commit_transaction
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	kjournald2
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	kthread
+ 50.00%	50.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	blk_peek_request
+ 50.00%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	ret_from_fork
+ 33.33%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	queue_unplugged
+ 33.33%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	blk_flush_plug_list
+ 33.33%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	blk_finish_plug
+ 16.67%	16.67%	kworker/3:1H	[kernel.kallsyms]	[k]	blk_peek_request
+ 16.67%	16.67%	kworker/u16:1	[kernel.kallsyms]	[k]	blk_peek_request
+ 16.67%	16.67%	swapper	[kernel.kallsyms]	[k]	blk_peek_request
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	blk_queue_bio
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	generic_make_request
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	submit_bio
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	blkdev_issue_discard
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	submit_bh_wbc
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	ext4_free_data_callback
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	submit_bh
+ 16.67%	0.00%	jbd2/sda1-8	[kernel.kallsyms]	[k]	ext4_journal_commit_callback

perf usage – perf record & report

```
Samples: 7 of event 'block:block_rq_complete', Event count (approx.): 7
Children Self Command Shared Object Symbol
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] scsi_end_request
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] scsi_io_completion
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] scsi_finish_command
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] scsi_softirq_done
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] blk_done_softirq
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] __do_softirq
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] irq_exit
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] __irqentry_text_start
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] ret_from_intr
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] cpuidle_enter
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] call_cpuidle
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] cpu_startup_entry
+ 85.71% 85.71% swapper [kernel.kallsyms] [k] blk_update_request
+ 85.71% 0.00% swapper [kernel.kallsyms] [k] start_secondary
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] scsi_end_request
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] scsi_io_completion
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] scsi_finish_command
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] scsi_softirq_done
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] blk_done_softirq
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] __do_softirq
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] irq_exit
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] __irqentry_text_start
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] ret_from_intr
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] __schedule
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] schedule
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] worker_thread
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] kthread
+ 14.29% 14.29% kworker/3:1H [kernel.kallsyms] [k] blk_update_request
+ 14.29% 0.00% kworker/3:1H [kernel.kallsyms] [k] ret_from_fork
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] blk_update_bidi_request
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] __blk_end_bidi_request
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] __blk_end_request_all
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] blk_flush_complete_seq
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] flush_end_io
+ 14.29% 0.00% swapper [kernel.kallsyms] [k] blk_finish_request
```


perf usage – Unit test

- Perf 툴을 이용하여 unit 단위로 정의된 Event 항목들을 단위별로 테스트 할수 있다.

```
root@hslee-15ZD960-GX5BK:/#  
root@hslee-15ZD960-GX5BK:/# perf test list  
1: vmlinux syms matches kallsyms  
2: detect openat syscall event  
3: detect openat syscall event on all cpus  
4: read samples using the mmap interface  
5: parse events tests  
6: Validate PERF_RECORD_* events & perf_sample fields  
7: Test perf pmu format parsing  
8: Test dso data read  
9: Test dso data cache  
10: Test dso data reopen
```

```
root@hslee-15ZD960-GX5BK:/# perf test 3 4 5  
3: detect openat syscall event on all cpus : ok  
4: read samples using the mmap interface : ok  
5: parse events tests : ok  
root@hslee-15ZD960-GX5BK:/# perf test -v 3 4 5  
3: detect openat syscall event on all cpus :  
--- start ---  
test child forked, pid 6721  
test child finished with 0  
---- end ----  
detect openat syscall event on all cpus: Ok  
4: read samples using the mmap interface :  
--- start ---  
test child forked, pid 6722  
mmap size 528384B  
test child finished with 0  
---- end ----  
read samples using the mmap interface: Ok  
5: parse events tests :  
--- start ---
```

perf usage – annotate

- 기록 된 프로파일 정보를 **오브젝트 코드의 실제 기능 및 명령어에 매핑하는** 기능을 제공한다.
- 코드 찾아보기 기능을 사용하여 프로파일 링 정보와 함께 코드 실행을 수행 할 수 있습니다.
- 리눅스 커널의 압축된 **vmlinux** 이미지에 대한 코드 분석이 가능 하다.

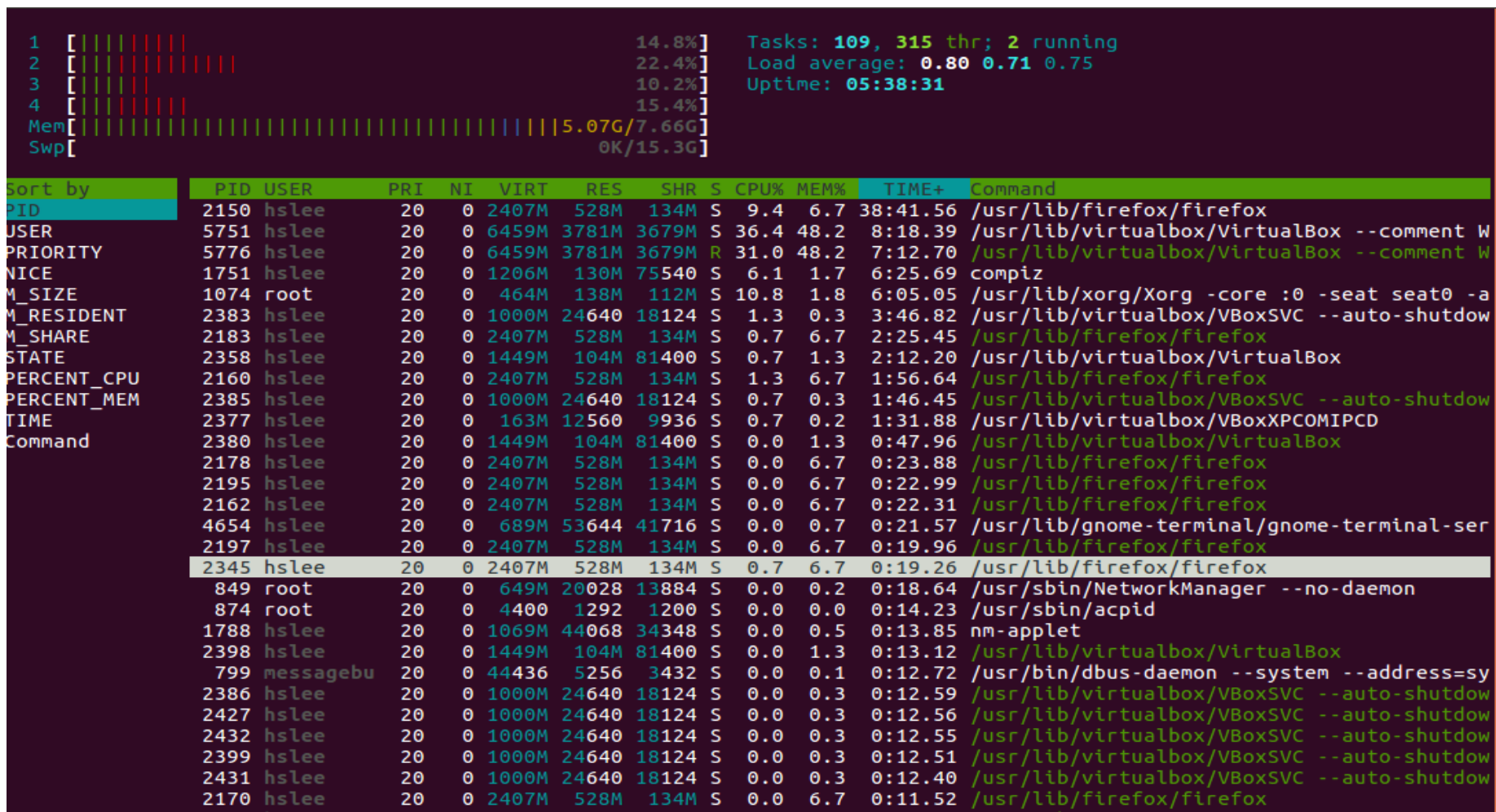
```
perf record ./vmlinux/vmlinux-4.4.0-87-generic
perf annoate -stdio
```

```
Percent | Source code & Disassembly of kcore for cycles:pp
-----|-----
:
:
:
: Disassembly of section load0:
:
: ffffffff81064500 <load0>:
0.00 : ffffffff81064500: push    %rbp
0.00 : ffffffff81064501: mov     %esi,%eax
0.00 : ffffffff81064503: mov     %edi,%ecx
0.00 : ffffffff81064505: mov     %rsp,%rbp
75.00 : ffffffff81064508: wrmsr
25.00 : ffffffff8106450a: xor     %eax,%eax
0.00 : ffffffff8106450c: pop     %rbp
0.00 : ffffffff8106450d: retq
0.00 : ffffffff8106450e: xchg    %ax,%ax
Percent | Source code & Disassembly of kcore for cycles:pp
-----|-----
:
```


기본적인 **perf** 툴을 이용하여 **Profiling** 해봅시다

perf top을 이용한 모니터링

- Pert top과 유사한 기능으로 htop을 쓰는 것도 일반적이지만, 각각의 영역별 (Kernel, User) 오버헤드 분석을 제공하지 않는다.
- Usage에 대해서는 분석이 가능하나, 어느 지점이 병목 지점인가에 대해서는 분석이 어렵다.



perf top을 이용한 모니터링

- Perf top을 이용하여 실시간 모니터링을 통한 각각의 object들에 대한 영역별 overhead 확인 가능

```
Samples: 224 of event 'cycles:pp', Event count (approx.): 31298405057314
Overhead Shared Object Symbol
 8.85% [unknown] [k] 0000000000000000
 8.35% VBoxC.so [.] 0x00000000000083287
 5.41% libdrm_intel.so.1.0.0 [.] 0x000000000000656b
 3.70% libxul.so [.] 0x00000000024ad9c8
 3.07% libxcb.so.1.1.0 [.] pthread_mutex_unlock@plt
 2.69% libxul.so [.] 0x00000000011e0952
 2.41% libunityshell.so [.] unity::UnityScreen::compizDamageNux
 2.35% i965_dri.so [.] 0x00000000001a34e0
 2.35% [kernel] [k] unix_poll
 2.35% [kernel] [k] mutex_spin_on_owner.isra.3
 2.35% libxul.so [.] 0x0000000001bc382a
 2.35% [kernel] [k] rcu_irq_exit
 2.06% libxul.so [.] 0x000000000097ec0a
 2.06% libxul.so [.] 0x0000000001856886
 2.03% [kernel] [k] menu_select
 1.85% libxul.so [.] 0x00000000009c093c
 1.80% libxul.so [.] 0x0000000001cccad6
 1.80% libxul.so [.] 0x0000000001270bda
 1.58% libnux-graphics-4.0.so.0.8.0 [.] nux::GetGraphicsDisplay
 1.49% [kernel] [k] ioread32
 1.42% libxul.so [.] 0x0000000002b3e88d
 1.38% [kernel] [k] __fget
 1.38% [kernel] [k] select_task_rq_fair
 1.38% Xorg [.] xf86Wakeup
 1.38% [kernel] [k] __pollwait
 1.30% [kernel] [k] timerqueue_add
 1.28% [kernel] [k] update_blocked_averages
 1.22% [kernel] [k] __switch_to_xtra
 1.21% [kernel] [k] pick_next_task_fair
 1.21% libglib-2.0.so.0.4800.2 [.] g_source_add_child_source
 1.11% VBoxXPCOMIPCC.so [.] 0x00000000000148bb
 1.06% [unknown] [.] 0x00007f3e519f5820
 1.06% i965_dri.so [.] 0x00000000000141e34
 1.06% i965_dri.so [.] 0x000000000001615c7
```

어느 누군가의 성능에 대한 불만



UEK 커널을 설치한 시스템과 **Vanila** 커널을 설치한 시스템을
비교 했을때 **UEK** 커널이 **CPU** 사용량이 **더 좋았다**, 어떠한 차이점이
있는지 분석해 달라.

황당하지만 재미삼아 접근해본다.

- CPU의 사용량이 더 좋았다?

- CPU와 관련된 Trace point의 수집을 통합 접근

- CPU의 사용률이 높은 process?

- Perf record -F 99 -p PID -g - sleep 10

- CPU의 사용률이 높은 Core?

- CPU Event와 관련된 Profiling Action

CPU Event Profiling

- `perf record -F 99 -ag --sleep 30`
- `perf record -F 99 -p 3345 -g --sleep 30`
- `perf record -e cycles:u -ag -- sleep 30`
- `perf record -e cycles:k -ag -- sleep 30`

참고 사이트

- <https://perf.wiki.kernel.org/index.php/Tutorial>
- <http://www.brendangregg.com/perf.html>
- http://www.oss.kr/oss_repository28/667575
- <https://github.com/brendangregg/perf-tools>

