An implementation of the trip counter

Sverre Hendseth

January 16, 2015

Contents

1 The main program: tripcounter.c

```
/*** #file "tripcounter.c" ***/
#include <stdio.h>
#include <unistd.h>
#include "hardwaresimulator.h"
#include "tripcounterstatemachine.h"
#include "timer.h"

int
main(){
  hw_init();
  fsm_init();

  // Assume no buttons pressed at startup.
  int largeButtonState = 0;
  int smallButtonState = 0;

  // Must detect whether anything insteresting has happened:

  while(1){
    if(largeButtonState == 0 && hw_button1Status() == 1){
      largeButtonState = 1;
      fsm_evLargeButtonPressed();
    }
    if(largeButtonState == 1 && hw_button1Status() == 0){
      largeButtonState = 0;
      fsm_evLargeButtonReleased();
    }
    if(smallButtonState == 0 && hw_button2Status() == 1){
      smallButtonState = 1;
      fsm_evSmallButtonPressed();
    }
    if(smallButtonState == 1 && hw_button2Status() == 0){
```

```
      smallButtonState = 0;
    }
    if(timer_isTimeOut()){
      fsm_evTimeout();
      timer_stop();
    }
    usleep(100000); // 0.1 sec
  }
}
/*** End of File ***/
```

2 Module: tripcounterstatemachine - Modelling the trip counter as a state machine.

2.1 What are the states?

The possible states of the trip counter.

running The user is swimming; we wait for the next event.

largeButtonPressed The large button has been pressed, but not released yet.

settingPoolLength Any keypresses of the large button should change the pool length

2.2 What are the events?

The interesting things that can happen in the system are:

- evLargeButtonPressed: The large button is pressed
- evLargeButtonReleased: The large button is released
- evTimeout: Timer goes out after 3 seconds.
- evSmallButtonPressed: The small button is pressed.
- The system starts. I see this is not an event in the UML design. But some things needs to be done at this point: read the poolLength and distance from file. I make it an init function in the state machine module to be called at startup...

- the on/off button is pushed? Do we care to implement this? Depending on hardware? We can skip it if if we always store changed data...

2.3 Module interface

```
/*** #file "tripcounterstatemachine.h" ***/
#ifndef TRIPCOUNTERSTATEMACHINE_H
#define TRIPCOUNTERSTATEMACHINE_H

void fsm_init();
void fsm_evLargeButtonPressed();
void fsm_evLargeButtonReleased();
void fsm_evTimeout();
void fsm_evSmallButtonPressed();

#endif
/*** End of File ***/
```

2.4 Implementation

```
/*** #file "tripcounterstatemachine.c" ***/
#include <stdio.h>
#include <assert.h>
#include "distances.h"
#include "hardwaresimulator.h"
#include "timer.h"

typedef enum {
  state_running,
  state_largeButtonPressed,
  state_settingPoolLength
} TState;

static TState g_state = state_running;

static void printEventStatus(char * str);
static void setDistance(double newDistance);

void
fsm_init(){
```

```
  hw_on();
  dist_restore();
  hw_setDisplayNumber(dist_getDistance());
  g_state = state_running;
}

void
fsm_evLargeButtonPressed(){
  printEventStatus("evLargeButtonPressed");
  switch(g_state){
    case state_running: {
      timer_start();
      g_state = state_largeButtonPressed;
      break;
    }
    case state_largeButtonPressed: {
      break;
    }
    case state_settingPoolLength: {
      dist_rotatePoolLength();
      timer_start();
      hw_setDisplayNumber(dist_getPoolLength());
    }
  }
}

void
fsm_evLargeButtonReleased(){
  printEventStatus("evLargeButtonReleased");
  switch(g_state){
    case state_running: {
      // Do nothing - this happens after the timeout when resetting the distance.
      break;
    }
    case state_largeButtonPressed: {
      setDistance(dist_getDistance() + 2*dist_getPoolLength());
      g_state = state_running;
      timer_stop();
      break;
    }
```

```
    case state_settingPoolLength: {
      break;
    }
  }
}

void
fsm_evTimeout(){
  printEventStatus("evTimeout");
  switch(g_state){
    case state_running: {
      // This should never happen
      fprintf(stderr,"\nWARNING: Unexpected timeout happened in running state!\n\n");
      assert(0);
      break;
    }
    case state_largeButtonPressed: {
      setDistance(0);
      g_state = state_running;
      break;
    }
    case state_settingPoolLength: {
      hw_setDisplayNumber(dist_getDistance());
      g_state = state_running;
      break;
    }
  }
}

void
fsm_evSmallButtonPressed(){
  printEventStatus("evSmallButtonPressed");
  switch(g_state){
    case state_running: {
      timer_start();
      g_state = state_settingPoolLength;
      hw_setDisplayNumber(dist_getPoolLength());
      break;
    }
    case state_largeButtonPressed: {
```

```
      // No... Seriously? two keys at once?
      // Just ignore it.
      break;
    }
    case state_settingPoolLength: {
      timer_start();
      break;
    }
  }
}

// ------- Two helper functions

static void
setDistance(double newDistance){
  dist_setDistance(newDistance);
  hw_setDisplayNumber(dist_getDistance());
  dist_store();
}

static void
printEventStatus(char * str){
/*
  printf("Trace: Event %s happened in state ",str);
  switch(g_state){
    case state_running: {
      printf("running\n");
      break;
    }
    case state_largeButtonPressed: {
      printf("largeButtonPressed\n");
      break;
    }
    case state_settingPoolLength: {
      printf("settingPoolLength\n");
      break;
    }
  }
*/
}
```

```
/*** End of File ***/
```

3 Module: timer - starting and stopping a timer and detecting timeouts.

```
/*** #file "timer.h" ***/
#ifndef TIMER_H
#define TIMER_H

void timer_start();
void timer_stop();
int timer_isTimeOut();

#endif
/*** End of File ***/
```

3.1 Implementation

The time() function used here has precision of seconds only - should be replaced with something else.

```
/*** #file "timer.c" ***/
#include <time.h>
#include <assert.h>

static time_t g_startTime = -1;

void
timer_start(){
  g_startTime = time(0);
  assert(g_startTime != -1);
}

void
timer_stop(){
  g_startTime = -1;
}

int
timer_isTimeOut(){
```

```
  if(g_startTime <0){
    // There is no timeout, because the timer is not started
    return 0;
  }

  time_t now = time(0);
  if(now - g_startTime > 3){
    return 1;
  }else{
    return 0;
  }
}
/*** End of File ***/
```

4 Module: distances - handling and storing distance and poolLength

The tripcounter should keep track of and remember the pool length and total length between sessions.

```
/*** #file "distances.h" ***/
#ifndef DISTANCES_H
#define DISTANCES_H

void dist_rotatePoolLength();
double dist_getPoolLength();

void dist_setDistance(double distance);
double dist_getDistance();

// Persistent storage
void dist_store();
void dist_restore();

#endif
/*** End of File ***/
```

4.1 Implementation

```
/*** #file "distances.c" ***/
```

```
#include <stdio.h>

static double g_poolLength = 25;
static double g_distance = 0;

void
dist_rotatePoolLength(){
  if(g_poolLength == 25){
    g_poolLength = 50;
  }else if(g_poolLength == 50){
    g_poolLength = 12.5;
  }else if(g_poolLength == 12.5){
    g_poolLength = 25;
  }
}

double
dist_getPoolLength(){
  return g_poolLength;
}

void
dist_setDistance(double distance){
  g_distance = distance;
}

double
dist_getDistance(){
  return g_distance;
}

void
dist_store(){
  FILE * f;
  f = fopen("tripcounter.dat","w");
  fprintf(f,"%lf %lf\n",g_poolLength,g_distance);
  fclose(f);
}

void
```

```
dist_restore(){
  FILE * f;
  int res = 0;
  f = fopen("tripcounter.dat","r");
  if(f != NULL) res = fscanf(f,"%lf %lf\n",&g_poolLength,&g_distance);
  if(res != 2){
    printf("WARNING: did not find data from last session\n");
  }
  if(f != NULL) fclose(f);
}
/*** End of File ***/
```

5 Module: The Trip Counter Window - handling buttons and the display

A console application do not have direct access to key and button, press and release events. To get the mouse buttons to work as the counter buttons I needed to create a gui application.

Chose xlib, just because I found a nice hello world program on the internet. Should work on any linux pc - and mac I assume - if you get the xlib headers installed.

Do not read this code :-) It is irrelevant for what you are supposed to learn from this document.

5.1 Module Interface

```
/*** #file "hardwaresimulator.h" ***/
#ifndef HARDWARESIMULATOR_H
#define HARDWARESIMULATOR_H

// hw_init() Creates a thread, that processes all gui events and updates the state
// that can be checked with other functions in the interface.
void hw_init();

// Checking inputs
int hw_button1Status();
int hw_button2Status();

// Controlling the display
```

```
void hw_setDisplayNumber(int i); // Set the number to display
void hw_on();  // The on and off functionality is a bit artificial compared to
void hw_off(); // the spec, since we need to "simulate" a display that is off.
// hw_off() is not even used as it is.

#endif
/*** End of File ***/
```

5.2 Implementation

```
/*** #file "hardwaresimulator.c" ***/
/*

This code is mainly ripped from

http://www.paulgriffiths.net/program/c/srcs/helloxsrc.html

In addition I have created a thread to run the window handling:
 - updating the window when a new value is to be shown.
 - detecting the mouse events that represents the buttons.
 - Made a nice and clean module of it :-) Check hardwaresimulator.h

Other useful references was
 - http://stackoverflow.com/questions/10785491/how-to-allow-a-worker-thread-to-updata-
 - http://tronche.com/gui/x/xlib/graphics/XClearWindow.html

This module is a hack to make the tripcounter code compile and run. It
is not part of the tripcounter code ment to be read or understood.

Sverre

*/


/*

  HelloX.C
  ========
  (c) Copyright Paul Griffiths 1999
  Email: mail@paulgriffiths.net
```

```
  "Hello, World!", X Window System style.

*/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <math.h>

/*  some module variables  */

static Window       win;
static Display *     display;
static int           screen_num;
static char * appname;

static double displayNumber = 0;
static int button1 = 0;
static int button2 = 0;
static int onOffButton = 0;

void
hw_setDisplayNumber(double i){
  displayNumber = i;

  // Need to force a redraw of the window.
  XEvent exppp;
  //printf ("Sending event of type Expose\n");
  memset(&exppp, 0, sizeof(exppp));
  exppp.type = Expose;
  exppp.xexpose.window = win;
  XSendEvent(display,win,False,ExposureMask,&exppp);
```

```
  XFlush(display);
}
int
hw_button1Status(){
  return button1;
}
int
hw_button2Status(){
  return button2;
}
int
hw_onOffButtonStatus(){
  return onOffButton;
}

static int on = 0;

void
hw_on(){
  on = 1;
}
void
hw_off(){
  on = 0;
}

void * windowThread(void * threadData /*Unused*/);

void hw_init(){
  pthread_t threadId;
  pthread_create(&threadId, NULL, windowThread, NULL);
  usleep(100000);
}

void * windowThread(void * threadData /*Unused*/) {

  /*  Window variables  */
  int          x, y;
  unsigned int width, height;
  unsigned int border_width = 0;
```

```
char *       window_name = "TripCounter!";
char *       icon_name   = "TripCounter";

/*  Display variables  */
char *       display_name = NULL;
// unsigned int display_width, display_height;


/*  Miscellaneous X variables  */
XSizeHints *  size_hints;
XWMHints   *  wm_hints;
XClassHint *  class_hints;
XTextProperty windowName, iconName;
XEvent        report;
XFontStruct * font_info;
XGCValues     values;
GC            gc;

appname = "Tripcounter";

/*  Allocate memory for our structures  */
if( !( size_hints  = XAllocSizeHints() ) ||
    !( wm_hints    = XAllocWMHints()   ) ||
    !( class_hints = XAllocClassHint() )    ) {
  fprintf(stderr, "%s: couldn't allocate memory.\n", appname);
  exit(EXIT_FAILURE);
}

/* Connect to X server  */

if ( (display = XOpenDisplay(display_name)) == NULL ) {
  fprintf(stderr, "%s: couldn't connect to X server %s\n", appname, display_name);
  exit(EXIT_FAILURE);
}

/*  Get screen size from display structure macro  */
screen_num     = DefaultScreen(display);
// display_width  = DisplayWidth(display, screen_num);
// display_height = DisplayHeight(display, screen_num);
```

```
/*  Set initial window size and position, and create it  */
x = y = 50;
//width  = display_width / 3;
//height = display_width / 3;
width  = 200;
height = 40;

win = XCreateSimpleWindow(display, RootWindow(display, screen_num),
                          x, y, width, height, border_width,
                          BlackPixel(display, screen_num),
                          WhitePixel(display, screen_num));

/*  Set hints for window manager before mapping window  */
if ( XStringListToTextProperty(&window_name, 1, &windowName) == 0 ) {
  fprintf(stderr, "%s: structure allocation for windowName failed.\n", appname);
  exit(EXIT_FAILURE);
}

if ( XStringListToTextProperty(&icon_name, 1, &iconName) == 0 ) {
  fprintf(stderr, "%s: structure allocation for iconName failed.\n",appname);
  exit(EXIT_FAILURE);
}

size_hints->flags       = PPosition | PSize | PMinSize;
size_hints->min_width   = 200;
size_hints->min_height  = 100;

wm_hints->flags         = StateHint | InputHint;
wm_hints->initial_state = NormalState;
wm_hints->input         = True;

class_hints->res_name   = appname;
class_hints->res_class  = "hellox";

XSetWMProperties(display, win, &windowName, &iconName, NULL/*argv*/, 0/*argc*/,
                 size_hints, wm_hints, class_hints);

/*  Choose which events we want to handle  */
XSelectInput(display, win, ExposureMask | KeyPressMask | ButtonReleaseMask | ButtonP
```

```
/*  Load a font called "9x15"  */
if ( (font_info = XLoadQueryFont(display, "9x15")) == NULL ) {
  fprintf(stderr, "%s: cannot open 9x15 font.\n", appname);
  exit(EXIT_FAILURE);
}

/*  Create graphics context  */
gc = XCreateGC(display, win, 0, &values);

XSetFont(display, gc, font_info->fid);
XSetForeground(display, gc, BlackPixel(display, screen_num));

/*  Display Window  */
XMapWindow(display, win);

/*  Enter event loop  */
while ( 1 ) {
  static char message[128];
  static int    length;
  static int    font_height;
  static int    msg_x, msg_y;

  XNextEvent(display, &report);

  switch ( report.type ) {
    case Expose:
      if ( report.xexpose.count != 0 ) break;

      XClearWindow(display,win);

      if(on){
        if(displayNumber-trunc(displayNumber) < 0.1){
          // Round and present as integer.
          sprintf(message,"%4i",(int)displayNumber);
        }else{
          sprintf(message,"%4.1lf",displayNumber);
        }
      }else{
        sprintf(message," ");
      }
```

```
/*  Output message centrally in window  */

length = XTextWidth(font_info, message, strlen(message));
msg_x  = (width - length) / 2;

font_height = font_info->ascent + font_info->descent;
msg_y  = (height + font_height) / 2;

XDrawString(display, win, gc, msg_x, msg_y,
            message, strlen(message));
break;

case ConfigureNotify:
  /*  Store new window width & height  */
  width  = report.xconfigure.width;
  height = report.xconfigure.height;
  break;

case ButtonPress:           /*  mouse buttons  */
  // printf("buttonevent: state = %d button = %d \n",report.xbutton.state,repo
  if(report.xbutton.button == 1) button1 = 1; // These should be mousebutton 1
  if(report.xbutton.button == 2) onOffButton = 1;
  if(report.xbutton.button == 3) button2 = 1;
  break;

case ButtonRelease:           /*  mouse buttons  */
  // printf("buttonreleaseevent: state = %d button = %d \n",report.xbutton.sta
  if(report.xbutton.button == 1) button1 = 0;
  if(report.xbutton.button == 2) onOffButton = 0;
  if(report.xbutton.button == 3) button2 = 0;
  break;

case KeyPress:
  printf("keypressevent: state = %d button = %d \n",report.xkey.state,report.xk
  printf("Quitting because of keypress! Do not press a key if you do not want

  /*  Clean up and exit  */
  XUnloadFont(display, font_info->fid);
  XFreeGC(display, gc);
```

```
          XCloseDisplay(display);
          exit(EXIT_SUCCESS);
      }
  }
  return EXIT_SUCCESS;   /*  We shouldn't get here  */
}
/*** End of File ***/
```

6 The Makefile

```
/*** #file "Makefile" ***/
OBJS = tripcounter.o hardwaresimulator.o distances.o \
       timer.o tripcounterstatemachine.o

%.o:%.c
        gcc -c -g -Wall $<

all: tripcounter

tripcounter: $(OBJS)
        gcc $(OBJS) -g -o $@ -lX11 -lpthread -lm

# All the source files are extracted from the org-document.
Makefile : ../tripcounter.org
        tinyclip -v $<

/*** End of File ***/
```

7 Order of work.

Just out of interest I record the order I implemented the different thing in.

- First, even before reading the spec carefully, I explored how to simulate the buttons and display. Ended up with making an XLib window as an easy way to get access to keyboard and mouse events. 2-3 hours.
- Read the spec. 10 minutes.
- How to make on-off work? Need persistent storage. Add an on-off button and a persistenStorage module. 30 min.

- I will probably need a timer interface of some kind also... - will make it when I need it. All uncertainties clarified; Time to set up state machine. (States and events defined, skeleton made, timer module made 1 hour)
- Filling in the event function cases and testing - 1 hr.
- Straightening the document (the one you are reading). 1 hour. Sending to Anders and Øyvind for comments.
- Discussing spec and UML consistency with Øyvind, Anders and Gorm 1h.
- Reverting to english state and event names (Øyvind will rather change language in the UML design). 20 mins
- Packing handling the distances into a separate module, merging this with the persistent storage module... (not elegant, but still...) 1h

8 TODO & Comments

- Should one adjust pool length or distance per press? I feel the second is simpler, also for the user... Need to negociate with marketing (Øyvind).
- Comparison between float numbers should never be trusted.
- There are some unused code relating to the on-off button. As it is now we store the two values every time they change. If we rather wanted to store the numbers only when turning off, or to extend the hardware simulator to simulate on and off better we would need them...

8.1 Known Bugs

- Quitting by killing the window leaves some messages in the console. Close event should be handled.