Embedded Linux 3.

RAMDisk를 이용한 root FS

http://network.hanbat.ac.kr

- Empos II에서 사용할 수 있는 ramdisk의 생성
  - Basic FS
  - Busybox
  - System V init

Root FS의 기본 디렉토리 생성

TARGET_PREFIX=/root/embedded/gcc-3.4.5-glibc-2.3.6/arm-unknown-linux-gnu/arm-unknown-linux-gnu

- 디렉토리 이름이다. (파일 이름이 아님)

```
mkdir /root/embeddedLinux/rootfs
ROOT_FS=/root/embedded/rootfs
EMBED_ROOT=/root/embedded
cd ${ROOT_FS}
mkdir bin dev etc lib proc sbin tmp usr var
chmod 1777 tmp
mkdir usr/bin usr/lib usr/sbin
mkdir var/lib var/lock var/log var/run var/tmp
mkdir root
chmod 1777 var/tmp
```

- 1: set sticky bit
  - other의 쓰기 권한에 대한 특별한 퍼미션인데 /tmp 디렉토리와 /var/tmp 디렉토리에의 퍼미션이 stickbit가 포함되어 있다.
  - sticky bit는 8진수 모드로는 1000으로 설정되고 심볼릭 모드로는 't' 또는 'T‘ 로 설정된다.
  - 이 sticky bit가 포함되어 있는 디렉토리에 other에 쓰기 권한이 있을 경우 other에 해당하는 사용자들은 디렉토리 안에 파일을 만들 수는 있어도 디렉토리 삭제는 할 수 없다.
  - sticky bit가 포함되어 있는 대표적인 디렉토리가 /tmp 디렉토리인데 분명히 /tmp 디렉토리의 퍼미에는 other에게 쓰기 권한이 주어져 있다. 그렇기 때문에 /tmp 디렉토리에는 누구나 디렉토리와 파일을 만들 수 있다. 하지만 other에 해당하는 사용자는 쓰기 권한이 있음에도 불고 하고 /tmp 디렉토리를 지울수 없는 것이다. 그 이유는 /tmp 디렉토리에는 sticky bit가 설정되어 있기 때문이다. 당연히 소유자는 sticky bit가 설정되어 있는 디렉토리를 삭제할 수 있다.
  - 디렉토리가 아닌 파일에 sticky bit가 설정되어 있을 때는 other 에게 쓰기 퍼미션이 있어도 파일을 수정할 순 있지만 그 파일을 삭제할 수는 없다.
    sticky bit를 적용할려면 당연히 쓰기 권한도 주어야 되다.
- 4: set sticky bit
  - 해당 파일이 실행되는 동안의 해당 파일의 owner 권한으로 실행 (/usr/bin/passwd 실행파일)

Library 복사

```
cp ${TARGET_PREFIX}/lib/*-*.so ${ROOT_FS}/lib
cp -d ${TARGET_PREFIX}/lib/*.so.[*0-9]
   ${ROOT_FS}/lib
cd ${TARGET_PREFIX}/lib/
cp libSegFault.so libmemusage.so libpcprofile.so
   ${ROOT_FS}/lib
```

  - 이전에 다운받은 arm용 gcc의 library 를 복사
    - 우리는 이 library를 이용하여 보드에서 사용할 응용을 컴파일할 것임
    - 호스트에서 사용하는 라이브러리를 복사하면 동작하지 않음.

- 위의 예에서는 모든 library를 복사하였으나, 필요한 경우 보드에서 동작하는 응용에서 필요로 하는 특정 library 만을 복사할 수 도 있다.
  - 어떤 응용이 어떤 libray를 필요로 하는지를 알려면

```
HOST# arm-unknown-linux-gnu-readelf  -a
   rootfs/bin/busybox  |  grep library
0x00000001 (NEEDED)          Shared library: [libm.so.6]
0x00000001 (NEEDED)          Shared library: [libc.so.6]
```

```
arm-unknown-linux-gnu-strip ${ROOT_FS}/lib/*.so
```

- Symbolic information을 제거하여, 설치된 library의 크기를 reduce

Busybox 설치

- Busybox란?
  - 경량의 여러 UNIX 유틸리티들을 하나의 작은 실행 파일로 합친 것
  - 사용자 데스크탑 시스템에서 사용하는 대부분의 일반 유틸리티 (예, ls, cp, mv, mount, tar, 등등)를 대체하는 필수 프로그램을 제공
    - BusyBox에 포함된 유틸리티는 일반 GNU 유틸리티보다는 적은 수의 옵션을 제공
  - 사용할 수 있는 파일 시스템 용량의 제한이 있는 embedded system에 적합

wget http://busybox.net/downloads/busybox-1.13.3.tar.bz2

tar xvjf ${EMBED_ROOT }/src/busybox-1.13.3.tar.bz2

cd busybox-1.13.3/

make menuconfig 또는 make xconfig

- BusyBox Settings에서
- → Build Options의 마지막 줄: Cross Compiler Prefix: 예
  - **arm-unknown-linux-gnu-** 삽입
- **Init Utilities 부분에서 체크된 부분 모두 삭제**
- 저장 후 exit

more INSTALL

- 파일을 읽어보면 앞 부분에, 특정 위치에 install 하려면,
  - make CONFIG_PREFIX=/path/from/root install
  - 처럼 기술하라고 되어 있음
- 우리는 생성하고 있는 root fs에 이를 설치하여야 하므로

make CONFIG_PREFIX=/root/embedded/rootfs all install

System V init 설치 → 삭제

- **init** (short for initialization) is the program on Unix and Unix-like systems that spawns all other processes.
  - UNIX는 부팅 초기에 init proces가 생성되며, init은 미리 정해진 script에 따라서 다른 process 들을 생성한다.
  - It runs as a daemon and typically has PID 1.
  - 종류
    - BSD style
    - SystemV style

wget
ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/sysvinit-2.86.tar.gz

tar xvzf ../src/sysvinit-2.86.tar.gz

cd sysvinit-2.86/src/

make CC=arm-unknown-linux-gnu-gcc

- Crypt에서 에러가 발생하면, man crypt의 출력을 보고, 해당 부분을 수정

mkdir ${ROOT_FS}/usr/include

mkdir -p ${ROOT_FS}/usr/share/man/man1

mkdir -p ${ROOT_FS}/usr/share/man/man5

mkdir -p ${ROOT_FS}/usr/share/man/man8

make ROOT=${ROOT_FS} install

- **대소문자 주의**
- **잘못하면 호스트의 init을 수정하게 되어, 호스트가 부팅되지 않을 수 있음.**

생성하려는 root FS의 /dev/ 필수 파일 생성

- UNIX는 'file의 개념'을 시스템에 연결되어 있는 '주변장치에까지 확장'시킨다.
  - printer, disk, tape, main memory 등과 같은 주변장치들은 file 구조상의 file 이름으로 나타난다.
    - 예: BD와의 tty 통신 시 /dev/ttyS0 사용
      - /dev/ttyS0에 write 하는 것은 실제로 장치에 출력됨
  - 즉, 각각의 I/O device는 적어도 하나의 special file에 연결된다.
- special file은 ordinary file과 달리 관련 device를 구동하여 데이타를 read/write할 수 있도록 하는 channel을 형성해 준다. device driver와 연결해 주는 special file로 /dev밑에 있음.
- system의 모든 input/output(I/O)는 /dev directory를 통해 이루어진다.
- "mknod" command로 special file을 만듦.

```
HOST#   mknod -m 600 ${ROOT_FS}/dev/mem c 1 1
HOST#   mknod -m 666 ${ROOT_FS}/dev/null c 1 3
HOST#   mknod -m 666 ${ROOT_FS}/dev/zero c 1 5
HOST#   mknod -m 644 ${ROOT_FS}/dev/random c 1 8
HOST#   mknod -m 600 ${ROOT_FS}/dev/tty0 c 4 0
HOST#   mknod -m 600 ${ROOT_FS}/dev/tty1 c 4 1
HOST#   mknod -m 600 ${ROOT_FS}/dev/ttyS0 c 4 64
HOST#   mknod -m 666 ${ROOT_FS}/dev/tty c 5 0
HOST#   mknod -m 600 ${ROOT_FS}/dev/console c 5 1
HOST#   mknod -m 660 ${ROOT_FS}/dev/ram0 b 1 0
```

  - Special file을 생성 (mknod)
  - Mode를 660으로 하여 (-m 660)
  - 이름은 ${ROOT_FS}/dev/ram0
  - Block special file (b)
  - Major #, minor # (1 0)

```
mknod -m 600 ${ROOT_FS}/dev/mem c 1 1
mknod -m 666 ${ROOT_FS}/dev/null c 1 3
mknod -m 666 ${ROOT_FS}/dev/zero c 1 5
mknod -m 644 ${ROOT_FS}/dev/random c 1 8
mknod -m 600 ${ROOT_FS}/dev/tty0 c 4 0
mknod -m 600 ${ROOT_FS}/dev/tty1 c 4 1
mknod -m 600 ${ROOT_FS}/dev/ttyS0 c 4 64
mknod -m 666 ${ROOT_FS}/dev/tty c 5 0
mknod -m 600 ${ROOT_FS}/dev/console c 5 1
mknod -m 660 ${ROOT_FS}/dev/ram0 b 1 0
```

Root FS 생성 및 파일 복사

```
rm -f ${EMBED_ROOT}/empos.ramdisk.fs.bin
    /root/embeddedLinux/empos.ramdisk.fs
dd if=/dev/zero of=${EMBED_ROOT}/empos.ramdisk.fs
    count=4096 bs=1024
```

- /dev/zero/로 부터 입력받아 empos.ramdisk.fs 파일을 생성 (내용이 전부 0으로 채워짐)
- Block 크기는 1024로 하고, 2049개의 block을 생성

```
mke2fs -F -m0 ${EMBED_ROOT}/empos.ramdisk.fs
```

- 파일 empos.ramdisk.fs를 파일 시스템으로 만듬
  - Force mke2fs to create a filesystem, even if… (-F)
  - the percentage of the filesystem blocks reserved for the super-user. (-m0, reserve된 superblock 크기를 0)
  - FS는 일정한 형식이 있음
    - 이 명령은 empos.ramdisk.fs 파일을 ext2 FS 형식으로 format 한다고 생각하면 됨

```
mkdir -p ${EMBED_ROOT}/ramdisk
mount -t ext2 -o loop ${EMBED_ROOT}/empos.ramdisk.fs
    ${EMBED_ROOT}/ramdisk/
```

- 생성한 FS를 ramdisk 디렉토리에 마운트

```
cp -rav ${EMBED_ROOT}/rootfs/*
    ${EMBED_ROOT}/ramdisk/
```

- 복사

```
umount ${EMBED_ROOT}/ramdisk

gzip < ${EMBED_ROOT}/empos.ramdisk.fs >
    ${EMBED_ROOT}/empos.ramdisk.fs.bin
```

- 압축

생성한 root FS의 검증

```
EMPOS# tftp empos.ramdisk.fs.bin ramdisk
```

```
EMPOS# flash ramdisk
EMPOS# reset
```

- 보드의 동작을 확인

추가로 필요한 작업

- System V init에서 사용하는 /etc/inittab 과 /etc/rc.d/ 디렉토리의 파일들을 수정해 줄 필요가 있다.
- 여기에는 시스템이 부팅되면서, 설정된 run-level에 따라서 수행할 필요가 있는 여러가지 명령들이 기록되어 있다.

- /etc/inittab 파일의 예

```
id:5:initdefault:

lc:0123456:wait:/etc/rc.d/rc.local

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

T0:12345:respawn:/sbin/getty -L ttyS0 115200 vt100
```

- /etc/rc.d/rc 파일의 예

```
# Source function library.
. /etc/rc.d/init.d/functions

# Now find out what the current and what the previous runlevel are.
argv1="$1"
set `/sbin/runlevel`
runlevel=$2
previous=$1
export runlevel previous

# See if we want to be in user confirmation mode
if [ "$previous" = "N" ]; then
   if grep -i confirm /proc/cmdline >/dev/null ; then
     CONFIRM=yes
   else
     CONFIRM=
   fi
fi
# Get first argument. Set new runlevel to this argument.
[ "$1" != "" ] && runlevel="$argv1"

# Tell linuxconf what runlevel we are in
[ -d /var/run ] && echo "/etc/rc.d/rc$runlevel.d" >
/var/run/runlevel.dir
```

```
# Is there an rc directory for this new runlevel?
if [ -d /etc/rc.d/rc$runlevel.d ]; then
    # First, run the KILL scripts.
    for i in /etc/rc.d/rc$runlevel.d/K*; do
        # Check if the script is there.
        [ ! -f $i ] && continue

        # Don't run [KS]??foo.{rpmsave,rpmorig} scripts
        [ "${i%.rpmsave}" != "${i}" ] && continue
        [ "${i%.rpmorig}" != "${i}" ] && continue
        [ "${i%.rpmnew}" != "${i}" ] && continue

        # Check if the subsystem is already up.
        subsys=${i#/etc/rc.d/rc$runlevel.d/K??}
        [ ! -f /var/lock/subsys/$subsys ] && \
            [ ! -f /var/lock/subsys/${subsys}.init ] && continue

        # Bring the subsystem down.
        if egrep -q "(killproc |action )" $i ; then
            $i stop
        else
            action "Stopping $subsys" $i stop
        fi
    done
```

```
# Now run the START scripts.
   for i in /etc/rc.d/rc$runlevel.d/S*; do
      # Check if the script is there.
      [ ! -f $i ] && continue

      # Don't run [KS]??foo.{rpmsave,rpmorig} scripts
      [ "${i%.rpmsave}" != "${i}" ] && continue
      [ "${i%.rpmorig}" != "${i}" ] && continue
      [ "${i%.rpmnew}" != "${i}" ] && continue

      # Check if the subsystem is already up.
      subsys=${i#/etc/rc.d/rc$runlevel.d/S??}
      [ -f /var/lock/subsys/$subsys ] || \
         [ -f /var/lock/subsys/${subsys}.init ] && continue

      # If we're in confirmation mode, get user confirmation
      [ -n "$CONFIRM" ]  &&
       {
        confirm $subsys
        case $? in
          0)
            :
          ;;
          2)
            CONFIRM=
          ;;
          *)
            continue
          ;;
        esac
       }
```

```
 # Bring the subsystem up.
      if egrep -q "(daemon |action )" $i ; then
         $i start
      else
         if [ "$subsys" = "halt" -o "$subsys" = "reboot" -o "$subsys" =
"single" ]; then
            $i start
         else
            action "Starting $subsys" $i start
         fi
      fi
   done
fi
```

- Root password 삭제
  - /etc/passwd
    root:x:0:0:root:/root:/bin/sh
  - /etc/shadow
    root::0:0:99999:7:-1:-1:33637592

Ramdisk 수정

- 사용하는 ramdisk 소스를 gunzip으로 풀고

```
% mount –o loop ramdiskName ./tmp
% cd tmp
    ## 필요한 작업
% cd ..
% umount tmp
% gzip < ramdiskName > new-ramdisk-name
```

  - new-ramdisk-name 파일을 다운로드하여, flash 저장하고 리부팅

JFFS (Journalling Flash File System)

- Flash 메모리에 파일 시스템을 저장해 놓고, linux 가 부팅된 이후 해당 부분을 mount 하여 사용
- MTD 드라이버는 플래시 메모리를 위한 강력한 기능을 제공한다.
  - 이것을 이용하면 플래시 메모리를 위해 특별히 고안된 JFFS, JFFS2 와 같은 실제 (read/write 가 가능한) 파일 시스템을 사용할 수 있다.
- 전체 과정
  - JFFS 형식의 파일 시스템을 만들어서,
  - empos boot 시에 생성된 jffs 파일 시스템을 다운로드하고 플래시에 저장한 후 (% flash usr),
  - 리눅스 부팅 후 flash 부분을 mount
- “Linux device driver 5.5 Empos 2.6 JFFS porting” 참조

http://hulryung.tistory.com/204

- 리눅스 시스템용 루트 파일 시스템을 구성하는데에는 많은 방법이 있다. 그냥 디렉토리 하나씩 만들어 가면서 만드는 방법도 있고, scratchbox, buildroot, 이미지를 이용하는 방법 등등등..

  나는 하나씩 하나씩 새로 만드는 방법을 한번 써보도록 하겠다~
  여기서 쓰는 시스템은 ubuntu 7.10(gutsy gibbon)이고 첫 타겟 시스템은 ARM920T를 이용한 프로세서(S3C2440A)를 이용한 하드웨어다. 참고로 시중에 판매되는 제품은 아니고 새로 설계된 하드웨어;;
  리눅스 커널은 현재 최신의 2.6.25에 개발 하드웨어에 대한 내용을 포팅했다.
  일단은 ramdisk를 만들고 ramdisk를 기반으로 해서 JFFS2 또는 yaffs용 파일 시스템을 만들어보겠다.

  ramdisk 이미지 만들기
  먼저 8MB짜리 이미지를 만든다.
  $dd if=/dev/zero of=ramdisk-image bs=1024 count=8192
  8192+0 records in
  8192+0 records out
  8388608 bytes (8.4 MB) copied, 0.164898 seconds, 50.9 MB/s

  이 파일에다가 파일시스템을 꾸미자
  $mkfs.ext2 ramdisk-image
  mke2fs 1.40.2 (12-Jul-2007)
  ramdisk-image is not a block special device.
  Proceed anyway? (y,n) y
  Filesystem label=
  OS type: Linux
  Block size=1024 (log=0)
  Fragment size=1024 (log=0)
  2048 inodes, 8192 blocks
  409 blocks (4.99%) reserved for the super user
  First data block=1
  Maximum filesystem blocks=8388608
  1 block group
  8192 blocks per group, 8192 fragments per group
  2048 inodes per group
- Writing inode tables: done
  Writing superblocks and filesystem accounting information: done
- This filesystem will be automatically checked every 38 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override.

- $file ramdisk-image
  ramdisk-image: Linux rev 1.0 ext2 filesystem data

  기본적인 ext2 파일 시스템으로 포맷된 이미지가 생성되었다. 이것을 마운트 해서 사용해보자
  $mkdir ramdisk
  $sudo mount -t ext2 -oloop ramdisk-image ramdisk
  $cd ramdisk
  ramdisk$ls
  lost+found

  ramdisk에 기본 디렉토리 생성하기
  디렉토리를 생성해보자
  ramdisk$mkdir -p bin dev etc home opt proc root sbin usr var lib/modules usr/include usr/share/man/man1 usr/share/man/man5 usr/share/man/man8 sys var/log var/run/utmp var/run/wtmp dev/pts var/lock/ var/lock/subsys /usr/bin

  ROOTFS = 루트파일시스템 위치
  TOOLCHAIN = 툴체인 위치

  glibc 복사
  일단 복사하자
  $ cd $TOOLCHAIN/arm-softfloat-linux-gnu
  $ cp lib usr $ROOTFS -a
  $ cp include share $ROOTFS/usr -a

  sysvinit 설치
  $ wget ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/sysvinit-2.86.tar.gz
  or $ wget http://downloads.openmoko.org/sources/sysvinit-2.86.tar.gz
  $ tar xzvf sysvinit-2.86.tar.gz
  $ cd sysvinit/src
  $ Makefile 수정
  CC=arm-linux-gcc
  BIN_OWNER = hulryung
  BIN_GROUP = hulryung
  $ ROOT=$ROOTFS make
  $ ROOT=$ROOTFS make install
  $ vi $ROOTFS/etc/inittab
  적절하게..

- **initscript 구성**

  $cd $ROOTFS/etc/
  $mkdir init.d rc0.d rc1.d rc2.d rc3.d rc4.d rc5.d rc6.d rcS.d
  $cp banner bootmisc.sh checkfs.sh checkroot.sh devices devpts.sh finish functions halt hostname.sh mountall.sh mountnfs.sh populate-volatile.sh ramdisk reboot rmnologin save-rtc.sh sendsigs single umountfs umountnfs.sh urandom sysfs.sh
  /home/hulryung/emdk2440s/rootfs/rootfs-base2/etc/init.d/
  $cp devpts $ROOTFS/etc/default
  $cp volatiles $ROOTFS/etc/default/volatile/00_core

  dev 구성

```
#!/bin/sh
mknod apm_bios c 10 134
mknod console c 5 1
mknod fb0 c 29 0
mknod hda b 3 0
for i in 1 2 3 4 5 6 7 8 9;do mknod hda$i b 3 $i;done
mknod kmem c 1 2
mknod mem c 1 1
for i in 0 1 2 3 4 5 6 7 8 9;do mknod mtd$i c 90 $((i * 2));done
for i in 0 1 2 3 4 5 6 7 8 9;do mknod mtdblock$i c 31 $i;done
mknod null c 1 3
mknod ptmx c 5 2
for i in 0 1 2 3;do mknod ram$i c 31 $i;done
mknod random c 1 8
mknod tty c 5 0
for i in 0 1 2 3 4 5 6 7 8;do mknod tty$i c 4 $i;done
mknod ttySAC0 c 204 64
mknod urandom c 1 9
mknod zero c 1 5
```

- 이제 ramdisk에 부팅에 필요한 파일들을 좀 올려보자.
  busybox
  $ wget http://busybox.net/downloads/busybox-1.10.2.tar.bz2
  $ tar xjvf busybox-1.10.2.tar.bz2
  $ cd busybox-1.10.2
  busybox-1.10.2$ make menuconfig
  세팅 대충하고..

  busybox-1.10.2$ ARCH=arm CROSS_COMPILE=arm-linux- make
  하면 메이킹 된다.
  busybox-1.10.2$ make install
  하면 _install이라는 디렉토리에 필요한 파일이 만들어진다.
  $cp _install/* /home/hulryung/emdk2440s/rootfs/rootfs-base/ -a

  tinylogin 설치
  Makefile 수정
  CROSS := arm-linux-
  $make
  $sudo make install
  login과 tinylogin만 루트파일시스템으로 복사한다.
  tinylogin에 setuid로 권한이 세팅되어 있다.. 그래서 root권한으로 사용되도록 해야 한다.
  그래서 바꿔준다.
  $sudo chown root:root /home/hulryung/emdk2440s/rootfs/rootfs-base/bin/tinylogin

  자.. 이제 기본적으로 부팅하는 이미지는 생성 완료
  여기에 팁으로 자동 로그인 되게 만들자
  $vi /home/hulryung/emdk2440s/rootfs/rootfs-base/bin/autologin
  #!/bin/sh
  /bin/tinylogin login -f root

  $chmod 777 /home/hulryung/emdk2440s/rootfs/rootfs-base/bin/autologin
  $vi /home/hulryung/emdk2440s/rootfs/rootfs-base/etc/inittab
  T0:12345:respawn:/sbin/getty -L -n -l /bin/autologin ttySAC0 115200 vt100

- **bash shell 설치**
  ./configure --host=arm-linux
  make
  Makefile에서 prefix 수정 한다.
  make install
  cd /home/hulryung/emdk2440s/rootfs/rootfs-base/bin
  ln -s bash sh

  etc디렉토리 passwd 파일 수정
  root 계정에 패스워드를 없애버리자

  패키징
  $sudo umount ramdisk
  $gzip -9 ramdisk-image
  $mkimage -n Ramdisk -A arm -O linux -T ramdisk -C gzip -a 0x30800000 -e 0x30800000 -d ramdisk-image.gz ramdisk-emdk2440s.img

  다운로드 후 테스트

  root=/dev/ram0 rw rootfstype=ext2 console=ttySAC0,115200n81 initrd=0x30800000 ramdisk_size=8192

  이정도 하면 뭐 되는거 거의 없지만 아~~주 간단하게 루트파일 시스템을 구성할 수 있다.
  앞으로는 이걸 베이스로 해서 sysvinit의 구성(rc.xx)하고 X도 올려보고 qt도 올려보고 할 수 있다
  자.. 한번 해보자.