ups 설정

---

0. NUT 설치 과정

Ubuntu 24.04 환경에서 NUT(Network UPS Tools)를 설치합니다.

```
sudo apt update
sudo apt install nut nut-client nut-server -y
```

설치 후 설정 파일 경로:

```
/etc/nut/
```

구성도

---

1. Ubuntu NUT 설정

UPS를 USB로 연결한 Ubuntu 서버에서 NUT을 이용하여 상태 모니터링 및 자동 종료를 설정합니다.

/etc/nut/ups.conf

```
[ups]
  driver    = usbhid-ups
  port      = auto
  vendorid  = 051d
  productid = 0003
  serial    = AS2422151019
  desc      = "APC SMC1500IC"
```

/etc/nut/nut.conf

```
MODE=standalone
```

/etc/nut/upsd.users

```
[monuser]
  password = secret
```

```
upsmon master
```

/etc/nut/upsmon.conf

```
MONITOR ups@localhost 1 monuser secret master
SHUTDOWNCMD "/sbin/shutdown -h now"
POWERDOWNFLAG /etc/killpower
FINALDELAY 5
```

서비스 등록 및 실행

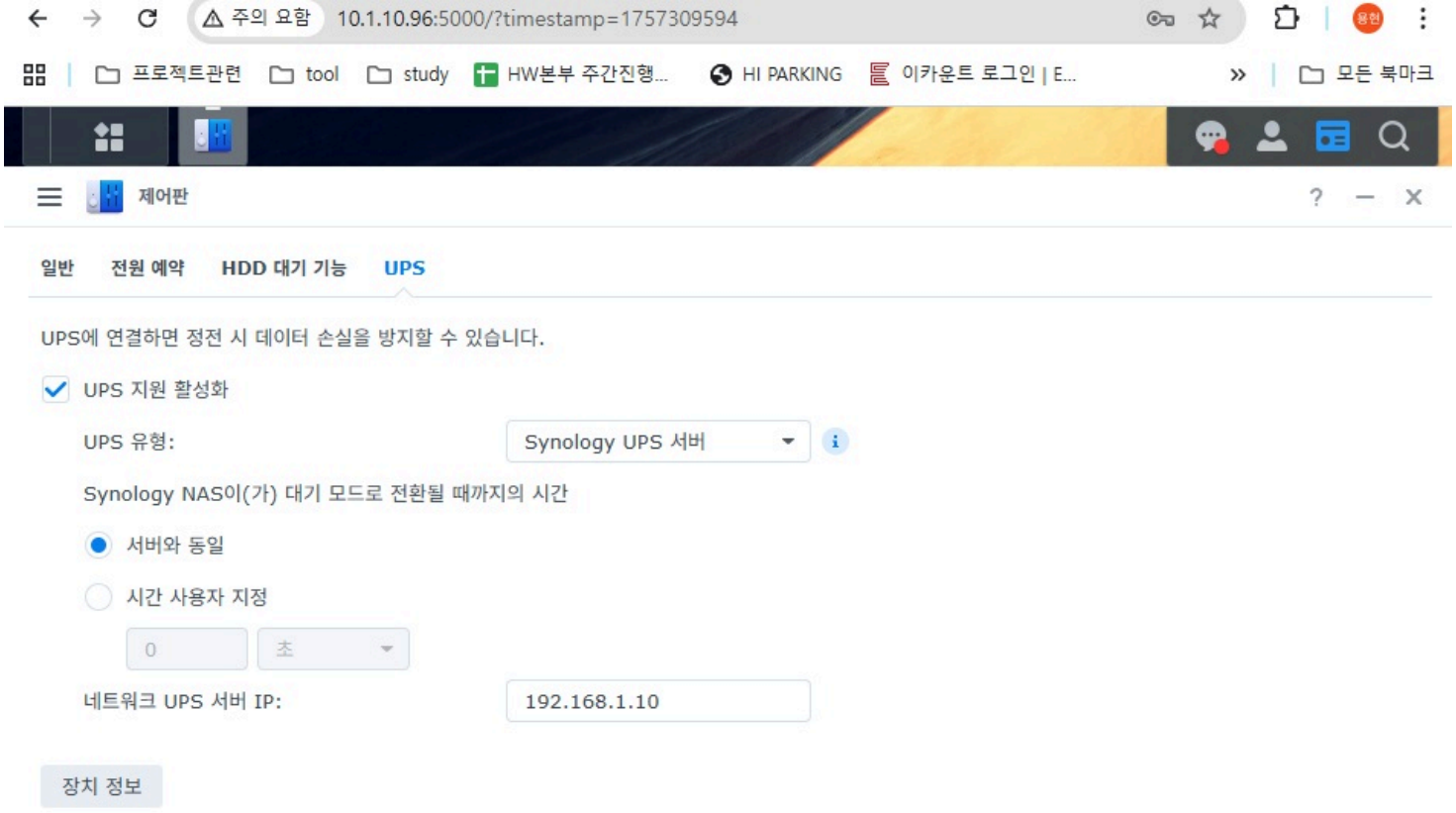
```
sudo systemctl enable nut-driver@ups.service nut-server nut-monitor
sudo systemctl start nut-driver@ups.service nut-server nut-monitor
```

---

2. Synology NAS 설정

Ubuntu 서버를 UPS 서버로 두고 Synology NAS와 연동해 전원 장애 시 동기화 종료를 수행합니다.

- **경로**: DSM → 제어판 → 하드웨어 및 전원 → UPS
- **UPS 유형**: Synology UPS 서버
- **서버 주소**: Ubuntu 서버 IP (예: `192.168.1.10`)
- **포트**: 3493 (기본, 방화벽 허용 필요)
- **설정**: UPS에서 일정 시간 이상 배터리 동작 시 NAS 자동 종료



3. UPS 상태 확인

UPS 상태 확인 명령:

```
upsc ups@localhost
```

정상 출력 예시:

```
battery.charge: 100
battery.runtime: 2966
device.model: Smart-UPS_1500
ups.status: OL
```

```
koast-user@koast:~$ upsc ups@localhost
Init SSL without certificate database
battery.charge: 100
battery.charge.low: 10
battery.charge.warning: 50
battery.runtime: 2966
battery.runtime.low: 150
battery.type: PbAc
battery.voltage: 26.3
battery.voltage.nominal: 24.0
device.mfr: American Power Conversion
device.model: Smart-UPS_1500
device.serial: AS2422151019
device.type: ups
driver.debug: 0
driver.flag.allow_killpower: 0
driver.name: usbhid-ups
driver.parameter.pollfreq: 30
driver.parameter.pollinterval: 2
driver.parameter.port: /dev/hidraw4
driver.parameter.productid: 0003
driver.parameter.serial: AS2422151019
driver.parameter.synchronous: auto
driver.parameter.vendorid: 051d
driver.state: quiet
driver.version: 2.8.1
driver.version.data: APC HID 0.100
driver.version.internal: 0.52
driver.version.usb: libusb-1.0.27 (API: 0x100010a)
ups.beeper.status: enabled
ups.delay.shutdown: 20
ups.firmware: UPS 02.0 / ID=1061
ups.mfr: American Power Conversion
ups.mfr.date: 2024/05/28
ups.model: Smart-UPS_1500
ups.productid: 0003
ups.serial: AS2422151019
ups.status: OL
ups.timer.reboot: -1
ups.timer.shutdown: -1
ups.vendorid: 051d
```

---

4. 전원 시나리오

원양어선 등 전원 불안정 환경에서 고려해야 할 동작 시나리오:

1. **정상 상태**
   - UPS는 상용 전원(OL 상태)에서 PC와 NAS에 전력 공급.
   - 모든 장비 정상 가동.
2. **발전기 교체 / 전원 장애 발생 (OB 상태)**
   - UPS 배터리 모드 진입.
   - NUT이 전원 장애 이벤트 감지 후 일정 시간 이상 지속되면 shutdown 로직 발동.
3. **PC 자동 종료**
   - NUT의 `upsmon`이 먼저 Ubuntu 서버(PC)에 shutdown 명령 실행.
   - 종료 직후 `/etc/killpower` 플래그 작성.
4. **NAS 자동 종료**
   - Synology NAS는 Ubuntu 서버의 NUT 서버 상태를 주기적으로 확인.
   - 동일 조건 충족 시 NAS도 안전하게 shutdown.
5. **UPS 방전 방지**

- PC와 NAS가 모두 꺼진 후 UPS는 최소 부하 상태.
- UPS 배터리 과방전 방지 → 전원 복구 대기.

6. **전원 복구**
   - 발전기/상용 전원 복구 시 UPS가 OL 상태로 전환.
   - PC와 NAS는 BIOS/DSM 설정에 따라 자동 켜짐.

---

5. 운영 지침

- **BIOS 설정**: PC BIOS에서 "AC Power Loss → Power On" 활성화.
- **Synology 설정**: DSM → 전원 복구 설정 → "전원 복구 시 자동 켜짐" 활성화.
- **테스트 권장**: 실제 발전기 전환 테스트로 PC/NAS 순차 종료 및 자동 재가동 확인.

---

6. 정전 2분 이상 지속 시 자동 종료 (upssched 활용)

일반적인 `upsmon.conf` 만으로는 단순 조건(BATTERYLEVEL, MINUTES) 기반 종료만 가능합니다.
발전기 교체 등으로 순간 정전이 자주 발생하는 환경에서는 **정전이 2분 이상 지속될 때만 PC를 종료**하는 방식이 적합합니다.
이를 위해 `upssched` 를 추가 설정합니다.

6.1 upsmon.conf 수정

`/etc/nut/upsmon.conf` 에 아래 내용을 추가합니다:

```
# 이벤트 발생 시 upssched 실행
NOTIFYCMD /usr/sbin/upssched
NOTIFYFLAG ONBATT  SYSLOG+EXEC
NOTIFYFLAG ONLINE  SYSLOG+EXEC
```

6.2 upssched.conf 생성

/etc/nut/upssched.conf:

```
CMDSCRIPT /etc/nut/upssched-cmd
PIPEFN /run/nut/upssched.pipe
LOCKFN /run/nut/upssched.lock



# UPS가 배터리 모드로 전환되면 120초 타이머 시작
AT ONBATT * START-TIMER onbatt 120

# 전원이 복구되면 타이머 취소
AT ONLINE * CANCEL-TIMER onbatt

# 120초 동안 전원이 복구되지 않으면 shutdown 실행
```

```
AT TIMER onbatt EXECUTE shutdown
```

6.3 커맨드 스크립트 작성

/etc/nut/upssched-cmd:

```
#!/bin/bash
case $1 in
    shutdown)
        logger -t upssched "UPS on battery >120s, shutting down system"
        /sbin/shutdown -h now
        ;;
    *)
        logger -t upssched "Unrecognized command: $1"
        ;;
esac
```

권한 부여:

```
sudo chmod +x /etc/nut/upssched-cmd
```

6.4 서비스 재시작

설정 반영을 위해 nut-monitor만 재시작합니다.

```
sudo systemctl restart nut-monitor
```

7. UPS 종료 시 애플리케이션 SIGTERM 처리

UPS 전원 장애가 2분 이상 지속되면 `nut-monitor`가 시스템에 셧다운 명령을 내립니다.
리눅스는 이 과정에서 **모든 프로세스에 SIGTERM (15)** 신호를 먼저 보내고, 일정 시간이 지나면 **SIGKILL (9)**로 강제 종료합니다.

- **SIGTERM**: 정상 종료 요청 신호. 프로그램이 이 신호를 처리하면 데이터 flush, 세션 정리, 로그 저장, DB 연결 해제 등의 작업을 안전하게 수행할 수 있음.
- **SIGKILL**: SIGTERM을 처리하지 못하면 강제 종료. 데이터 손실 발생 가능.

따라서, UPS 기반 자동 셧다운 환경에서는 **프로그램이 SIGTERM을 처리하도록 구현**해야 합니다.

7.1 Python 예제

```
import signal, sys, time, threading

shutdown = threading.Event()

def cleanup():
    print("cleanup... done")
```

```python
def handle_sigterm(signum, frame):
    print(f"received signal {signum}, shutting down...")
    shutdown.set()

signal.signal(signal.SIGTERM, handle_sigterm)
signal.signal(signal.SIGINT, handle_sigterm)  # Ctrl+C 도 동일 처리

try:
    while not shutdown.is_set():
        time.sleep(0.5)  # 메인 루프
finally:
    cleanup()
    sys.exit(0)
```

7.2 Java 예제

```java
public class App {
    private static volatile boolean running = true;

    public static void main(String[] args) throws Exception {
        Thread worker = new Thread(() -> {
            while (running) {
                try { Thread.sleep(500); } catch (InterruptedException ignored) {}
                // TODO: 주기 작업
            }
        });
        worker.start();

        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("Shutdown hook: cleanup start");
            running = false;
            try { worker.join(5000); } catch (InterruptedException ignored) {}
            // TODO: DB/파일/네트워크 정리
            System.out.println("Shutdown hook: cleanup done");
        }));

        worker.join();
    }
}
```

7.3 Node.js 예제

```javascript
const http = require('http');

let shuttingDown = false;
const server = http.createServer((req, res) => {
  if (shuttingDown) {
    res.writeHead(503);
    return res.end('Server is shutting down');
```

```
  }
  res.end('ok');
});

server.listen(3000, () => console.log('listening on 3000'));

async function cleanup() {
  console.log('cleanup start');
  await new Promise(r => setTimeout(r, 500));
  console.log('cleanup done');
}

function gracefulExit() {
  if (shuttingDown) return;
  shuttingDown = true;
  console.log('received SIGTERM, shutting down...');
  server.close(async (err) => {
    if (err) console.error('server close error:', err);
    await cleanup();
    process.exit(0);
  });

  setTimeout(() => {
    console.error('force exit after timeout');
    process.exit(1);
  }, 10000).unref();
}

process.on('SIGTERM', gracefulExit);
process.on('SIGINT', gracefulExit);
```