Warsaw, 25.04.2021

Linux for Embedded Systems - Laboratory ex. 3

Emilia Wróblewska 291674

Problem Description

The aim of the third assignment is to create a design with:

- separate „administrative" system (used for maintenance or as a rescue system in real embedded systems), and the „utility" system (used for normal operation).
- the bootloader enabling the selection of the system to be started.

The „Admin" system must use initramfs compiled into kernel to allow modification of the SD card. The „Utility" system should offer the permanent storage, and therefore it should use the root file system located in the second partition of the emulated SD card.
The selection of the started system should be done with the GUI buttons as described below. The templates (Admin) and (Utility) available in the demo project (BR_Internet_Radio repository, branch gpio_uboot_2021.02 or the example for Lecture 6 in Moodle) may be used as a starting point. Below are the detailed tasks:

1. Prepare the "administrative" Linux system similar to the „rescue" system used in our normal lab. This system should be:

1) working in initramfs;

2) equipped with the tools necessary to manage the SD card (partition it, format it, copy the new version of the system via a network, etc.).

2. Prepare a "utility" Linux operating system using the ext4 filesystem located in the 2nd partition as its root file system. This system should provide a file server controlled via WWW interface (suggested solution is Python with a certain web application framework – flask, Tornado etc.. However, you may choose another environment as well).

1) The server should serve files located in a certain directory (displaying the list of files and allowing selection of file to download).

2) The server should also allow the authenticated users to upload new files to that directory.

3. Prepare a script for the bootloader, which allows to select which system (the „administrative" or the „utility") should be booted.

1) Unfortunately, the current emulation of the GUI-connected GPIO does not correctly send the initial state of the inputs. Therefore the user must have certain time to press the „reconnect" button in the GUI as soon as the U-Boot starts.

2) Due to certain problems with current emulation of GPIO, the first change of the state of the LED is ignored. Please clear all GPIOs connected to the LEDs you are going to use, so the next operations will work correctly.

3) After two seconds the LED 24 should signal that the buttons will be checked.

4) After the next second, the buttons should be read to select the system. If the chosen button IS NOT pressed, the „utility" system should be loaded. If the chosen button IS pressed, the „administrative" system should be loaded.

5) After selection of the system, the LED 24 should be switched off. The LED 25 should be switched ON if the „utility" system was selected. The LED 26 should be switched ON if the „administrative" system was selected.

Procedure to recreate the design from the archive

Firstly, navigate into the "Utility" directory and run "build.sh" script to build the image of the Utility system. Then, run the "copy_to_admin.sh" script, change the working directory to "Admin/user_img" and run the "boot_user.sh" script. Finally, navigate back to the "Admin" directory and run "build.sh" script to build the Admin system. If the step finished without any errors, the buildroot is correctly created. Now to start the solution, firstly run the "run_before" script to launch the GUI and then in another terminal "runme" script to start the machine.

(Note: Due to some unknown reasons, when choosing which system to boot, I had to click the “Reconnect” button in GUI several times in order to activate the Utility system (without “Reconnect” the Admin system was started). On the contrary, if I wanted to choose the Admin system, clicking “Reconnect” once and then button 12 only once as well was enough.)

Description of the solution

1) I used the given templates of Admin and Utility system from moodle archive as a starting point. Firstly, I prepared the configuration of both systems by running “/user_img/boot_admin.sh” and then “build.sh” script in “Admin” and “build.sh” in the “Utility” directory (I needed to install libssl-dev and mtools package for Admin system to build correctly). After the images are built, I checked if required tools (resize2fs and fsck) are installed in the buildroot and I can start playing with partitions of the emulated SD card. I navigate to the “Admin/buildroot-2021.02/output/images” directory where the sdcard.img is located. Then I run:

```
$ truncate -s 500M sdcard.img
```

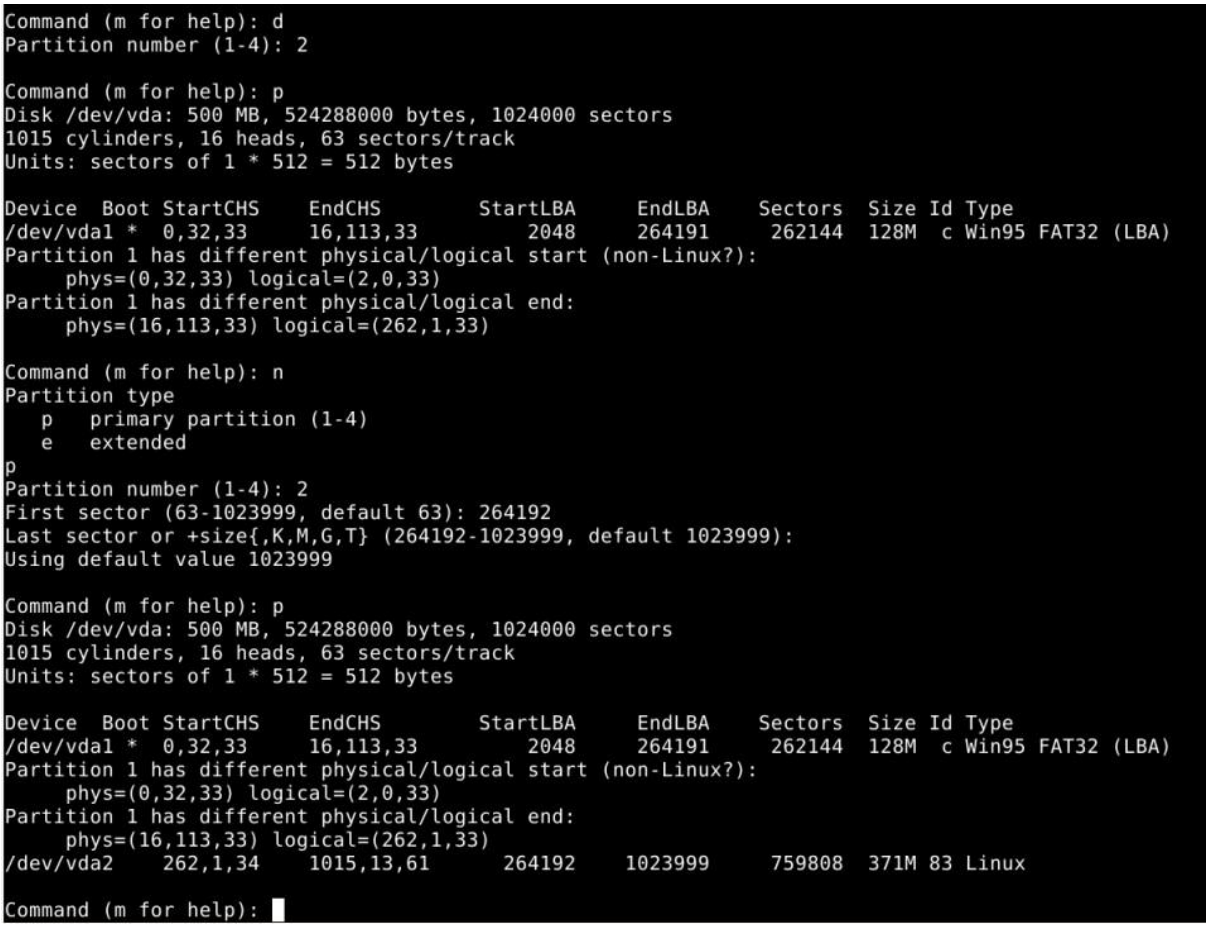
To change the size of the image. It is necessary to see the difference when modifying the partitions on the card. Then I navigate to the “Admin” directory and run the Admin system using the “runme” script. After starting the machine I run:

```
# fdisk -u /dev/vda
```

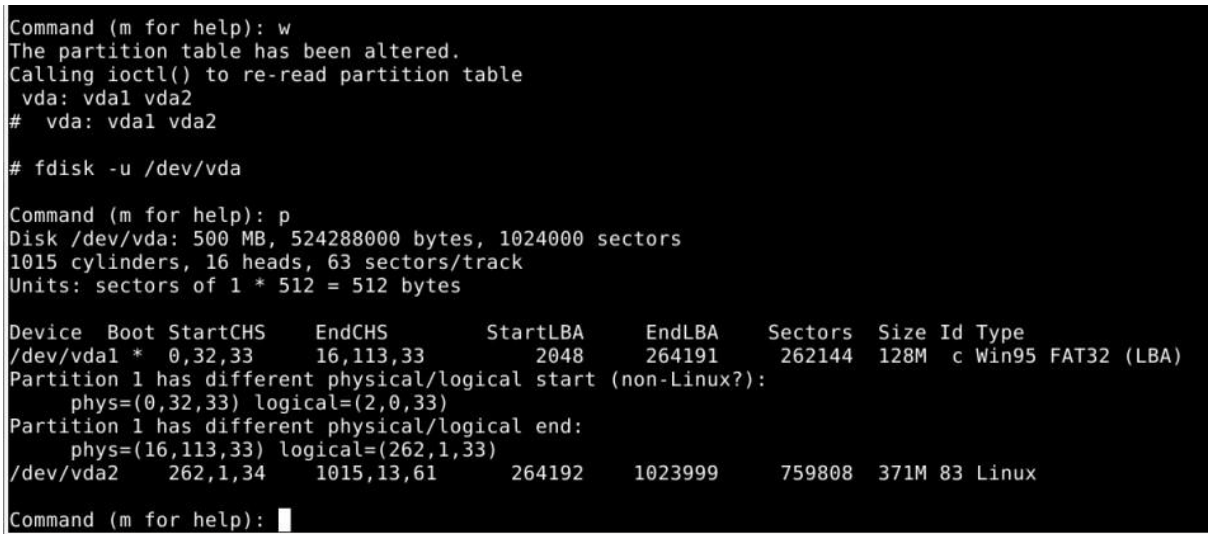
I can see 2 partitions with sizes 128M and 200M which together sum up to ≈ 336M, the previous size of the card.

```
Welcome to Buildroot
buildroot login: root
# fdisk -u /dev/vda

Command (m for help): p
Disk /dev/vda: 329 MB, 344981504 bytes, 673792 sectors
668 cylinders, 16 heads, 63 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device  Boot StartCHS    EndCHS        StartLBA     EndLBA    Sectors  Size Id Type
/dev/vda1 *  0,32,33     16,113,33         2048     264191     262144  128M  c Win95 FAT32 (LBA)
Partition 1 has different physical/logical start (non-Linux?):
     phys=(0,32,33) logical=(2,0,33)
Partition 1 has different physical/logical end:
     phys=(16,113,33) logical=(262,1,33)
/dev/vda2    16,113,34   41,240,7        264192     673791     409600  200M 83 Linux
Partition 2 has different physical/logical start (non-Linux?):
     phys=(16,113,34) logical=(262,1,34)
Partition 2 has different physical/logical end:
     phys=(41,240,7) logical=(668,7,7)
```

Then I can delete the second partition and create a new one in its place, occupying the rest of available space.



Now the size of the new partition is 371M. I save the changes and check them.



Then I resize the filesystem, so it could take the available space. After the filesystem was modified, I checked it once again using “fsck.ext4” command.

```
# resize2fs /dev/vda2
resize2fs 1.45.6 (20-Mar-2020)
Please run 'e2fsck -f /dev/vda2' first.

# e2fsck -f /dev/vda2
e2fsck 1.45.6 (20-Mar-2020)
rootfs: recovering journal
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 3A: Optimizing directories
Pass 4: Checking reference counts
Pass 5: Checking group summary information

rootfs: ***** FILE SYSTEM WAS MODIFIED *****
rootfs: 1716/51200 files (0.9% non-contiguous), 59978/204800 blocks
# resize2fs /dev/vda2
resize2fs 1.45.6 (20-Mar-2020)
Resizing the filesystem on /dev/vda2 to 379904 (1k) blocks.
The filesystem on /dev/vda2 is now 379904 (1k) blocks long.

# fsck.ext4 -f /dev/vda2
e2fsck 1.45.6 (20-Mar-2020)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
rootfs: 1716/96256 files (0.9% non-contiguous), 66170/379904 blocks
```

The last thing I tried to do was to replace the Utility system without doing full rebuild. To do that firstly, I navigate to the directory "Utility/buildroot-2021.02/output/images" where the Utility system - "rootfs.ext4" is located. I pack it using the command

$ gzip -k rootfs.ext4

and host it locally for my machine using python server in the same directory:

$ python3 -m http.server

I copy the new Image of Utility system to the first partition of SD card using:

wget -O - http://10.0.2.2:8000/rootfs.ext4.gz

And then install the new filesystem on the second partition and resize it:

wget -O - http://10.0.2.2:8000/rootfs.ext4.gz | gzip -d | dd of=/dev/vda2

```
# wget -O - http://10.0.2.2:8000/rootfs.ext4.gz | gzip -d | dd of=/dev/vda2
Connecting to 10.0.2.2:8000 (10.0.2.2:8000)
writing to stdout
-                    100% |********************************| 17.9M  0:00:00 ETA
written to stdout
409600+0 records in
409600+0 records out
#
```

```
# resize2fs /dev/vda2
resize2fs 1.45.6 (20-Mar-2020)
Resizing the filesystem on /dev/vda2 to 379904 (1k) blocks.
The filesystem on /dev/vda2 is now 379904 (1k) blocks long.

# fsck.ext4 -f /dev/vda2
e2fsck 1.45.6 (20-Mar-2020)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
rootfs: 1721/96256 files (0.9% non-contiguous), 66206/379904 blocks
#
```

Unfortunately, this part of the solution is not included in the archive because it would be very difficult to recreate the performed actions automatically. Hence, I included screenshots of consecutive steps above.

2) In this part I added a flask web application to the "Utility" system. To do this, firstly I need to add flask package to the buildroot configuration:

> Target packages → Interpreter languages and scripting → External python modules → python-flask

Now in order to apply the changes I needed to run $make clean all (with simple "make" not all python libraries were linked correctly). Then, I added the file "server.py" which implements my server to the "overlay/server" directory. In the same folder I put a "/files" folder with several dummy files for the server to host. After building the Utility image and running the machine I can now run the server executing the commands:

```
# cd  /server
# python3 server.py
```

Since the "runme" script for the Utility system already has the forwarding of the port, I host my server on port 22. Thanks to that the server is available at localhost:2222, ready to download (by entering the file name into input) and upload files (login credentials are "admin" and "password").

```
if __name__ == '__main__':

    app.run(debug=True, port=22, host='0.0.0.0')
```

The last step is to add a script which will run my server at the start of the Utility system. To do that, I create a script "S99initserver" and add it to "overlay/etc/init.d" directory.

```
#!/bin/sh
start() {
    printf "Starting utility server..."
    cd /server
    python3 server.py
}
stop() {
    printf "Server stopped."
    killall python3
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart|reload)
        stop
        sleep 1
        start
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac
```

3) To complete this part I modified the “boot_user.txt” file located in “Admin/user_img” directory to the following:

```
sleep 5
gpio clear 24
gpio clear 25
gpio clear 26
gpio set 24
sleep 1

if gpio input 12; then
  gpio clear 24
  gpio set 26
  setenv bootargs "console=ttyAMA0"
  setenv fdt_addr_r 0x40000000
  load virtio 0:1 $kernel_addr_r Image
  booti $kernel_addr_r - $fdt_addr_r
else
  gpio clear 24
  gpio set 25
  setenv bootargs "console=ttyAMA0 root=/dev/vda2 rootwait"
  setenv fdt_addr_r 0x40000000
  load virtio 0:1 $kernel_addr_r Image_user
  booti $kernel_addr_r - $fdt_addr_r
fi
```

The script waits 5 seconds for reconnecting and then waits 1 seconds for clicking the button 12. If button 12 is clicked, the administrator system is started and if there is no input the utility system is booted. The LEDs show the status of the machine – LED 24 lights up at the beginning during the reconnection period. If the button is clicked, LED 26 indicates that the “Admin” system is loading, otherwise LED 25 is lighted up and signalizes that the “Utility” system is booting.