Warsaw, 04.04.2021

Linux for Embedded Systems - Laboratory ex. 2

Emilia Wróblewska 291674

Problem Description

1. Prepare an application in C language, which uses the buttons and LED diodes. When reading the buttons, please take into consideration the „contact bounce" effect. The application should use the new „libgpiod" based handling of GPIOs.

2. It is required that application implements thorough error checking (you can't assume that e.g. all hardware communication functions always succeed).

3. The application must respond to changes of state of the buttons without active waiting (should sleep waiting for the required change).

4. You may define the functionality of the application (below are just examples). However, it is important that the functionality is sensitive to „bounce effect".

5. The application should be converted into the Buildroot package.

6. Please try debugging your application with the gdb debugger.

Description of the Solution

I started with the John_Smith demo application given by the teacher and used it as a base. After downloading and unpacking Buildroot, I added a package with my application. To do that I created a directory “timer” and put it inside the “BR_PATH/package” directory. Inside this folder I create “Config.in” and “timer.mk” files.

“Config.in” file:

```
config BR2_PACKAGE_TIMER
    bool "timer"
    depends on BR2_PACKAGE_LIBGPIOD
    help
      Simple timer application with lap function.

comment "timer depends on libgpiod library"
    depends on !BR2_PACKAGE_LIBGPIOD
```

“timer.mk” file:

```
################################################################################
#
# timer
#
################################################################################

TIMER_VERSION = 1.0
TIMER_SITE = $(TOPDIR)/../timer_src
TIMER_SITE_METHOD = local
TIMER_DEPENDENCIES = libgpiod
TIMER_INSTALL_STAGING = YES

define TIMER_BUILD_CMDS
    $(MAKE) $(TARGET_CONFIGURE_OPTS) timer -C $(@D)
endef
define TIMER_INSTALL_TARGET_CMDS
    $(INSTALL) -D -m 0755 $(@D)/timer $(TARGET_DIR)/usr/bin
endef
TIMER_LICENSE = Proprietary

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

In “Config.in” I used *depends on*, like it was described in the lecture.
Now I need to add a line to the “Config.in” file located in the “package” directory (I put it in the “Games” section).

```
source “package/timer/Config.in”
```

The application was created using generic-package, like it was described in the lecture. I put my resources just outside the Buildroot directory, which is specified in the “timer.mk” file. They are placed inside “timer_src” which consists of Makefile and “timer.c” files. Now I need to modify buildroot settings to add my package and to include gdb debugger with:

```
$ make menuconfig
Target packages → Debugging → gdb → gdbserver
Toolchain → Build cross gdb for the host
Build options → build packages with debugging symbols
Target packages → Games → timer
$ make
```

After building the image, I can start the GUI by using:

```
$ ./run_before
```

And then in the second terminal I start the virtual machine:

```
$ ./runme
```

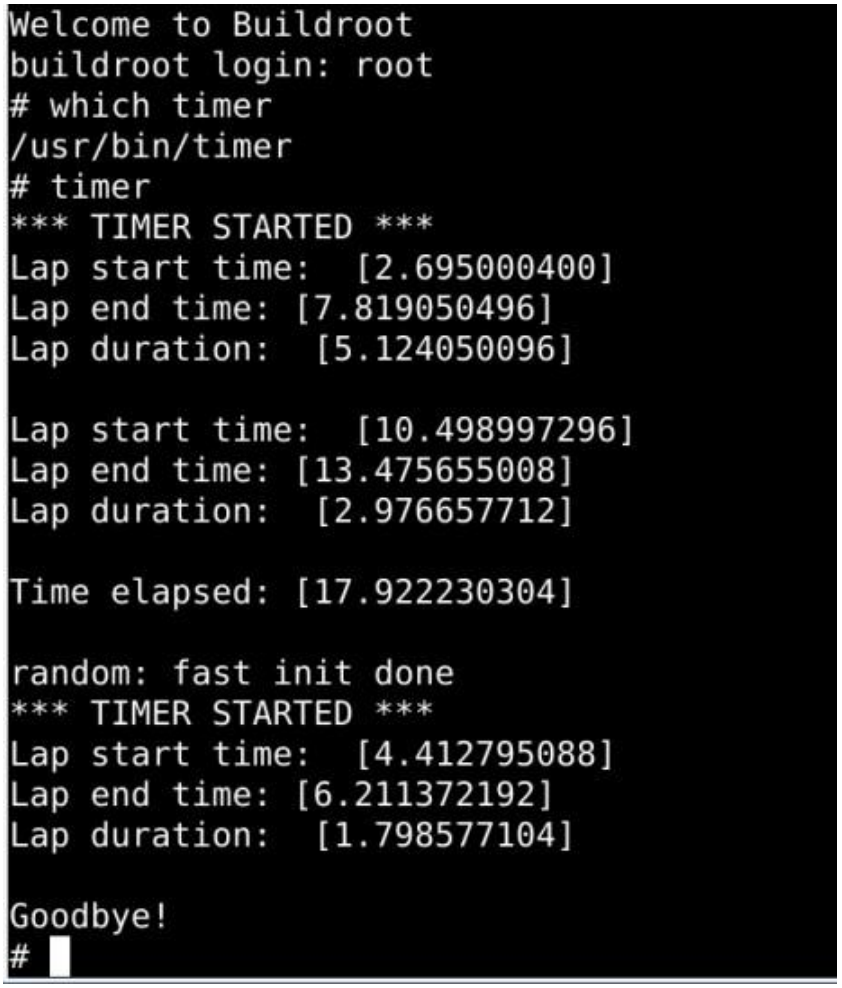
Now I can start my application with

```
# timer
```

My application is a simple timer with lap function. Timer is started by clicking Button 12, indicated by turning on LED 24, which stays turned on until the timer is stopped, and the message in the console. Then the user can click Button 13 to start the lap, which prints to the console time elapsed since the timer started (Lap start time) and turns on the LED 25. After clicking the Button 13 again the lap is ended - LED 25 is switched off and Lap end time as well as lap duration are printed in the console. Laps can be used indefinitely during timer operating time.

To stop the timer one should press Button 12 again which results in printing to the console the full time elapsed since the start of the timer. To exit the program one should press Button 14.
I dealt with the “bounce effect” by using a function which after processing one event recursively processes all others that follow immediately after and then just returns the last one.

The exemplary output of the timer application:



As for the gdb debugger, I successfully started my application on the machine via gdb server (port 8810 is used because “runme” script was used to start the machine):

```
# gdbserver host:8810 `which timer`
```

Then, in another terminal I change the working directory to “BRPATH/output/host/usr/bin” and run:

```
$ ./aarch64-buildroot-linux-gnu-gdb BRPATH/output/build/timer-1.0/timer
```

And then, in the debugger session:

```
(gdb) set sysroot BRPATH/output/staging
(gdb) target remote localhost:8888
```

Below are some exemplary screenshots of working in my debugger session:

```
(gdb) list main
57      {
58          struct gpiod_chip *chip;
59          struct gpiod_line_bulk buttons;
60          struct gpiod_line *led24;
61          struct gpiod_line *led25;
62          struct timespec ts = { 2, 0 };
63          struct gpiod_line_event event, start, lap;
64          bool timer = false;
65          bool isLap = false;
66          int seconds, nanoseconds;
(gdb) break main
Note: breakpoint 5 also set at pc 0x4009a0.
Breakpoint 6 at 0x4009a0: file timer.c, line 62.
(gdb) break 70
Note: breakpoint 3 also set at pc 0x4009e8.
Breakpoint 7 at 0x4009e8: file timer.c, line 72.
(gdb) info break
Num     Type           Disp Enb Address            What
3       breakpoint     keep y   0x00000000004009e8 in main at timer.c:64
5       breakpoint     keep y   0x00000000004009a0 in main at timer.c:62
6       breakpoint     keep y   0x00000000004009a0 in main at timer.c:62
7       breakpoint     keep y   0x00000000004009e8 in main at timer.c:72
(gdb)
```

```
(gdb) continue
Continuing.
warning: Could not load shared library symbols for 3 libraries, e.g. /usr/lib64/
libgpiod.so.2.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?

Breakpoint 1, main (argc=1, argv=0x7fd89aae28) at timer.c:62
62          struct timespec ts = { 2, 0 };
(gdb) p ts
$1 = {tv_sec = 548156915712, tv_nsec = 1}
(gdb) n
71          chip = gpiod_chip_open_by_name("gpiochip0");
(gdb) n
72          if(!chip){
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004009a0 in main at timer.c:62
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00000000004009a0 in main at timer.c:62
        breakpoint already hit 1 time
```

Unfortunately, I wasn't able to use the "aarch64-none-linux-gnu-gdb" debugger since for some unknown reason it wasn't present in the filesystem of my buildroot.

To recreate my solution, after unpacking the archive, one should run the "build.sh" script. After the compilation, run the "run_before" script to start the GUI and then start the virtual machine with "runme" script.