“ 라즈베리파이 , 서버 , IoT 성능 어디까지 쥐어짜봤니 ? ”

Linux Kernel – perf

:a profiler tool with performance events

NAVER DEVIEW 2016

Taeung Song
미래부 KOSS LAB. – Software Engineer
taeung@kosslab.kr

2016-10-25

Speaker

DEVIEW 2016

송태웅 (Taeung Song, https://github.com/taeung)

- 미래창조과학부 KOSS Lab. Software Engineer
- Linux Kernel Contributor 활동 중 (perf)

강의활동

- SK C&C Git/Github 사내 교육영상 제작
- 서강대 , 아주대 , OSS 포럼 등 Git/Github 강의
- 국민대 , 이화여대 등 Linux perf, Opensource 참여 관련 시간강사 활동

Points

1. 리눅스의 커널과 관련된 성능문제 어떻게 Troubleshooting ?

2. IoT, Embedded 환경 C/C++ program 성능개선작전

Contents

1. 리눅스 성능분석도구 perf 소개
2. 커널관련 성능문제 Troubleshooting (Cassandra 와 Kernel version)
3. perf 의 Events Sampling 은 어떻게 동작하는가 ?
4.  Raspberry pi 환경 C/C++ program 성능개선작전 (ioT, Embedded)

리눅스 성능분석도구 perf 란 ?

리눅스 성능분석도구 perf 란 ?

특정 프로그램 / 시스템 전반

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

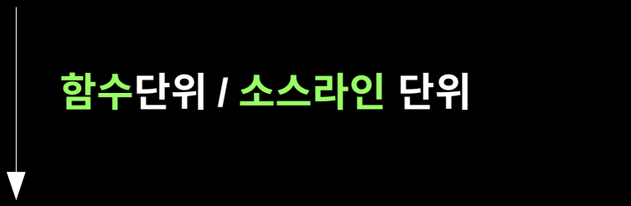
특정 프로그램 / 시스템 전반

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

특정 프로그램 / 시스템 전반 with Events Sampling

리눅스 성능분석도구 perf 란 ?



특정 프로그램 / 시스템 전반 with Events Sampling

성능 측정가능한 초점 Focus

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

특정 프로그램 / 시스템 전반 with Events Sampling

성능 측정가능한 초점 Focus (CPU cycles,

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

특정 프로그램 / 시스템 전반 with Events Sampling

성능 측정가능한 초점 Focus (CPU cycles, cache-misses

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

특정 프로그램 / 시스템 전반 with Events Sampling

성능 측정가능한 초점 Focus (CPU cycles, cache-misses
, page-fault,

리눅스 성능분석도구 perf 란 ?

함수단위 / 소스라인 단위

특정 프로그램 / 시스템 전반 with Events Sampling

성능 측정가능한 초점 Focus (CPU cycles, cache-misses
, page-fault, system calls, etc.)

기존에 많이 사용하는
성능 모니터링 도구들

타 성능측정도구

- top

```
top - 11:40:30 up 23:57,  5 users,  load average: 1.40, 0.90, 0.63
Tasks: 382 total,   3 running, 379 sleeping,   0 stopped,   0 zombie
Cpu(s): 53.7%us,  2.7%sy,  0.0%ni, 43.4%id,  0.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1024212k total,   986976k used,    37236k free,    16476k buffers
Swap:  4088500k total,   313288k used,  3775212k free,   207380k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7087 rameshj   20   0  231m 9840 5772 R   94  1.0   1:31.81 java
 2304 root      20   0  427m  89m 6548 R    9  8.9  15:27.31 X
 5701 rameshj   20   0  179m  10m 6548 S    2  1.0  24:37.23 knotify4
 5855 rameshj   20   0  691m 275m  20m S    1 27.5 113:37.47 firefox-bin
 5767 rameshj    9 -11  156m 3452 2936 S    1  0.3  68:22.04 pulseaudio
 6089 rameshj   20   0  347m  20m 7400 S    1  2.0  81:30.14 plugin-containe
 1570 rameshj   20   0 73764  10m 7168 S    0  1.1   0:08.28 konsole
 1859 rameshj   20   0  579m 198m 2092 S    0 19.9   2:55.70 java
 5702 rameshj   20   0  255m  13m 8024 S    0  1.3   0:26.34 plasma
 5755 rameshj    9 -11  156m 3452 2936 S    0  0.3  33:29.66 pulseaudio
 5887 rameshj   20   0  691m 275m  20m S    0 27.5   8:38.24 firefox-bin
 5906 rameshj   20   0  691m 275m  20m S    0 27.5   0:54.29 firefox-bin
```

타 성능측정도구

- iperf

```
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-30.00  sec   794 MBytes   222 Mbits/sec                  sender
[  4]   0.00-30.00  sec   794 MBytes   222 Mbits/sec                  receiver
[  6]   0.00-30.00  sec   795 MBytes   222 Mbits/sec                  sender
[  6]   0.00-30.00  sec   795 MBytes   222 Mbits/sec                  receiver
[  8]   0.00-30.00  sec   786 MBytes   220 Mbits/sec                  sender
[  8]   0.00-30.00  sec   786 MBytes   220 Mbits/sec                  receiver
[ 10]   0.00-30.00  sec   795 MBytes   222 Mbits/sec                  sender
[ 10]   0.00-30.00  sec   795 MBytes   222 Mbits/sec                  receiver
[ 12]   0.00-30.00  sec   772 MBytes   216 Mbits/sec                  sender
[ 12]   0.00-30.00  sec   771 MBytes   216 Mbits/sec                  receiver
[ 14]   0.00-30.00  sec   754 MBytes   211 Mbits/sec                  sender
[ 14]   0.00-30.00  sec   754 MBytes   211 Mbits/sec                  receiver
[ 16]   0.00-30.00  sec   756 MBytes   211 Mbits/sec                  sender
[ 16]   0.00-30.00  sec   756 MBytes   211 Mbits/sec                  receiver
[ 18]   0.00-30.00  sec   758 MBytes   212 Mbits/sec                  sender
[ 18]   0.00-30.00  sec   758 MBytes   212 Mbits/sec                  receiver
[ 20]   0.00-30.00  sec   782 MBytes   219 Mbits/sec                  sender
[ 20]   0.00-30.00  sec   781 MBytes   219 Mbits/sec                  receiver
[ 22]   0.00-30.00  sec   765 MBytes   214 Mbits/sec                  sender
[ 22]   0.00-30.00  sec   765 MBytes   214 Mbits/sec                  receiver
[SUM]   0.00-30.00  sec  7.57 GBytes  2.17 Gbits/sec                  sender
[SUM]   0.00-30.00  sec  7.57 GBytes  2.17 Gbits/sec                  receiver

iperf Done.
```

타 성능측정도구

- iotop

```
Total DISK READ: 0.00 B/s | Total DISK WRITE: 120.50 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
  286 be/3 root        0.00 B/s   38.87 K/s  0.00 %  0.10 % [jbd2/dm-0-8]
  943 be/4 root        0.00 B/s    3.89 K/s  0.00 %  0.00 % rsyslogd -i /var/run/syslogd.pid -c 5
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
 1027 be/4 adamowen    0.00 B/s    0.00 B/s  0.00 %  0.00 % transmission-daemon -g /home/adamowen1/.config/transmission
    4 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
    5 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
    6 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [watchdog/0]
    7 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [events/0]
    8 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [cgroup]
    9 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khelper]
   10 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [netns]
   11 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [async/mgr]
   12 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [pm]
   13 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [sync_supers]
   14 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [bdi-default]
   15 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kintegrityd/0]
   16 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kblockd/0]
   17 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kacpid]
   18 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kacpi_notify]
    3 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
   20 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ata_aux]
   21 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ata_sff/0]
   22 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksuspend_usbd]
   23 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khubd]
   24 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kseriod]
   25 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [md/0]
   26 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [md_misc/0]
   27 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [linkwatch]
   28 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khungtaskd]
   29 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kswapd0]
   30 be/5 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksmd]
   31 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [aio/0]
   32 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [crypto/0]
 1569 be/4 nobody      0.00 B/s    0.00 B/s  0.00 %  0.00 % bprobe --tls-key /etc/bprobe/key~run/bprobe.pid --syslog bprobe -G
 1570 be/4 nobody      0.00 B/s    0.00 B/s  0.00 %  0.00 % bprobe --tls-key /etc/bprobe/key~run/bprobe.pid --syslog bprobe -G
```

개발자 입장에서 top, iperf, iotop 의 한계

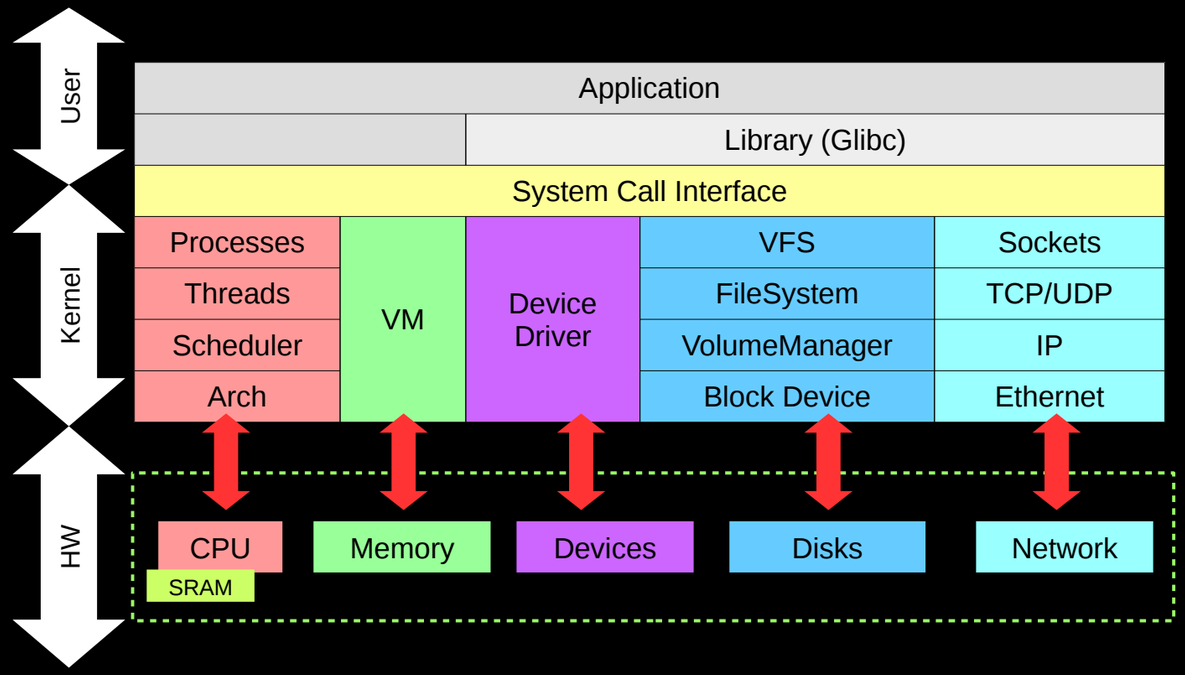
- 결과적인 CPU 점유율만 표시
- 시간당 데이터처리량만 표시
- I/O 발생 정도만 확인 가능
- 어떤 소스라인 / 함수 가 병목지점인지 알수 없음
- Disk/Network I/O 가 심하다면 상세한 진단 불가능

그런데 **perf** 에서는 **각종 event** 중에서

System Layer 상에서 원하는 **성능분석 지점** (Focus) 을 **선택**가능

- CPU cycles 중심 성능분석
- Block Device 레벨 성능분석
- File System(ext4) 레벨 성능분석
- Socket 레벨 성능분석
- Ethernet (NIC) 레벨 성능분석
- ...

System Layer 상에서 원하는 성능분석 지점 (Focus) 을 선택가능

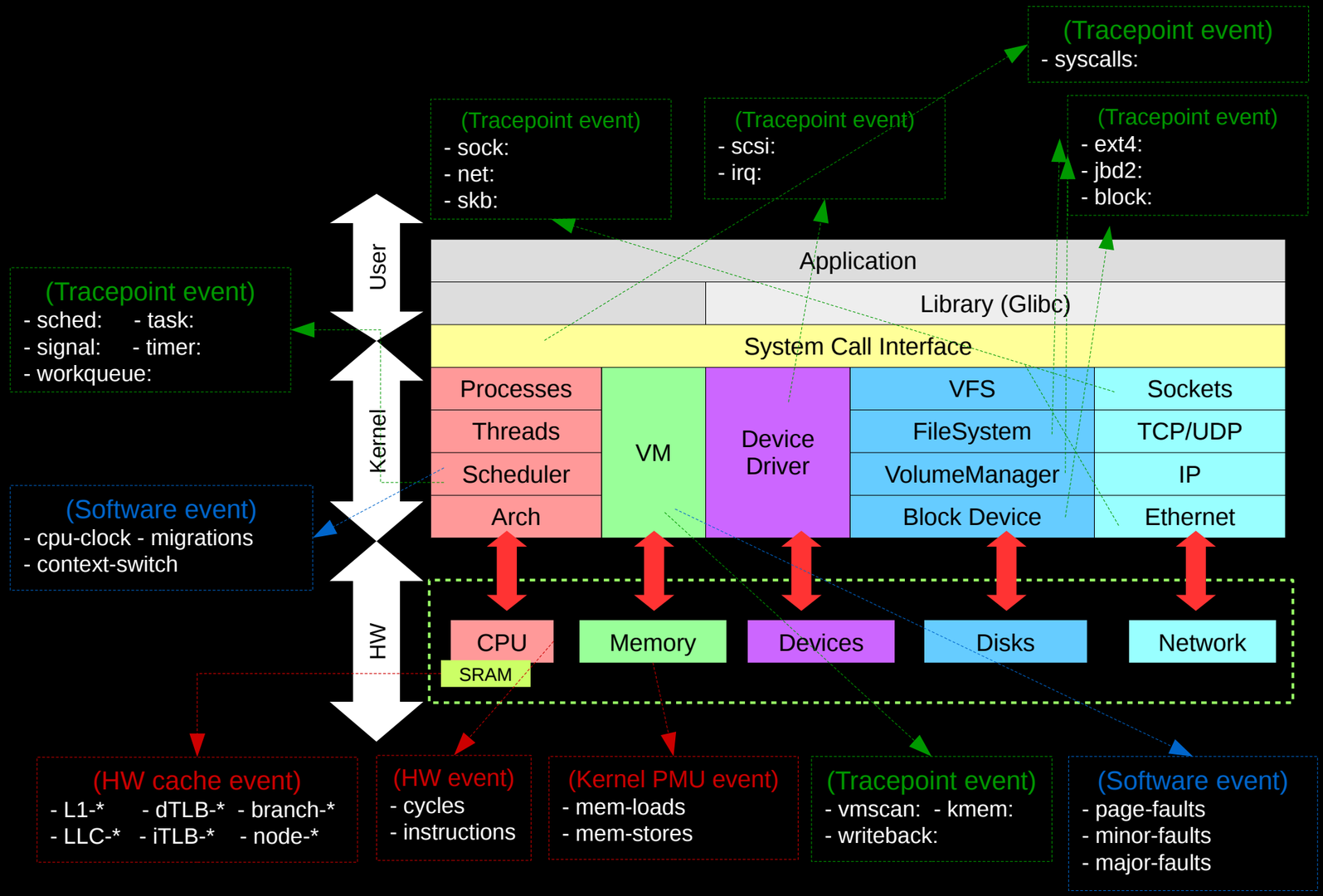


< Linux kernel 의 주요 5 가지 subsystem 기준 >

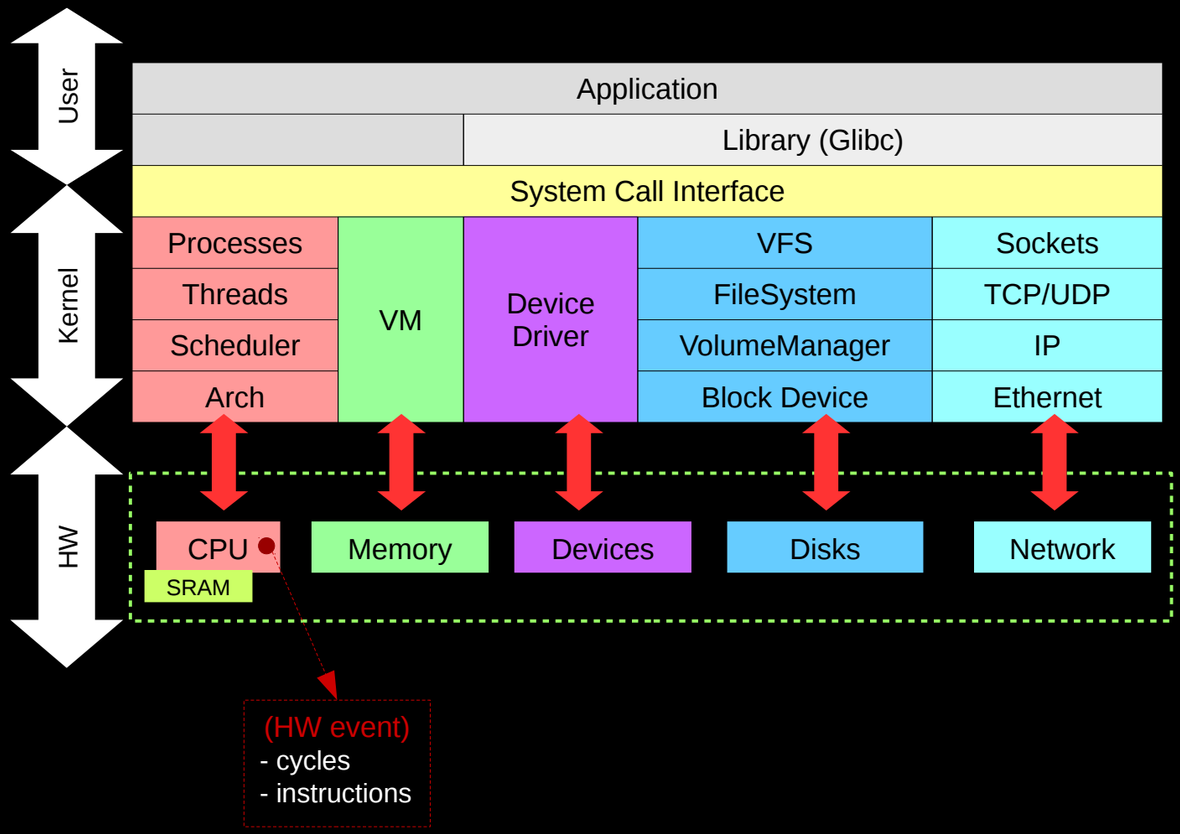
1) **프로세스 관리** (Process management)
2) **메모리 관리** (Memory management)
3) **디바이스 드라이버** (Device Driver)
4) **파일 시스템** (File System)
5) **네트워킹** (Networking)

http://www.makelinux.net/kernel_map/
http://www.brendangregg.com/perf.html

System Layer 상에서 원하는 성능분석 지점 (Focus) 을 선택가능



System Layer 상에서 원하는 성능분석 지점 (Focus) 을 선택가능



따라서 perf 는 단순성능측정 도구보다

한층 더 상세한 성능 분석이 가능하다

perf 의 사용목적 ?

- Profiling
- Tracing

perf 의 사용목적 ?

- Profiling : 병목지점 (Bottlenecks) 을 찾아내기 위해

perf 의 사용목적 ?

- **Profiling :** **병목지점** (Bottlenecks) **을 찾아내기 위해**

```
...
21 int foo ( int a, int b)
22 {
23  ...
24
25
26 }
27
28
29
30
31
32
33
...
```

소스코드 program.c

perf 의 사용목적 ?

- **Profiling :** 병목지점 (Bottlenecks) 을 찾아내기 위해

  ↓

  - 프로그램 전체에서 어떤함수가 CPU 를 많이 사용하는지 ?

```
...
21 int foo ( int a, int b)
22 {
23  ...
24
25
26 }
27
28
29
30
31
32
33
...
```

소스코드 program.c

perf 의 사용목적 ?

- **Profiling :** **병목지점** (Bottlenecks) **을 찾아내기 위해**

  - 프로그램 전체에서 **어떤함수**가 **CPU** 를 많이 **사용**하는지 ?
  - **소스코드**에서 어떤 **라인**이 **CPU** 를 많이 **사용**하는지 ?

```
...
21 int foo ( int a, int b)
22 {
23  ...
24
25
26 }
27
28
29
30
31
32
33
...
```

소스코드 program.c

perf 의 사용목적 ?

- Profiling : 병목지점 (Bottlenecks) 을 찾아내기 위해

HW / SW Events

  - 프로그램 전체에서 어떤함수가 CPU 를 많이 사용하는지 ?
  - 소스코드에서 어떤 라인이 CPU 를 많이 사용하는지 ?

```
...
21 int foo ( int a, int b)
22 {
23  ...
24
25
26 }
27
28
29
30
31
32
33
...
```

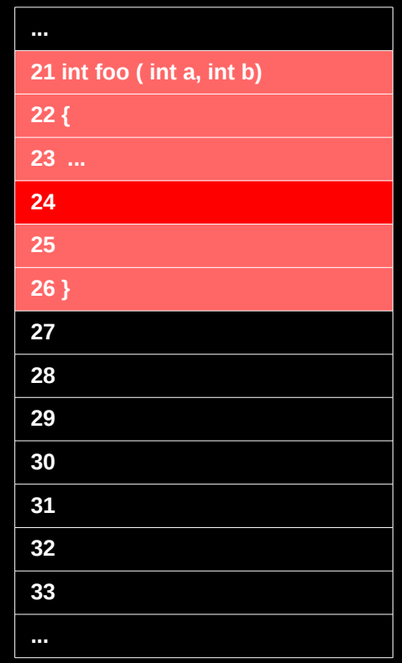
소스코드 program.c

perf 의 사용목적 ?

- **Profiling :** 병목지점 (Bottlenecks) 을 찾아내기 위해

  ↓ HW / SW Events (CPU cycles, cache-misses ...)

  - 프로그램 전체에서 어떤함수가 CPU 를 많이 사용하는지 ?
  - 소스코드에서 어떤 라인이 CPU 를 많이 사용하는지 ?



소스코드 program.c

perf 의 사용목적 ?

- **Tracing** : 특정 Event 발생에 대한 경위 ,
  과정 (function call graph) 을 살펴보기 위해

perf 의 사용목적 ?

- **Tracing** : 특정 Event 발생에 대한 경위 ,
  과정 (function call graph) 을 수색하기 위해

  - 특정 커널 함수가 왜 불려 졌을까 ?

perf 의 사용목적 ?

- **Tracing** : 특정 Event 발생에 대한 경위 ,
  과정 (function call graph) 을 **수색**하기 위해

  - 특정 **커널 함수**가 왜 불려 졌을까 ?
  - 그 커널함수가 **호출된** (mapping 된 event 가 **발생된**) **과정** (call graph) 어땠을까 ?

perf 의 사용목적 ?

- **Tracing** : 특정 Event 발생에 대한 경위 ,
  과정 (function call graph) 을 **수색**하기 위해

Tracepoint / Probe Events

- 특정 **커널 함수**가 왜 불려 졌을까 ?
- 그 커널함수가 **호출된** (mapping 된 event 가 **발생된**) **과정** (call graph) 어땠을까 ?

perf 의 사용목적 ?

- **Tracing** : 특정 Event 발생에 대한 경위 ,
  과정 (function call graph) 을 **수색**하기 위해

  ↓ Tracepoint / Probe Events (여러 커널함수 , system calls, page fault ...)

  - 특정 **커널 함수**가 왜 불려 졌을까 ?
  - 그 커널함수가 **호출된** (mapping 된 event 가 **발생된**) **과정** (call graph) 어땠을까 ?

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

Tracepoint Events

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
...
-    0.36%     0.36%  8,0 R 0 () 483479880 + 56 [chrome]

     page_fault
     do_page_fault
     __do_page_fault
     handle_mm_fault
     __do_fault
     filemap_fault
     __do_page_cache_readahead
     blk_finish_plug
     blk_flush_plug_list
     __elv_add_request

+    0.36%     0.36%  8,0 R 0 () 483479960 + 8 [chrome]
...
```

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
...
-    0.36%     0.36%  8,0 R 0 () 483479880 + 56 [chrome]

     page_fault
     do_page_fault
     __do_page_fault
     handle_mm_fault
     __do_fault
     filemap_fault
     __do_page_cache_readahead
     blk_finish_plug
     blk_flush_plug_list
     __elv_add_request

+    0.36%     0.36%  8,0 R 0 () 483479960 + 8 [chrome]
...
```

chrome 이 block_rq_insert 이벤트를 발생시킴 (block I/O 요청)
(== 커널함수 __elv_add_request 호출함 Read 목적으로)

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
...
-    0.36%     0.36%  8,0 R 0 () 483479880 + 56 [chrome]

     page_fault
     do_page_fault
     __do_page_fault
     handle_mm_fault
     __do_fault
     filemap_fault
     __do_page_cache_readahead
     blk_finish_plug
     blk_flush_plug_list
     __elv_add_request

+    0.36%     0.36%  8,0 R 0 () 483479960 + 8 [chrome]
...
```

Tracepoint Events

chrome 이 block_rq_insert 이벤트를 발생시킴 (block I/O 요청)
(== 커널함수 __elv_add_request 호출함 Read 목적으로)

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
...
-    0.36%     0.36%  8,0 R 0 () 483479880 + 56 [chrome]

     page_fault
     do_page_fault
     __do_page_fault
     handle_mm_fault
     __do_fault
     filemap_fault
     __do_page_cache_readahead
     blk_finish_plug
     blk_flush_plug_list
     __elv_add_request

+    0.36%     0.36%  8,0 R 0 () 483479960 + 8 [chrome]
...
```

왜 __elv_add_request 가 호출이 되었나 ?

경위를 찾아 거슬러 올라가보면 ..

Chrome 에서 파일 업로드 하는 동안

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
...
-    0.36%     0.36%  8,0 R 0 () 483479880 + 56 [chrome]

     page_fault
     do_page_fault
     __do_page_fault
     handle_mm_fault
     __do_fault
     filemap_fault
     __do_page_cache_readahead
     blk_finish_plug
     blk_flush_plug_list
     __elv_add_request

+    0.36%     0.36%  8,0 R 0 () 483479960 + 8 [chrome]
...
```

chrome 이 Block I/O 를 요청 (block_rq_insert 이벤트 발생시킨) 한 이유

: page fault 가 발생 했기 때문에 실제 Read 를 요청 했다 .

커널관련 성능문제 Troubleshooting

소스는 그대로 커널은 버전업 , 성능문제가 생겼다 ?

Netflix 의 Cassandra DB 와 커널버전

성능문제 Troubleshooting

문제상황 : 커널버전 업그레이드 이후

DISK I/O 가 많아짐 , iowait 도 높아짐

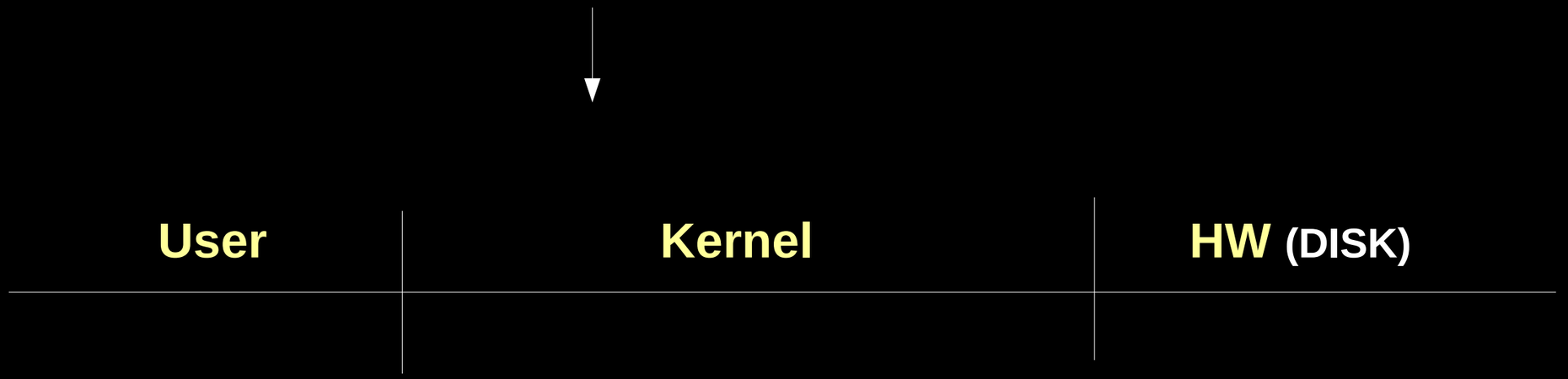
문제상황 : 커널버전 업그레이드 이후

DISK I/O 가 많아짐 , iowait 도 높아짐

어디부터 봐야할까 ?

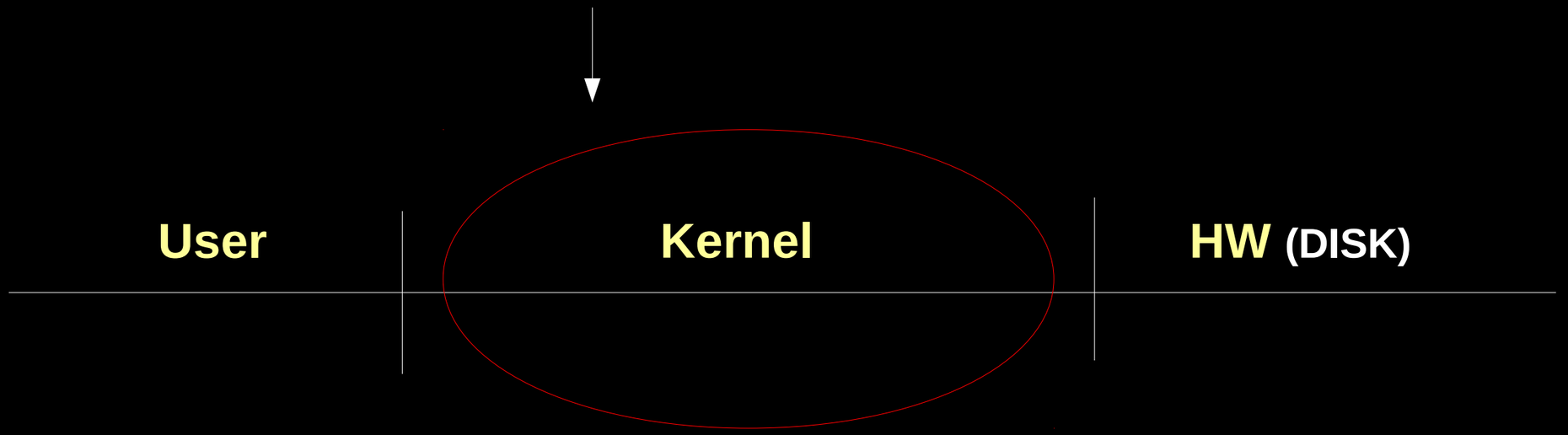
문제상황 : 커널버전 업그레이드 이후

DISK I/O 가 많아짐 , iowait 도 높아짐



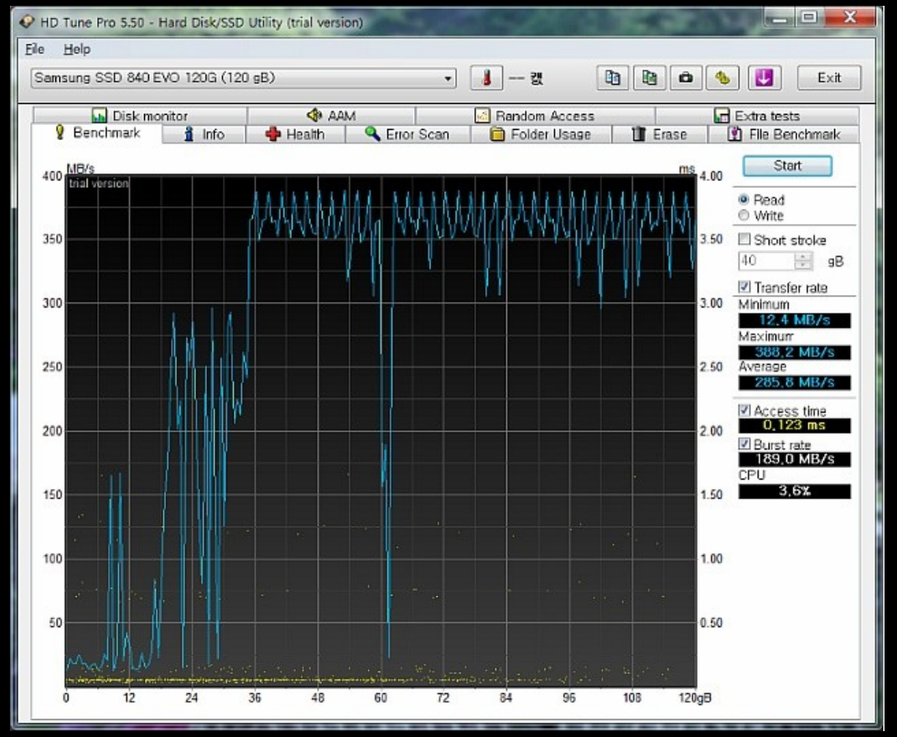
문제상황 : 커널버전 업그레이드 이후

DISK I/O 가 많아짐 , iowait 도 높아짐



DISK(HW) 문제 가능성 : SSD 펌웨어 관련

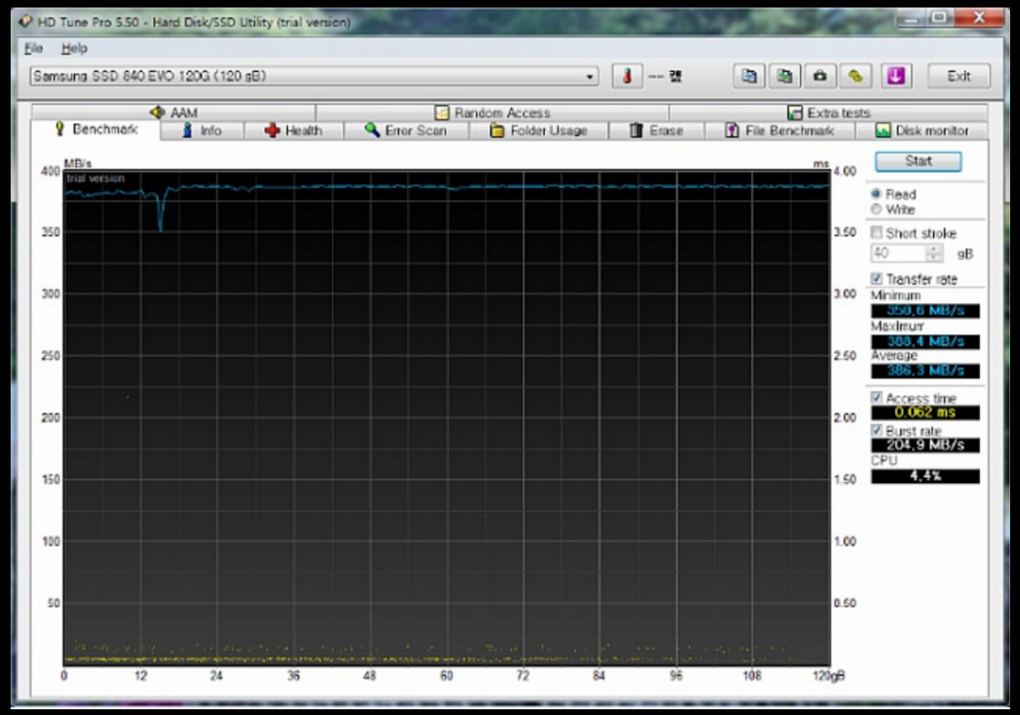
S 사 SSD 840 EVO 읽기성능 이슈 (펌웨어 업데이트 전)



http://dsct1472.tistory.com/405

DISK(HW) 문제 가능성 : SSD 펌웨어 관련

S 사 SSD 840 EVO 읽기성능 이슈 (펌웨어 업데이트 후)



http://dsct1472.tistory.com/405

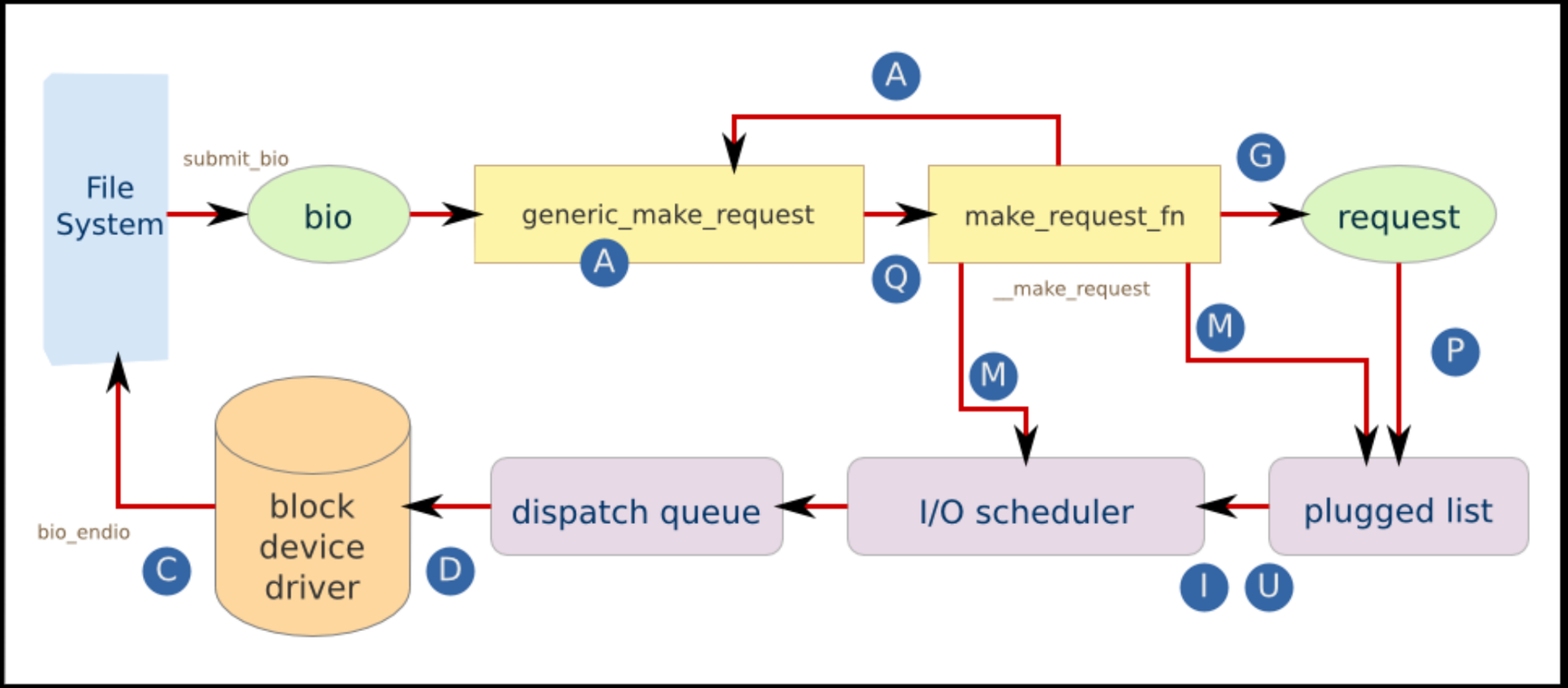
DISK(HW) 문제 가능성 : Smartctl 등을 활용한 간단한 확인

```
smartctl -a /dev/sda | grep SATA
SATA Version is:  SATA 3.1, 6.0 Gb/s (current: 6.0 Gb/s)
```

DISK(HW) 문제 가능성 : Smartctl 등을 활용한 간단한 확인

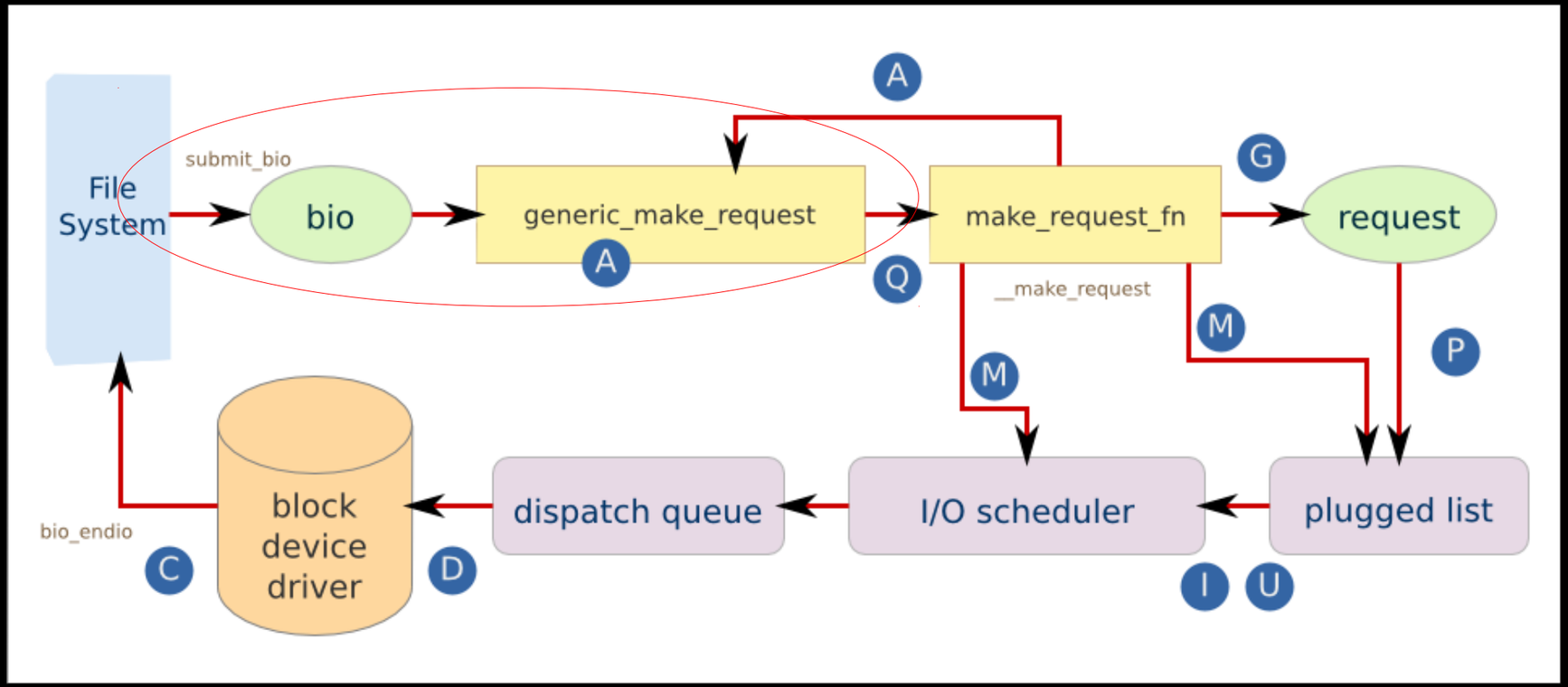
```
smartctl -a /dev/sda | grep SATA
SATA Version is:  SATA 3.1, 6.0 Gb/s (current: 6.0 Gb/s)

6.0 Gb -> 750MB

 :> dd if=/dev/zero of=testfile bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 1.41711 s, 758 MB/s
```

Kernel 영역에서 어디를 봐야할까 ?



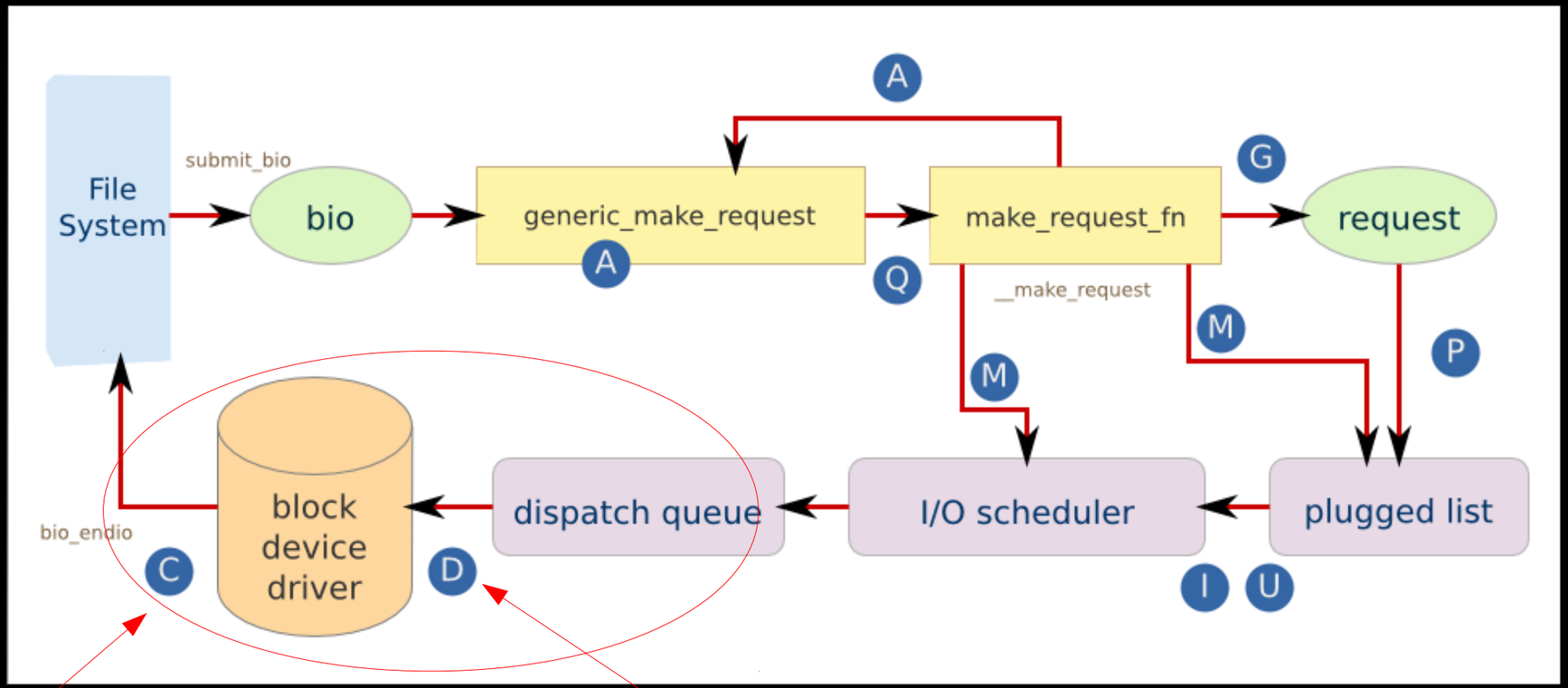
http://studyfoss.egloos.com/5585801

Tracepoint Events 지점별로 나눠서 지연시간 확인



http://studyfoss.egloos.com/5585801

Tracepoint Events 지점별로 나눠서 지연시간 확인



http://studyfoss.egloos.com/5585801

block:block_rq_complete

block:block_rq_issue

block:block_rq_insert

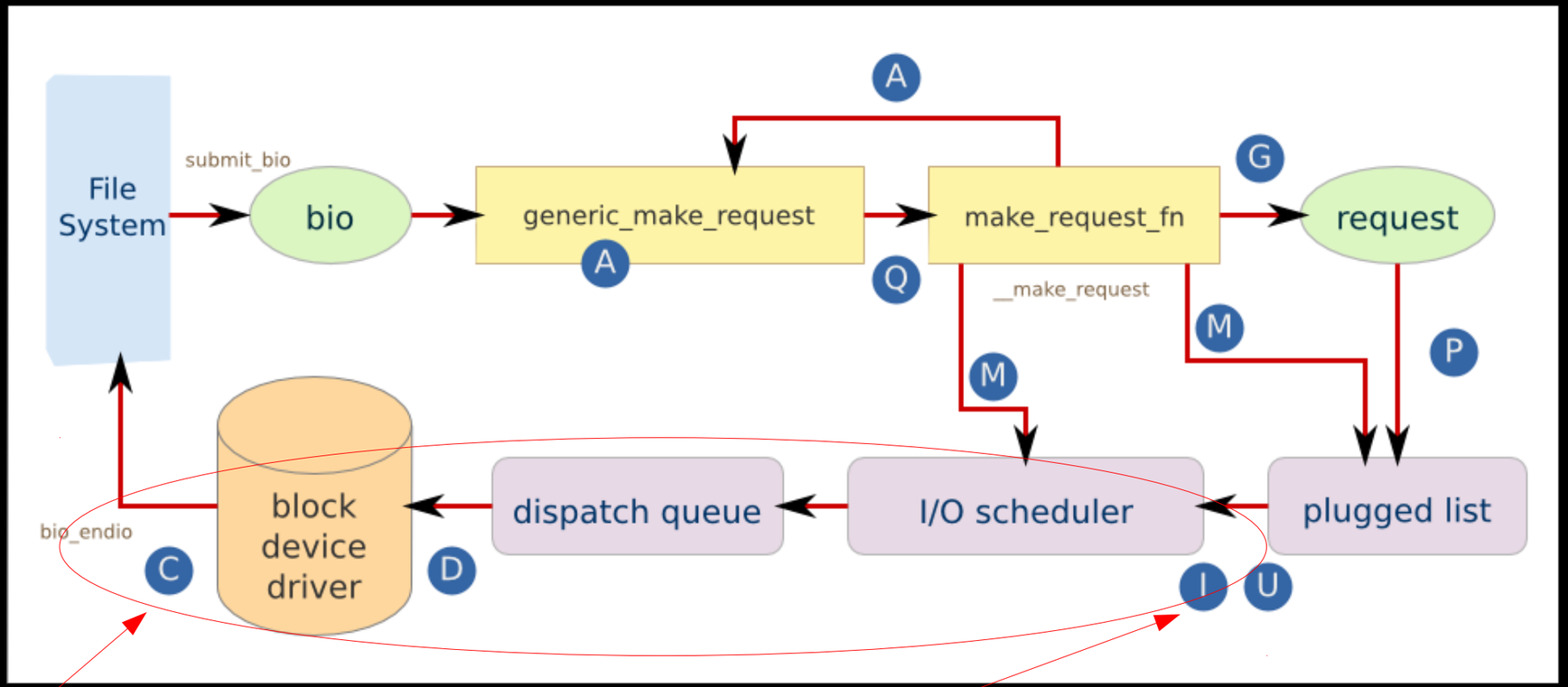
block_rq_issue 에서 block_rq_complete 까지

```
# cat /sys/kernel/debug/tracing/events/block/block_rq_complete/format
name: block_rq_complete
ID: 931
format:
        field:unsigned short common_type;    offset:0;size:2;  signed:0;
        field:unsigned char common_flags;    offset:2;size:1;  signed:0;
        field:unsigned char common_preempt_count;     offset:3;size:1;  signed:0;
        field:int common_pid;       offset:4;size:4;  signed:1;

        field:dev_t dev;   offset:8;size:4;  signed:0;
        field:sector_t sector;      offset:16;        size:8;  signed:0;
...

# cat /sys/kernel/debug/tracing/events/block/block_rq_issue/format
name: block_rq_issue
ID: 937
format:
        field:unsigned short common_type;    offset:0;size:2;  signed:0;
        field:unsigned char common_flags;    offset:2;size:1;  signed:0;
        field:unsigned char common_preempt_count;     offset:3;size:1;  signed:0;
        field:int common_pid;       offset:4;size:4;  signed:1;

        field:dev_t dev;   offset:8;size:4;  signed:0;
        field:sector_t sector;      offset:16;        size:8;  signed:0;
...
```

block_rq_issue 에서 block_rq_complete 까지

```
# perf record -e block:block_rq_issue,block:block_rq_complete -p <pid>
...
# python perf-script.py
dev=202, sector=32, nr_sector=1431244248, rwbs=R, comm=java, lat=0.58
dev=202, sector=32, nr_sector=1431244336, rwbs=R, comm=java, lat=0.58
dev=202, sector=32, nr_sector=1431244424, rwbs=R, comm=java, lat=0.59
dev=202, sector=32, nr_sector=1431244512, rwbs=R, comm=java, lat=0.59
...
```

Tracepoint Events 지점별로 나눠서 지연시간 확인



block:block_rq_complete

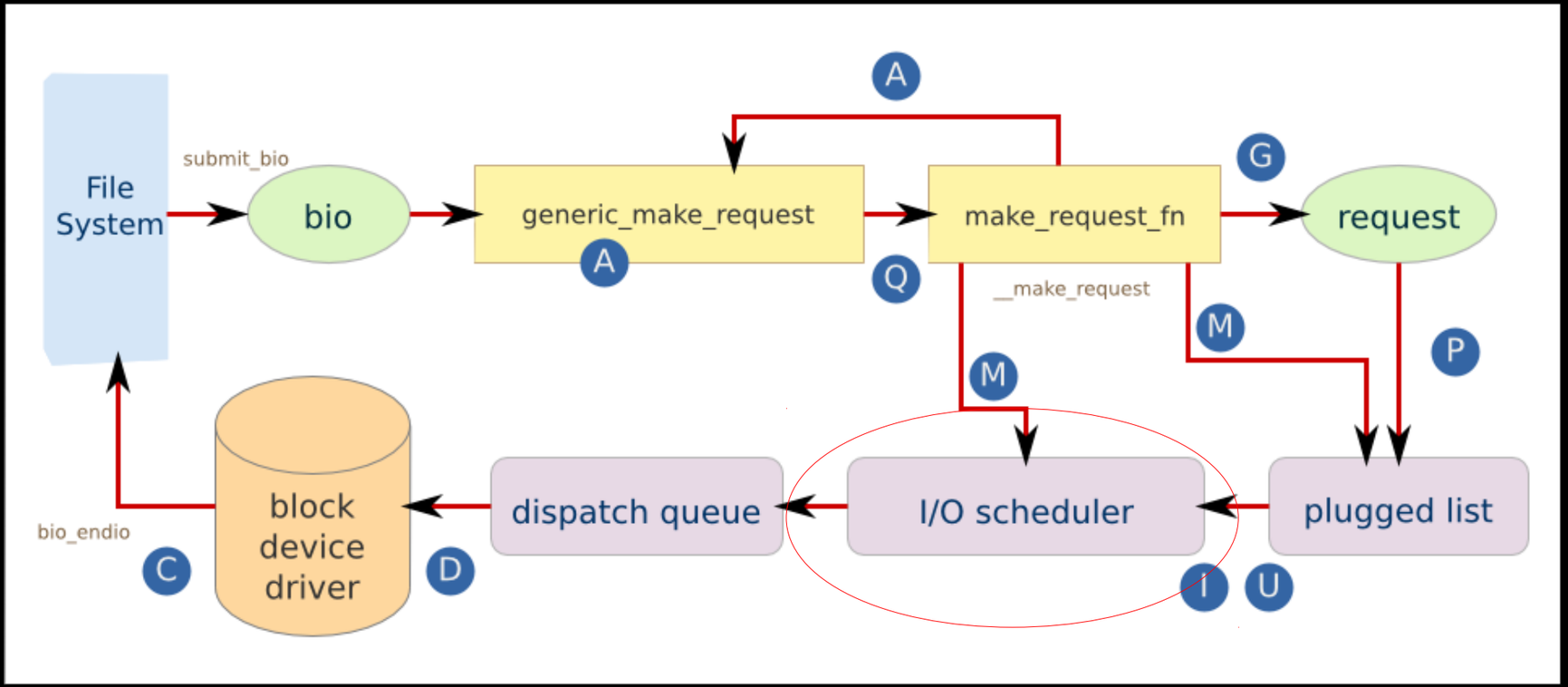
block:block_rq_insert

http://studyfoss.egloos.com/5585801

block_rq_insert 에서 block_rq_complete 까지

```
# perf record -e block:block_rq_issue,block:block_rq_complete -p <pid>
...
# python perf-script.py
dev=202, sector=32, nr_sector=1431244248, rwbs=R, comm=java, lat=0.58
dev=202, sector=32, nr_sector=1431244336, rwbs=R, comm=java, lat=0.58
dev=202, sector=32, nr_sector=1431244424, rwbs=R, comm=java, lat=0.59
dev=202, sector=32, nr_sector=1431244512, rwbs=R, comm=java, lat=0.59
...

# perf record -e block:block_rq_insert,block:block_rq_complete -p <pid>
...
# python perf-script.py
dev=202, sector=32, nr_sector=1596381840, rwbs=R, comm=java, lat=3.85
dev=202, sector=32, nr_sector=1596381928, rwbs=R, comm=java, lat=3.87
dev=202, sector=32, nr_sector=1596382016, rwbs=R, comm=java, lat=3.88
dev=202, sector=32, nr_sector=1596382104, rwbs=R, comm=java, lat=3.89
...
```

왜 queue 에서 지연이 될까 ?



http://studyfoss.egloos.com/5585801

deadline 에서 noop 으로 변경 실험

```
# cat /sys/block/sda/queue/scheduler
noop [deadline] cfq

# echo deadline > /sys/block/sda/queue/scheduler

# cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

누가 block_rq_insert 을 발생시키나 ? (많이 호출하나 ?)

호출된 과정을 거슬러 올라가며 수색해보자 ..

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-    13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
        page_fault
        do_page_fault
        handle_mm_fault
        handle_pte_fault
        __do_fault
        filemap_fault
        do_sync_mmap_readahead.isra.24
        ra_submit
        __do_page_cache_readahead
        read_pages
        xfs_vm_readpages
        mpage_readpages
        do_mpage_readpage
        submit_bio
        generic_make_request
        generic_make_request.part.50
        blk_queue_bio
        blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-    13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
        page_fault
        do_page_fault
        handle_mm_fault
        handle_pte_fault
        __do_fault
        filemap_fault
        do_sync_mmap_readahead.isra.24
        ra_submit
        __do_page_cache_readahead
        read_pages
        xfs_vm_readpages
        mpage_readpages
        do_mpage_readpage
        submit_bio
        generic_make_request
        generic_make_request.part.50
        blk_queue_bio
        blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-   13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
      page_fault
      do_page_fault
      handle_mm_fault
      handle_pte_fault
      __do_fault
      filemap_fault
      do_sync_mmap_readahead.isra.24
      ra_submit
      __do_page_cache_readahead
      read_pages
      xfs_vm_readpages
      mpage_readpages
      do_mpage_readpage
      submit_bio
      generic_make_request
      generic_make_request.part.50
      blk_queue_bio
      blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-    13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
       page_fault
       do_page_fault
       handle_mm_fault
       handle_pte_fault
       __do_fault
       filemap_fault
       do_sync_mmap_readahead.isra.24
       ra_submit
       __do_page_cache_readahead
       read_pages
       xfs_vm_readpages
       mpage_readpages
       do_mpage_readpage
       submit_bio
       generic_make_request
       generic_make_request.part.50
       blk_queue_bio
       blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-    13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
       page_fault
       do_page_fault
       handle_mm_fault
       handle_pte_fault
       __do_fault
       filemap_fault
       do_sync_mmap_readahead.isra.24
       ra_submit
       __do_page_cache_readahead
       read_pages
       xfs_vm_readpages
       mpage_readpages
       do_mpage_readpage
       submit_bio
       generic_make_request
       generic_make_request.part.50
       blk_queue_bio
       blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

cassandra 가 block_rq_insert 이벤트를 발생 (block I/O 요청) 시키는

이유가 page fault, readahead 등과 관련된거 였다 ..

Cassandra 동작중에 ..

block_rq_insert 이벤트가 발생되기까지의 과정 (call graph) 수색하면 ?

```
# perf record -g -e block:block_rq_insert -p <pid> && perf report
...
-    13.41%     13.41%  202,16 R 0 () 1431480000 + 8 [java]
       page_fault
       do_page_fault
       handle_mm_fault
       handle_pte_fault
       __do_fault
       filemap_fault
       do_sync_mmap_readahead.isra.24
       ra_submit
       __do_page_cache_readahead
       read_pages
       xfs_vm_readpages
       mpage_readpages
       do_mpage_readpage
       submit_bio
       generic_make_request
       generic_make_request.part.50
       blk_queue_bio
       blk_flush_plug_list
+    12.32%     12.32%  202,16 R 0 () 1431480024 + 32 [java]
...
```

cassandra 가 block_rq_insert 이벤트를 발생 (block I/O 요청) 시키는

이유가 page fault, readahead 등과 관련된거 였다 ..

테스트 당시 Ubuntu 환경

- Page size: 2MB direct-mapped pages;
  huge pages (이전 환경 : 4KB)
- Readahead size: 2048KB (이전환경 : 128KB)

I/O 요청의 원인 page fault, read pages, readahead 와 관련된거라면 ..

I/O 요청의 원인 page fault, read pages, readahead 와 관련된거라면 ..

submit_io 는 몇번 불리고 ?

filemap_fault 는 몇번이나 불리는가 ?

동적으로 특정 커널함수 probe events 로 지정

```
# perf probe --add submit_bio
Added new event:
  probe:submit_bio     (on submit_bio)

You can now use it in all perf tools, such as:

      perf record -e probe:submit_bio -aR sleep 1
```

동적으로 특정 커널함수 probe events 로 지정

```
# perf probe --add filemap_fault
Added new event:
  probe:filemap_fault     (on filemap_fault)

You can now use it in all perf tools, such as:

      perf record -e probe:filemap_fault -aR sleep 1
```

```
# perf stat -e probe:submit_bio,probe:filemap_fault
...
             27881      probe:submit_bio
             2203       probe:filemap_fault
...
```

```
# perf stat -e probe:submit_bio,probe:filemap_fault
...
             27881      probe:submit_bio
             2203       probe:filemap_fault
...
```

왜 filemap_fault 보다 submit_bio 가 더 많이 부릴까 ? 거의 10 배 차이로

Filemap Fault 로 가져와야할 page 가 많으니까 ..

Filemap Fault 로 가져와야할 page 가 많으니까 ..

왜 많은가 ?

Filemap Fault 로 가져와야할 page 가 많으니까 ..

왜 많은가 ?

테스트 당시 Ubuntu 환경

- Page size: 2MB direct-mapped pages;
  huge pages (이전 환경 : 4KB)
- Readahead size: 2048KB (이전환경 : 128KB)

uftrace: A function graph tracer for C/C++ userspace programs

https://github.com/namhyung/uftrace

https://github.com/CppCon/CppCon2016/blob/master/Posters/uftrace%20-%20A%20function%20graph%20tracer%20for%20userspace%20programs/uftrace%20-%20A%20function%20graph%20tracer%20for%20userspace%20programs%20-%20Namhyung%20Kim%20and%20Honggyu%20Kim%20-%20CppCon%202016.pdf

```
# uftrace record -K5 -F filemap_fault@kernel
...
# uftrace replay -k -D5

            [13289] |   filemap_fault() {
            [13289] |     pagecache_get_page() {
   0.244 us [13289] |       find_get_entry();
   0.614 us [13289] |     } /* pagecache_get_page */
   0.170 us [13289] |     max_sane_readahead();
            [13289] |     __do_page_cache_readahead() {
            [13289] |       __page_cache_alloc() {
   2.162 us [13289] |         alloc_pages_current();
   2.531 us [13289] |       } /* __page_cache_alloc */
...
            [13289] |       wait_on_page_bit_killable() {
  89.298 us [13289] |         __wait_on_bit();
  89.685 us [13289] |       } /* wait_on_page_bit_killable */
  90.393 us [13289] |     } /* __lock_page_or_retry */
   0.222 us [13289] |     put_page();
 112.220 us [13289] |   } /* filemap_fault */
```

```
# uftrace record -K5 -F filemap_fault@kernel
...
# uftrace replay -k -D5

            [13289] |   filemap_fault() {
            [13289] |     pagecache_get_page() {
   0.244 us [13289] |       find_get_entry();
   0.614 us [13289] |     } /* pagecache_get_page */
   0.170 us [13289] |     max_sane_readahead();
            [13289] |     __do_page_cache_readahead() {
            [13289] |       __page_cache_alloc() {
   2.162 us [13289] |         alloc_pages_current();
   2.531 us [13289] |       } /* __page_cache_alloc */
...
            [13289] |       wait_on_page_bit_killable() {
  89.298 us [13289] |         __wait_on_bit();
  89.685 us [13289] |       } /* wait_on_page_bit_killable */
  90.393 us [13289] |     } /* __lock_page_or_retry */
   0.222 us [13289] |     put_page();
 112.220 us [13289] |   } /* filemap_fault */
```

```
# perf probe -e probe:__do_page_cache_readahead
...
      java-8714  [000] 13445354.703793: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8716  [002] 13445354.819645: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8734  [001] 13445354.820965: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8709  [000] 13445354.825280: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
...
```

0x200 = 512 Pages * 4KB = 2048KB

readahead 값 변경 후에도 변화가 없는 문제 ..

```
# perf probe -e probe:__do_page_cache_readahead
...
      java-8714  [000] 13445354.703793: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8716  [002] 13445354.819645: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8734  [001] 13445354.820965: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8709  [000] 13445354.825280: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
...
```

0x200 = 512 Pages * 4KB = 2048KB

vfs_open 발생시 한번만 file_ra_state_init() 로 readahead 값 초기화가 문제

```
# perf probe -e probe:__do_page_cache_readahead
...
      java-8714  [000] 13445354.703793: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8716  [002] 13445354.819645: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8734  [001] 13445354.820965: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
      java-8709  [000] 13445354.825280: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=200
...
```

0x200 = 512 Pages * 4KB = 2048KB

Cassandra DB Restart 로 해결

```
# perf probe -e probe:
...
      java-8714  [000] 13445354.703793: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=80
      java-8716  [002] 13445354.819645: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=80
      java-8734  [001] 13445354.820965: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=80
      java-8709  [000] 13445354.825280: probe:__do_page_cache_readahead: (0x0/0x180) nr_to_read=80
...
```

0x80 = 128 Pages * 4KB = 512KB

문제상황종료 : Ubuntu 의 default readahead 설정이 핵심원인

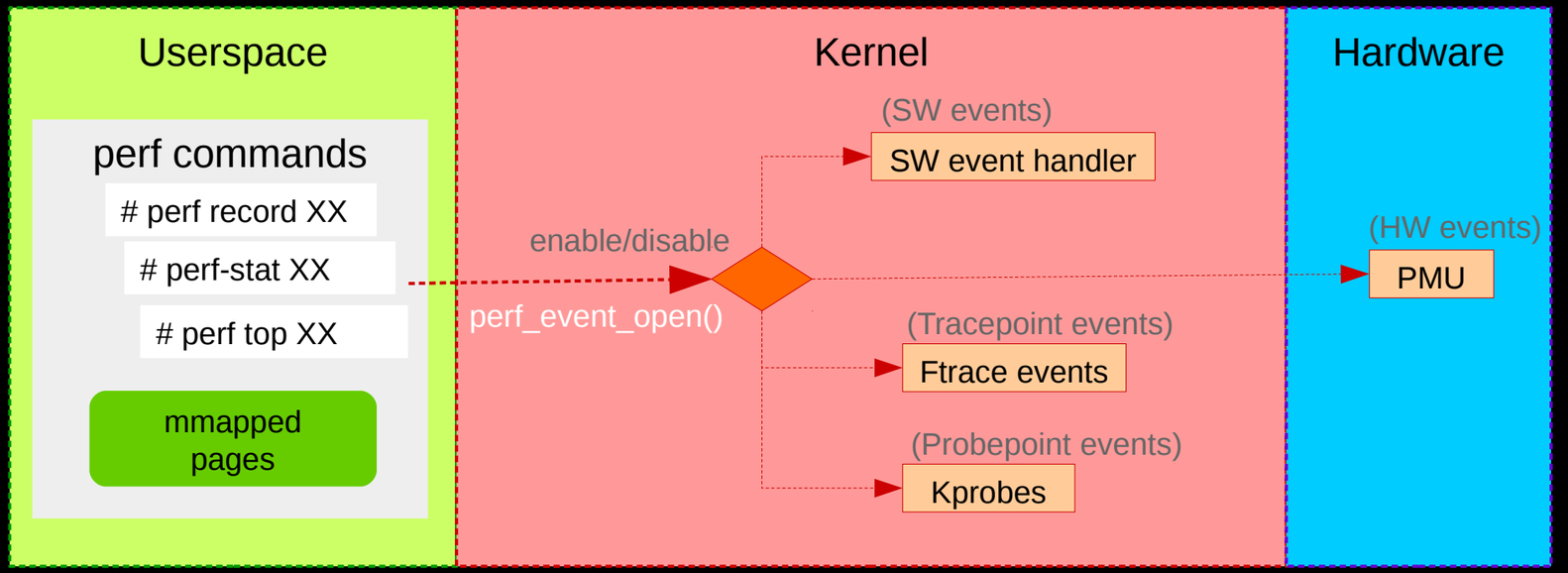
Readahead 세팅변경 (2048KB → 512KB) 후 해결

```
# cat /sys/block/sda/queue/read_ahead_kb
2048

# blockdev --getra /dev/sda
4096

# blockdev --setra 1024 /dev/sda

# cat /sys/block/sda/queue/read_ahead_kb
512
```

perf 의 Event Sampling 은
어떻게 동작 하는 가 ?

Events Sampling 원리

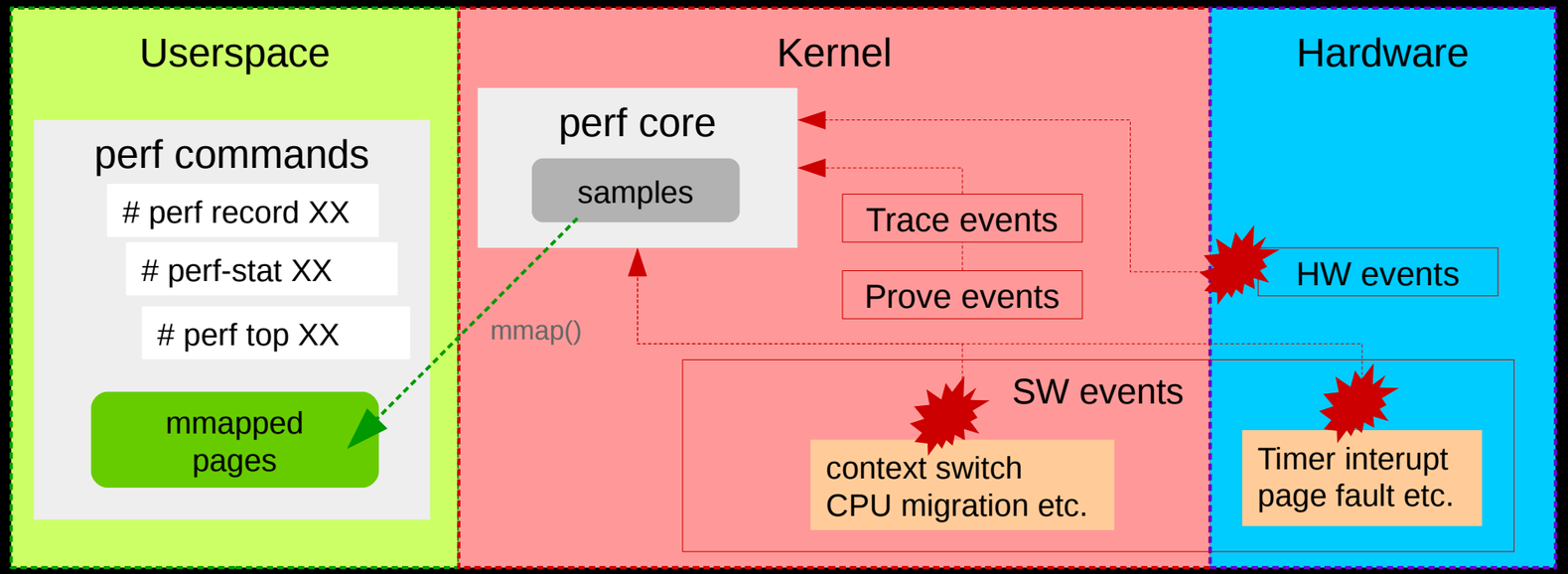
1. **perf_event_open()** 시스템 콜을 하게 되면 (file descriptor 리턴)
2. **성능정보 측정 할 수 있는 file descriptor 로 각각의 event 들과 연결**
   (동시에 여러이벤트들을 수집가능하게 Grouping 가능)
3. 각 event 들을 **enable** or **disable** 한다 (by ioctl() / prctl())



참조 : http://events.linuxfoundation.org/sites/events/files/lcjp13_takata.pdf

Events Sampling 원리

4. **counting** : event 들이 **발생횟수**를 누적하여 센다 .
5. **sampling** : **측정 정보**를 **버퍼에 쓴다** (mmap() 로 받을 준비를 한뒤 ,
   특정이벤트 발생시 (interrupt) 매번 수집 또는 지정된 특정 주기로 수집)
6. perf command 가 **mmap()** 을 통해서 **성능정보 수집** (kernel 에서 user 의 copying 없이)



참조 : http://events.linuxfoundation.org/sites/events/files/lcjp13_takata.pdf

내 개발환경에 perf 를 어떻게 적용시킬까 ?

각종 유용한 Event 소개

* 참고

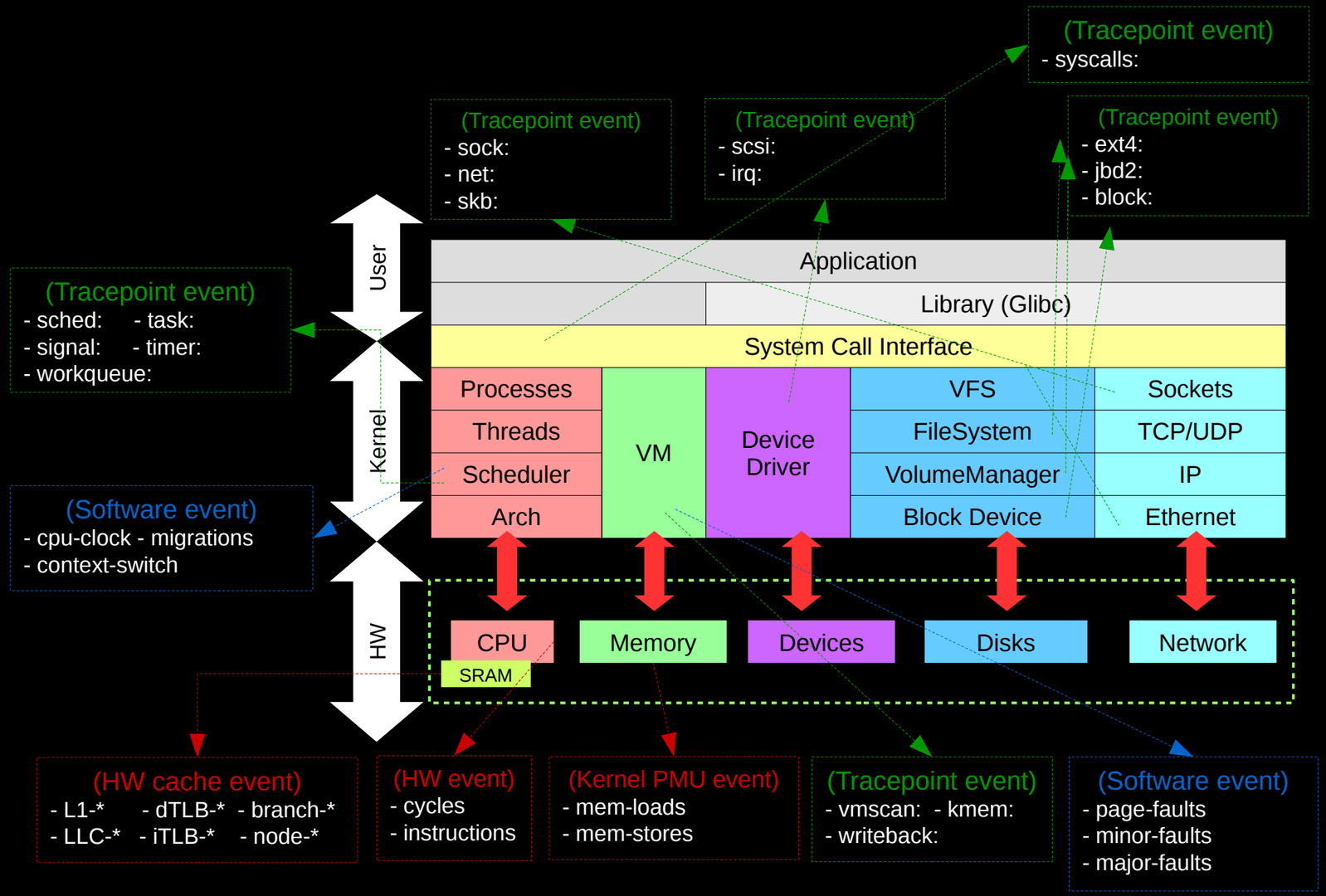
perf 설치하기

sudo apt-get install linux-tools-common linux-generic

perf 사용법 익히기

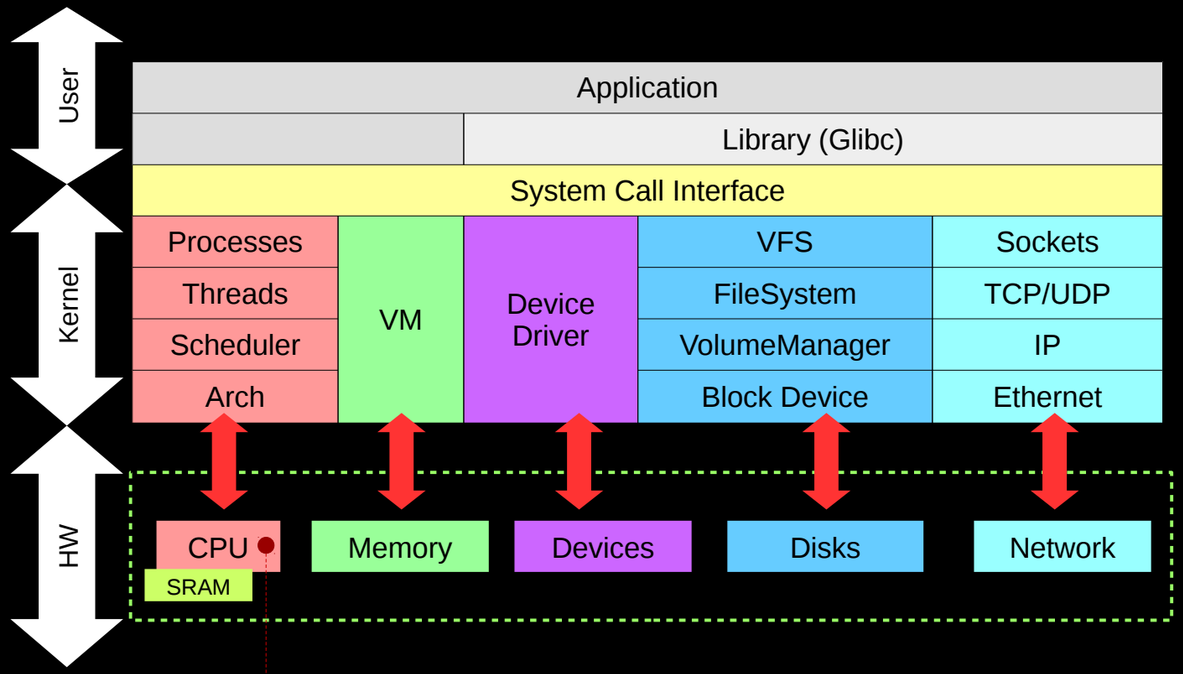
http://www.brendangregg.com/perf.html

System Layer 상에서 원하는 성능분석 지점 (Focus) 을 선택가능



Events on CPU

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

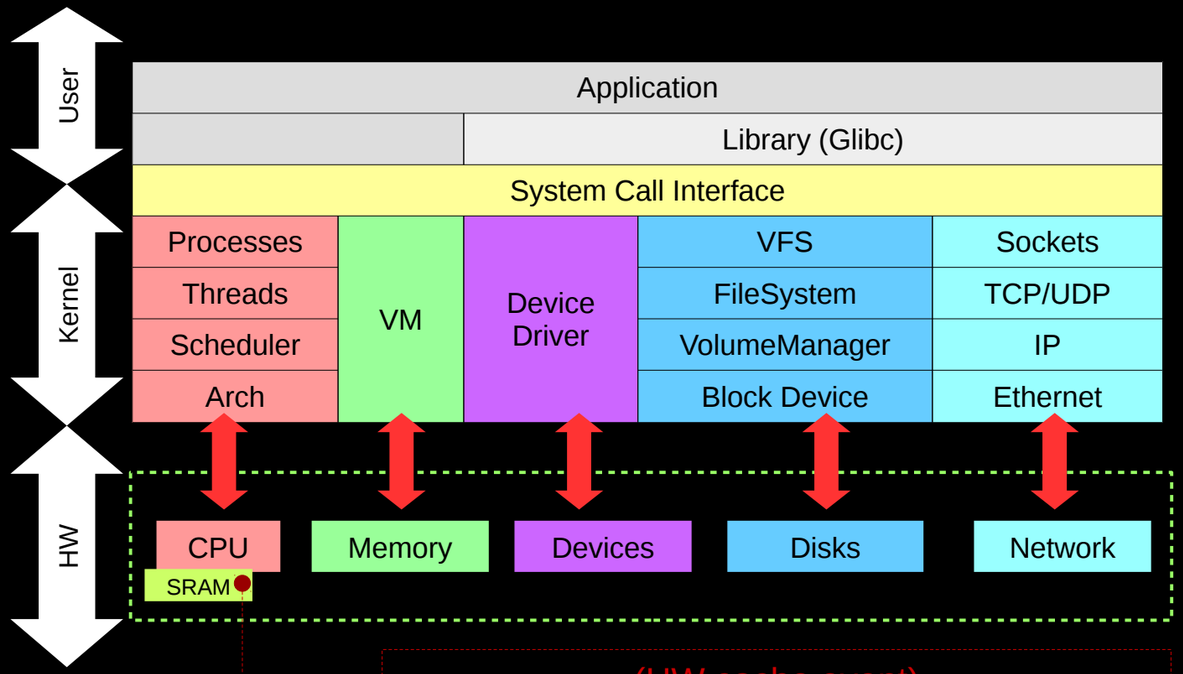


(HW event)

- cycles : CPU 클럭 싸이클 수
- instructions : 실행한 명령어의 개수
- branch-instructions : 실행한 분기명령의 개수
- branch-misses : 분기예측 (branch prediction) 의 실패 수

Events on SRAM

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz



(HW cache event)

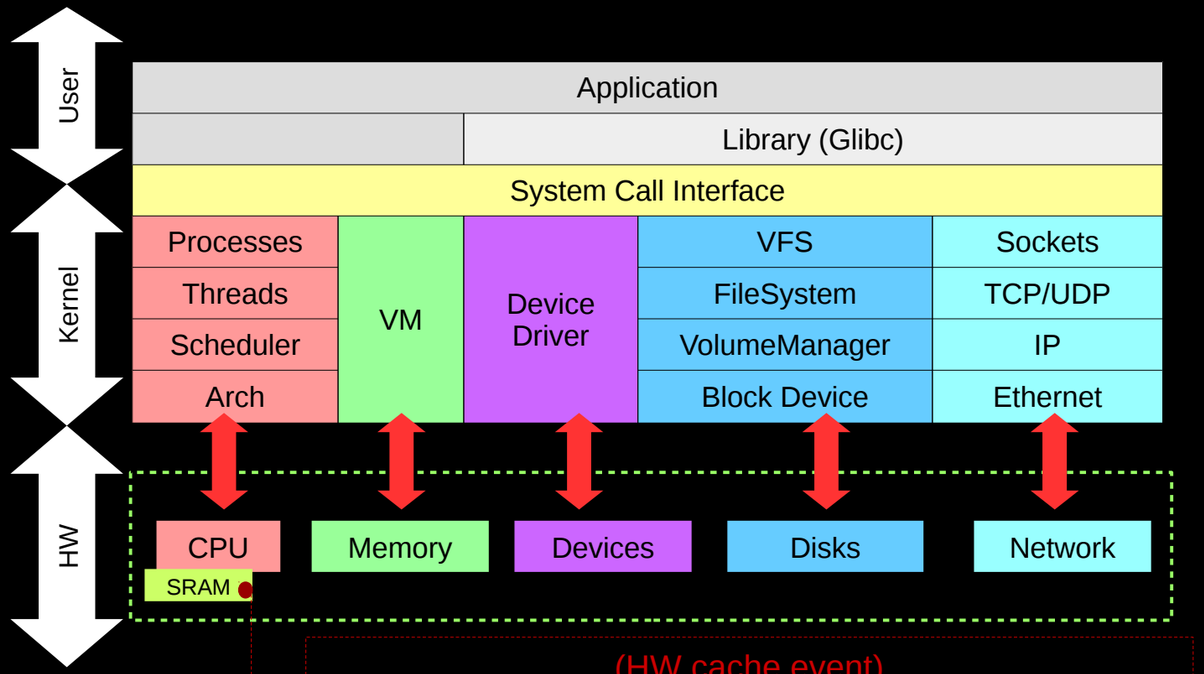
- L1-dcache-load-misses : 데이터 캐시 로드 실패 수
- L1-dcache-loads : 데이터 캐시 로드 실패 수
- L1-icache-load-misses : 명령어 캐시 로드 실패 수

(LLC : Last Level Cache ; CPU 와 가장 멀리있는 캐시)

- LLC-loads, LLC-load-misses : LLC 로드 관련
- LLC-stores, LLC-store-misses : LLC 저장 관련

Events on SRAM

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

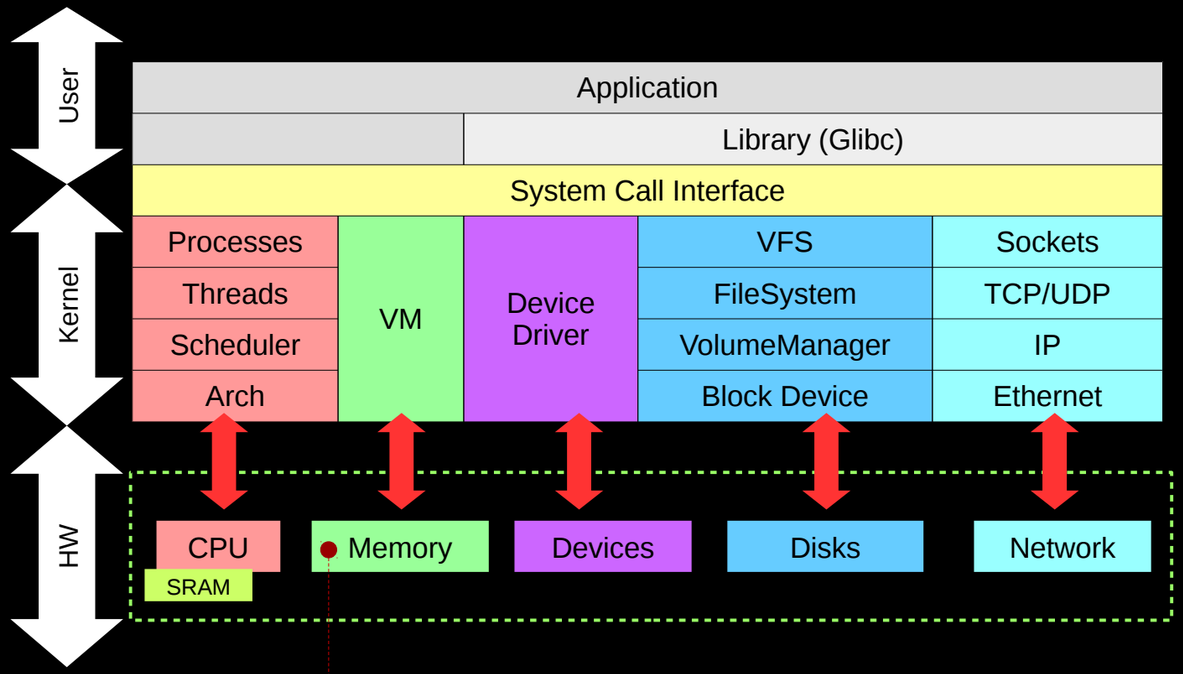


(HW cache event)

- branch-loads, branch-load-misses : branch prediction unit 관련
(TLB: Translation lookaside buffer; 가상 메모리 주소를 위한 캐시)
- dTLB-loads, dTLB-load-misses : 데이터 TLB 로드 관련
- dTLB-stores, dTLB-store-misses : 데이터 TLB 저장 관련
- iTLB-loads, iTLB-load-misses : 명령어 TLB 로드 관련
- node-loads, node-load-misses : 로컬 메모리 로드 관련
- node-stores, node-store-misses : 로컬 메모리 저장 관련

Events on Memory

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

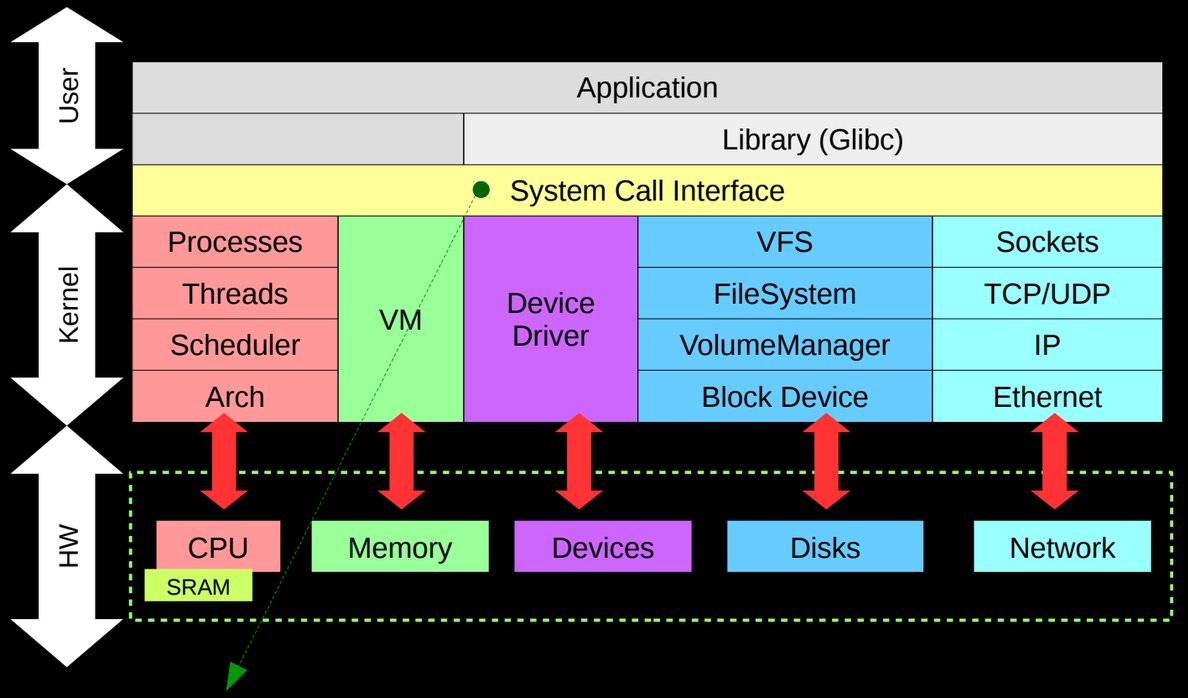


(Kernel PMU event)

- mem-loads : 메모리로 부터 cpu 로 데이터 / 명령어 가져오는 횟수
- mem-stores : cpu 의 레지스터 (ex. AC (누산기 , Acummulator)) 로 부터 데이터를 메모리에 저장하는 횟수

Events for System Calls

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

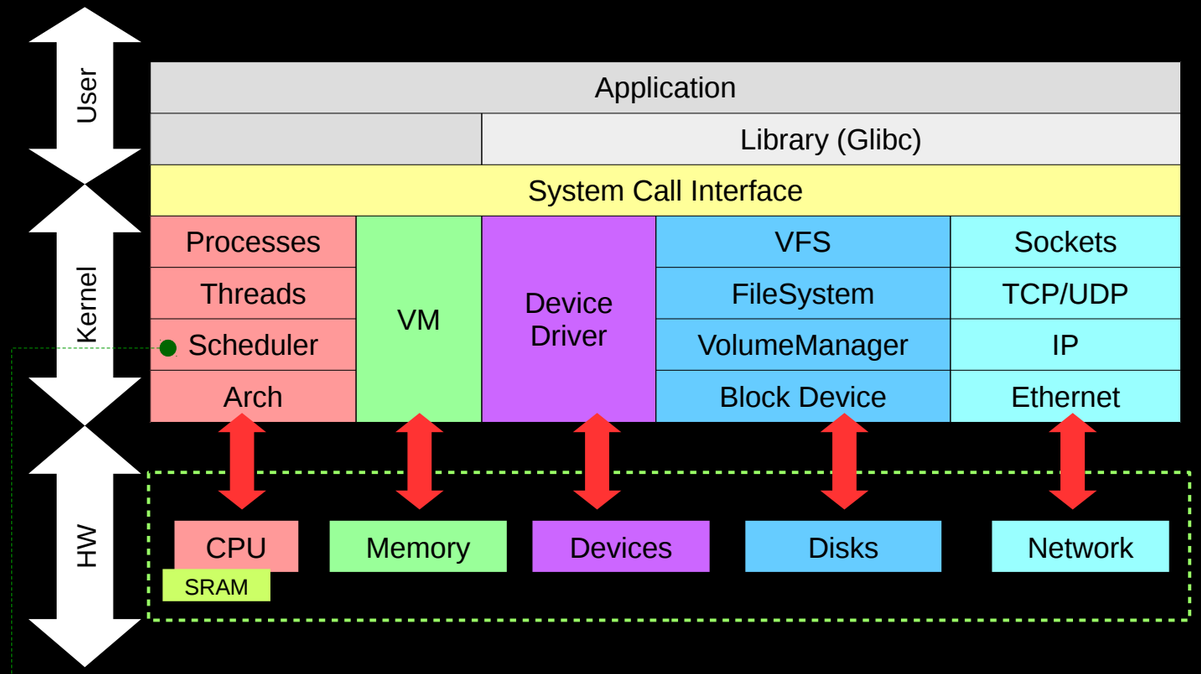


(Tracepoint event)

- syscalls: 각종 시스템 콜이 호출 (sys_enter_*) 되고 수행이 끝나는 (sys_exit_*) point 를 확인한다 .
  어떤 코드부분에서 얼마나 불려졌는지를 확인 할 수 있다 .
  시스템콜은 300 가지 정도 확인이 가능하다 . (perf version 4.5.rc2 기준)

Events for Scheduler

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

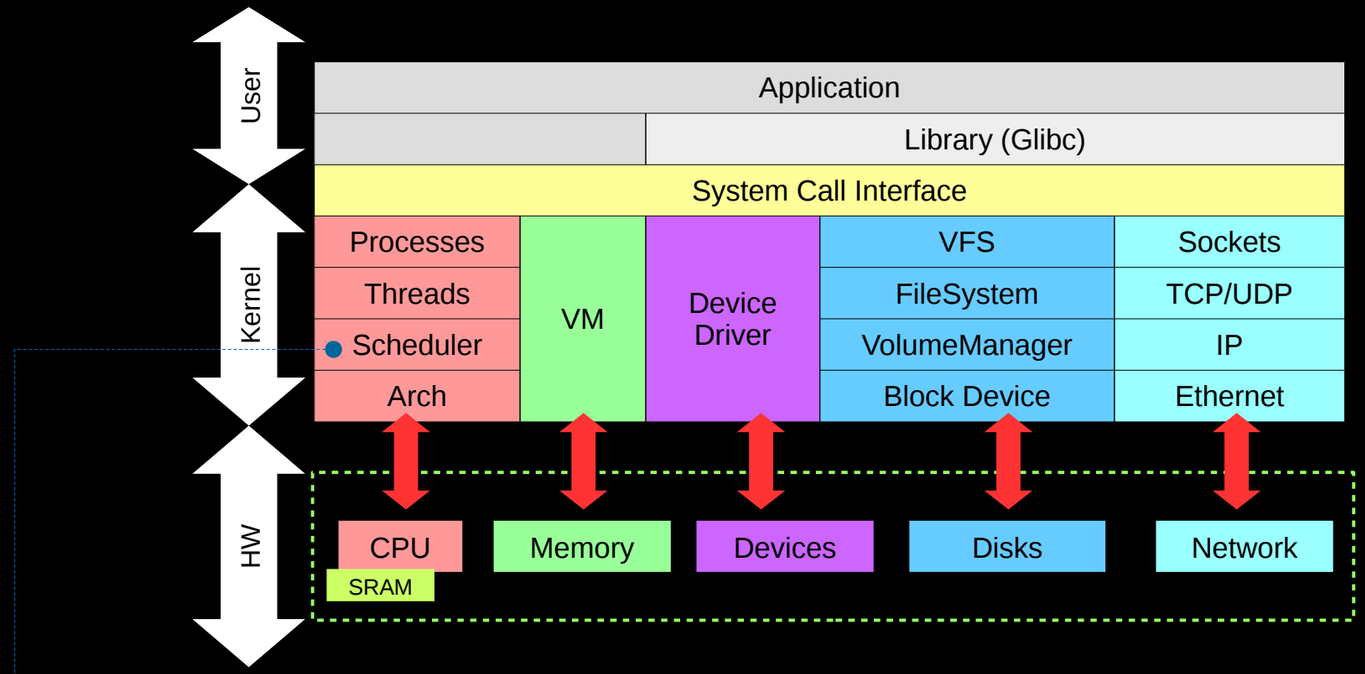


(Tracepoint event)

- sched: 스케줄링 관련
- signal: 시그널 생성 및 전달 관련
- workqueue: 워크큐가 언제 시작되고 끝났는지 현재 active 한 작업 관련 확인가능
- task: 새로운 테스트가 생성 되었는지 또는 이름이 변경 되었는지 확인가능
- timer: 타이머의 상태 초기화 등 관련 정보 확인

Events for Scheduler

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

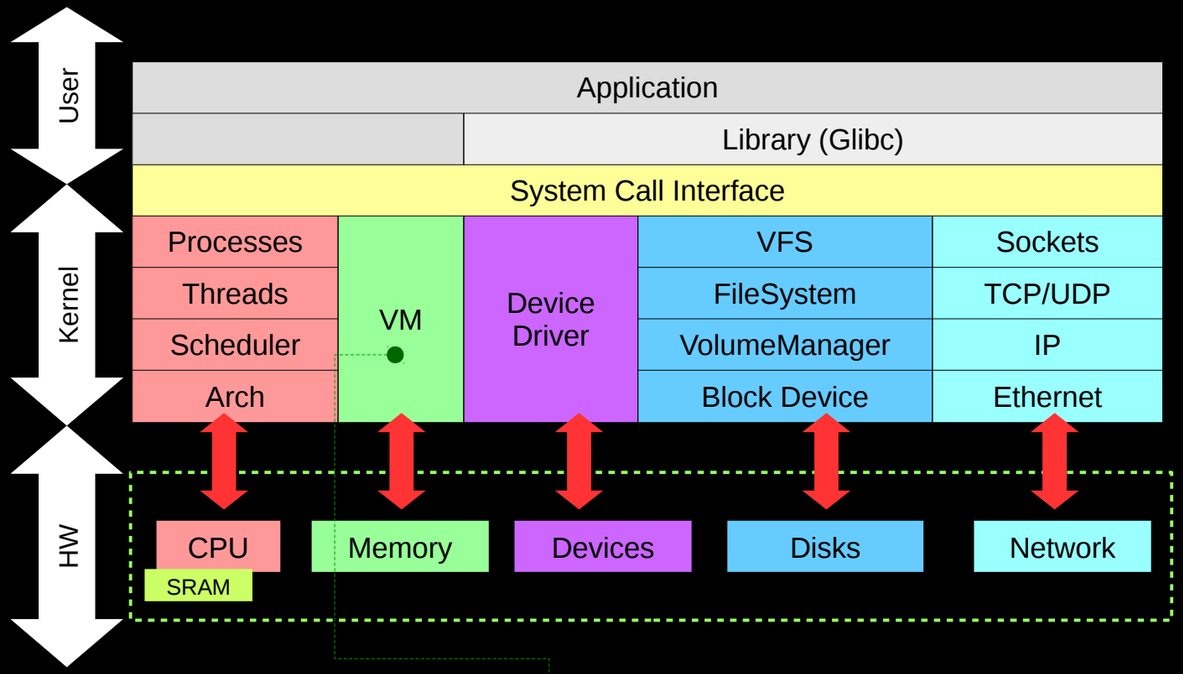


(Software event)

- cpu-clock: perf 가 record, stat 하는 등의 수행시간을 cpu 시간 단위로 확인 (msec 단위)
- migrations: cpu migration – context-switch: 컨택스트 스위치

Events for Virtual Memory

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

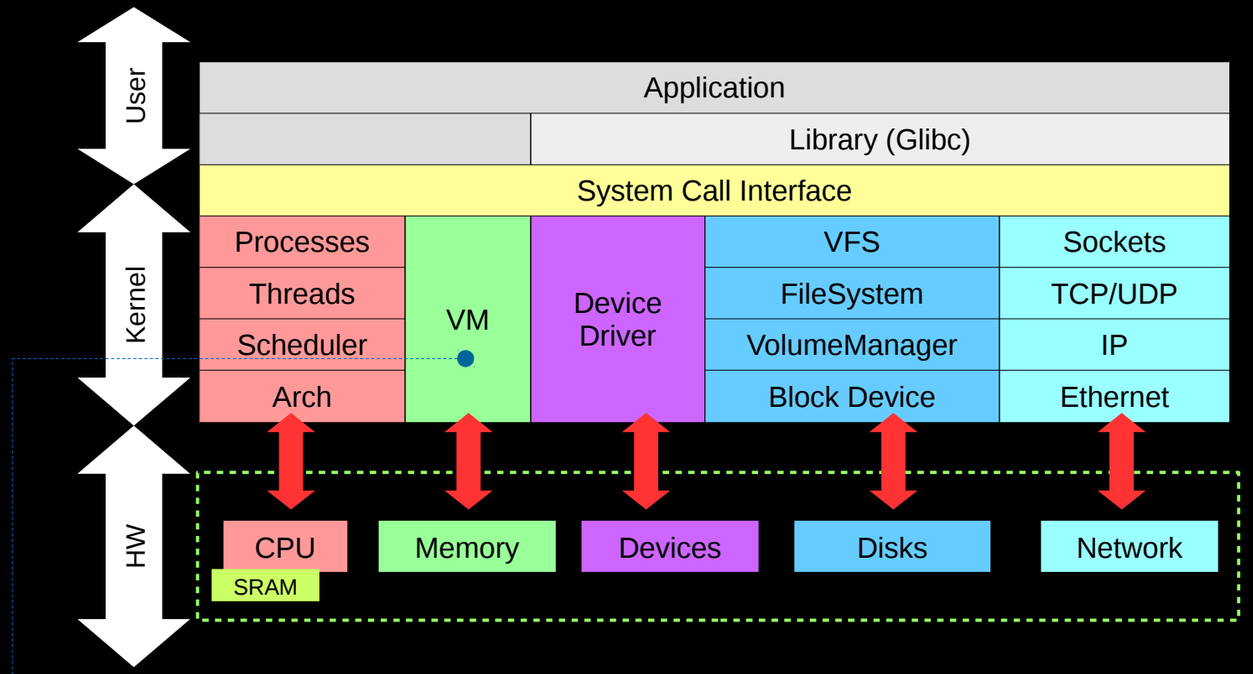


(Tracepoint event)

- vmscan: page reclaim (회수) 가 언제시작되고 끝났는지 등을 확인 가능
– kmem: kmalloc, page_alloc 등을 확인 가능
- writeback: writeback 관련 event 확인 가능

Events for Virtual Memory

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

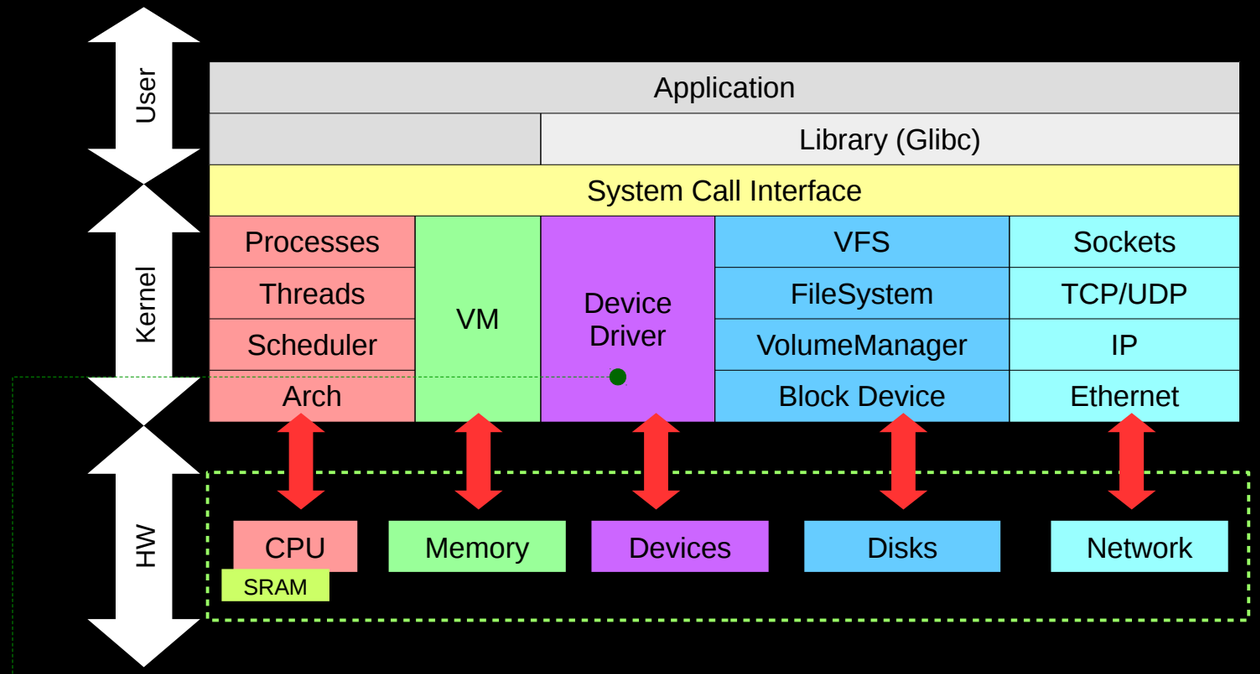


(Software event)

- page-faults : 페이지폴트 발생이 어디서 얼마나 났는지 확인 가능
- minor-faults: I/O 발생없이 MMU 만 로드정보를 없을때의 페이지폴트 관련
- major-faults : Disk I/O 발생이 필요한 페이지폴트 관련

Events for Device Driver

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

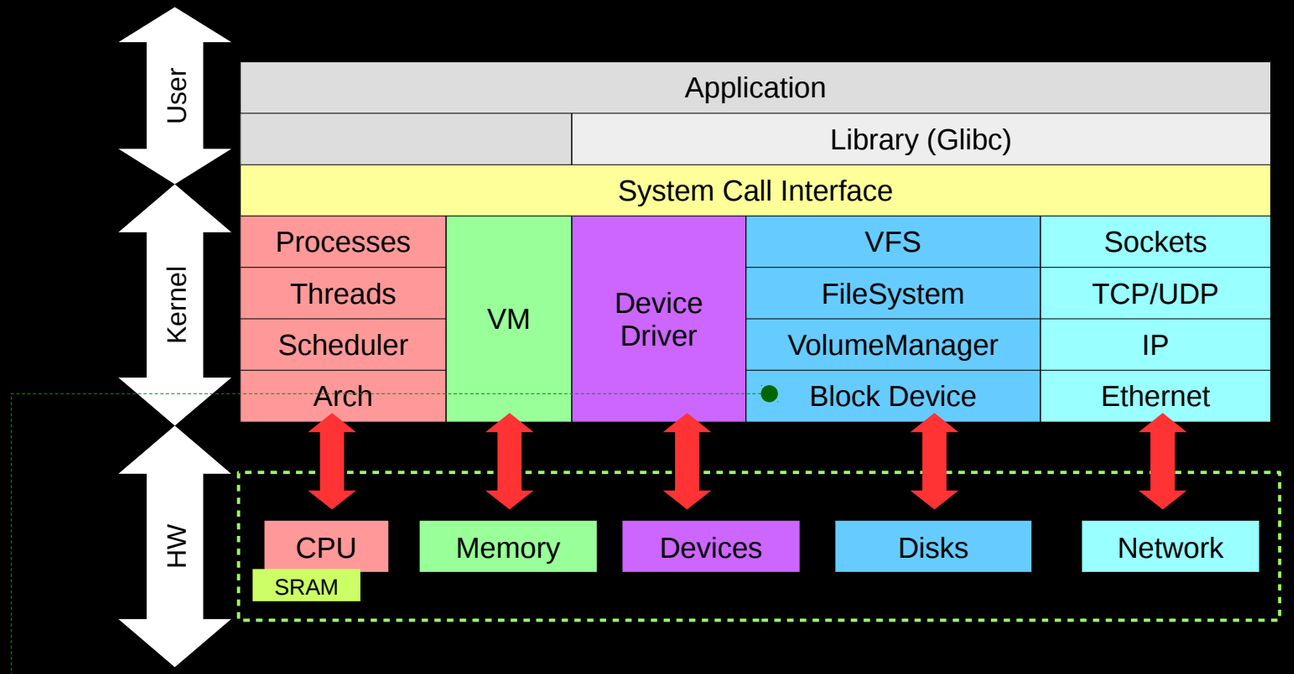


(Tracepoint event)

- scsi: 해당 기기에 명령이 오류가 났는지 수행이 끝났는지 등을 확인 가능 하다 .
  (Small Computer System Interface)
- irq: 인터럽트 핸들링 , 인터럽트 백터 관련한 작업을 확인 가능하다 .

Events for Block Device

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

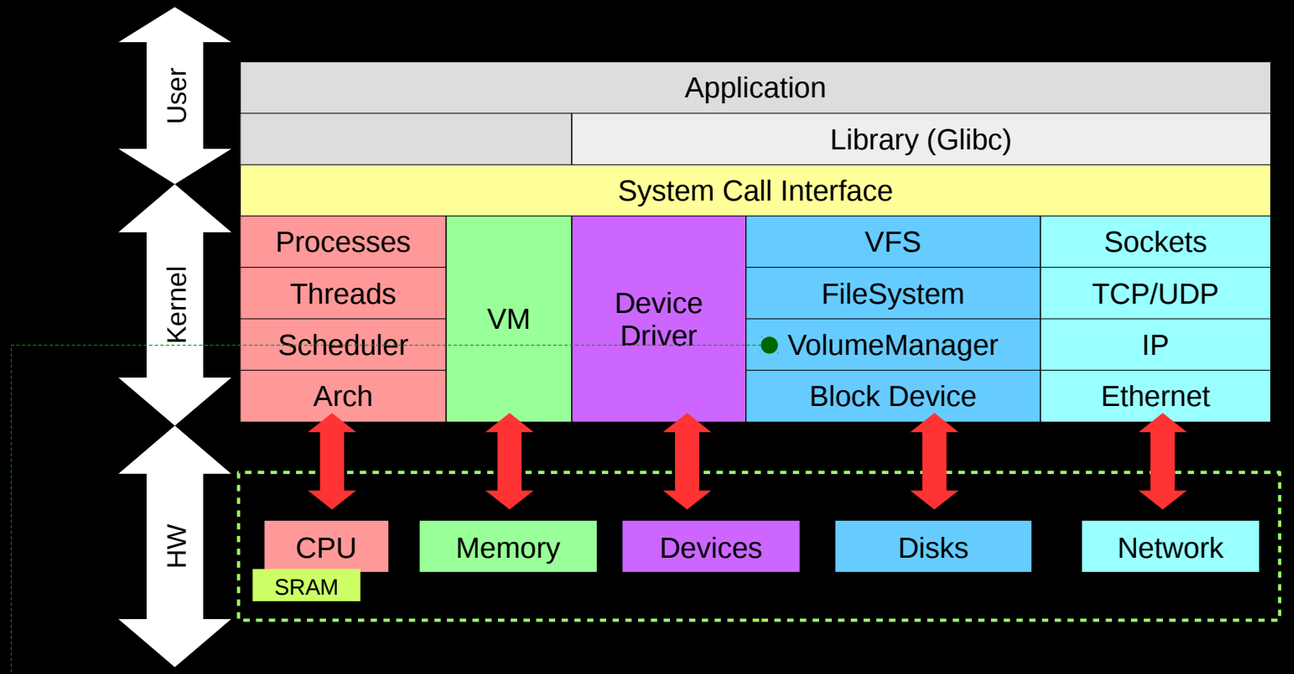


(Tracepoint event)

- block: 블록 I/O 관련 하여 request 가 완료됐는지 혹은 취소됐는지
  언제 어디서 됐는지를 확인 할 수 있다 .

Events for Volume Manager

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

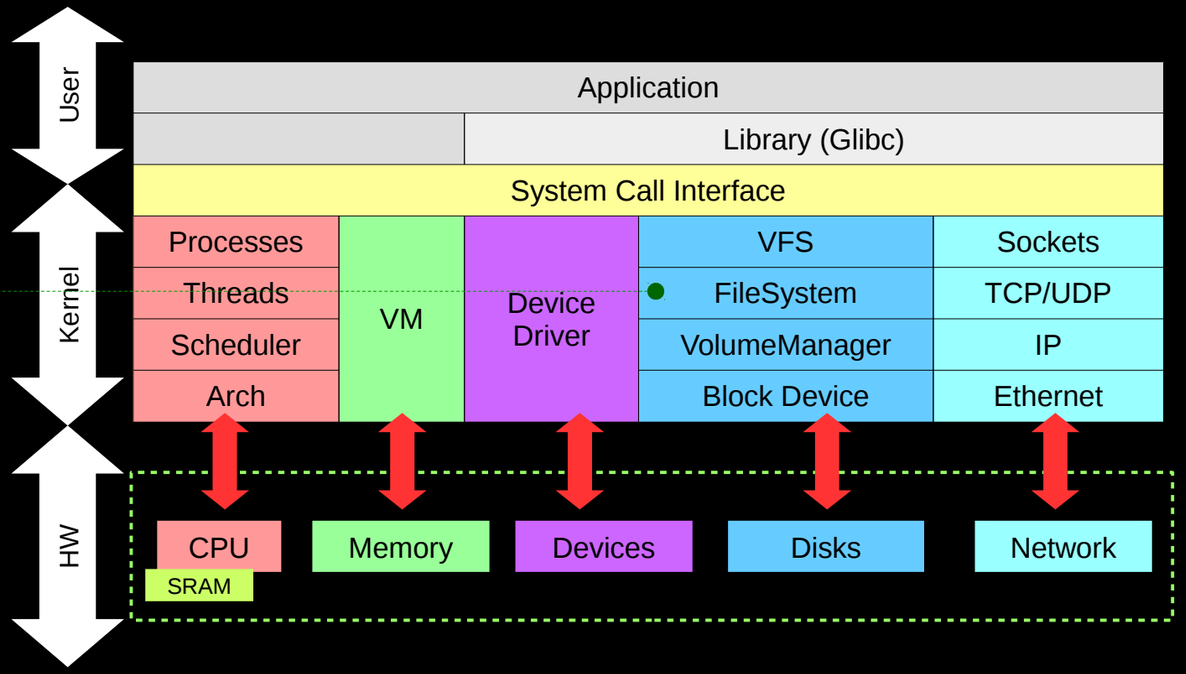


(Tracepoint event)

- jbd2: commit flushing 또는 locking 또는 commit 이 언제 시작하는지 등을 확인 가능하다 .

Events for File System

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

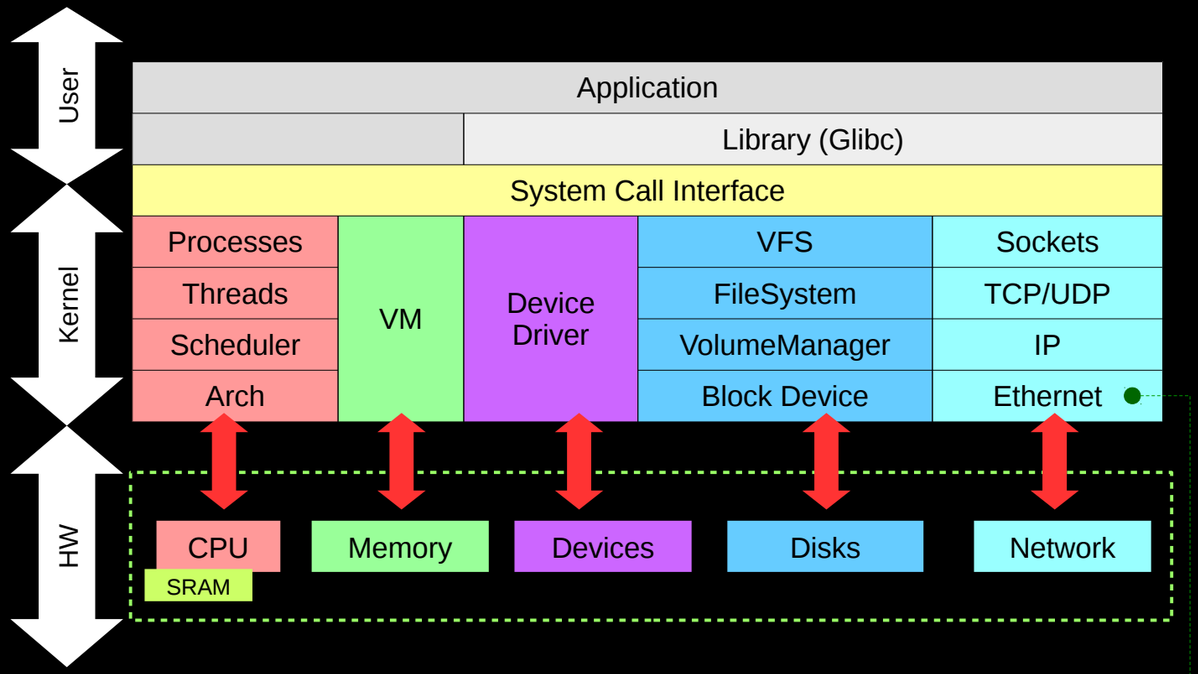


(Tracepoint event)

- ext4: inode 를 free 했는지 write 를 언제 시작했는지 등을 확인 할 수 있다 .

Events for Ethernet

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

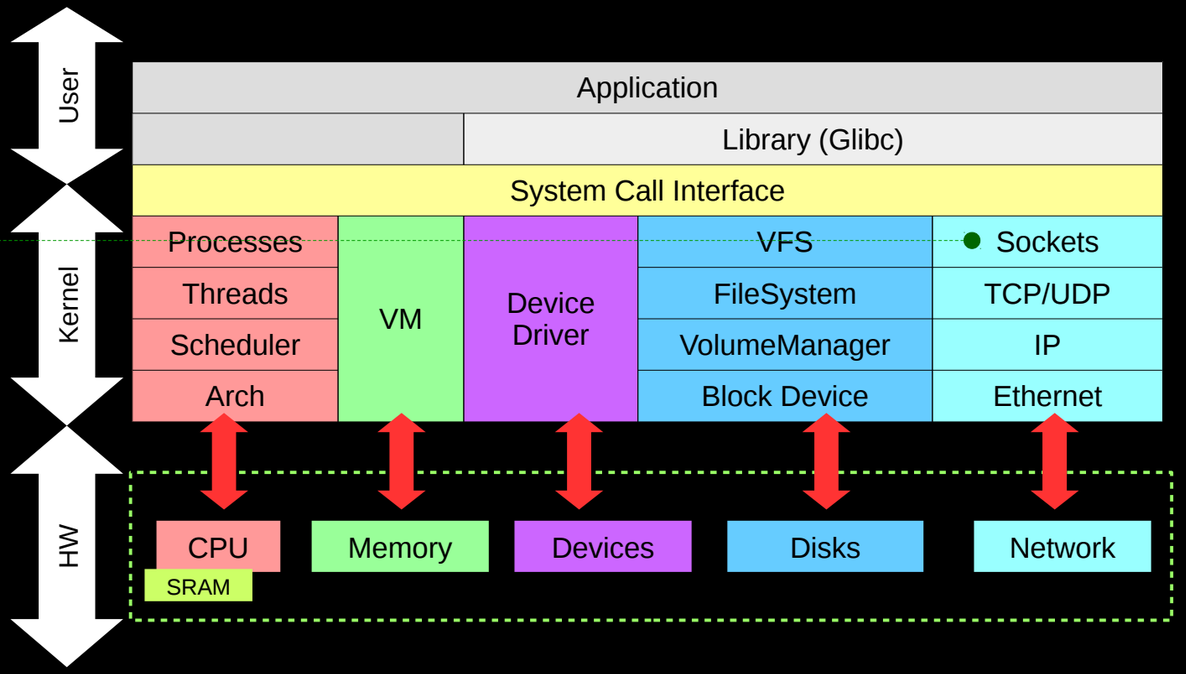


(Tracepoint event)

- net: rx 등을 확인 한다 .
- skb: 소켓버퍼를 얼마나 소비했는지 kfree 했는지 등을 확인 한다 .

Events for Sockets

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz



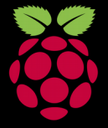
(Tracepoint event)

- sock: 소켓 receive 큐가 꽉 찼는지 버퍼한계가 초과 되었는지등을 확인 한다 .

Performance Event Sampling 을 통해

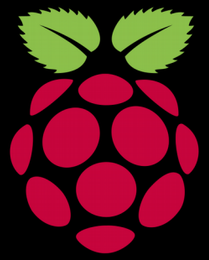
기존 성능도구보다 한층 더 날카롭게 다각도로

성능분석을 해보자



Raspberry pi 환경

C/C++ program 성능개선작전 (ioT, Embedded)

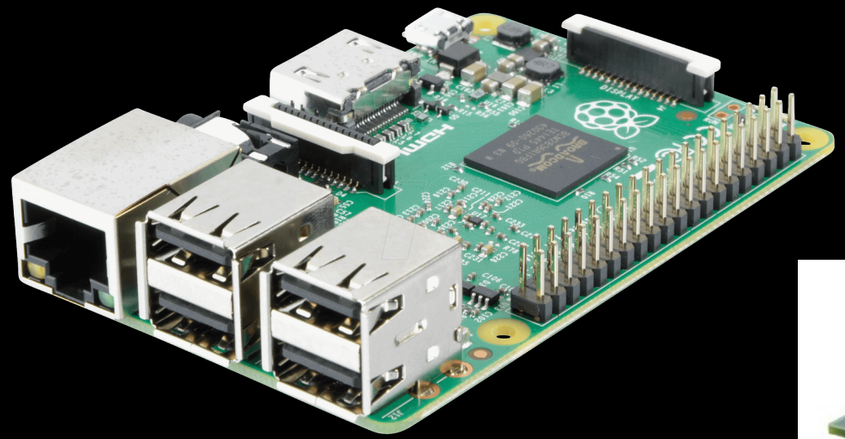


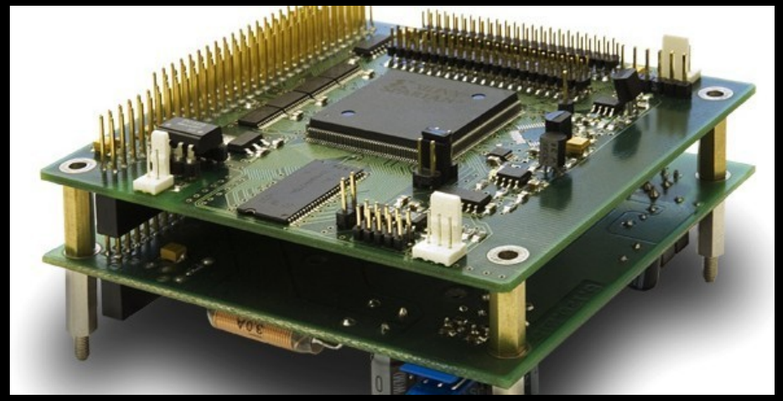
Raspberry Pi 2 with a 900MHz ARM Cortex-A7 quad-core processor

- 열악한 터미널 환경
- Overhead 가 적고 가벼운 성능도구 필요
- HW 에 맞춤형 성능도구 선호

임베디드 개발환경

Board 의 한정된 자원 (열악한 spec)

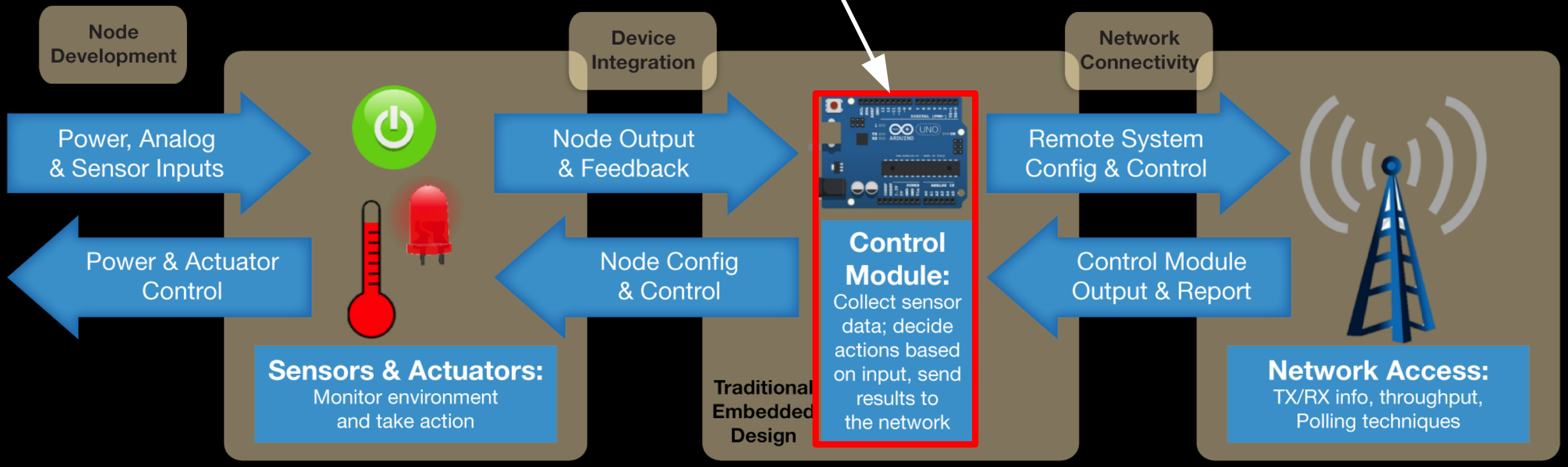




http://vistaembedded.com/embedded-training-projects-Gandhinagar-Ahmedabad-Gujarat.aspx

IoT 개발환경

Control Module 성능개선이 필요한경우



http://www.rigolna.com/iot/

이미 개선한 임베디드 / IoT 프로그램 , HW 한계 때문에

한번 성능 쥐어짜기 ?

패션모델들의 다이어트 ?

패션모델들의 다이어트 ?

↓

마른 수건을 짠다

1 단계

time check

2 단계

uftrace

3 단계

perf

1 단계

time check

2 단계

uftrace

3 단계

perf

```
# time ./old
6623856

real    0m4.125s
user    0m4.120s
sys     0m0.000s
```

```
# time ./old
6623856

real    0m4.125s
user    0m4.120s
sys     0m0.000s


# time ./improved
6623856

real    0m1.559s
user    0m1.560s
sys     0m0.000s
```

1 단계

time check

2 단계

uftrace

3 단계

perf

```
# uftrace record ./old
6623856

# uftrace graph
```

```
# uftrace record ./old
6623856

# uftrace graph
```

```
calling functions
================================
  19.037  s : (1) main
  36.927 us :  +-(1) fopen
            :  |
   2.232 ms :  +-(941) fgets
            :  |
  18.144 ms :  +-(941) get_values_from
   2.528 ms :  |  +-(941) strdup
            :  |  |
   1.625 ms :  |  +-(941) strsep
            :  |  |
   3.742 ms :  |  +-(1882) atoi
            :  |
  29.583 us :  +-(2) malloc
            :  |
  19.009  s :  +-(940) pack_knapsack
   8.566  s :  | (4264680) get_cond_maxprice
            :  |
 100.104 us :  +-(1) printf
            :  |
  16.615 us :  +-(2) free
```

```
# uftrace record ./improved
6623856

# uftrace graph
```

```
# uftrace record ./improved
6623856

# uftrace graph
```

```
calling functions
================================
  16.209  s : (1) main
  42.865 us :  +-(1) fopen
            :  |
   2.509 ms :  +-(941) fgets
            :  |
  20.303 ms :  +-(941) get_values_from
   2.604 ms :  |  +-(941) strdup
            :  |  |
   1.783 ms :  |  +-(941) strsep
            :  |  |
   4.179 ms :  |  +-(1882) atoi
            :  |
  82.916 us :  +-(2) malloc
            :  |
  16.178  s :  +-(940) pack_knapsack
   5.936  s :  | (4264680) get_cond_maxprice
            :  |
 106.249 us :  +-(1) printf
            :  |
  16.354 us :  +-(2) free
```

1 단계

time check

2 단계

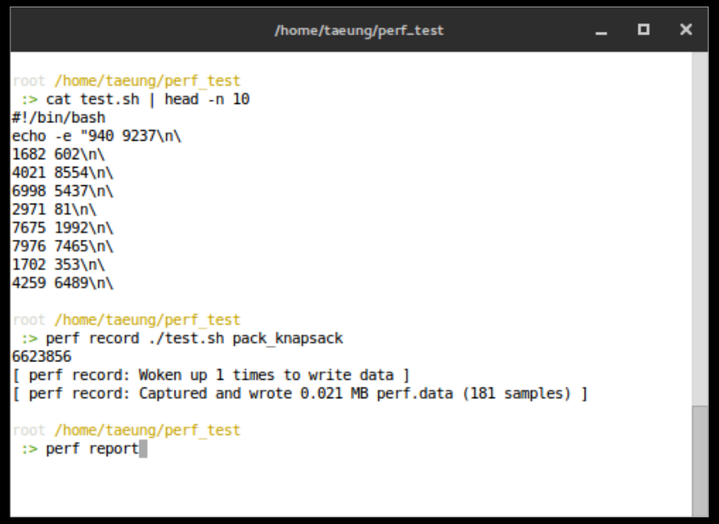
uftrace

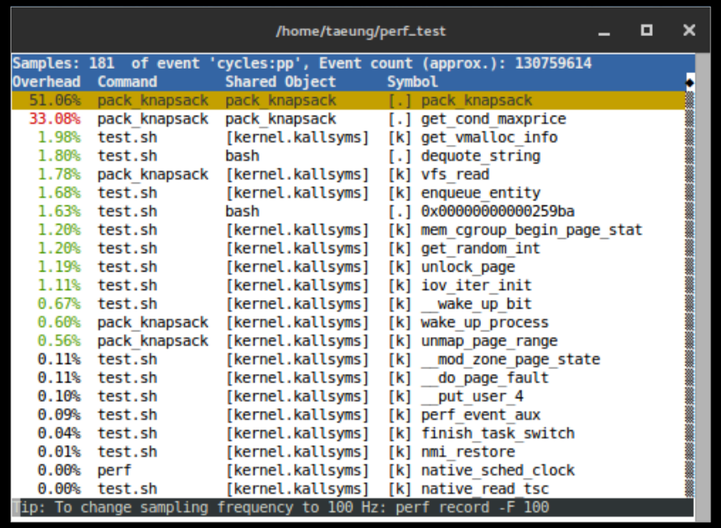
3 단계

perf

perf **record** ./test.sh pack_knapsack (성능분석 정보수집)

perf **report** (성능분석 결과 view)

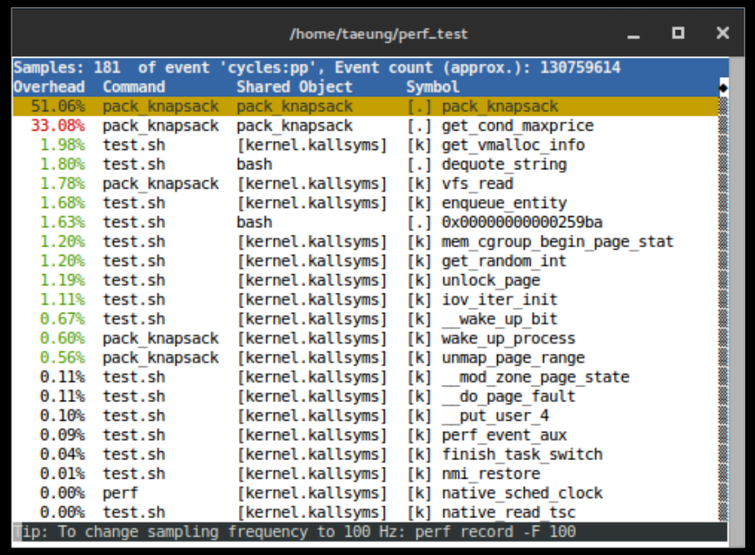


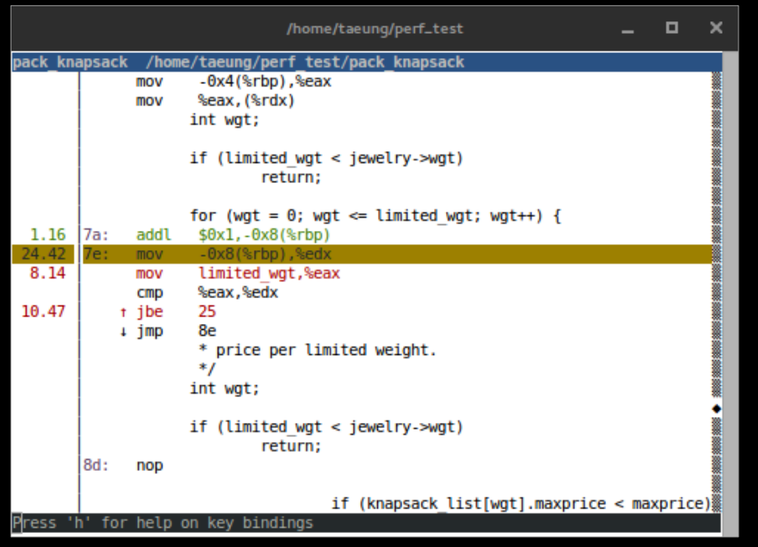


perf **record** ./test.sh pack_knapsack (성능분석 정보수집)

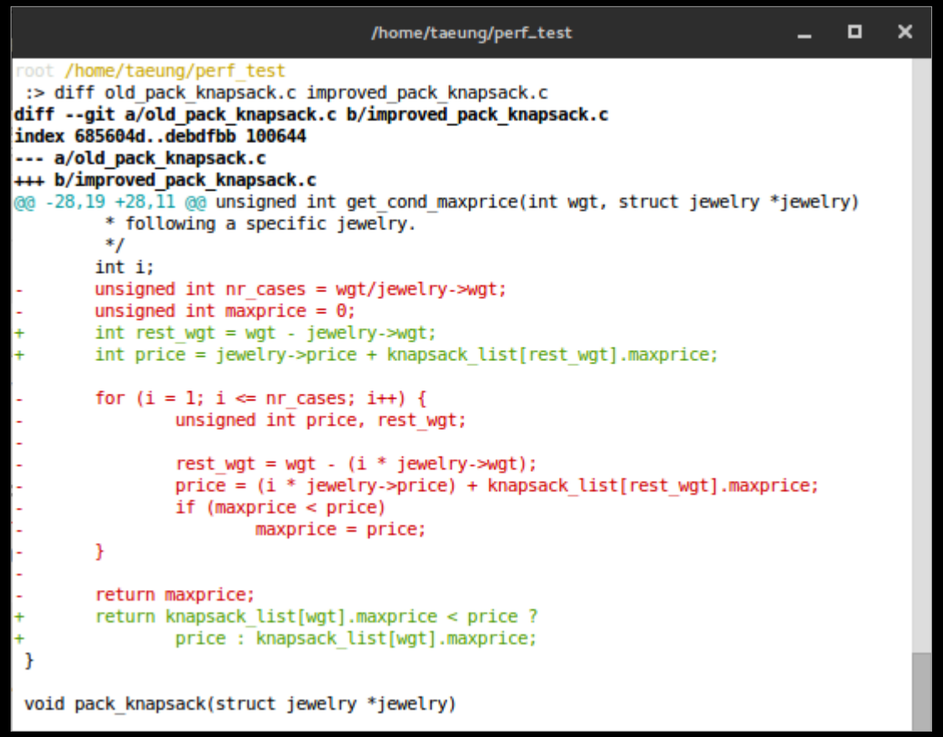
perf **report** (성능분석 결과 view)

코드 어느부분이 overhead 가 높은지 성능분석결과 확인 (perf-annotate)





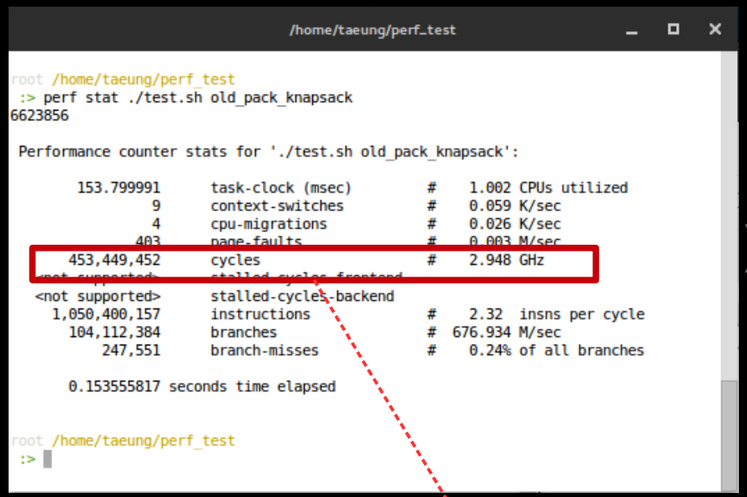
patch pack_knapsack.c < improvement.patch (**개선** : overhead 가 높은 **코드수정**)

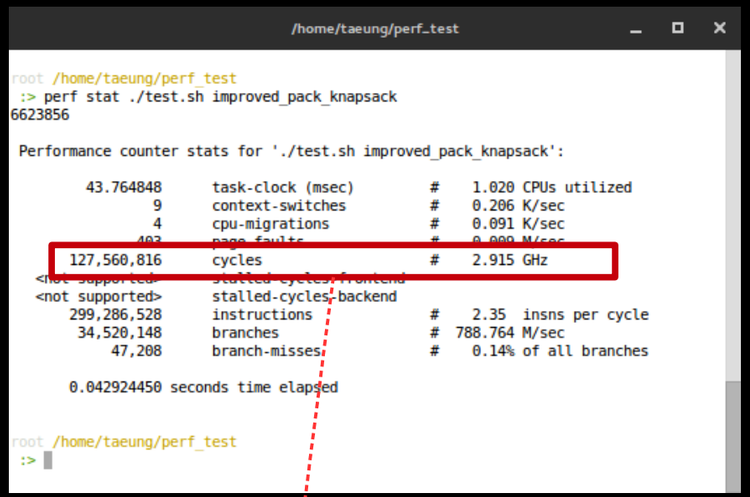


perf **stat** ./test.sh old_pack_knapsack (event 카운트)

overhead 가 높았던 pack_knapsack() 함수 코드 개선 후

(성능개선후 다시 event 카운트)





총 70% cpu-cycles 감소

453,449,452 − 325,888,636 = 127,560,816 cycles

ARMv7 performance event table for

Raspberry pi

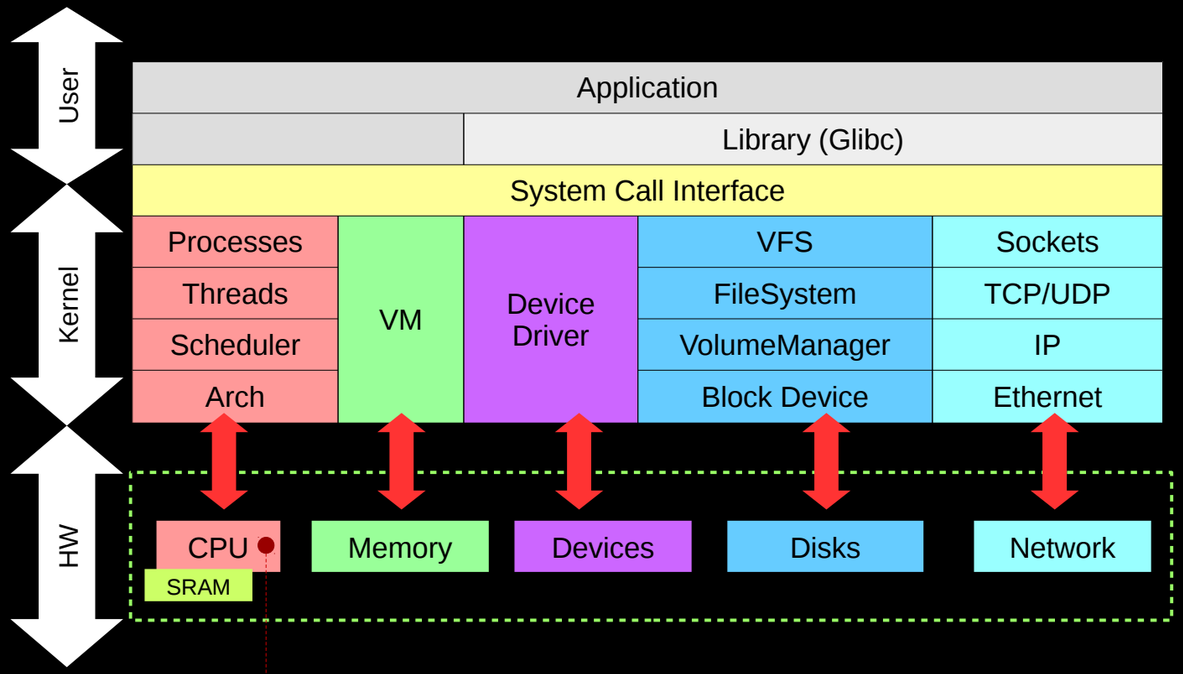
PERF hardware performance events

EVENT NAME	ARMV7 EVENT ID
cpu-cycles OR cycles	0xFF
instructions	0x08
cache-references	0x04
cache-misses	0x03
branch-instructions OR branches	0x0C
branch-misses	0x10
bus-cycles	0x1D
L1-dcache-loads	0x04
L1-dcache-load-misses	0x03
L1-dcache-stores	0x04
L1-dcache-store-misses	0x03
L1-icache-loads	0x14

...

Events on CPU

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz

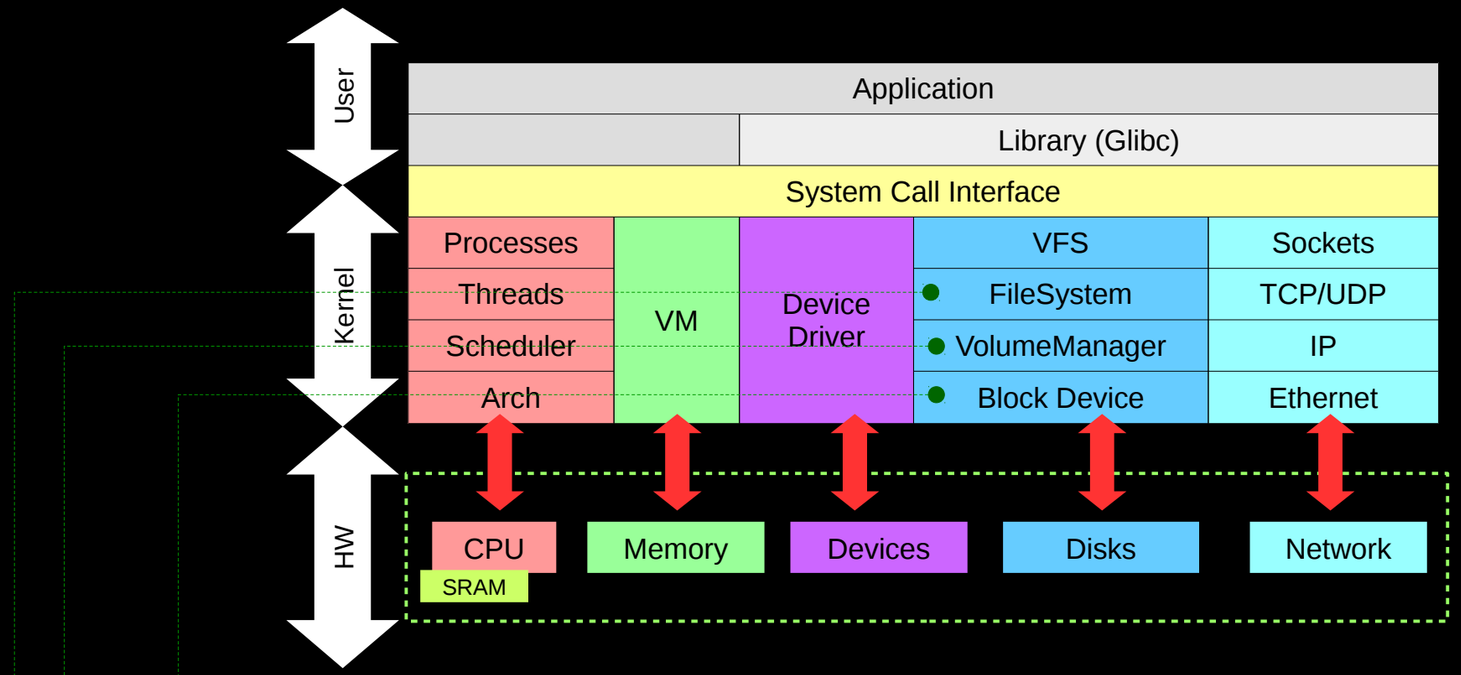


(HW event)

- cycles : CPU 클럭 싸이클 수
- instructions : 실행한 명령어의 개수
- branch-instructions : 실행한 분기명령의 개수
- branch-misses : 분기예측 (branch prediction) 의 실패 수

Events for Disks

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz



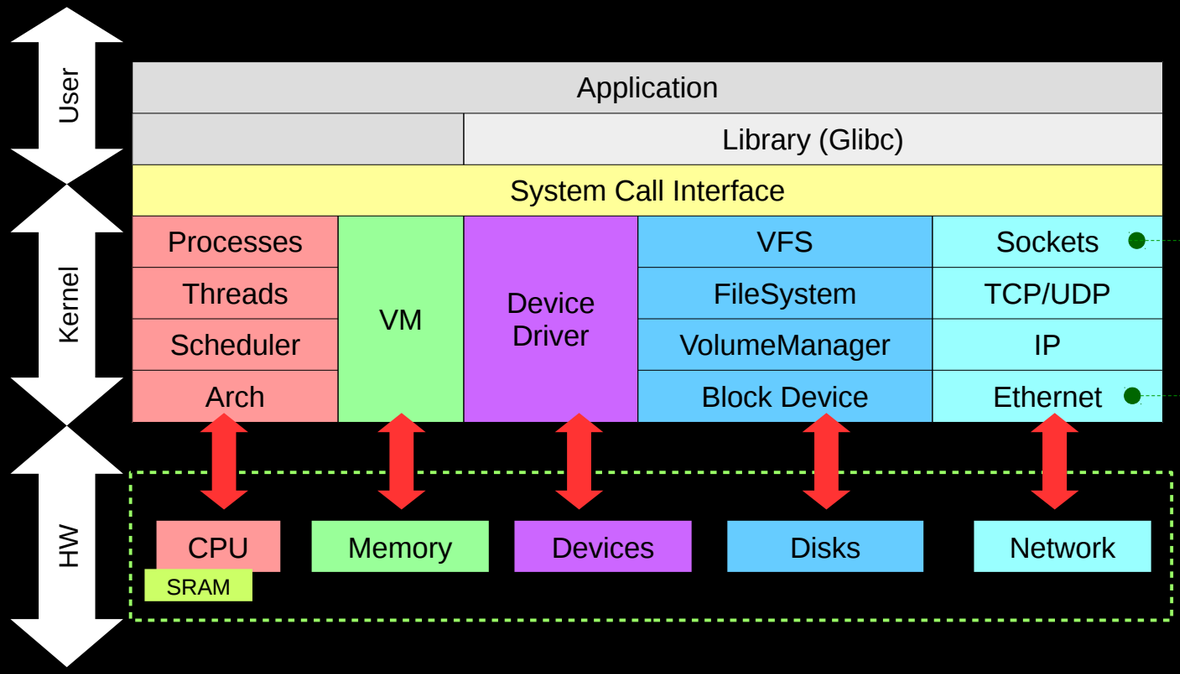
(Tracepoint event)
- block: 블록 I/O 관련 하여 request 가 완료됬는지 혹은 취소됬는지 언제 어디서 됬는지를 확인 할 수 있다 .

(Tracepoint event)
- jbd2: commit flushing 또는 locking 또는 commit 이 언제 시작하는지 등을 확인 가능하다 .

(Tracepoint event)
- ext4: inode 를 free 했는지 write 를 언제 시작했는지 등을 확인 할 수 있다 .

Events for Network

kernel version : 4.2.0-27-generic
perf version : 4.5.rc2.ga7636d9
CPU : Intel® Core(TM) i7-5500U CPU @ 2.40GHz



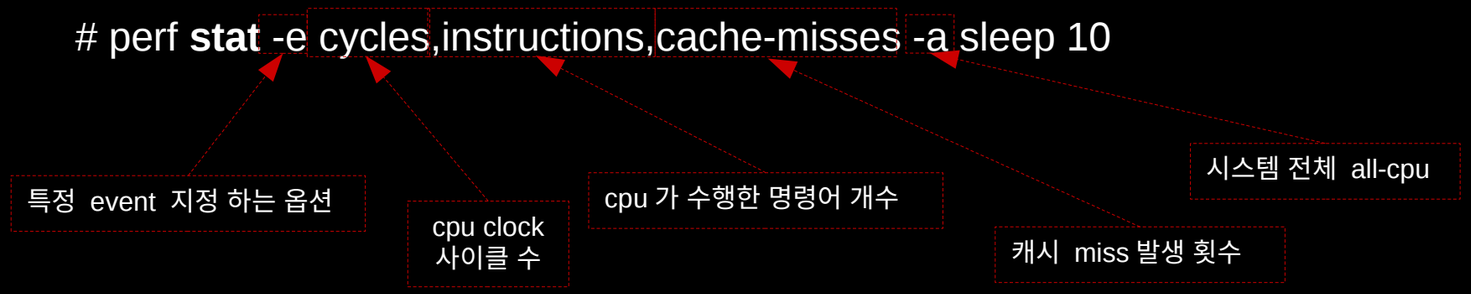
(Tracepoint event)

- net: rx 등을 확인 한다 .
- skb: 소켓버퍼를 얼마나 소비했는지 kfree 했는지 등을 확인 한다 .

(Tracepoint event)

- sock: 소켓 receive 큐가 꽉 찼는지 버퍼한계가 초과 되었는지등을 확인 한다 .

perf **stat** -e cycles,instructions,cache-misses -a sleep 10




```
# perf stat -e cycles,instructions,cache-misses -a sleep 10

 Performance counter stats for 'system wide':

    13,662,665,568      cycles
     8,022,790,410      instructions              #    0.59  insns per cycle
       153,551,391      cache-misses

      10.000686049 seconds time elapsed
```

10 초동안 시스템 전체에 대한 block device I/O event 발생 수 통계보기

```
# perf stat -e 'block:*' -a sleep 10
```

블록 디바이스 관련 각종 I/O event 발생수
- block_bio_backmerge
- block_bio_queue
- block_rq_insert
- block_dirty_buffer
- etc..

< 결과 출력 >

```
# perf stat -e 'block:*' -a sleep 10

 Performance counter stats for 'system wide':

           353      block:block_touch_buffer
           374      block:block_dirty_buffer
             0      block:block_rq_abort
             0      block:block_rq_requeue
            46      block:block_rq_complete
            37      block:block_rq_insert

          ..( 생략 )
      10.000728364 seconds time elapsed
```

10 초동안 모든 CPU 상에서 처리되는 kernel instructions 데이터 샘플링

perf **record** -e cycles:k -ag -- sleep 10

Kernel instructions 만 선별하여 샘플링
('u' 로 user instructions 선별만 가능)

(event):u → 유저레벨 (priv level 3,2,1)
(event):k → 커널레벨 (priv level 0)
(event):h → 가상환경의 하이퍼바이저레벨 이벤트
(event):H → 가상환경의 Host 머신 대상
(event):G → 가상환경의 Guest 머신 대상

< 결과출력 >

```
#perf report --stdio

# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 931  of event 'cycles:k'
# Event count (approx.): 333717883
#
# Overhead  Command          Shared Object         Symbol
# ........  ...............  ....................  ................................................
#
    19.88%  swapper          [kernel.kallsyms]     [k] intel_idle
     2.03%  swapper          [kernel.kallsyms]     [k] menu_select
     1.79%  pulseaudio       [kernel.kallsyms]     [k] policy_zonelist
     1.44%  swapper          [kernel.kallsyms]     [k] native_write_msr_safe
     1.42%  Xorg             [kernel.kallsyms]     [k] sock_poll
     1.27%  swapper          [kernel.kallsyms]     [k] rcu_note_context_switch
     1.18%  chromium-browse  [kernel.kallsyms]     [k] __bpf_prog_run
```

사용자 임의 event (probe:tcp_sendmsg) 에 대해서 4 초간 정보수집

```
# perf record -e probe:tcp_sendmsg -aRg sleep 4
```

< 결과 출력 >

임의의 event:
tcp_sendmsg 커널함수 지정

--raw-samples 옵션으로 열려있는 모든 카운터들로 부터의 sample 기록 모두를 수집한다 .
tracepoint counter 들에 대해서 이미 기본옵션으로 적용되지만 추가했다 .

여기서 sample 관련된 모든 정보를 수집 , 저장한다는 말은 /sys/kernel/debug/tracing/events/*/format 에 나타나는 field 들의 각 값을 모두저장해 언제 / 어떻게 / 왜 event 가 발생했는지 까지 모두 알수가 있다 .

```
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 36  of event 'probe:tcp_sendmsg'
# Event count (approx.): 36
#
# Children      Self  Trace output
# ........  ........  ..................
#
   100.00%   100.00%  (ffffffff81752c40)
             |
             |- 69.44%--0xfdad
             |          entry_SYSCALL_64_fastpath
             |          sys_write
             |          vfs_write
             |          __vfs_write
             |          sock_write_iter
             |          sock_sendmsg
             |          tcp_sendmsg
```

기본적인 reporting view 의 형식

perf **report**

전체 sample 들 중에서 특정 함수에 해당되는 sample 의 퍼센트 비율이다 .

sample 들이 수집된 ELF 이미지 (커널로 부터온 sample 이라면 [kernel.kallsyms] 가 된다 .)

Symbol 이름은 함수명이나 함수주소를 뜻한다 .

```
# Events: 1K cycles
#
# Overhead       Command          Shared Object          Symbol
# ........  ............  .....................  .....................................
#
    28.15%       firefox-bin  libxul.so              [.] 0xd10b45
     4.45%           swapper  [kernel.kallsyms]      [k] mwait_idle_with_hints
     4.26%           swapper  [kernel.kallsyms]      [k] read_hpet
     2.13%       firefox-bin  firefox-bin            [.] 0x1e3d
     1.40%  unity-panel-ser  libglib-2.0.so.0.2800.6 [.] 0x886f1
     [...]
```

sample 들이 수집된 각 프로세스명

동작레벨 (privilege level) 을 뜻한다 .

[.] : 유저레벨
[k] : 커널레벨
[g] : 가상환경의 Guest 커널레벨
[H] : 하이퍼바이저 레벨
[u] : 가상환경의 Guest 유저레벨

정렬기능 소개 (cpu 별)

```
# perf report --sort=cpu

# Events: 354  cycles
#
# Overhead  CPU
# ........  ...
#
    65.85%  1
    34.15%  0
```

CPU 넘버별로 overhead(%) 를 보여준다.

정렬기능 소개 (source line 별 ; -g 옵션 컴파일 후 특정 프로세스 기준 , packing_knapsack.c 예제)

```
# gcc -g packing_knapsack.c -o pack_knapsack
# perf record ./test.sh pack_knapsack
# perf report --sort=srcline

# To display the perf.data header info, please use --header/--
header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 626  of event 'cycles:ppp'
# Event count (approx.): 451409501
#
# Overhead  Source:Line
# ........  ..........................................
#
    27.28%  packing_knapsack.c:38
    24.15%  packing_knapsack.c:37
    14.17%  packing_knapsack.c:39
     8.71%  packing_knapsack.c:34
     5.34%  packing_knapsack.c:56
     4.23%  packing_knapsack.c:58
     3.19%  packing_knapsack.c:57
     2.92%  packing_knapsack.c:60
     2.41%  packing_knapsack.c:26
```

어떤 소스파일의 몇번째 라인이
Overhead 가 몇 % 퍼센트 인지를 보여준다 .

assembly, source code 와 함께 보여주는 view (packing_knapsack.c 예제 재활용)

```
# perf record -g ./test.sh pack_knapsack
# perf annotate -d pack_knapsack –stdio
```

```
 Percent  |      Source code & Disassembly of pack_knapsack for cycles:ppp
------------------------------------------------------------------------------
          :
          :
          :
          :      Disassembly of section .text:
          :
          :      0000000000400826 <get_cond_maxprice>:
          :      get_cond_maxprice():
          :      };
          :
          :      unsigned int limited_wgt;
          :
          :      unsigned int get_cond_maxprice(int wgt, struct jewelry *jewelry)
          :      {
     1.27 :        400826:       push   %rbp
     0.85 :        400827:       mov    %rsp,%rbp
     0.21 :        40082a        mov    %edi,-0x24(%rbp)
     0.00 :        40082d:       mov    %rsi,-0x30(%rbp)
          :              /* Get maximum price based on a specific weight
          :               * following a specific jewelry.
          :               */
          :              int i;
          :              int rest_wgt = wgt - jewelry->wgt;
     0.21 :        400831:       mov    -0x24(%rbp),%eax
     0.42 :        400834:       mov    -0x30(%rbp),%rdx
```

타겟프로그램 pack_knapsack 소스 내의 get_cond_maxprice() 라는 함수를 assembly 그리고 source code 까지 함께 확인이 가능하다

References

- http://studyfoss.egloos.com/5585801
- http://man7.org/linux/man-pages/man2/perf_event_open.2.html
- https://en.wikipedia.org/wiki/Readahead
- https://news.ycombinator.com/item?id=8206692
- http://www.brendangregg.com/perf.html
- https://github.com/brendangregg/perf-tools/blob/master/iosnoop
- http://www.brendangregg.com/blog/2014-07-16/iosnoop-for-linux.html
- https://lwn.net/Articles/608497/

Thank you, Let’s work with perf & uftrace

NAVER DEVIEW 2016

Taeung Song
미래부 KOSS LAB. – Software Engineer
taeung@kosslab.kr