

Linux w systemach wbudowanych

Laboratorium L3 G1

Sprawozdanie

Adam Grącikowski, 327350

Warszawa, 23 kwietnia 2025

Spis treści

1	Cel ćwiczenia laboratoryjnego	2
2	Wymagania	2
3	Opis modyfikacji i konfiguracji Buildroota	3
3.1	Wspólna konfiguracja dla systemu Admin i User	3
3.2	Konfiguracja dla systemu Admin	3
3.3	Konfiguracja dla systemu User	4
4	Opis plików konfiguracyjnych i skryptów dołączonych do archiwum	4
4.1	<i>Overlay</i> na system User	4
4.2	Logika Bootloader-a	5
5	Opis implementacji serwera WWW	5
6	Uruchamianie	6

1 Cel ćwiczenia laboratoryjnego

Celem ćwiczenia jest:

- zapoznanie się z bootloader'em U-Boot,
- implementacja prostego serwera uruchamianego na platformie Raspberry Pi.

2 Wymagania

- System administratora **Admin** powinien:
 - Pracować z **initramfs**.
 - Zawierać narzędzia do zarządzania kartą SD (partycjonowanie, formatowanie, kopiowanie poprzez sieć, naprawianie systemu plików itp.)
- Karta SD powinna zostać podzielona na 3 partycje:
 - VFAT z systemem ratunkowym (w katalogu **rescue**), oraz administracyjnym i jądrem systemu użytkowego (w katalogu **user**).
 - **ext4** z systemem plików systemu użytkowego.
 - **ext4** z danymi systemu użytkowego.
- System użytkowy **User** powinien:
 - Pracować z systemem plików **ext4** na drugiej partycji.
 - Zawierać serwer WWW, zrealizowany przy pomocy środowiska **Tornado** języka **Python**:
 - * udostępniający pliki z partycji 3 na karcie SD - wyświetlający listę tych plików,
 - * pozwalający na wybranie pliku do pobrania,
 - * umożliwiający uwierzytelnionym użytkownikom wgrywanie nowych plików na tę partycję.
- Bootloader powinien umożliwiać określenie, który system (**Admin** czy **User**) ma zostać załadowany. Działanie bootloadera powinno być sygnalizowane w następujący sposób:
 - Żółta dioda powinna sygnalizować, że za chwilę sprawdzony zostanie stan przycisku.
 - Po sekundzie odczytywany jest stan przycisku. Jeżeli nie jest wciśnięty, powinien zostać załadowany system **User**.
 - Po sprawdzeniu przycisku żółta dioda powinna zostać zgaszona. Jeżeli został wybrany system **User**, powinna zostać zapalona zielona dioda, a jeżeli został wybrany system **Admin**, powinna zostać zapalona dioda czerwona.

3 Opis modyfikacji i konfiguracji Buildroota

3.1 Wspólna konfiguracja dla systemu Admin i User

1. Pobranie środowiska Buildroot, rozpakowanie oraz dokonanie wstępnej konfiguracji poprzez:
`make raspberrypi4_64_defconfig`.
2. System configuration > Run a getty (login prompt) after boot > TTY port ustawione na console.
3. Build options > Mirrors and Download locations > Primary download site ustawione na `http://192.168.137.24/dl`.
4. Toolchain > Toolchain type ustawione na External toolchain.
5. Build options > Enable compiler cache zaznaczone i Compiler cache location ustawione na `/malina/gracikowskia/ccache-br`.

3.2 Konfiguracja dla systemu Admin

1. Filesystem images zaznaczona opcja initial RAM filesystem linked into linux kernel oraz Compression method ustawione na gzip.
2. System configuration > Enable root login and password > Root password ustawione na przykładowe hasło admin,123.
3. System configuration > System banner ustawione na Welcome Admin.
4. Target packages > BusyBox zaznaczona opcja Show packages that are also provided by busybox.
5. Target packages > Networking applications zaznaczona opcja dhcp (ISC) oraz dhcp client, a także netplug, ntp i ntpd.
6. System configuration > System hostname ustawione na admin.
7. Target packages > Filesystem and flash utilities zaznaczona opcja dosfstools, a w niej opcje fatlabel, fsck.fat oraz mkfs.fat. Dodatkowo zaznaczona opcja e2fsprogs, a w niej fsck oraz resize2fs.
8. Filesystem images > ext2/3/4 root filesystem zaznaczona opcja additional mke2fs options.
9. Target packages > System tools > util-linux zaznaczona opcja mount/umount
10. Target packages > Libraries > Javascript > flot zaznaczona opcja resize. Target packages > Hardware handling > u-boot tools zaznaczona opcja mkimage.
11. Host utilities zaznaczona opcja host imx-mkimage.
12. Bootloaders zaznaczona opcja U-Boot oraz Board defconfig ustawione na rpi_4.

3.3 Konfiguracja dla systemu User

1. System configuration > Enable root login and password > Root password ustawione na przykładowe hasło `user,123`.
2. System configuration > System banner ustawione na `Welcome User`.
3. System configuration > System hostname ustawione na `user`.
4. Filesystem images > ext2/3/4 variant ustawione na `ext4`.
5. W pliku `board/raspberrypi4-64/genimage.cfg.in` parametr `size` ustawiony na `256M`.
6. System configuration > Root filesystem overlay directories ustawione na `board/overlay`.
7. Target packages > Networking applications zaznaczona opcja `dhcp` (ISC) oraz `dhcp client`, a także `netplug`, `ntp` i `ntpd`.
8. Target packages > System tools zaznaczona opcja `start-stop daemon`.
9. Target packages > Interpreter languages and scripting zaznaczona opcja `python3` oraz w External python modules zaznaczone opcje `python-tornado` i `python-urllib3`.

4 Opis plików konfiguracyjnych i skryptów dołączonych do archiwum

4.1 *Overlay* na system User

Rysunek 1 przedstawia strukturę plików dodanych w ramach nakładki (ang. *overlay*) na system User.

```
/board
├── overlay/
│   ├── usr/
│   │   ├── file-server/
│   │   │   ├── ...
│   │   │   └── server.py
│   └── etc/
│       ├── init.d/
│       │   ├── S98mount
│       │   └── S99server
```

Rysunek 1: Struktura plików nakładce *overlay*

Folder `file-server` zostanie omówiony w rozdziale 5. Najważniejszym jego elementem jest jednak plik `server.py`, zaznaczony na rysunku 1, do którego odwołuje się skrypt `S99server`.

Skrypt ten jest odpowiedzialny za zarządzanie uruchamianiem i zatrzymywaniem serwera (w naszym przypadku serwera opartego na `Tornado`).

Skrypt `S98mount` montuje trzecią partycję z urządzenia `mmcblk0p3` do katalogu `/mnt` przy pomocy `mount`:

```
mount /dev/mmcblk0p3 /mnt
```

4.2 Logika Bootloader-a

Dynamiczny wybór trybu pracy (Admin/User) został osiągnięty za pomocą jednego przycisku i wizualne potwierdzenie diodami. Logika ta znajduje się w pliku `boot.txt`.

Skrypt znajdujący się w pliku `boot.txt` został zmodyfikowany poleceniem:

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

5 Opis implementacji serwera WWW

Implementacja logiki serwera WWW została umieszczona w projekcie `file-server` o strukturze przedstawionej na rysunku 2.

```
/file-tree
├── templates/
│   ├── index.html
│   ├── items.html
│   ├── login.html
│   └── login_failed.html
├── handlers/
│   ├── base.py
│   ├── auth.py
│   ├── files.py
│   └── __init__.py
├── file_tree.py
├── routes.py
└── server.py
```

Rysunek 2: Struktura plików w projekcie `file-tree`

Główny skrypt uruchamiany podczas startu systemu to `server.py`. Przyjmuje on dwa pozycyjne argumenty - `port` o domyślnej wartości 8888 oraz `address` o domyślnej wartości 127.0.0.1.

Funkcja `make_app()` jest odpowiedzialna za stworzenie i konfigurację obiektu `Application` z modułu `tornado.web`. Funkcja `main()` jest natomiast odpowiedzialna za uruchomienie serwera oraz ustawienie obsługi sygnału `SIGINT` oraz `SIGTERM` w celu ładnego zakończenia działania serwera w przypadku nadejścia sygnału.

W pliku `routes.py` znajdują się stałe określające obsługiwane ścieżki API serwera (klasa `Routes`).

W pliku `file_tree.py` znajduje się funkcja `build_file_tree()` odpowiedzialna za rekurencyjne zbudowanie drzewa plików partycji.

W folderze `handlers` znajdują się implementacja odpowiednio logiki prostej autoryzacji użytkownika oraz operacji na plikach.

Zawartość wszystkich plików wymienionych na rysunku 2 została udostępniona wraz ze sprawozdaniem.

6 Uruchamianie

Po skonfigurowaniu obrazu systemu **Admin**, został on wygenerowany przy pomocy polecenia **make**. Następnie na kartę SD przesłane zostały pliki **u-boot.bin** oraz **Image** z podfolderu **output/images** głównego katalogu Buildroot'a. Nazwy tych plików zostały zmienione odpowiednio na **Image** oraz **Image.admin**.

Aby uruchomić system **Admin** należało uruchomić polecenie **reboot**, a następnie wywołać ręcznie z konsoli odpowiednie linie z pliku **boot.txt**, prowadzące do załadowania obrazu **Image.admin**.

Kolejnym krokiem było pobranie na kartę SD przygotowanego wcześniej pliku **boot.txt** i zmodyfikowanie go przy pomocy polecenia:

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

Plik **boot.scr** został następnie umieszczony w katalogu głównym pierwszej partycji.

Po skonfigurowaniu obrazu systemu **User**, został on wygenerowany przy pomocy polecenia **make**. Następnie na kartę SD przesłane zostały pliki **Image** oraz **rootfs.ext2** z podfolderu **output/images** głównego katalogu Buildroot'a.

Nazwa pliku z obrazem systemu **User** została zmieniona na **Image.user**.

Kolejnym krokiem było stworzenie i sformatowanie partycji zgodnie z wymaganiami. Stworzenie partycji zrealizowano przy pomocy interaktywnego narzędzia **fdisk**:

```
fdisk /dev/mmcblk0
D
3
D
2
N
P
2
<enter>
+500M
N
P
3
<enter>
+500M
W
```

Partycje zostały następnie sformatowane:

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

dane przeniesione:

```
dd of=/dev/mmcblk0p2 if=rootfs.ext2 bs=4096
```

a rozmiar dopasowany:

```
resize2fs /dev/mmcblk0p2
```

Literatura

- [1] dr hab. inż. Wojciech Zabołotny, prezentacja do *Wykładu 6*.
- [2] Biblioteka Tornado - <https://www.tornadoweb.org/en/stable/>, dostęp 23 kwietnia 2025
- [3] Szkielet skryptu zarządzającego serwerem - <https://gist.github.com/mrowe/8b617a8b12a6248d48b8>,
dostęp 23 kwietnia 2025
- [4] Podział i formatowanie dysku w trybie tekstowym - <https://linuxiarze.pl/partycje4/>, dostęp
23 kwietnia 2025
- [5] Resize2fs Command - <https://www.thegeekdiary.com/resize2fs-command-examples-in-linux/>,
dostęp 23 kwietnia 2025