15

2024년 11월 21일 목요일 오전 8:32

전원 관리

전력 사용량 측정

- **전력 사용량 측정 방법**

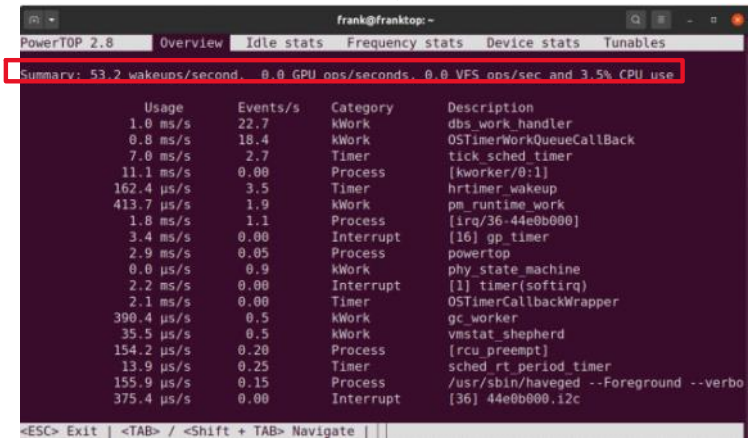
1. **외부에서 측정하는 방식**

   전류계와 전압을 측정하기 위한 전압계 필요 (두 값을 곱해서 와트 값 얻을 수 있음)
   => 전력량을 보여주는 기본 계량기, 데이터 로깅이 결합돼 밀리 초 단위로 값이 변할 때 전력 변화를 확인할 수 있는 계량기, USB 전원 측정기 등 사용

2. **내부에서 측정하는 방식**

   리눅스에 내장된 모니터링 시스템 사용
   -> sysfs 사용, 다양한 소스에서 정보를 모아 한곳에서 볼 수 있는 PowerTOP 프로그램

   (Debian Stretch IoT에서 비글본 블랙에 PowerTop 설치 후 실행 예제)



   CPU 사용량이 3.5%에 불과

클럭 주파수 스케일링

CPU를 낮은 주파수에 실행하면 에너지 절약 가능
코드를 실행할 대 CPU가 소비하는 전력은 다른 것들 중 게이트 누설 전류로 인한 정적 컴포넌트와 게이트를 스위칭할 때 발생하는 동적 컴포넌트의 합

$$P_{cpu} = P_{static} + P_{dyn}$$

동적 전력 컴포넌트는 스위칭되는 논리 게이트의 총 정전 용량, 클럭 주파수, 전압의 제곱에 따라 달라짐

$$P_{dyn} = CfV^2$$

주파수를 낮추면 CPU가 유휴 상태로 돌아가는 데 시간이 더 걸려 실제 전력 소비가 증가
=> 높은 주파수를 사용하는 것이 더 효율적이며, 이 과정을 "유휴 상태로의 전환 (race to idle)" 이라고 함

전력 사용을 줄이려면 CPU 코어의 전압과 주파수를 함께 조정해야 함
이를 위해 SoC에는 동적 전압과 주파수 스케일링(DVFS) 기능이 구현되어 있으며, 최적의 전압과 주파수 조합은 OPP(Operating Performance Point)로 정의
P-state(Performance State)는 이러한 OPP를 나타내며, 주파수와 전압의 조합을 관리
=> P-state 전환은 커널 드라이버와 거버너에 의해 제어

CPUFreq 드라이버

리눅스에는 OPP간 전환을 관리하는 CPUFreq 라는 컴포넌트가 존재
SoC용 패키지에 대한 자원의 일부로, 하나의 OPP에서 다른 OPP로 전환할 수 있게 해주는 드라이버와 언제 전환해야 할지에 대한 정책을 구현하는 거버너 목록으로 이루어져 있음
/sys/devices/system/cpu/cpuN/cpufreq 디렉터리를 통해 CPU 단위로 제어
여기서 N은 CPU의 번호

CPU 주파수 관련 정보:

- **cpuinfo_cur_freq** / 현재 CPU의 주파수 (현재 실행 중인 속도)
- **cpuinfo_max_freq, cpuinfo_min_freq** / CPU가 지원하는 최대, 최소 주파수
- **cpuinfo_transition_latency** / CPU가 한 OPP(Operating Performance Point)에서 다른 OPP로 전환하는 데 걸리는 시간 (단위: 나노초), 값을 모르는 경우: -1로 설정
- **scaling_available_frequencies** / 이 CPU에서 사용할 수 있는 주파수 목록 (지원하는 OPP들)
- **scaling_available_governors** / 이 CPU에서 사용할 수 있는 거버너 목록
- **scaling_governor** / 현재 사용 중인 CPUFreq 거버너
- **scaling_max_freq** / 현재 거버너가 설정한 최대 주파수
- **scaling_min_freq** / 현재 거버너가 설정한 최소 주파수
- **scaling_setspeed** / userspace 거버너 모드일 때, 사용자가 수동으로 CPU 주파수를 설정할 수 있는 파일

```
user@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ ls
affected_cpus     cpuinfo_max_freq  cpuinfo_transition_latency  scaling_available_frequencies  scaling_cur_freq  scaling_governor  scaling_min_freq  stats
cpuinfo_cur_freq  cpuinfo_min_freq  related_cpus                scaling_available_governors    scaling_driver    scaling_max_freq  scaling_setspeed
```

CPU 거버너:
거버너는 CPU 주파수를 조절하는 정책을 설정
각 거버너는 특정 상황에서 CPU의 성능과 전력 소비를 조절

```
user@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_governor
ondemand
```

- **powersave** / CPU가 가능한 가장 낮은 주파수를 사용하여 전력을 절약
- **performance** / CPU가 가능한 가장 높은 주파수를 사용하여 성능을 극대화
- **ondemand** / CPU 사용률을 기준으로 주파수를 자동으로 조정
  CPU 유휴 상태가 20% 미만일 경우 주파수를 최대값으로 설정
  CPU 유휴 상태가 30% 이상일 경우 주파수를 5%씩 낮추어 설정
- **conservative** / ondemand와 유사하지만, 최고 주파수로 바로 전환하는 대신, 5%씩 차례로 주파수를 조정
- **userspace** / 사용자가 직접 주파수를 설정 (주로 scaling_setspeed 파일을 사용)

CPUFreq 사용

```
user@raspberrypi:/sys/devices/system/cpu/cpu0/cpufreq $ cat scaling_available_frequencies
600000 700000 800000 900000 1000000 1100000 1200000 1300000 1400000 1500000 1600000 1700000 1800000
```

(사용가능한 주파수를 보면 이들이 시동 시 사용 중인 OPP인지 확인 가능)

최적의 idle 상태 선택

프로세서가 할 작업이 없으면 halt(중지)를 실행하여 유휴 상태로 돌입
이때, CPU는 유휴 상태에서 전력을 절약하며, 이벤트가 발생하면 유휴 상태를 종료하고 다시 작동을 시작

ACPI(Advanced Configuration and Power Interface) 사양에서는 이러한 유휴 상태를 C-state(CPU Idle State)라고 부름
C-state는 여러 단계로 나뉘며, 각 단계마다 유휴 상태에서 정상 작동 상태로 돌아오는 시간이 다름

깊은 C-state에서는 더 많은 회로가 꺼지고, 일부 상태 값이 지워져서 CPU가 다시 실행될 때 주메모리에서 정보를 다시 로딩해야 할 수 있음
이때 돌아오는 데 시간이 더 걸리고, 이 과정은 비용이 많이 소요

C-state 선택 기준:
- 부하가 높으면 CPU가 지속적으로 활동할 가능성이 크므로 깊은 C-state로 진입하는 것이 비효율적

- 부하가 적더라도 곧 만료될 타이머 이벤트가 있을 경우, 깊은 C-state로 진입하기 전에 해당 이벤트를 고려
- 부하가 없고 타이머 이벤트도 없다면, 더 깊은 C-state로 진입하여 전력을 절약하는 것이 좋음

CPUIdle 드라이버

리눅스에서 가장 적합한 유휴 상태를 선택하는 부분
(Documentation/cpuidle 디렉터리에 정보 찾아볼 수 있음)

CPUIdle은 BSP 일부인 드라이버와 정책을 결정하는 거버너로 구성
CPUFreq와 달리 시동 시 거버너를 변경할 수 없으며, 사용자 공간 거버너에 대한 인터페이스도 없음

(유휴 상태에 대한 정보: /sys/devices/system/cpu/cpu0/cpuidle)

```
user@raspberrypi:/sys/devices/system/cpu/cpu0 $ ls
cache  cpu_capacity  cpufreq  of_node  power  regs  subsystem  topology  uevent
```

?
=> 해당 디렉터리에는 state0~stateN까지 절전 상태에 대한 서브 디렉터리 존재
state0은 가장 가벼운 절전 상태, stateN 가장 깊은 상태

다음과 같은 파일 존재

desc	상태에 관한 간단한 설명
disable	이 파일에 1을 쓰면 상태를 비활성화 할 수 있음
latency	상태를 종료하고 CPU 코어가 정상 작동을 재개하는 데 걸리는 시간
name	상태의 이름
power	유휴 상태일 때 소비되는 전력
time	유휴 상태로 소요된 총 시간
usage	이 상태로 돌입한 횟수

```
$ cd /sys/devices/system/cpu/cpu0/cpuidle
$ grep "" state0/*
state0/desc:ARM WFI
state0/disable:0
state0/latency:1
state0/name:WFI
state0/power:4294967295
state0/residency:1
state0/time:1023898
state0/usage:1426
```

상태 이름: WFI (Wait For Interrupt)
단순한 중지 명령이므로, 지연 시간은 겨우 1마이크로초, 소모되는 전력은 -1로서 전력 예산을 알 수 없다는 뜻

```
$ cd /sys/devices/system/cpu/cpu0/cpuidle
$ grep "" state1/*
state1/desc:mpu_gate
state1/disable:0
state1/latency:130
state1/name:mpu_gate
state1/power:0
state1/residency:300
state1/time:139156260
state1/usage:7560
```

상태 이름:mpu_gate
지연 시간은 130 마이크로초, 이러한 유휴 상태는 CPUIdle 드라이버에 하드코딩돼 있거나 자이 트리에 표시돼 있을 수 있음

CPUIdle은 2개의 거버너를 가지고 있음

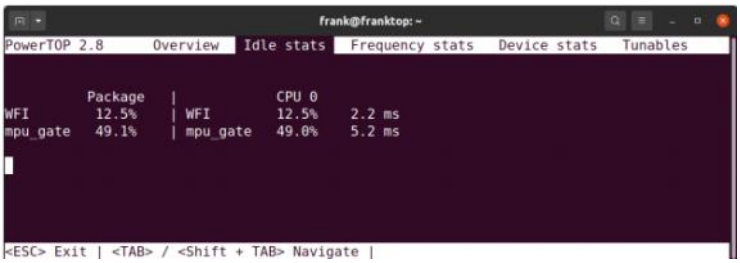
ladder	마지막 유휴 기간에 소요된 시간에 따라 유휴 상태를 한 단계씩 위로 또는 아래로 이동 일반 타이머 틱(tick)에서는 잘 동작하지만 동적 틱에서는 잘 통하지 않음
menu	예상 유휴 시간을 기반으로 유휴 상태를 선택 동적인 틱에서 잘 동작함

사용자와의 상호 작용은 sysfs 파일시스템을 통해 이뤄짐
/sys/devices/system/cpu/cpuidle 디렉터리에는 다음과 같은 파일 존재

current_driver	cpuidle 드라이버의 이름
current_governor_ro	거버너의 이름

유휴 상태의 값은 PowerTOP 의 idle stat 탭에서 확인 가능

(menu 거버너를 이용한 비글본의 예)



=> 시스템이 유휴 상태일 때, 대부분 더 깊은 mpu_gate 유휴 상태로 이동한다는 것을 보여줌

<틱리스 작업>

동적 틱 구성 옵션인 CONFIG_NO_HZ_IDLE은 타이머 처리 루틴 마지막에 타이머 큐를 확인하고 다음 이벤트가 발생할 때
다음 인터럽트를 예약해 불필요하게 CPU를 깨우는 일을 피함으로써 CPU가 더 오랜 시간동안 유휴 상태가 될 수 있도록 함

전력에 민감한 애플리케이션에서는 이 옵션을 활성화한 상태에서 커널을 구성해야 함

주변 기기 전원 끄기

리눅스 커널에서는 런타임 전원 관리 시스템(runtime power management system, pm)이 주변 기기 전력 관리
드라이버가 런타임 pm을 지원한다면, 사용하지 않는 드라이버는 종료하고 다음에 필요할 때 다시 활성화 가능
하드웨어 관리 기능을 구현하는 것은 장치 드라이버에 달려있지만, 일반적으로 클럭 게이팅 이라고 하는 서브 시스템의 클럭을 종료하는 기능과
가능한 경우 코어 회로를 끄는 기능이 포함돼 있음

런타임 전원관리 시스템은 sysfs 인터페이스를 통해 제공
Power라는 서브 디렉터리가 있으며 다음과 같은 파일이 포함

control	이 장치에서 런타임 pm을 사용할지 여부를 사용자 공간에서 결정할 수 있음 auto로 설정하면 런타임 pm이 활성화되지만, on으로 설정하면 장치가 항상 실해중인 상태로 런타임 pm을 사용하지 않음
runtime_enables	런타임 pm이 enabled 되거나 disabled된 상태를 기록하며 control 값이 on이라면 forbidden으로 기록
runtime_status	장치의 현재 상태를 기록 active, suspended, unsupported 값을 가질 수 있음
autosuspend_delay_ms	장치가 일시 중단되기 전의 시간 -1인 경우 영원히 기다린다는 뜻 일부 드라이버는 장치를 정지하는데 상당한 비용이 드는 경우, 이를 구현해 너무 빠른 일시 중지/재개 주기를 방지

(비글본 블랙의 MMC 드라이버)

```
$ cd /sys/devices/platform/ocp/481d8000.mmc/mmc_host/mmc1/
mmc1:0001/power
$ grep "" *
async:enabled
autosuspend_delay_ms:3000
control:auto
runtime_active_kids:0
runtime_active_time:14464
runtime_enabled:enabled
runtime_status:suspended
runtime_suspended_time:121208
runtime_usage:0
```

런타임 pm이 활성화돼 있고, 장치가 현재 일시 중지 상태며, 마지막으로 사용된 후 다시 일시 중지 상태로 변경되기 전에 3000밀리초를 기다린다는 것을 알 수 있음

(한 블록을 읽어보고 값이 변경된 게 맞는지 확인)

```
$ sudo dd if=/dev/mmcblk1p3 of=/dev/null count=1
1+0 records in
1+0 records out
512 bytes copied, 0.00629126 s, 81.4 kB/s
$ grep "" *
async:enabled
autosuspend_delay_ms:3000
control:auto
runtime_active_kids:0
runtime_active_time:17120
runtime_enabled:enabled
runtime_status:active
runtime_suspended_time:178520
runtime_usage:0
```

이제 MMC 드라이버가 활성화돼 보드의 전력 값이 320mW에서 500mW로 증가한 것을 확인 가능
3초후에 다시 값을 확인하면, 다시 일시 중지 상태로 바뀌고 전력이 320mW로 되돌아 오는 것을 확인 가능

시스템을 절전 모드로 전환하기

- **시스템 절전**

한동안 시스템을 다시 사용하지 않는다면 전체 시스템을 절전 모드로 만드는 것

- 일시 중지(suspend)
  => 시스템 메모리를 제외한 모든 기능을 종료, 시스템이 깨어나면 메모리는 이전 상태를 모두 유지하므로 노트북은 몇초 내에 동작 가능한 상태가 됨
  약간의 전력을 소비해야 함

- 최대 절전 모드(hibernate)
  => 메모리의 내용이 하드 드라이버에 저장
  시스템은 전력을 전혀 소비하지 않으므로 무기한으로 이 상태를 유지할 수 있으나, 시스템이 깨어나면 디스크에서 메모리를 복원하는데 시간이 걸림

<전력 상태>

ACPI 스펙에서는 절전 상태를 S-state(System Sleep State)라고 부름
총 4개의 절전 상태를 지원

freeze(s0)	사용자 공간에서 모든 활동을 중지하지만 CPU와 메모리는 정상적으로 동작
stanby(s1)	Freeze와 비슷하지만 부팅 CPU를 제외한 모든 CPU를 오프라인으로 전환
mem(s3)	시스템의 전원을 끄고 메모리를 self-refresh 모드로 설정 Suspend to RAM으로 알려져 있음
Disk(s4)	메모리를 하드디스크에 저장하고 전원을 끔 suspend to disk로 알려져 있음

```
# cat /sys/power/state
freeze standby mem disk
```

```
user@raspberrypi:/sys/power $ cat state
user@raspberrypi:/sys/power $
```

?

<웨이크업 이벤트>

장치를 일시 중지하기 전에 다시 깨우는 방법이 반드시 있어야 하며, 커널이 도움을 줌
적어도 하나의 웨이크업 소스가 없으면 시스템은 다음 메시지를 보낸 후 일시 중지를 거절할 것임

```
No sources enabled to wake-up! Sleep abort.
```

웨이크업 이벤트는 sysfs를 통해 제어
sys/device 내 각 장치에는 서브디렉터리인 power가 있으며, 해당 디렉터리에는 다음 문자열 중 하나가 작성된 wakeup이란 이름의 파일 존재

Enabled	웨이크업 이벤트 생성
Disabled	웨이크업 이벤트를 생성하지 않음
(비어 있음)	웨이크업 이벤트를 생성할 수 없음

Enabled나 disabled 값이 적힌 wakeup 파일을 가진 장치를 검색하면 됨

```
[ 1646.348264] PM: Wakeup source UART
[ 1646.368482] PM: noirq resume of devices complete after
19.963 msecs
[ 1646.372482] PM: early resume of devices complete after 3.192
msecs
[ 1646.795109] net eth0: initializing cpsw version 1.12 (0)
[ 1646.798229] net eth0: phy found : id is : 0x7c0f1
[ 1646.798447] libphy: PHY 4a101000.mdio:01 not found
[ 1646.798469] net eth0: phy 4a101000.mdio:01 not found on
```

<실시간 클럭의 시간 지정 웨이크업>

대부분의 시스템에는 향후 24시간까지 알람 인터럽트를 생성할 수 있는 실시간 클럭(RTC,Real-Time clock)이 존재

(/sys/class/rtC/rtc0 디렉터리에 wakealarm 파일로 존재)

```
user@raspberrypi:/sys/class/rtc $ ls
user@raspberrypi:/sys/class/rtc $
```

Wakealarm 파일에 숫자를 작성하면 해당 초 이후에 알람이 생성
또한, rtc에서 wakeup 이벤트를 활성화하면 RTC는 일시 중지된 장치를 다시 시작할 수 있음

(rtcwake 명령은 RTC가 5초 후에 시스템을 깨우면서 시스템을 standby 모드 상태로 만들음)

```
[  187.500941] pm33xx pm33xx: PM: Successfully put all
powerdomains to target state
[  187.500941] PM: Wakeup source RTC Alarm
[  187.529729] net eth0: initializing cpsw version 1.12 (0)
[  187.605061] SMSC LAN8710/LAN8720 4a101000.mdio:00: attached
PHY driver [SMSC LAN8710/LAN8720] (mii_bus:phy_addr=4a101000.
mdio:00, irq=POLL)
[  187.731543] OOM killer enabled.
[  187.731563] Restarting tasks ... done.
[  187.756896] PM: suspend exit

$ sudo su -
# rtcwake -d /dev/rtc0 -m standby -s 5
  rtcwake: assuming RTC uses UTC ...
  rtcwake: wakeup from "standby" using /dev/rtc0 at Tue Dec 1
19:34:10 2020
[  187.345129] PM: suspend entry (shallow)
[  187.345148] PM: Syncing filesystems ... done.
[  187.346754] Freezing user space processes ... (elapsed 0.003
seconds) done.
[  187.350688] OOM killer disabled.
[  187.350789] Freezing remaining freezable tasks ... (elapsed
0.001 seconds) done.
[  187.352361] Suspending console(s) (use no_console_suspend to
debug)
[  187.500906] Disabling non-boot CPUs ...
```