5

2024년 9월 26일 목요일 오전 8:42

루트 파일시스템 만들기

RYO (Roll Your Own)

임베디드 리눅스 초기에 루트 파일시스템을 만드는 유일한 방법이었음

- **RYO 루트 파일시스템을 적용할 수 있는 경우**
  - 램이나 저장소의 크기가 매우 제한적인 경우
  - 빠른 시연이 필요한 경우
  - 요구 사항이 표준 빌드 시스템 도구로 충족되지 않는 모든 경우

루트 파일시스템에는 무엇이 있어야 하는가?

- 최소한의 루트 파일시스템을 만들기 위해 필요한 요소

init	모든 것을 시작시키는 프로그램
셸	명령 프롬프트를 보여주기 위해 필요 init과 기타 프로그램이 호출하는 셸 스크립트를 실행하기 위해 필요
데몬	다른 프로그램에게 서비스를 제공하는 백그라운드 프로그램 (ex. Syslogd, sshd) Init 프로그램은 주 시스템 애플리케이션들을 지원하기 위해 초기 데몬들을 시작해야 함
공유 라이브러리	대부분의 프로그램은 공유 라이브러리와 링크됨 라이브러리는 루트 파일시스템에 있어야 함
구성 파일	Init과 기타 데몬용 구성 파일들은 일련의 텍스트 파일로, 보통 /etc 디렉터리에 저장
장치 노드	다양한 장치 드라이버에 접근할 수 있도록 해주는 특수 파일들
Proc과 sys	커널 자료구조를 디렉터리와 파일의 계층 구조로 나타내는 2개의 가상 파일시스템 여러 프로그램과 라이브러리 함수가 /proc과 /sys에 의존
커널 모듈	커널의 일부를 모듈로 구성했다면, 루트 파일시스템 (/lib/modules/[커널 버전]) 에 설치돼야 함

<디렉터리 레이아웃>

리눅스 시스템의 기본 레이아웃은 Linux Filesystem Hierarchy Standard (FHS)에 정의
FHS는 가장 큰 것부터 가장 작은 것까지 리눅스 운영체제의 구현을 모두 다룸

/bin	기본적인 사용자 명령어가 포함된 디렉터리
/dev	장치 노드와 기타 특수 파일들
/etc	시스템 구성
/lib	필수 공유 라이브러리
/proc	가상 파일로 표현되는 프로세스에 대한 정보
/sbin	시스템 관리 명령어가 포함된 디렉터리
/sys	가상 파일로 표시되는 장치와 드라이버에 대한 정보
/tmp	임시 파일이나 휘발성 파일 담아두는 곳
/user	/usr/bin, /usr/lib, /usr/sbin 디렉터리에는 각각 추가 프로그램, 라이브러리, 시스템 관리 유틸리티
/var	실행 중 변경될 수도 있는 파일과 디렉터리

- **/bin과 /sbin 차이**
  - /sbin은 일반 사용자 검색 경로에 포함되지 않는 것이 일반적

/usr은 루트 파일시스템과 다른 파티션에 있어도 되므로 시스템을 부트할 때 필요한 것을 담고 있어서는 안됨

<스테이징 디렉터리>

```
user@raspberrypi:~/s32g-linux-bsp300-master $ mkdir ~/rootfs
user@raspberrypi:~/s32g-linux-bsp300-master $ cd ~/rootfs/
user@raspberrypi:~/rootfs $ mkdir bin dev etc home lib proc sbin sys tmp usr var
user@raspberrypi:~/rootfs $ mkdir usr/bin usr/lib usr/sbin
user@raspberrypi:~/rootfs $ mkdir -p var/log
```

```
user@raspberrypi:~/rootfs $ tree -d
.
├── bin
├── dev
├── etc
├── home
├── lib
├── proc
├── sbin
├── sys
├── tmp
├── usr
│   ├── bin
│   ├── lib
│   └── sbin
└── var
    └── log

15 directories
```

디렉터리 계층 구조 확인 tree
디렉토리만 표시 -d
디렉터리 생성 시 중간 디렉터리도 자동으로 생성 -p

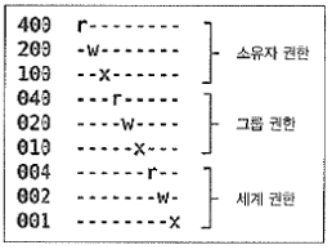
<POSIX 파일 접근 권한>

사용자 ID(UID): 32비트 숫자로 표현
사용자 관련 정보: /etc/passwd에 저장
그룹 ID(GID): /etc/group에 저장

- UID 0: 루트 사용자 (superuser)
- GID 0: 루트 그룹

기본 구성에서 대부분의 권한 검사를 우회하고 시스템의 모든 자원에 접근할 수 있음

각 파일과 디렉터리도 소유자 존재, 하나의 그룹에 속함
프로세스의 파일/디렉터리 접근 수준은 파일의 '모드'라는 접근 권한 플래그의 집합에 의해 제어




```
user@raspberrypi:~ $ ls -l
total 181596
-rw-r--r--  1 user user         0 Oct  2 10:21
-rw-r--r--  1 user user       344 Sep 27 15:56
-rw-r--r--  1 user user         0 Oct  2 10:21
drwxr-xr-x 27 user user      4096 Sep 20 11:45
drwxr-xr-x  2 user user      4096 Sep  6 10:09
drwxr-xr-x 13 user user      4096 Sep 26 10:26
drwxr-xr-x 25 user user      4096 Sep 27 09:33
-rw-r--r--  1 user user 185926511 Sep 20 14:01
drwxr-xr-x 26 user user      4096 Sep 20 09:20
```

$ ls -l : 파일 접근 권한 확인

Ex) 1번 예시
- **소유자 권한**: 읽기 및 쓰기 가능 (rw-) 6
- **그룹 권한**: 읽기만 가능 (r--) 4
- **다른 사용자 권한**: 읽기만 가능 (r--) 4

SUID (4) (Set User ID)	프로그램 실행 시 프로세스의 effective UID가 파일 소유자의 UID로 바꿈
SGID (2) (Set Group ID)	프로그램 실행 시 프로세스의 effective GID가 파일 소유자의 GID로 바꿈
Sticky (1)	디렉터리에서 삭제를 제한해 다른 사용자 소유의 파일을 지울 수 없도록 함 보통 /tmp와 /var/tmp 에 설정

<스테이징 디렉터리의 파일 소유권과 권한>

민감한 자원은 루트만 접근할 수 있도록 제한, 최소한의 프로그램을 비루트 사용자를 이용해 실행해야 함
소유자 외의 모든 읽기 쓰기 접근을 거부하도록 모드는 600이어야 함

```
user@raspberrypi:~/s32g-linux-bsp300-master $ sudo chown -R root:root *
```

-> 소유권 변경

<루트 파일시스템용 프로그램>

루트 파일시스템 작동에 필요한 필수 프로그램

- **init 프로그램**

실행되는 첫 번째 프로그램

- **셸**

스크립트를 실행하고 명령줄 프롬프트를 표시해서 사용자가 시스템과 상호작용하도록 하려면 셸이 필요
개발, 디버깅, 유지보수용으로는 유용

임베디드 시스템에서 많이 쓰이는 셸

Bash	유닉스 본 셸의 확대 집합으로 여러 확장과 고유 기능 존재
Ash	본 셸에 기반을 둔 셸, BusyBox는 bash와 더 호환되도록 확장 된 버전의 ash를 가지고 있음 Bash보다 훨씬 작음
Hush	매우 작은 셸, 메모리가 매우 작은 장치에 유용 BusyBox용 버전 존재

- **유틸리티**

셸이 유용해지기 위해서는 유닉스 명령줄이 기반을 두고 있는 유틸리티 프로그램들이 필요
기본적인 루트 파일시스템도 약 50가지 유틸리티 필요

< BusyBox >

- 등장 배경

1. 각각의 소스 코드를 찾아 크로스 컴파일하는 어려움
2. 파일 크기 문제 (여러 유틸리티들은 각각 크기가 큰데 리눅스 한경에서는 저장공간이 제한적)

필수 리눅스 유틸리티의 필수 기능을 수행하도록 처음부터 작성
80:20 규칙 활용 (유용한 기능 80%는 20% 코드로 작성됨)
모든 도구를 하나의 바이너리로 묶어서 도구들 사이에 코드를 공유하기 쉽도록 하는 것

```
user@raspberrypi:~/s32g-linux-bsp300-master $ busybox
BusyBox v1.30.1 (Debian 1:1.30.1-6+b3) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
   or: busybox --list[-full]
   or: busybox --show SCRIPT
   or: busybox --install [-s] [DIR]
   or: function [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
        [, [[, acpid, adjtimex, ar, arch, arp, arping, ash, awk, basename, bc, blkdiscard, blockdev, brctl, bunzip2, bzcat, bzip2, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp, cp,
        cpio, cttyhack, cut, date, dc, dd, deallocvt, depmod, devmem, df, diff, dirname, dmesg, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, egrep, env, expand, expr, factor,
        fallocate, false, fatattr, fgrep, find, fold, free, freeramdisk, fsfreeze, fstrim, ftpget, ftpput, getopt, getty, grep, groups, gunzip, gzip, halt, head, hexdump, hostid, hostname,
        httpd, hwclock, i2cdetect, i2cdump, i2cget, i2cset, id, ifconfig, ifdown, ifup, init, insmod, ionice, ip, ipcalc, ipneigh, kill, killall, klogd, last, less, link, linux32, linux64,
        linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, ls, lsmod, lsscsi, lzcat, lzma, lzop, md5sum, mdev, microcom, mkdir, mkdosfs, mke2fs, mkfifo, mknod, mkpasswd,
        mkswap, mktemp, modinfo, modprobe, more, mount, mt, mv, nameif, nc, netstat, nl, nologin, nproc, nsenter, nslookup, nuke, od, openvt, partprobe, paste, patch, pidof, ping, ping6,
        pivot_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, resume, rev, rm, rmdir, rmmod, route, rpm, rpm2cpio, run-init, run-parts, sed, seq, setkeycodes,
        setpriv, setsid, sh, sha1sum, sha256sum, sha512sum, shred, shuf, sleep, sort, ssl_client, start-stop-daemon, stat, strings, stty, svc, svok, swapoff, swapon, switch_root, sync, sysctl,
        syslogd, tac, tail, tar, taskset, tee, telnet, test, tftp, time, timeout, top, touch, tr, traceroute, traceroute6, true, truncate, tty, ubirename, udhcpc, udhcpd, uevent, umount, uname,
        uncompress, unexpand, uniq, unix2dos, unlink, unlzma, unshare, unxz, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, w, watch, watchdog, wc, wget, which, who, whoami, xargs, xxd,
        xz, xzcat, yes, zcat
```

-> busybox 입력 시, 컴파일된 모든 애플릿 목록 확인 가능

Busybox가 cat 애플릿을 실행하도록 하는 더 나은 방법은 /bin/cat에서 /bin/busybox로의 심볼릭 링크를 만드는 것

BusyBox는 전형적으로 하나의 프로그램과 각 애플릿을 위한 심볼릭 링크로 설치되지만, 개별 애플리케이션의 묶음인 것처럼 동작
vi 편집기도 존재하여 텍스트 파일 수정 가능

<BusyBox 빌드하기>

Busybox 컴파일

```
user@raspberrypi:~/s32g-linux-bsp300-master $ ls -l bin/cat bin/busy
ls: cannot access 'bin/cat': No such file or directory
ls: cannot access 'bin/busy': No such file or directory
user@raspberrypi:~/s32g-linux-bsp300-master $ git clone git://busybox.net/busybox.git
Cloning into 'busybox'...
remote: Enumerating objects: 117946, done.
remote: Counting objects: 100% (117946/117946), done.
remote: Compressing objects: 100% (26764/26764), done.
remote: Total 117946 (delta 94647), reused 113317 (delta 90427), pack-reused 0
Receiving objects: 100% (117946/117946), 28.91 MiB | 6.27 MiB/s, done.
Resolving deltas: 100% (94647/94647), done.
user@raspberrypi:~/s32g-linux-bsp300-master $ cd busybox
```

Git 아카이브 복제한 후 원하는 버전 체크 아웃

make distclean -> 빌드 과정에서 생성된 파일들 청소 및 깨끗한 상태 만들기
make defconfig -> 기본 구성을 설정하고 해당 설정에 따라 .config 파일을 생성

```
user@raspberrypi:~/s32g-linux-bsp300-master/busybox $ make distclean
user@raspberrypi:~/s32g-linux-bsp300-master/busybox $ make defconfig
```

Busybox 기본 구성 설정

+)

```
user@raspberrypi:~/s32g-linux-bsp300-master/busybox $ make ARCH=arm CROSS_CONPILE=arm-coretex_a8-linux-gnueabihf- install
```

BusyBox를 스테이징 영역에 설치 (CONFIG_PREFIX로 설정된 디렉터리로 복사하고, 이를 가리키는 심볼릭 링크를 만듦)

<루트 파일시스템용 라이브러리>

프로그램은 라이브러리와 링크됨
둘 이상의 프로그램이 있는 경우 불필요하게 많은 양의 저장 공간을 차지, 툴체인으로부터 스테이징 디렉터리로 공유 라이브러리를 복사해야 함

어느 라이브러리로 복사할지 알아내는 방법

1. 툴체인의 sysroot 디렉터리에 있는 모든 .so 파일을 복사하기 (모든 라이브러리가 필요할 것이라고 가정)
2. 필요한 라이브러리만 고르기 (라이브러리 의존 관계 찾아낼 방법 필요)

```
user@raspberrypi:/bin $ aarch64-linux-gnu-readelf -a busybox | grep "program interpreter"
      [Requesting program interpreter: /lib/ld-linux-aarch64.so.1]
user@raspberrypi:/bin $ aarch64-linux-gnu-readelf -a busybox | grep "shared library"
user@raspberrypi:/bin $ aarch64-linux-gnu-readelf -a busybox | grep "Shared library"
 0x0000000000000001 (NEEDED)             Shared library: [libresolv.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [ld-linux-aarch64.so.1]
```

...
?

<스트립을 통한 크기 축소>

라이브러리와 프로그램은 종종 디버깅과 추적을 돕기 위해 심볼 테이블에 저장된 약간의 정보를 포함한 채로 컴파일됨
공간 절약을 하는 빠르고 간단한 방법은 바이너리에서 심볼테이블을 스트립 하는 것

```
$ file rootfs/lib/libc-2.22.so
lib/libc-2.22.so: ELF 32-bit LSB shared object, ARM, EABI5
version 1 (GNU/Linux), dynamically linked (uses shared libs),
for GNU/Linux 4.3.0, not stripped
$ ls -og rootfs/lib/libc-2.22.so
-rwxr-xr-x 1 1542572 Mar 3 15:22 rootfs/lib/libc-2.22.so

$ arm-cortex_a8-linux-gnueabihf-strip rootfs/lib/libc-2.22.so
$ file rootfs/lib/libc-2.22.so
rootfs/lib/libc-2.22.so: ELF 32-bit LSB shared object, ARM,
EABI5 version 1 (GNU/Linux), dynamically linked (uses shared
libs), for GNU/Linux 4.3.0, stripped
$ ls -og rootfs/lib/libc-2.22.so
-rwxr-xr-x 1 1218200 Mar 22 19:57 rootfs/lib/libc-2.22.so
```

커널 모듈 스트립 (재배치에 필요한 정보를 남겨두고 디버그 심볼을 제거하는 법)
-> strip --strip-unneeded <모듈명>

<디바이스 노드>

디바이스 노드 위치: /dev
디바이스 노드는 블록 장치 (대용량 저장 장치와 관련된 장치)나 문자 장치 (데이터가 문자 단위로 처리되는 장치)를 가리킴

- **디바이스 노드 생성**

- mknod로 생성 가능

  mknod <이름> <종류> <주번호> <부번호>

- <이름> - 디바이스 노드의 이름
- <종류> - 문자 장치 c, 블록 장치 b
- <주번호>,<부번호> - 한쌍의 숫자, 커널이 파일 요청을 적절한 장치 드라이버 코드로 보낼 때 쓰임
  (Documentation/devices.txt 파일에 표준 주번호/부번호 목록 존재)

최소 루트 파일시스템에서 BusyBox로 부팅하기 위해서는 2개의 노드(console과 null)만 있으면 됨

console
노드 소유자인 루트만 접근할 수 있으면 됨
접근 권한: 600 (rw-------)

null
모든 사람이 읽고 쓸 수 있어야 함
접근 권한: 666 (rw-rw-rw-)

```
user@raspberrypi:~/rootfs $ sudo mknod -m 666 dev/null c 1 3
user@raspberrypi:~/rootfs $ sudo mknod -m 600 dev/console c 5 1
user@raspberrypi:~/rootfs $ ls -l dev
total 0
crw------- 1 root root 5, 1 Sep 27 13:55 console
crw-rw-rw- 1 root root 1, 3 Sep 27 13:55 null
```

mknod 시, -m 옵션을 쓰면 노드를 만들면서 모드를 설정할 수 있음
rm으로 디바이스 노드 삭제 가능

<proc과 sysfs 파일시스템>

Proc과 sysfs 는 커널의 내부 동작을 보여주는 창과 같은 유사 파일시스템
둘다 커널 데이터를 디렉터리 계층 구조상의 파일들로 나타냄

Proc과 sysfs는 장치 드라이버 및 기타 커널 코드와 상호작용하는 또 다른 방법 제공

mount -t proc proc /proc
mount -t sysfs sysfs /sys

Proc
프로세스에 대한 정보를 사용자 공간에 노출하는 것
/proc/<PID>라는 디렉터리 존재
프로세스 목록 명령 ps는 파일을 읽어 출력물을 만들어 냄
커널의 다른 부분에 대한 정보를 알려주는 파일 존재 (CPU 정보 -> /proc/cpuinfo / 인터럽트 정보 -> /proc/interrupts)

/proc/sys
커널 서브시스템의 상태와 동작을 보여주고 제어하는 파일들 존재
man 5 proc-> 매뉴얼 페이지

Sysf
장치와 장치 드라이버에 관련된 파일의 계층 구조와 이들이 서로 연결된 방식에 대한 정보 제공

<파일시스템 마운트하기>

mount 명령-> 하나의 파일시스템을 다른 파일시스템 내의 디렉터리에 붙여서 파일시스템의 계층 구조를 만들 수 있음
커널이 부트될 때 맨 꼭대기에 마운트되는 파일시스템을 루트 파일시스템이라고 함

mount [-t vfstype] [-o 옵션] 장치 디렉터리

vfstype -> 파일시스템 종류
옵션 -> 콤마로 분리된 mount 옵션 목록
장치 -> 파일시스템이 존재하는 블록 장치 노드
디렉터리 -> 파일시스템을 마운트 하고자 하는 디렉터리
-o 뒤에 다양한 옵션을 지정할 수 있음 (매뉴얼 페이지 mount (8) 참조)

마운트가 성공하면 지정한 디렉터리에서 해당 파일시스템의 내용 확인 가능

장치노드 /dev/proc은 없음-> why? 진짜가 아니라 유사 파일시스템이기 때문
mount 명령은 procfs와 nodevice 문자열을 무시

루트 파일시스템을 타깃으로 전송

루트 파일 시스템을 타깃으로 전송하는 세 가지 방법

1. initramfs
   부트로더에 의해 램에 로드되는 파일시스템 이미지
   램디스크는 만들기 쉽고 대용량 저장 장치 드라이버에 의존하지 않음
   주 루트 파일시스템을 업데이트 해야할 때 유지보수 모드로 사용할 수 있음
   작은 임베디드 장치에서 주 루트 파일시스템으로 사용 가능, 주류 리눅스 배포판에서 초기 사용자 공간으로 흔히 사용
   루트 파일시스템의 내용은 휘발성 (영구 저장 원할 시 또 다른 종류의 저장소가 필요)

2. 디스크 이미지
   포맷되고 타깃의 대용량 저장 장치에 로드될 준비가 된 루트 파일시스템의 복사본

3. 네트워크 파일시스템
   네트워크를 통해 파일 시스템을 공유하는 프로토콜
   스테이징 디렉터리는 NFS 서버를 통해 네트워크로 export될 수 있고 타깃 부팅 때 마운트 될 수 있음

부트 initranfs 만들기

초기 램 파일시스템, 즉 압축된 cpio 아카이브
cpio는 오래된 유닉스 아카이브 포맷
Initranfs를 지원하려면 커널을 COMFIG_BLK_DEV_INITRD로 구성해야함

부트 램디스크를 만드는 세가지 방법
1. 단독형 cpio 아카이브
2. 커널 이미지에 내장된 cpio 아카이브
3. 커널 빌드 시스템이 빌드의 일부로 처리하는 장치 테이블로 만드는 것

<단독형 initramfs>

```
user@raspberrypi:~/rootfs $ find . | cpio -H newc -ov --owner root:root > ../initramfs.cpio
.
./lib
./tmp
./proc
./bin
./var
./var/log
./sys
./dev
./dev/null
./dev/console
./etc
./sbin
./usr
./usr/lib
./usr/bin
./usr/sbin
./home
5 blocks
```

- owner root:root 옵션으로 실행한 이유는 파일 소유권 문제에 대한 해결책 (cpio아카이브 안에 있는 모든 파일들의 UID와 GID를 0으로 설정)

보드를 부트하기 위해 큰 용량의 저장소 필요
크기 문제 해결법
1. 필요 없는 드라이버와 기능을 제거해서 커널 작게 만들기
2. 필요 없는 유틸리티를 제고해서 BusyBox 작게 만들기
3. glibc 대신 musl libc나 uclibc-ng 사용
4. BusyBox를 정적으로 컴파일

<initramfs를 커널 이미지에 넣기>

리눅스는 initramfs를 커널 이미지에 넣도록 구성할 수 있음
-> 커널 구성을 바꾸고 CONFIG_INITRAMFS_SOURCE를 미리 만들어둔 cpio 아카이브의 전체 경로로 설정하면 됨

루트 파일시스템의 내용을 바꿀 때 마다 커널을 다시 빌드하고 .cpio 파일을 다시 만들어야 함

<장치 테이블을 이용해 initramfs 빌드하기>

장치 테이블
-> 아카이브나 파일시스템 이미지에 저장되는 파일, 디렉터리, 디바이스 노드, 링크의 목록을 담고 있는 텍스트파일

장점
-> 루트 특권 없이도 아카이브 파일 안에 루트 사용자나 다른 UID가 소유하는 항목들을 만들 수 있음
-> 루트 특권 없이 디바이스 노드 생성 가능

장치 테이블 파일을 작성하고, 이 파일에서 CONFIG_INITRAMFS_SOURCE를 해당 장치 테이블 파일로 설정
커널을 빌드하고 장치 테이블에 적힌 지시사항에 따라 cpio 아카이브가 생성

- **장치 테이블 명령어**
  - dir <name> <mode> <uid> <gid>:
    - 디렉터리를 생성합니다.
  - file <name> <location> <mode> <uid> <gid>:
    - 파일을 생성합니다. <location>은 파일의 원본 위치
  - nod <name> <mode> <uid> <gid> <dev_type> <maj> <min>:
    - 디바이스 노드를 생성합니다. 여기서 <dev_type>은 문자 장치 또는 블록 장치, <maj>는 주요 번호, <min>은 부 번호
  - slink <name> <target> <mode> <uid> <gid>
    - 심볼릭 링크를 생성합니다. <target>은 링크가 가리킬 위치

init 프로그램

Init은 구성 파일 /etc/inittab을 읽으며 시작
Busybox init은 루트 파일시스템에 inittab이 없으면 기본 inittab 제공

<데몬 프로세스 시작하기>

::respawn:/sbin/syslogd -n

시스템 부팅 시 syslogd 데몬을 시작하고, 만약 이 데몬이 중단되면 자동으로 다시 시작

syslogd:
-> 다른 프로그램의 로그 메시지를 기록
Respawn
-> 프로그램이 종료되면, 자동으로 다시 실행
-n
-> 포그라운드 프로세스로 실행해야 함

로그는 /var/log/messages에 기록

사용자 계정 구성하기

특권이 없는 사용자 계정을 만들고 루트 권한 전부가 필요치 않은 경우에 사용하는 것이 좋음
사용자 이름은 /etc/passwd 에 설정
한 줄이 하나의 사용자를 나타내며 콜론으로 구분된 7개의 필드가 존재

- 로그인 이름
- 패스워드를 검증하기 위한 해시코드 (보통은 민감한 정보의 노출을 줄이기 위해 패스워드가 /etc/shadow에 저장돼 있음을 나타내는 x가 들어있음)
- 코멘트 필드, 종종 빈칸으로 남겨둠
- 사용자의 홈 디렉터리
- 사용자가 사용할 셸

- /etc/passwd 파일에서 나타나는 루트 사용자 계정의 예시

```
root:x:0:0:root:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/false
```

-> 루트 사용자 계정에 대한 정보로, 패스워드는 안전하게 보호되고 있으며, 루트 계정의 UID와 GID는 0으로 설정

- /etc/shadow 파일에서 나타나는 사용자 계정의 예시 (민감한 정보의 노출을 줄이기 위해 패스워드는 다음에 저장되고 패스워드 필드에는 이를 나타내는 x를 적어둠)

```
root::10933:0:99999:7:::
daemon:*:10933:0:99999:7:::
```

-> root::10933:0:99999:7:::는 루트 사용자 계정에 대한 정보로, 패스워드 해시가 없고, 패스워드 변경과 관련된 다양한 설정(최소/최대 변경 일수, 만료 경고 기간 등)을 포함
일반 사용자가 이 파일을 접근할 수 없도록 권한이 제한

그룹 이름은 /etc/groups에 비슷한 방식으로 저장

한 줄이 하나의 그룹을 나타내며 콜론으로 구분된 4개의 필드가 존재
- 그룹 이름
- 그룹 패스워드, 일반적으로는 그룹 패ㅐ스워드가 없음을 나타내는 x 문자
- GID, 즉 그룹 ID
- 선택 사항으로, 그룹에 속하는 사용자의 목록. 콤마로 구분

- /etc/group 파일에서 나타나는 그룹 정보의 예시

```
root:x:0:
daemon:x:1:
```

root:x:0:는 루트 사용자 계정을 나타내며, 패스워드는 보안을 위해 별도로 저장되어 있고, 루트 사용자는 UID가 0인 특권 있는 계정

<루트 파일시스템에 사용자 계정 추가하기>

스테이징 디렉터리에 etc/passwd, etc/shadow, et/group 파일을 추가해야 함

Shadow 파일의 권한은 0600이어야함
그 다음, getty라는 프로그램을 기동해서 로그인 절차를 시작해야함
Busybox에는 getty가 있음
Getty는 로그인 셸이 종료될 때 inittab에서 respawn 키워드를 통해 실행됨

```
::sysinit:/etc/init.d/rcS
::respawn:/sbin/getty 115200 console
```

<장치 노드를 관리하는 더 좋은 방법>

장치 노드 만드는 방법

- **devtmpfs:**
  부트 때 /dev에 마운트 할 수 있는 의사 파일시스템
  노드의 소유자는 루트, 기본 권한은 0666
  Devtmpfs 파일시스템 지원은 커널 구성 변수 CONFIG_DEVTMPFS에 의해 제어
  커널 구성에서 CONFIG_DEVTMPFS_MOUNT를 활성화하면, 커널은 루트 파일시스템을 마운트한 직후 자동으로 devtmpfs를 마운트

- **mdev:**
  디렉터리를 장치 노드로 채우고 필요에 따라 새로운 노드를 만드는 데 쓰이는 Busybox 애플릿
  구성 파일인 /etc/mdev.conf는 노드의 소유권과 권한에 대한 규칙을 담고 있음
  만들어지는 장치 노드의 권한을 수정할 수 있도록 해줌
  -s 옵션으로 mdev를 실행하면, mdev가 /sys 디렉터리를 스캔해서 현재 장치에 대한 정보를 찾고 /dev 디렉터리를 해당 노드로 채움
  추가되는 새로운 장치들을 계속해서 추적해서 필요한 노드들을 만들어주고 싶다면, /proc/sys/kernel/hotplug에 씀으로써 mdev에게 핫플러그 클라이언트를 만들어줘야함
  다음 명령을 /etc/init.d/rcSdp 추가하면 됨

```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

기본 모드는 660, 소유권은 root::root (변경 원할 시, /etc/mdev.conf에 규칙 추가)

- **Udev:**
  주류 리눅스에서 mdev에서 해당하는 기능이며, 현재 systemd의 일부
  데스크톱 리눅스와 일부 임베디드에서 찾아볼 수 있음

네트워크 구성하기

주 네트워크 구성은 /etc/network/interfaces에 저장

<glibc용 네트워크 요소>

Glibc는 NSS(Name Service Switch)라는 매커니즘을 사용해서 이름을 네트워크나 사용자 관련 숫자로 변환하는 방식을 제어
/etc/nsswitch.conf 로 설정됨

호스트 이름을 제외하면, 모두 /etc/ 안에 있는 해당 이름의 파일을 통해 변환됨
호스트 이름은 /etc/hosts 에 없다면 추가적으로 DNS 검색을 통해 변환
/etc/에 이들 파일을 넣으면 작동함.
네트워크, 프로토콜, 서비스는 모든 리눅스 시스템에서 동일하므로 개발 PC의 /etc로부터 복사해도됨
/etc/hosts는 최소한 루프백 주소를 담고있어야 함

```
passwd:         files
group:          files
shadow:         files
gshadow:        files

hosts:          files mdns4_minimal [NOTFOUND=return] dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

장치 테이블을 이용해 파일시스템 이미지 만들기

genext2fs 도구를 설치해야 함

Genext2fs는 <name> <type> <mode> <uid> <gid> <major> <minor> <start> <inc> <count>형식의 장치 테이블 파일을 사용

Name: 파일명
Type:(f) 보통 파일
    (d) 디렉터리
    (c) 문자 특수 장치 파일
    (b) 블록 특수 장치 파일
    (p) FIFO
uid: 파일의 uid
gid: 파일의 gid
Major와 minor: 장치 번호
start, inc, count: minor 번호가 start에서 시작하는 장치 노드 그룹을 만들 수 있도록 함

Genext2fs를 이용해 4MiB 의 파일시스템 이미지를 만들음
결과로 만들어진 이미지를 SD 카드 등에 복사 가능

<NFS를 이용해 루트 파일시스템 마운트하기>

장치에 네트워크 인터페이스가 있으면 개발 중에는 네트워크를 통해 루트 파일시스템을 마운트하는 것이 가장 좋음

호스트에 NFS 서버를 설치하고 구성해야 함
NFS 서버는 /etc/exports로 제어됨, 각줄은 하나의 export를 나타냄

호스트에 있는 루트 파일시스템을 export 하려면?

```
/home/chris/rootfs *(rw,sync,no_subtree_check,no_root_squash)
```

*는 로컬 네트워크의 모든 주소에 대해 디렉터리를 export
*와 여는 괄호 사이에는 빈칸이 없어야 함

rw: 디렉터리를 읽고 쓰기로 export
sync: 동기 버전 NFS 프로토콜을 선택, 동기버전은 비동기 버전보다 더 견고하지만 약간 느림
no_subtree_check: 서브 트리 확인을 비활성화, 보안에 영향이 있지만 경우에 따라 신뢰성이 높아짐
no_root_squash: 사용자 ID 0으로 전달된 요청을 다른 사용자 ID로 바꾸지 않고 처리하도록 허용

/etc/exports를 수정했으면 NFS 서버를 재기동해 수정 사항이 반영되도록 함
타깃이 NFS를 통해 루트 파일시스템을 마운트하도록 설정해야 함
이를 위해서는 커널이 CONFIG_ROOT_NFS 로 구성돼있어야 함

```
root=/dev/nfs rw nfsroot=<호스트-ip>:<루트-디렉터리> ip=<타깃-ip>
```

rw: 루트 파일시스템을 읽고 쓰기로 마운트
Nfsroot: 호스트의 ip 주소, export된 루트 파일시스템의 경로를 지정
Ip: 타깃에 할당된 IP 주소, 실행 때 할당됨, 이 경우 루트 파일시스템이 마운트되고 init이 시작하기 전에 인터페이스가 구성돼야 함 (커널 명령줄에 구성)

<파일 권한 문제>

커널 소스 코드나 DTB를 수정할 때 NFS가 제공하는 빠른 속도와 동일한 이점을 얻는 방법
-> TFTP

TFTP를 이용해 커널 로드하기

TFTP 는 매우 간단한 파일 전송 프로토콜, U-Boot 같은 부트로더에서 구현하기 쉽도록 설계
호스트에 TFTP 데몬을 설치해야 함

```
user@raspberrypi:~/rootfs/etc $ sudo apt install tftpd-hpa
```

Ttftpd-hpa 는 디폴트로 /var/lib/tftpboot 디렉터리에 있는 파일들에 대한 읽기 전용 접근을 허용
Tftpd-hpa가 설치되고 실행되면, 타깃으로 복사하고 싶은 파일을 /var/lib/tftpboot 디렉터리로 복사
u-boot 명령 프롬프트에 다음 명령 입력

```
setenv serverip 192.168.1.1
setenv ipaddr 192.168.1.101
tftpboot 0x80200000 zImage
tftpboot 0x80f00000 am335x-boneblack.dtb
setenv npath [path to staging]
setenv bootargs console=ttyO0,115200 root=/dev/nfs rw
nfsroot=${serverip}:${npath} ip=${ipaddr}
bootz 0x80200000 - 0x80f00000
```

Tftpboot 도중 끊임없이 글자 T를 출력하여 진행이 안되는 경우 존재 (이는 TFTP 요청 시간 제한을 초과했다는 뜻)
이런 일이 일어나는 이유

- 서버의 IP 주소가 잘못됨
- 서버에서 TFTP 데몬이 동작하지 않고 있음
- 서버의 방화벽이 TFTP 프로토콜을 막고 있음, 대부분의 방화벽은 디폴트 설정에서 실제로 TFTP포트 69번을 막음