Report

Laboratory 1
14.05.2018

Maciej Korzeniewski

Prepare SD card

SD card had to be prepared in the following fashion:

1. First partition (mmcblk0p1) should be formatted in VFAT. It will be used as administrative system responsible for managing SD card.
2. Second partition (mmcblk0p2) should be formatted in Ext4. It will be used as utility system containing web server controlled via WWW interface.
3. Third partition (mmcblk0p3) should be formatted in Ext4. It will be used as a storage for files uploaded via web server running on partition 2.

Administrative System Configuration

First of all we prepare buildroot configuration for administrative system. It will be working in initramfs. In .config file set:

BR2_TARGET_ROOTFS_INITRAMFS=y

Administrative system needs tools to partition, format, resize and repair file systems. We will make use of:

dosfstools - consists of the programs mkfs.fat, fsck.fat and fatlabel to create, check and label file systems of the FAT family.

```
BR2_PACKAGE_DOSFSTOOLS=y
BR2_PACKAGE_DOSFSTOOLS_FATLABEL=y
BR2_PACKAGE_DOSFSTOOLS_FSCK_FAT=y
BR2_PACKAGE_DOSFSTOOLS_MKFS_FAT=y
```

e2fsprogs - It provides the filesystem utilities for use with the ext2 filesystem. It also supports the ext3 and ext4 filesystems.

```
BR2_PACKAGE_E2FSPROGS_FSCK=y
```

resize2fs - ext2/ext3/ext4 file system resizer

```
BR2_PACKAGE_E2FSPROGS_RESIZE2FS=y
```

dropbear - provides communication via ssh.

```
BR2_PACKAGE_DROPBEAR=y
BR2_PACKAGE_DROPBEAR_CLIENT=y
BR2_PACKAGE_DROPBEAR_SMALL=y
```

Utility System Configuration

In contrary to to administrative system, where initramfs was used, zImage generated by buildroot compilation will consists of kernel only. Filesystem should be provide manually - by coping rootfs.cpio content into mounted partition two. rootfs.cpio file is generated when the following option is turned on:

```
BR2_TARGET_ROOTFS_CPIO=y
BR2_TARGET_ROOTFS_CPIO_GZIP=y
```

Besides, we want our utility system to be formatted in Ext4:

```
BR2_TARGET_ROOTFS_EXT2=y
BR2_TARGET_ROOTFS_EXT2_4=y
BR2_TARGET_ROOTFS_EXT2_GEN=4
BR2_TARGET_ROOTFS_EXT2_REV=1
BR2_TARGET_ROOTFS_EXT2_LABEL=""
BR2_TARGET_ROOTFS_EXT2_SIZE="300M"
BR2_TARGET_ROOTFS_EXT2_INODES=0
BR2_TARGET_ROOTFS_EXT2_RESBLKS=5
BR2_TARGET_ROOTFS_EXT2_MKFS_OPTIONS="-O ^64bit"
BR2_TARGET_ROOTFS_EXT2_NONE=y
```

Utility system should provide web server which will serve files from partition three. Server has been written in NodeJS and WWW interface has been written in HTML and JavaScript.

For NodeJS set:

```
BR2_PACKAGE_NODEJS_ARCH_SUPPORTS=y
BR2_PACKAGE_NODEJS=y
BR2_PACKAGE_NODEJS_NPM=y
BR2_PACKAGE_NODEJS_MODULES_ADDITIONAL=""
BR2_PACKAGE_NODEJS_MODULES_ADDITIONAL_DEPS=""
```

Setting up administrative system on Raspberry

Administrative system in being set up as we did with previous systems so far, mainly, enter rescue mode, mount partition and copy zImage using scp from our host machine, then reboot.

Setting up utility system on Raspberry

In order to properly setup utility system we first enter our administrative system. We need to prepare partitions (partition 2 and partition 3) for our utility system. Run:

```
$ fdisk -u /dev/mmcblk0
```

We can see partition table using `p`, save our changes using `w`, delete partition with `d` and create new partition with `n`.

Let’s create new partition 2 with following sequence of commands:

```
d -> 2 -> n -> p -> 2 -> +{first_block} -> +{last_block} -> {enter}
```

Then create partition 3:

```
n -> p -> 3 -> +{first_block} -> +{last_block} -> {enter}
```

Do not forget about saving changes with:

```
w -> {enter}
```

Now let’s format partition 2 and 3 to Ext4:

```
mkfs.ext4 /dev/mmcblk0p2
mkfs.ext4 / dev/mmcblk0p3
```

When our partitions are ready we can fill partition 2 with file system. Navigate to ./buildroot-{version}/output/images on our host machine (this is a buildroot folder configured for our utility system) and execute the following command:

```
$ cat rootfs.cpio | ssh root@192.168.143.XXX ‘cd /tmp/user; cpio -i -d’
```

Since utility system will make use of partition 3 as a storage for files we need to ensure that partition 3 is always mounted on utility system's start. Hence, we will create an overlay on a /etc/fstab file:

```
# <file system>       <mount pt>   <type>         <options>     <dump>       <pass>
...
/dev/mmcblk0p3  /mnt/files  ext4    defaults    0   1
```

Moreover we need to ensure that our application start on system boot. Hence, we will create an overlay on a /etc/init.d folder with a file named S50filebrowser:

```
#!/bin/sh
start() {
  cd /home/filebrowser
  ./run.sh
}
case "$1" in
  start)
    start
    ;;
  *)
    echo "Usage: $0 {start}"
esac
```

Bootloader

We need to setup a functionality which allows user to load different image (either zImage_admin or zImage_user) depending on button click during boot.

Enter rescue mode and copy properly prepared boot.txt file from host machine. Make sure we have set

root=/dev/mmcblk0p2 rootwait

in bootargs.

Then compile it using:

```
$ mkimage -T script -C none -n 'Start script' -d boot.txt boot.scr
```

Now, when everything is finished we can reboot and enjoy uploading files through web server running on our Raspberry Pi!

Solution files:

1. ./admin/.config
2. ./user/.config
3. ./user/overlay - contains filebrowser application and other files
4. ./boot.txt

References:

https://buildroot.org/downloads/manual/manual.html