Linux w systemach wbudowanych – Laboratorium 4

Tymon Felski

13 maja 2017

1 Treść zadania

Podczas czwartego laboratorium należało przygotować przy pomocy płytki Raspberry Pi urządzenie wyposażone w złożony interfejs użytkownika, w którym:

- przyciski i diody LED powinny być użyte do podstawowej obsługi urządzenia,
- interfejs WWW lub inny interfejs sieciowy powinien być użyty do bardziej zaawansowanych funkcji,
- przetwarzany powinien być dźwięk lub obraz.

2 Odtwarzanie projektu z załączonego archiwum

Dostarczone archiwum należy umieścić w dowolnym miejscu na dysku i rozpakować. Uruchomienie skryptu `install.sh`, znajdującego się w środku archiwum, spowoduje odtworzenie konfiguracji środowiska z laboratorium. Jako parametr należy podać ścieżkę do katalogu z rozpakowanymi plikami Buildroota.

Skrypt utworzy katalog `felskit-lab4/` równoległy do katalogu przekazanego jako argument. Następnie do obu z nich zostaną przekopiowane odpowiednie pliki i katalogi. Ponadto, skrypt zastosuje domyślną konfigurację dla płytki Raspberry Pi i właściwie spatchuje pliki konfiguracyjne Buildroota. Na koniec skrypt zapyta czy rozpocząć kompilację jądra systemu.

3 Opis rozwiązania

3.1 Przygotowanie

W celu przygotowania środowiska Buildroot, ustawiono następujące opcje:

1. *System configuration* → *Port to run a getty (login prompt) on* na *ttyAMA0*
2. *Build options* → *Mirrors and Download locations* → *Primary download site* na *http://192.168.137.24/dl*
3. *Target options* → *Target ABI* na *EABI*
4. *Toolchain* → *Toolchain type* na *External toolchain*
5. *Toolchain* → *Toolchain* na *Sourcery CodeBench ARM 2014.05*

Ponadto, zaznaczono opcję

Filesystem images → *initial RAM filesystem linked into linux kernel*

oraz włączono kompresję obrazu ustawiając opcję

Filesystem images → *tar the root filesystem, Compression method (gzip)*

3.2 Zadania

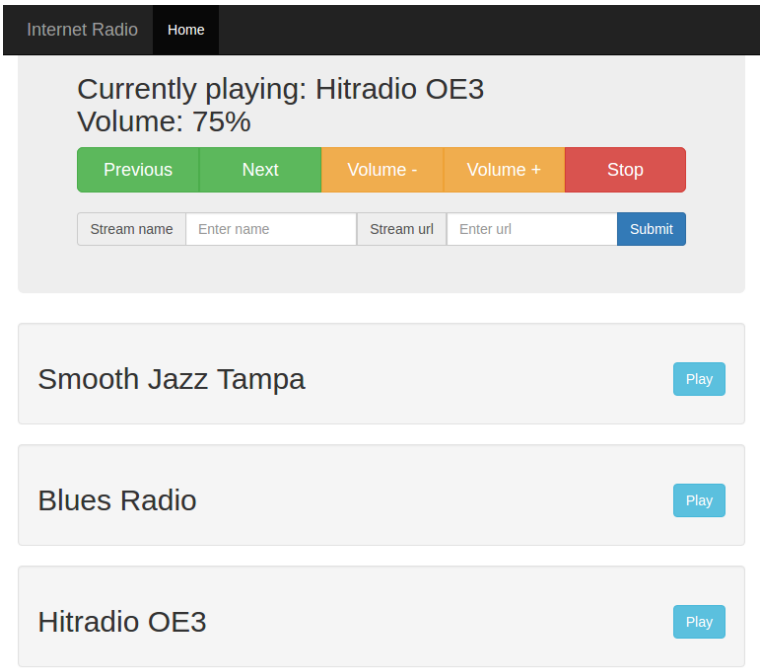
3.2.1 Aplikacja

Stworzona w ramach laboratorium aplikacja to internetowe radio napisane w Pythonie, którym można sterować przy pomocy interfejsu `GPIO` oraz interfejsu webowego. Do obsługi przycisków i diod LED na płytce Raspberry Pi wykorzystano pythonowy moduł `RPi.GPIO`, natomiast interfejsem webowym zarządza framework `Flask`.

Przycisk o numerze pinu 27 służy do zmiany stacji na następną, a przyciski o numerach 10 i 22 odpowiednio zmniejszają i zwiększają głośność odtwarzania. Interfejs webowy jest bardziej rozbudowany i udostępnia dodatkowe funkcjonalności, takie jak zmiana stacji na poprzednią, wybranie konkretnej stacji z listy oraz całkowite zatrzymanie streamu.

Oba interfejsy wyświetlają aktualnie graną stację radiową i głośność odtwarzania. Diody LED na płytce Raspberry Pi odzwierciedlają obecną głośność odtwarzania, gdzie przy głośności 0% wszystkie diody będą zgaszone, a przy 100% - zapalone.

Interfejs webowy, zdefiniowany w pliku `templates/main.html`, wykorzystuje `jQuery` i style `Bootstrapowe`, dzięki czemu jest responsywny. Dodatkowa część zadania wymagała dodania `HTML-owego` formularza, przyjmującego dwa napisy, do interfejsu webowego oraz obsługi odpowiedniego zapytania typu `POST` po stronie serwera. Po wpisaniu nazwy stacji i adresu pod którym można znaleźć stream oraz wciśnięciu przycisku `Submit`, stacja zostanie dodana do listy widocznej pod głównym panelem.



Rysunek 1: Zrzut ekranu interfejsu webowego

Stacje radiowe są wczytywane z pliku `stations.txt` na początku działania aplikacji. Muzyka jest odtwarzana przy pomocy `mpc`, prostego klienta do `mpd` (*Music Player Deamon*), który z kolei jest skonfigurany do współpracy z `ALSA` (*Advanced Linux Sound Architecture*). Pythonowe funkcje obsługujące odtwarzacz korzystają z pomocniczej funkcji `run_cmd`, która przy pomocy modułu `subprocess` tworzy proces potomny do wykonywania komend w linii poleceń.

```
def run_cmd(cmd):
    p = Popen(cmd, shell=True, stdout=PIPE, stderr=STDOUT)
    output = p.communicate()[0]
    return output
```

Listing 1: Funkcja pomocnicza run_cmd

Przygotowano trzy wersje aplikacji. Pierwsza, której źródło znajduje się w pliku `radio-blocking.py`, działa na czterech wątkach, z których trzy obsługują przyciski na płytce Raspberry Pi przy pomocy blokujących funkcji `wait_for_edge`, a na czwartym działa serwer `Flaskowy`. Niestety ze względu na występujący obecnie problem w module obsługującym interfejs `GPIO`, nie jest możliwe, aby wszystkie trzy wątki czekały jednocześnie na wciśnięcie przycisku.

```
def gpio10_handler():
    while True:
        GPIO.wait_for_edge(10, GPIO.FALLING)
        with lock:
            mpc_vol_down()
            mpc_print()

def gpio22_handler():
    while True:
        GPIO.wait_for_edge(22, GPIO.FALLING)
        with lock:
            mpc_vol_up()
            mpc_print()

def gpio27_handler():
    while True:
        GPIO.wait_for_edge(27, GPIO.FALLING)
        with lock:
            mpc_next()
            mpc_print()
```

Listing 2: Funkcje obsługujące interfejs GPIO (z radio-blocking.py)

Druga wersja aplikacji, która znajduje się w pliku `radio-polling.py` jest modyfikacją poprzedniej. Interfejs `GPIO` jest obsługiwany przez jeden wątek, który odpytuje przyciski pięć razy na sekundę. Pozostała część aplikacji została zrealizowana w ten sam sposób.

```
def poll_btns():
    if GPIO.input(10) == GPIO.LOW:
        return "down"
    if GPIO.input(22) == GPIO.LOW:
        return "up"
    if GPIO.input(27) == GPIO.LOW:
        return "next"
    return "none"

def gpio_handler():
    global exit
    while not exit:
        pressed = poll_btns()
        if pressed != "none":
            with lock:
                if pressed == "down":
                    mpc_vol_down()
                elif pressed == "up":
                    mpc_vol_up()
                else:
                    mpc_next()
                mpc_print()
        time.sleep(0.2)
```

Listing 3: Funkcje obsługujące interfejs GPIO (z radio-polling.py)

Trzecia wersja, znajdująca się w pliku `radio-events.py`, korzysta z jednego wątku do obsługi serwera, jednak przed jego uruchomieniem zapisuje przy pomocy `add_event_detect` trzy funkcje obsługujące interfejs `GPIO` na eventy odpowiadające wciśnięciu odpowiednich przycisków. W ten sposób w aplikacji nie ma oczekiwania aktywnego, a callbacki wywołają się na dodatkowych wątkach w przypadku wciśnięcia któregoś z przycisków.

```
def gpio10_handler():
    with lock:
        mpc_vol_down()
        mpc_print()

def gpio22_handler():
    with lock:
        mpc_vol_up()
        mpc_print()

def gpio27_handler():
    with lock:
        mpc_next()
        mpc_print()
```

Listing 4: Funkcje (callbacki) obsługujące interfejs GPIO (z radio-events.py)

Wszystkie wersje zostały wyposażone w handler przerwania. Funkcja `register` z modułu `atexit` zapisuje przekazają jako argument funkcję `interrupt` na event odpowiadający wciśnięciu `^C`. Odpowiada ona za wyczyszczenie zasobów, czyli przywrócenie początkowych stanów pinów `GPIO` oraz zatrzymanie odtwarzania muzyki. Wątki w wersjach 1. i 2. są uruchamiane z parametrem `daemon = True`, który powoduje, że główny wątek nie musi czekać na ich zakończenie (zostają one zabite razem z nim). Mimo to w wersji 2. zastosowano zakończenie wątku przy pomocy flagi logicznej i wywołanie funkcji `join`. W wersji 3. funkcja `interrupt` dodatkowo wyrejestrowuje callbacki obsługujące interfejs `GPIO`.

```
def interrupt():
    for button in [10, 22, 27]:
        GPIO.remove_event_detect(button)
    GPIO.cleanup()
    mpc_stop()
```

Listing 5: Przykładowy handler przerwania (z radio-events.py)

Serwer `Flaskowy` działa na zasadzie komunikacji poprzez zapytania. Do danych ścieżek przypisane są pythonowe funkcje. Zostają one wywołane, przetwarzają zapytanie i odpowiadają na nie, co skutkuje wyświetleniem odpowiedniej strony w przeglądarce po stronie użytkownika. Wszystkie funkcje są zabezpieczone przy pomocy mechanizmu `Reentrant Lock`, aby aplikacja mogła działać równolegle z wątkami obsługującymi interfejs `GPIO`.

3.2.2 Obraz systemu

Obraz systemu musiał zostać wyposażony w szereg narzędzi i paczek, aby mógł odtwarzać dźwięk i uruchomić wcześniej opisywaną aplikację. Dodanie interpretera języka skryptowego Python do obrazu systemu polega na zaznaczeniu opcji

Target packages → *Interpreter languages and scripting* → *python3*

Następnie zaznaczono opcje odpowiadające niezbędnym pythonowym modułom

Target packages → *Interpreter languages and scripting* → *External python modules* → *python-rpi-gpio*
Target packages → *Interpreter languages and scripting* → *External python modules* → *python-flask*

Odtwarzacz muzyki `mpd` wraz z wymaganym klientem `mpc` można znaleźć pod opcjami

Target packages → *Audio and video applications* → *mpd*
Target packages → *Audio and video applications* → *mpd-mpc*

Aby `mpc` mógł obsługiwać streamy stacji radiowych z internetu wymagany jest plugin `curl`, który można wybrać zaznaczając

Target packages → *Audio and video applications* → *mpd (Input plugins)* → *curl*

Współdziałanie `mpd` oraz `ALSA` zapewni opcja

Target packages → *Audio and video applications* → *mpd (Output plugins)* → *alsa*

Wybranie powyższej opcji powinno automatycznie zaznaczyć

Target packages → *Libraries* → *Audio/Sound* → *alsa-lib*

Konieczne jest także dodanie

Target packages → *Audio and video applications* → *alsa-utils*

Wybrano ponadto pomocnicze paczki potomne `alsa-utils`, takie jak *alsaconf*, *alsactl*, *alsamixer*, *amixer* oraz *aplay*.

Aplikację umieszono w obrazie systemu przy pomocy mechanizmu Overlay. W opcji

System configuration → *Root filesystem overlay directories*

podano ścieżkę do katalogu `overlay/`, do którego zostały skopiowane wszystkie pliki źródłowe.

Konfiguracja `mpd` wymagała zdefiniowania urządzenia wyjściowego korzystającego z `ALSA`. Dodano plik `/etc/mpd.conf` rozszerzony o linie

```
audio_output {
        type "alsa"
        name "ALSA Device"
        mixer_type "software"
}
```

Został on utworzony na podstawie domyślnego pliku `mpd.conf` wygenerowanego podczas kompilacji obrazu systemu. Przed końcem kompilacji domyślna wersja tego pliku zostanie nadpisana tą z katalogu `overlay/`.

Ostatnim krokiem było zmodyfikowanie `device tree` generowanego podczas kompilacji. Przy pomocy `dtc` (*Device Tree Compiler*) rozkompilowano plik `bcm2708-rpi-b.dtb` poleceniem

```
# dtc -I dtb -O dts -o bcm2708-rpi-b.dts bcm2708-rpi-b.dtb
```

Następnie znaleziono opcję `status` odpowiadającą `brcm,bcm2835-audio` i zmieniono jej wartość z `disabled` na `okay`, po czym ponownie skompilowano plik poleceniem

```
# dtc -I dts -O dtb -o bcm2708-rpi-b.dtb bcm2708-rpi-b.dts
```

Tak zmodyfikowany plik `bcm2708-rpi-b.dtb` mógł zostać umieszczony na karcie pamięci płytki Raspberry Pi obok obrazu `zImage`.

3.2.3 Konfiguracja

Po uruchomieniu obrazu systemu należy załadować moduł jądra odpowiedzialny za dźwięk przy pomocy polecenia

```
# modprobe snd_bcm2835
```

Wówczas jeżeli widoczna jest karta dźwiękowa, a `mpd` oraz `alsa` działają poprawnie, można uruchomić radio jednym z plików o rozszerzeniu `.py` po przełączeniu się do katalogu `/radio`.