Warsaw 17.04.2020

Linux for Embedded Systems – Laboratory ex. 3

The student: Patryk Walczak

Description of the assignment

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 permament 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 given templates (Admin) and (Utility) 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 certain web application framework – flask, Tornado etc.. However, you may chose another environment as well).
   1. The server should serve files located in 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 attached archive

Firstly, open the Utility directory and start „build.sh", after that copy the build root to 'Admin' by „copy_to_admin.sh". After the step, change the current folder to Admin/user_img and run „boot_user.sh". The last step of the build is to run the „build.sh" in the Admin directory. If the step finished without any error the buildroot is correctly created. If you want to start the solution start „run_before" and after that „runme".

Description of the solution

I used the given templates (Admin) and (Utility) as a starting point, and firstly I prepared Utility configuration with the build script. Then, using „copy_to_admin.sh” I copied the required files (e.g. zImage) to Admin build. Then using script „boot_user” I changed the boot options. In the paragraph containing information about third task the file „boot_user.txt” is described, which had to be changed to do the last task. Then with „build.sh” script in Admin directory I created the build. After that step, I checked if the base configuration works correctly.

At the moment almost all the requirements of the first task are in the buildroot. However, to the administrative system, I added mkfs, fsck, resize2fs.

Target packages → Filesystem and flash utilities → dosfstools → mkfs
Target packages → Filesystem and flash utilities → e2fsprogs → fsck
Target packages → Filesystem and flash utilities → e2fsprogs → resize2fs

The first task is done.

Then I prepared flask web application, firstly I added to utility

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

And I repeated the steps of copying the changes to the administrative system.

The first endpoint „/” which shows all files in the current directory of the server:

```
@app.route("/")
def hello():
    entries = os.listdir('.')
    output=""
    for entry in entries:
        if(output!=""):
            output+=", "

        output+= entry
    return output
```

Here I prepared two solutions, the first one established upper, and the second one which is not used in the final version of the application.

```
 # stream = os.popen("ls")
# output = stream.read()
# return output
```

Both are working correctly and return the list of files in the current location. It can be changed to some directory. After that, I programmed the bigger part – to log in and upload a file. For that in the application are „/login”, „/upload” and „/logout” endpoints. The first one is used to log in the user with „admin” and „password” as the name and password. Then the user has access to the „/upload” endpoint, can choose the file and send it to the server. Then with the main „/” route user can check if a new file was correctly uploaded. In the end, the user can log out with the last mentioned endpoint.

In our case, when we use the virtual machine I had to change the runme script. I added „-net user,hostfwd=tcp::2222-:22” to forward the port to host and establish the access to the web application on our utility system. With the option, the address of the web service is http:/$HOST_IP$:2222/.

The third task I done with changing the „boot_user.txt” file.

```
#Below you may add additional options, like: root=/dev/mmcblk0p2 rootwait
setenv fdt_addr_r $ramdisk_addr_r
setenv kernel_addr_r 0x61000000
setenv fdt_addr_r 0x80008000
sleep 2
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"
 load mmc 0:1 $fdt_addr_r vexpress-v2p-ca9.dtb
 load mmc 0:1 $kernel_addr_r zImage
 bootz $kernel_addr_r - $fdt_addr_r
else
 gpio clear 24
 gpio set 25
 setenv bootargs "console=ttyAMA0 root=/dev/mmcblk0p2 rootwait"
 load mmc 0:1 $fdt_addr_r vexpress-v2p-ca9-user.dtb
 load mmc 0:1 $kernel_addr_r zImage_user
 bootz $kernel_addr_r - $fdt_addr_r
fi
```

The script waits 2 seconds for reconnecting and then waits 1 second for clicking the button 12. If clicked start up the administrator system otherwise the utility system starts. The LEDs show the status of the buildroot. At the beginning the 24 is lighting. And if the button was clicked and the ‘Admin’ system is loading the LED number 26 is lighting in other case the 25th LED is shining and ‘Utility’ system is establishing.