Linux w systemach wbudowanych – Laboratorium 2

Tymon Felski

22 kwietnia 2017

1 Treść zadania

Podczas drugiego laboratorium należało przygotować aplikację w języku C, która powinna:

- obsługiwać przyciski i diody LED płytki Raspberry Pi za pomocą interfejsu GPIO,
- reagować na zmiany stanu przycisków za pomocą przerwań (bez oczekiwania aktywnego),
- realizować funkcjonalność ustaloną przez studenta.

Celem ćwiczenia było również przygotowanie pakietu Buildroota ze stworzoną aplikacją i zdalne przetestowanie jej z komputera laboratoryjnego za pomocą debuggera `gdb`.

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-lab2/` 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 w języku C jest licznikiem, który wykorzystuje cztery diody LED i trzy przyciski. Diody służą wyświetlaniu binarnej reprezentacji liczby z zakresu $[0, 15]$, natomiast przyciski odpowiadają kolejno za dodawanie, odejmowanie i zakończenie obliczeń.

Dodatkowa część zadania przewidywała dodanie funkcjonalności polegającej na resetowaniu licznika w przypadku wciśnięcia kombinacji dwóch klawiszy.

Obsługa pinów GPIO jest realizowana przy pomocy interfejsu `sysfs` dostarczonego z systemem Linux. Program jest w stanie zapalać i gasić diody oraz reagować na wciskanie przycisków poprzez wykonywanie operacji na plikach specjalnych.

```
int main(void) {
	int num = 0, work = 1, pressed;
	int btnfds[BTN_COUNT];
	int ledfds[LED_COUNT];
	int btnpins[BTN_COUNT] = { BTN_LEFT, BTN_RIGHT, BTN_TOP };
	int ledpins[LED_COUNT] = { LED_BLUE, LED_WHITE, LED_GREEN, LED_RED };

	export_fds(btnfds, btnpins, "in", BTN_COUNT);
	export_fds(ledfds, ledpins, "out", LED_COUNT);
	setup_edges(btnpins, "falling", BTN_COUNT);

	while (work) {
		light_leds(ledfds, LED_COUNT, num);
		while ((pressed = poll_btns(btnfds, BTN_COUNT)) < 0);
		if (pressed == 3) {
			num = 0;
			continue;
		}
		switch (btnpins[pressed]) {
			case BTN_LEFT:
				num = mod(--num, MAX_NUM);
				break;
			case BTN_RIGHT:
				num = mod(++num, MAX_NUM);
				break;
			case BTN_TOP:
				work = 0;
		}
	}

	unexport_fds(btnfds, btnpins, BTN_COUNT);
	unexport_fds(ledfds, ledpins, LED_COUNT);
	return EXIT_SUCCESS;
}
```

Listing 1: Funkcja main w pliku counter.c

Aplikacja rozpoczyna swoje działanie od właściwego skonfigurowania pinów, które odpowiadają obsługiwanym przyciskom i diodom.

Funkcja `export_fds` wpisuje do pliku `/sys/class/gpio/export` odpowiednie numery pinów, czego efektem jest utworzenie katalogów `/sys/class/gpio/gpioNN`, gdzie `NN` jest numerem pinu. Kolejnym krokiem jest ustalenie kierunku na `in` dla przycisków i `out` dla diod poprzez wpisanie tych wartości do plików `direction` odpowiadających każdemu pinowi. Ostatecznie, otwierane są pliki `value`, a ich deskryptory zapamiętane w tablicach `btnfds` oraz `ledfds`.

Funkcja `setup_edges` otwiera pliki `edge` odpowiadające przyciskom i wpisuje `falling` do każdego z nich. Dzięki temu aplikacja będzie otrzymywać przerwanie w momencie wciśnięcia przycisku.

Główna pętla aplikacji zaczyna się od wywołania funkcji `light_leds`, która wpisując 0 lub 1 do plików `value` odpowiednio gasi i zapala diody LED w zależności od obecnej wartości zmiennej `num`. Następnie, w pętli wywoływana jest funkcja `poll_btns`, której zadaniem jest zidentyfikowanie wciśniętego przycisku i zwrócenie jego indeksu, dopóki zwracany przez nią indeks jest mniejszy od zera. Będzie on bowiem wynosił -1, jeżeli zawiedzie debouncing lub zostaną wciśnięte dwa przyciski jednocześnie, z wyjątkiem przycisków odpowiadających dodawaniu i odejmowaniu, co będzie skutkować zresetowaniem licznika. Funkcja `poll_btns` wykrywa ich jednoczesne wciśnięcie i zwraca specjalną wartość, która jest sprawdzana przed instrukcją `switch` w funkcji `main`.

Rozpoznawanie wciśniętego przycisku jest realizowane przy pomocy funkcji `poll`, która reaguje na przerwania generowane przez przyciski. Wykorzystano dwa wywołania tej funkcji - pierwsze z parametrem `timeout` wynoszącym -1, a drugie z wynoszącym 50. Pierwsze wywołanie będzie czekać na wciśnięcie przycisku, a drugie zakończy oczekiwanie po pięćdziesięciu milisekundach. Jeżeli drugie wywołanie nie zaobserwuje zmiany w stanach przycisków, nastąpi sprawdzenie wartości w plikach `value` odpowiadających przyciskom, aby zidentyfikować wszystkie wciśnięte. Taka implementacja ma na celu wyleminowanie drgania (*ang. bouncing*) przycisku, które polega na kilkukrotnej zmianie stanu przycisku krótko po jego pierwszym wciśnięciu lub odciśnięciu.

Przed każdym wywołaniem funkcji `poll` następuje przeczytanie wartości we wszystkich plikach `value` odpowiadających przyciskom, po wcześniejszym przesunięciu się na ich początek przy pomocy funkcji `lseek`. Jest to konieczne, ponieważ wywołania te mogłyby się skończyć natychmiast z powodu wcześniej występujących przerwań.

Po zidentyfikowaniu wciśniętego przycisku wykonywana jest odpowiednia akcja - dodawanie, odejmowanie lub wyjście z głównej pętli. Ta ostatnia spowoduje, że wywołane zostaną funkcje `unexport_fds`, które wpiszą do pliku `/sys/class/gpio/unexport` numery pinów wyeksportowanych na początku działania programu.

3.2.2 Pakiet Buildroota

Aby stworzyć pakiet Buildroota zawierający wyżej opisaną aplikację, do katalogu `package/` dodano `gpio-counter/`, w którym utworzono dwa pliki: `Config.in` oraz `gpio-counter.mk`.

Pierwszy z nich ma za zadanie zdefiniowanie pakietu.

```
config BR2_PACKAGE_GPIO_COUNTER
        bool "gpio-counter"
        help
          Binary counter using GPIO.

          https://github.com/felskit
```

Listing 2: Zawartość pliku Config.in

Drugi plik odpowiada za definicję metody instalacji źródeł, do których podano w nim ścieżkę.

```
################################################################################
#
# gpio-counter
#
################################################################################

GPIO_COUNTER_VERSION = 1.0
GPIO_COUNTER_SITE = $(TOPDIR)/../felskit-lab2/gpio-counter
GPIO_COUNTER_SITE_METHOD = local
GPIO_COUNTER_LICENSE = MIT
```

```
define GPIO_COUNTER_BUILD_CMDS
   $(MAKE) $(TARGET_CONFIGURE_OPTS) all -C $(@D)
endef

define GPIO_COUNTER_INSTALL_TARGET_CMDS
    $(INSTALL) -D -m 0755 $(@D)/counter $(TARGET_DIR)/usr/bin
endef

$(eval $(generic-package))
```

Listing 3: Zawartość pliku gpio-counter.mk

Tutaj plikiem źródłowym jest `counter.c`, który zgodnie z instrukcją zawartą w powyższym pliku zostanie skompilowany do pliku `counter` przy pomocy dostarczonego Makefile'a.

```
CC=$(CROSS_COMPILE)gcc
CFLAGS=-Wall
all:
	$(CC) $(CFLAGS) counter.c -o counter
.PHONY: clean
clean:
	-rm counter
```

Listing 4: Zawartość pliku Makefile

W pliku `package/Config.in` w sekcji *Debugging, profiling and benchmark* należy dodać linię

```
source "package/gpio-counter/Config.in"
```

Wówczas możliwe będzie dodanie naszej paczki poprzez zaznaczenie opcji

Target packages → *Debugging, profiling and benchmark* → *gpio-counter*

3.2.3 Korzystanie z debuggera

Komputer laboratoryjny do zdalnego debugowania potrzebuje działającej na płytce instacji serwera `gdb`. Wobec tego należy zaznaczyć opcję

Target packages → *Debugging, profiling and benchmark* → *gdb*

Opcja *gdbserver* będzie domyślnie zaznaczona. Aby komputer mógł się połączyć z działającą instancją serwera i zdalnie debugować zbudowaną aplikację, konieczne jest wybranie opcji

Toolchain → *Build cross gdb for the host*

Aby aplikacje stworzone w trakcie kompilacji obrazu systemu mogły być debugowane, konieczne jest włączenie dodawania symboli, co można zrobić zaznaczając opcję

Build options → *build packages with debugging symbols*

Pierwszym krokiem do zdalnego debugowania naszej aplikacji jest uruchomienie instacji serwera `gdb` na płytce przy pomocy polecenia

```
$ gdbserver host:56000 /usr/bin/counter
```

Następnie, na stacji roboczej można zestawić połączenie z płytką poleceniami

```
$ /malina/felskit/buildroot-2016.11.2/output/host/usr/bin/arm-linux-gdb \
  /malina/felskit/buildroot-2016.11.2/output/build/gpio-counter-1.0/counter
$ (gdb) target remote 192.168.143.188:56000
```

Powyższe operacje pozwolą nam na zdalne debugowanie kodu na płytce z komputera laboratoryjnego. Możliwe jest między innymi ustawianie breakpointów przy pomocy polecenia `break <line>`, kontynuowanie działania programu do kolejnego breakpointu w kodzie poleceniem `continue` oraz wyświetlanie wartości zmiennej za pomocą polecenia `display <name>`.