13

2024년 11월 18일 월요일 오전 10:05

시스템 구동: init 프로그램

커널 구동 이후

- **커널 부트스트랩**
  initramfs나 커널 명령줄에서 root = 로 지정한 파일시스템 중 하나인 루트 파일시스템을 찾아 프로그램을 실행

- 이 프로그램은 initramfs에서는 /init을, 일반적인 파일시스템에서는 /sbin/init 실행
- Init 프로그램은 root 권한을 갖고 있으며, 첫번째로 실행하는 프로세스이므로 해당 프로그램의 프로세스 아이디(PID)는 1이 됨

  (pstree 명령어로 Init 프로세스가 모든 프로세스의 부모임을 확인)

```
# pstree -gn
init(1)-+-syslogd(63)
        |-klogd(66)
        |-dropbear(99)
        `-sh(100)---pstree(109)
```

- **Init 프로그램의 주요역할**

- **부팅 시 데몬 프로그램 시작:**
  시스템이 동작하는 데 필요한 각종 데몬을 실행하고, 시스템 매개변수를 설정
- **로그인 셸 실행:**
  터미널에서 로그인 셸을 실행할 수 있는 데몬(예: getty)을 실행
- **고아 프로세스 처리:**
  부모 프로세스가 종료되면 자식 프로세스가 고아가 되는데, 이를 init가 채택하여 처리
- **좀비 프로세스 방지:**
  자식 프로세스가 종료되면 SIGCHLD 시그널을 처리하고 종료된 자식의 리턴값을 수집하여 좀비 프로세스가 되지 않도록 함
- **데몬 재시작:**
  종료된 데몬이 있을 경우 이를 재시작
- **시스템 종료 처리:**
  시스템 종료 시 필요한 작업을 수행

Init 프로그램의 소개

임베디드 장치에서 가장 많이 접하게 될 세가지 init 프로그램

메트릭	BusyBox init	System V init	systemd
복잡도	낮음	중간	높음
부팅 속도	빠름	느림	중간
필요 셸	Ash	Ash나 bash	없음
실행 파일 수	1	4	50*
Libc	모두 사용 가능	모두 사용 가능	glibc
크기(MB)	< 0.1 *	0.1	34 **

BusyBox init

BusyBox는 /etc/inittab 구성 파일을 사용하는 매우 작은 init 프로그램을 갖고 있으며, 해당 파일에 부팅 시 시작할 프로그램과 종료 시 멈출 프로그램에 대한 규칙을 정의
실제 작업은 대부분 /etc/init.d 디렉터리에 있는 셸 스크립트에 의해 수행

init은 /etc/inittab을 읽으면서 시작

- **형식**
  <id>::<action>:<program>

- **각 매개변수의 역할**

  - **Id:** 명령의 제어 터미널
  - **Action:** 아래에 나열되는 내용처럼 명령어를 실행하기 위한 조건을 포함
  - **Program:** 실행할 프로그램

- **Action의 내용**

  - **sysinit:** init이 시작될 때, 다른 유형의 작업보다 가장 먼저 이 프로그램을 실행
  - **respawn:** 프로그램을 실행하고 종료되는 경우 다시 시작, 대부분 프로그램을 데몬으로 실행할 때 사용
  - **askfirst:** respawn 과 동일한 역할을 하지만, 먼저 please press Enter to activate this console 이라는 메시지를 콘솔에 출력. 이후 Enter를 눌러야만 프로그램이 실행
    이는 사용자 이름이나 암호를 묻지 않고 터미널에서 대화형 셸을 시작할 때 사용
  - **Once:** 프로그램을 실행하고 종료됐으면 다시 시작 안함
  - **wait:** 프로그램을 실행하고 완료될 때까지 기다림
  - **Restart:** init이 inittab 파일을 다시 읽어와야 하는 SIGHUP 신호를 받으면 프로그램을 실행
  - **Ctrlaltdel:** init 이 SIGINT 시그널을 받을 때 프로그램을 실행, 해당 시그널은 보통 콘솔에서 Ctrl+Alt+Del 눌렀을 때 전송
  - **Shutdown:** init이 종료될 때 프로그램 실행

<Buildroot init 스크립트>

Buildroot의 /etc/init.d에는 rcS라는 2개의 스크립트가 존재

- **부팅 시**: /etc/init.d/에 있는 Sxx 형식의 **start 스크립트**들이 숫자 순서대로 실행
- **종료 시**: /etc/init.d/에 있는 Kxx 형식의 **kill 스크립트**들이 숫자 순서대로 실행

System V init

1980년대 중반, 유닉스 시스템 V의 프로그램에서 영감을 얻음

- **장점**

  - 부트 스크립트가 잘 알려진 모듈 형식으로 작성되므로 빌드 시 또는 런타임에 새 패키지를 추가하는 것이 쉬움
  - System V init 은 런레벨이라는 개념을 갖고 있어, 하나의 런레벨에서 다른 런레벨로 전환될 때 한 번에 여러 프로그램을 시작하거나 중지할 수 있음

System V init 에는 s를 포함해 0부터 6까지 총 8개의 런 레벨이 존재

S	시동 작업 실행
0	시스템 정지
1~5	일반 작업을 위해 사용
6	시스템 재시작

데스크톱 리눅스 배포판에서는 일반적으로 다음과 같이 할당

1	단일 사용자
2	네트워크 구성 요소를 사용하지 않는 다중 사용자
3	네트워크 구성 요소를 사용하는 다중 사용자
4	사용하지 않음
5	그래픽 로그인을 하는 다중 사용자

Init 프로그램은 다음과 같이 /etc/inittab의 initdefault 행에 작성된 대로 기본 런 레벨 시작

```
id:3:initdefault:
```

런타임 시 init에 메시지 보내는 telinit [runlevel] 명령을 사용하면 런레벨을 변경 가능
runlevel 명령을 사용하면 현재 런레벨과 이전 런레벨을 확인할 수 있음

```
# runlevel
N 5
# telinit 3
```

→ N 5로 이전 런레벨 값이 없는데, 이는 런레벨이 부팅한 후 변경되지 않았기에 부팅 시와 현재 런레벨이 모두 5이기 때문

런레벨(Runlevel) 란?

시스템의 동작 상태를 정의하는 번호 (예: 0, 1, 2, 3, 4, 5, 6)
각 런레벨은 시스템의 실행 상태에 따라 다름

```
# runlevel
N 5
# telinit 3
```

→ N 5로 이전 런레벨 값이 없는데, 이는 런레벨이 부팅한 후 변경되지 않았기에 부팅 시와 현재 런레벨이 모두 5이기 때문

```
INIT: Switching to runlevel: 3
# runlevel
5 3
```

→ 런레벨이 5에서 3으로 변경이 일어났음을 확인할 수 있음

Halt(시스템을 종료하는 명령)와 reboot 명령어는 각각 0과 6의 런레벨로 전환
(단일 사용자 모드로 런레벨을 변경하고 싶다면 커널 명령줄의 마지막에 1을 붙이면 됨)

```
console=ttyAMA0 root=/dev/mmcblk1p2 1
```

Sysyrm V의 init은 Buildroot와 Yocto 프로젝트에서 옵션으로 사용 가능
두경우 모두, init 스크립트는 bash 의 특성을 갖고 있지 않아서 BusyBox ash 셸로 작업

<inittab>

Init 프로그램은 /etc/inittab을 읽으면서 시작

inittab의 각 행의 형식

id:runlevels:action:process

- **id:** 최대 4자의 고유 식별자
- **runlevels:** 이 항목이 실행돼야 할 런레벨 (BusyBox inittab의 경우 이 항목이 빈칸)
- **action:** 시스템이 해당 항목을 처리할 방식 (예: respawn, wait, initdefault 등)
- **process:** 실행할 명령

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:5:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin
# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is, single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin
AMA0:12345:respawn:/sbin/getty 115200 ttyAMA0
# /sbin/getty invocations for the runlevels
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:


 # <id>:<runlevels>:<action>:<process>
 #

1:2345:respawn:/sbin/getty 38400 tty1
```

1. **id:5**
   : 시스템의 기본 런레벨을 5로 설정, 즉 시스템이 부팅할 때 런레벨 5로 시작

2. **si::sysinit:/etc/init.d/rcs**:
   부팅 시 /etc/init.d/rcs 스크립트를 실행

3. **10:0:wait:/etc/init.d/rc 0**:
   런레벨이 변경될 때마다 /etc/init.d/rc 스크립트를 실행하여 start 및 kill 스크립트를 처리

4. **런레벨 1~5에 대한 getty 실행**:
   각 런레벨(1~5)마다 /dev/ttyAMA0에서 로그인 프롬프트를 제공하는 getty 데몬을 실행하여 사용자가 로그인 후 대화형 셸을 사용할 수 있음

   => inittab 파일은 시스템 부팅과 런레벨 변화에 따라 필요한 스크립트와 로그인 서비스를 설정하는 역할

```
AMA0:12345:respawn:/sbin/getty 115200 ttyAMA0
```

1. **ttyAMA0 (시리얼 콘솔)**
   ARM 기반 QEMU에서 Versatile 보드의 시리얼 콘솔로 사용됨
   시리얼 포트를 통해 시스템과 통신

2. **tty1 (가상 콘솔)**
   가상 콘솔로 사용되며, getty 프로세스가 실행됨
   런레벨 2~5에서 활성화
   커널 설정에 따라 그래픽 콘솔로도 사용할 수 있음

3. **가상 터미널 (Virtual Consoles)**
   tty1 ~ tty6: 6개의 getty 프로세스가 실행되며, Ctrl + Alt + F1~F6 키로 전환 가능
   tty7: 그래픽 화면 전용으로 예약됨
   (우분투(Ubuntu)와 아치 리눅스(Arch Linux)는 tty1을 그래픽 콘솔로 사용)

4. **sysinit과 /etc/init.d/rcS**
   sysinit 항목이 /etc/init.d/rcS 스크립트를 실행하여 시스템 초기화 작업을 수행
   런레벨 s보다 더 많은 역할을 담당

<init.d 스크립트>

- **런레벨 변경 시 처리 흐름**

  /etc/init.d/rc 스크립트가 런레벨 변경 작업을 처리
  런레벨마다 rc<runlevel>.d라는 디렉터리가 존재
  각 디렉터리에는 start 및 stop 작업을 수행할 스크립트들이 있음

```
# ls -d /etc/rc*
/etc/rc0.d /etc/rc2.d /etc/rc4.d /etc/rc6.d
/etc/rc1.d /etc/rc3.d /etc/rc5.d /etc/rcS.d
```

- **스크립트 이름**

- **Sxx:** 시작 시 실행될 스크립트 (start)
- **Kxx**: 종료 시 실행될 스크립트 (kill)
  xx: 실행 순서를 나타내는 숫자 (작을수록 먼저 실행)

```
# ls /etc/rc5.d
S01networking S20hwclock.sh S99rmnologin.sh S99stop-bootlogd
S15mountnfs.sh S20syslog
```

- **/etc/rc5.d 디렉터리:**
  S01networking: 네트워크를 시작하는 스크립트
  K20hwclock.sh: 하드웨어 시계를 종료하는 스크립트
  S99rmnologin.sh: 로그인 제한을 해제하는 스크립트
  각 스크립트는 숫자 순서대로 실행

- **실행 흐름**

- 런레벨 변경 시, K로 시작하는 스크립트를 stop 매개변수로 실행한 뒤,
  S로 시작하는 스크립트를 start 매개변수로 실행

<새로운 데몬 추가>

- **simpleserver 프로그램 실행**

- simpleserver는 백그라운드에서 실행되는 데몬
- 이를 관리하는 스크립트를 /etc/init.d/simpleserver에 작성

```
#! /bin/sh

case "$1" in
    start)
        echo "Starting simpelserver"
        start-stop-daemon -S -n simpleserver -a /usr/bin/simpleserver
        ;;
    stop)
        echo "Stopping simpleserver"
        start-stop-daemon -K -n simpleserver
        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
esac

exit 0
```

- **start-stop-daemon**

  목적: 데몬을 시작하고 종료하는 작업을 간편하게 해주는 유틸리티

  -S: 데몬 시작 (한 번에 하나만 실행)
  -K: 데몬 종료 (SIGTERM 신호)

- **자동 실행 설정**

  simpleserver를 원하는 런레벨에서 실행되도록 링크 추가

  Ex) 기본 런레벨인 5에서 실행하도록 설정

```
# cd /etc/init.d/rc5.d
# ln -s ../init.d/simpleserver S99simpleserver
```

- S99simpleserver는 런레벨 5에서 시작되는 프로그램 중 마지막 순서라는 의미

- **종료 시 설정**

  종료 시, 데몬을 종료하려면 K로 시작하는 심볼릭 링크를 추가

  (데몬 종료를 위해 런레벨 0, 6에 링크 추가)

```
# cd /etc/init.d/rc0.d
# ln -s ../init.d/simpleserver K01simpleserver
# cd /etc/init.d/rc6.d
# ln -s ../init.d/simpleserver K01simpleserver
```

  런레벨과 스크립트 순서를 우회하여 테스트할 수 있음

<서비스 시작과 종료>

- /etc/init.d의 스크립트를 직접 호출 가능

```
# /etc/init.d/syslog --help
Usage: syslog { start | stop | restart }

# /etc/init.d/syslog stop
Stopping syslogd/klogd: stopped syslogd (pid 198)
stopped klogd (pid 201)

done

# /etc/init.d/syslog start
Starting syslogd/klogd: done
```

스크립트 필수 함수
- 모든 서비스 스크립트에는 start, stop, help 함수가 필요
- 일부 스크립트에는 status 함수도 있어, 서비스가 실행 중인지 확인 가능

서비스 관리 (service 명령어)
- System V init을 사용하는 배포판에서는 service 명령어로 서비스를 시작하고 중지
- 서비스 스크립트를 직접 호출하는 방법은 숨겨져 있음

Systemd

Systemd는 시스템 및 서비스 관리자
기능: 시스템 초기화, 서비스 관리, 장치 관리(udev), 로깅 등 다양한 기능을 포함

Systemd가 System V init보다 더 좋은 이유

- **간단하고 논리적인 구성:**
  System V init의 복잡한 셸 스크립트 대신, 단순하고 잘 정의된 유닛 구성 파일을 사용
- **명확한 서비스 의존성:**
  서비스 간의 의존성을 명시적으로 정의하여, 실행 순서를 설정하는 두 자리 코드가 필요 없음
- **보안 강화:**
  각 서비스에 대한 사용 권한과 리소스 제한 설정이 용이
- **서비스 모니터링 및 자동 재시작:**
  서비스를 모니터링하고, 필요 시 자동으로 재시작할 수 있음
- **병렬 서비스 실행:**
  여러 서비스를 병렬적으로 실행하여, 부팅 시간이 단축됨

임베디드 시스템에서의 장점
- System V init보다 더 효율적이고 관리하기 쉬운 시스템을 제공
- 임베디드 시스템에서도 복잡한 장치들을 관리하기에 적합

<Yocto 프로젝트와 Buildroot에서 system 빌드하기>

- Yocto 프로젝트의 기본 init 데몬은 System V
- systemd를 선택하려면 conf/local.conf 파일에 다음 행 추가

```
INIT_MANAGER = "systemd"
```

- Buildroot의 경우, BusyBox init 을 기본으로 사용

- Buildroot 메뉴에서 Init 시스템을 systemd로 변경
  System Configuration > Init System에서 systemd를 선택

- glibc 사용
  systemd는 uClibc-ng나 musl을 지원하지 않으므로 glibc를 사용하도록 설정

- 커널 버전과 구성
  systemd를 사용할 때 요구하는 커널 버전과 구성을 확인하고 맞추기

- systemd의 README 파일
  systemd 소스 코드의 상위 README 파일을 참고하여 라이브러리 및 커널 요구사항을 확인

<타깃, 서비스, 유닛의 소개>

유닛	타깃, 서비스, 기타 여러 가지 사항을 설명하는 구성 파일로, 속성과 값이 포함된 텍스트 파일
서비스	system V의 init 서비스와 비슷하게 시작하거나 중지할 수 있는 데몬
타깃	system V의 init 런레벨과 비슷한 개념이지만, 그보다 좀 더 일반적인 서비스 그룹으로, 부팅 시 시작되는 서비스들이 그룹인 기본 타깃 존재

<유닛>

설정의 기본 항목은 유닛 파일로, 다음 세 곳에 저장

- /etc/systemd/system: 로컬 설정
- /run/systemd/system: 런타임 설정
- /lib/systemd/system: 배포판 설정

유닛 검색 순서:

systemd는 위 디렉터리 순서대로 유닛 파일을 검색해야 함
일치하는 유닛을 찾으면, /etc/systemd/system에 동일한 이름의 유닛을 배치하여 배포판 설정 유닛의 재정의 할 수 있도록 함

유닛 비활성화:
비어 있거나 /dev/null에 연결된 로컬 파일 생성

유닛 파일의 기본 구조:
모든 유닛 파일은 [Unit] 섹션으로 시작하며, 이 섹션은 유닛의 기본 정보와 의존성 포함

```
[Unit]
Description=D-Bus System Message Bus
Documentation=man:dbus-daemon(1)
Requires=dbus.socket
```

- **유닛 의존성 키워드**
  - **Requires:**
    해당 유닛이 시작될 때 반드시 함께 시작해야 하는 다른 유닛 목록
    의존성이 강한 관계
  - **Wants**
    Requires보다 덜 강한 의존성. 여기에 나열된 유닛이 시작되지 않아도 상관없이 유닛을 시작할 수 있음
  - **Conflicts**
    역방향 의존성, 나열된 유닛이 시작되면 이 유닛은 중지되고, 반대로 이 유닛이 시작되면 나열된 유닛은 중지

=> 이 세 가지 키워드는 **outgoing dependency**를 정의하는데 사용
이는 시스템이 한 상태에서 다른 상태로 이동할 때 필요한 유닛을 지정하는 방식

Incoming Dependency (WantedBy 키워드를 사용하여 생성)
- 서비스와 타깃 간의 링크를 생성하며, 특정 상태에서 시작하거나 중지해야 할 서비스를 정의

- **유닛 시작 순서 지정**
  - **Before:**
    해당 유닛이 나열된 유닛보다 먼저 실행되도록 지정
  - **After:**
    해당 유닛이 나열된 유닛이 시작된 후에 실행되도록 지정

```
[Unit]
Description=Lighttpd Web Server
After=network.target
```

- Before이나 After 지시문이 없다면, 유닛은 특별한 순서 없이 동시에 시작되거나 중지됨

<서비스>

- **서비스 유닛 파일**: .service 확장자를 가진 파일로, 시스템 데몬을 정의 (시작과 중지가 가능)

```
[Service]
ExecStart=/usr/sbin/lighttpd -f /etc/lighttpd/lighttpd.conf -D
ExecReload=/bin/kill -HUP $MAINPID
```

- **[Service] 섹션**: 서비스 실행 방법을 지정하는 필수 섹션
  - ExecStart: 서비스 시작 시 실행할 명령어
  - ExecReload: 서비스 재시작 시 실행할 명령어

<타깃>

타깃 유닛 (.target): 서비스를 그룹화한 또 다른 형태의 유닛을 말함
의존성만 가진 유닛으로, 시스템의 동기화 지점 역할

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
```

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

- **Requires**: basic.target이 먼저 실행되어야 함
- **Conflicts**: rescue.service와 rescue.target과 충돌하므로, 이들이 시작되면 multi-user.target은 종료됨
- **After**: basic.target, rescue.service, rescue.target이 먼저 시작된 후에 실행되어야 함
- **AllowIsolate**: 타깃을 독립적으로 실행할 수 있게 허용

위 예제는 basic 타깃이 multi-user 타깃보다 먼저 실행돼야 한다는 것을 알려줌
rescue 타깃과 충돌하기때문에 rescue 타깃이 시작되면, multi-user 타깃이 먼저 종료됨을 보여줌

<systemd로 시스템을 구동하는 방법>

systemd 부트스트랩 과정 요약

1. **시작 프로세스**
   - **/sbin/init**는 **/lib/systemd/systemd**로 심볼릭 링크되어 있으며, 커널이 이를 실행
   - 이 과정에서 **기본 타깃**인 default.target이 실행

```
/etc/systemd/system/default.target -> /lib/systemd/system/multi-user.target
```

기본 타깃은 커널 명령줄에서 system.unit=<new target> 매개변수를 넘겨 재설정 할 수 있음

(systemctl 명령어로 확인 가능)

```
# systemctl get-default
multi-user.target
```

2. **타깃과 의존성**
   - multi-user.target은 시스템이 동작 상태로 시작될 때 필요한 의존성 트리를 구성
   - 예: multi-user.target → basic.target → sysinit.target → 초기 서비스들

- Systemctl list-dependencies 명령어를 이용하면 텍스트 그래프 출력 가능

(모든 서비스와 그들의 현재 상태 출력)

```
# systemctl list-units --type service
```

(타깃도 동일)

```
# systemctl list-units --type target
```

<직접 만든 서비스 추가>

(기존의 simpleserver 예제를 사용한 서비스 유닛)

```
[Unit]
Description=Simple server

[Service]
Type=forking
ExecStart=/usr/bin/simpleserver

[Install]
WantedBy=multi-user.target
```

- **[Unit] 절**
  유닛에 대한 설명만 포함되며, 의존성은 정의되지 않음
  systemctl 등 명령어로 유닛 정보를 확인할 때 사용

- **[Service] 절**
  실행 파일 경로와 포크(forking) 플래그를 설정

만약 포그라운드 환경에서 실행되는 간단한 파일이면, Type=forking은 필요하지 않음
(systemd가 자동으로 데몬화 처리)

- **[Install] 절**
  시스템이 multi-user 모드로 전환될 때 해당 서버가 시작되도록 multi-user.target에 incoming dependency를 설정

- **서비스 실행 및 관리**
  유닛 파일(/etc/systemd/system/simpleserver.service)이 설정되면,
  Systemctl (start or stop) simpleserver 명령어로 서비스를 시작/중지할 수 있음
  Systemctl 명령어로 현재 상태 파악

```
# systemctl status simpleserver
simpleserver.service - Simple server
  Loaded: loaded (/etc/systemd/system/simpleserver.service;
disabled)
  Active: active (running) since Thu 1970-01-01 02:20:50 UTC;
8s ago
Main PID: 180 (simpleserver)
  CGroup: /system.slice/simpleserver.service
          └─180 /usr/bin/simpleserver -n

Jan 01 02:20:50 qemuarm systemd[1]: Started Simple server.
```

  - 이 시점에서는 명령어를 통해서만 유닛을 시작하고 멈출 수 있음

  Systemctl enable을 이용하면 해당 서비스 활성화 가능

```
# systemctl enable simpleserver
Created symlink from /etc/systemd/system/multiuser.target.
wants/simpleserver.service to /etc/systemd/system/simpleserver.service.
```

- 타깃의 의존성 추가:
  타깃은 서비스에 대한 링크를 저장할 수 있는 wants 디렉터리를 가짐
  이 디렉터리에 서비스 링크를 추가하면, 타깃의 [Wants] 섹션에 의존성을 추가하는 것과 동일한 효과
- 서비스 재시작 설정:
  서비스가 실패한 경우 자동으로 재시작하려면 [Service] 섹션에 Restart=on-abort를 추가

```
/etc/systemd/system/multi-user.target.wants/simpleserver.service -> /etc/
systemd/system/simpleserver.service
```

- **Restart 옵션의 값**

  - **on-success**: 서비스가 성공적으로 종료되면 재시작
  - **on-failure:** 실패 시 재시작
  - **on-abnormal:** 비정상 종료 시 재시작
  - **on-watchdog:** watchdog 타이머 초과 시 재시작
  - **always:** 어떤 경우든 재시작

< 워치독 추가>

대부분의 임베디드 SoC에는 /dev/watchdog 장치 노드를 통해 접근할 수 있는 하드웨어 워치독 존재
워치독은 부팅 시 일정 타이머를 갖고 초기화 되며 그 기간 내에 반복적으로 리셋 돼야 함
그렇지 못했을 경우 워치독이 트리거 돼 시스템이 재부팅됨
드라이버 코드: drivers/watchdog

(서비스 유닛에 워치독 활성화시키는 방법 - [Service] 섹션에 다음을 추가)

```
WatchdogSec=30s
Restart=on-watchdog
StartLimitInterval=5min
StartLimitBurst=4
StartLimitAction=reboot-force
```

- **워치독의 역할**

- 임베디드 시스템에서 중요한 서비스가 작동하지 않으면 시스템을 자동으로 재설정하기 위해 하드웨어 워치독을 사용
- 워치독은 일정 시간 내에 주기적으로 리셋되어야 하며, 이 작업이 실패하면 시스템이 재부팅

On-abort와 on-failure의 차이점?

- on-abort: 서비스가 **강제로 종료**되었을 때 동작
  서비스가 외부 또는 내부에서 의도치 않게 중단된 경우
  (예: kill 신호, watchdog에 의한 종료 등)

- on-failure: 서비스가 **실패 코드**나 **에러**로 종료될 때 동작
  서비스가 예기치 않게 실패한 한 경우
  (예: 실행 중 오류 발생, 코드 반환값 1 이상 등)

- 대부분의 임베디드 SoC에서는 /dev/watchdog 장치 노드를 통해 하드웨어 워치독에 접근 가능

- **문제점**
- 여러 중요한 서비스가 워치독 보호를 받아야 할 경우, 각 서비스가 워치독 타이머를 공유하게 되면 문제가 발생

- **systemd의 해결 방법**
  - systemd는 **서비스별 소프트웨어 워치독**을 설정해 각 서비스가 독립적으로 워치독 타이머를 관리하도록 할 수 있음
  - 각 서비스가 주기적으로 **keepalive** 메시지를 보내도록 하여, 만약 지정된 시간 내에 메시지가 오지 않으면 **재시작** 등의 조치를 취하게 됨
  - 이를 위해 서비스 코드에 **WATCHDOG_USEC** 환경 변수를 설정하고, 지정된 시간(워치독 타이머 절반 시간) 내에 sd_notify(false, "WATCHDOG=1")를 호출해야 함

(서비스 유닛에 워치독 활성화 하는 방법 - [Service] 섹션에 다음을 추가)

```
WatchdogSec=30s
Restart=on-watchdog
StartLimitInterval=5min
StartLimitBurst=4
StartLimitAction=reboot-force
```

- **예제 설명**

  - 위의 예제에서 서비스는 30초마다 keepalive 호출을 해야 하고, keepalive 호출이 전달되지 않으면, 서비스가 자동으로 재시작 됨
  - 5분 내에 네 번 이상 재시작되면, systemd는 시스템을 강제로 재부팅

- **systemd 자체 워치독 설정**

  - systemd 자체 실패나 커널 충돌, 하드웨어 문제 등으로 시스템이 멈추면, 시스템을 자동으로 재부팅해야 할 경우가 존재
    => RuntimeWatchdogSec=NN을 /etc/systemd/system.conf에 추가하여 systemd의 워치독을 활성화

  - 지정된 기간 내에 워치독이 리셋되지 않으면 시스템을 재부팅하게 만듦