

Текст программы

Абдуллин Азат

29 мая 2017 г.

Оглавление

1	Текст программы	2
---	---------------------------	---

1 Текст программы

Листинг 1: Файл main.c

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <wlc/wlc.h>
5 #include <linux/input.h>
6
7 #include "config.h"
8
9 #define DEFAULT_CONFIG "~/.config/xxwm"
10
11 pid_t statusbar_pid;      // pid of statusbar process
12 wlc_handle statusbar = 0; // statusbar view
13 pid_t desktop_pid;       // pid of desktop app
14 wlc_handle desktop = 0;  // desktop view
15
16 // current view and details of active interactive action
17 static struct {
18     struct {
19         wlc_handle view;
20         struct wlc_point grab;
21         uint32_t edges;
22     } action;
23 } compositor;
24
25 // функция, которая запускает интерактивное действие
26 static bool start_interactive_action(wlc_handle view, const struct wlc_point *
    origin) {
27     // если нет активного окна, выходим
28     if (compositor.action.view)
29         return false;
30
31     // запоминаем активное окно
32     compositor.action.view = view;
33     // изначальную точку расположения окна
34     compositor.action.grab = *origin;
35     // выводит активное окно на первый план
36     wlc_view_bring_to_front(view);
37     return true;
38 }
39
40 // начинает интерактивное передвижение окна
41 static void start_interactive_move(wlc_handle view, const struct wlc_point *origin
    ) {
42     start_interactive_action(view, origin);
43 }
44
45 // начинает интерактивное изменение размеров окна
46 static void start_interactive_resize(wlc_handle view, uint32_t edges, const struct
    wlc_point *origin) {
47     const struct wlc_geometry *g;
48     // получаем параметры текущего окна
49     // и начинаем интерактивное действие
```

```

50     if (!(g = wlc_view_get_geometry(view)) || !start_interactive_action(view,
origin))
51         return;
52
53     // определяем, какую грань окна передвигать
54     const int32_t halfw = g->origin.x + g->size.w / 2; // координата середины окна по
горизонтали
55     const int32_t halfh = g->origin.y + g->size.h / 2; // координата середины окна по
вертикали
56     if (!(compositor.action.edges = edges)) {
57         // если начальная точка окна левее середины, перемещаем левую грань
58         // если начальная точка окна правее середины, перемещаем правую грань
59         // точно так же с верхней и нижней границей
60         compositor.action.edges = (origin->x < halfw ? WLC_RESIZE_EDGE_LEFT : (
origin->x > halfw ? WLC_RESIZE_EDGE_RIGHT : 0)) |
61         (origin->y < halfh ? WLC_RESIZE_EDGE_TOP : (
origin->y > halfh ? WLC_RESIZE_EDGE_BOTTOM : 0));
62     }
63
64     // устанавливаем флаг, что окно в текущий момент меняет свой размер
65     wlc_view_set_state(view, WLC_BIT_RESIZING, true);
66 }
67
68 // завершение интерактивного действия
69 static void stop_interactive_action(void) {
70     // если нет активного окна, ничего не делаем
71     if (!compositor.action.view)
72         return;
73
74     // снимаем флаг изменения размеров окна
75     wlc_view_set_state(compositor.action.view, WLC_BIT_RESIZING, false);
76     // обнуляем все флаги действий активного окна
77     memset(&compositor.action, 0, sizeof(compositor.action));
78 }
79
80 // получить самое верхнее окно по смещению
81 static wlc_handle get_topmost(wlc_handle output, size_t offset) {
82     size_t memb;
83     // получаем все окна
84     const wlc_handle *views = wlc_output_get_views(output, &memb);
85     // возвращаем окно по смещению, или 0 если окон нет
86     return (memb > 0 ? views[(memb - 1 + offset) % memb] : 0);
87 }
88
89 // перерисовка всех окон
90 static void relayout(wlc_handle output) {
91     const struct wlc_size *r;
92     // получаем разрешение ОМ
93     if (!(r = wlc_output_get_virtual_resolution(output)))
94         return;
95
96     // получаем самое верхнее окно
97     wlc_handle topview = get_topmost(output, 0);
98     uint32_t shift = 30;
99     // получаем имя верхнего окна
100     const char* app = wlc_view_get_title(topview);
101     // создаем структуру с параметрами отрисовки окна статусбара

```

```

102  const struct wlc_geometry gstatus = {
103      .origin = {
104          .x = 0,
105          .y = 0
106      },
107      .size = {
108          .w = r->w,
109          .h = shift
110      }
111  };
112  // получаем тип окна и определяем как его рисовать
113  uint32_t viewtype = wlc_view_get_type(topview);
114  bool isPopup = (viewtype & WLC_BIT_POPUP) == WLC_BIT_POPUP;
115  bool isUnmanaged = (viewtype & WLC_BIT_UNMANAGED) == WLC_BIT_UNMANAGED;
116  // если статусбар до сих пор не запущен, значит первое окно это статусбар
117  if (statusbar == 0 && wlc_view_get_pid(topview) == statusbar_pid) {
118      statusbar = topview;
119  // если статусбар запущен, значит перерисовывается другое окно которое( на данный момент
активно)
120  } else {
121      if (wlc_view_get_pid(topview) == desktop_pid) {
122          desktop = topview;
123      }
124      if (isUnmanaged) {
125          // если установлен флаг unmanaged, значит нам не надо менять параметры данного
окна,
126          // тк.. это меню приложения с уже верно заданными параметрами
127
128      } else if (isPopup) {
129          // если установлен флаг popup, значит создается всплывающее уведомление
130          const struct wlc_geometry* anchor_rect =
wlc_view_positioner_get_anchor_rect(topview);
131          // получаем размеры окна
132          struct wlc_size size_req = *wlc_view_positioner_get_size(topview);
133          if ((size_req.w <= 0) || (size_req.h <= 0)) {
134              const struct wlc_geometry* current = wlc_view_get_geometry(topview
);
135              size_req = current->size;
136          }
137          // задаем размеры окна такие же, как и были при создании
138          struct wlc_geometry gpopup = {
139              .origin = anchor_rect->origin,
140              .size = size_req
141          };
142          // если есть родительское окно, смещаем popup относительно его начала
143          wlc_handle parent = wlc_view_get_parent(topview);
144          if (parent) {
145              const struct wlc_geometry* parent_geometry = wlc_view_get_geometry
(parent);
146              gpopup.origin.x += parent_geometry->origin.x;
147              gpopup.origin.y += parent_geometry->origin.y;
148          }
149          // перерисовываем окно
150          wlc_view_set_geometry(topview, 0, &gpopup);
151
152      } else {
153          // иначе создается обычное окно

```

```

154         // задаем параметры отрисовки активного окна
155         const struct wlc_geometry gview = {
156             .origin = {
157                 .x = 0,
158                 .y = shift
159             },
160             .size = {
161                 .w = r->w,
162                 .h = r->h - shift
163             }
164         };
165         // перерисовываем активное окно
166         wlc_view_set_geometry(topview, 0, &gview);
167     }
168 }
169 // на всякий случай перерисовываем статусбар тк(.. могло поменяться разрешение экрана)
170 if (statusbar != 0) wlc_view_set_geometry(statusbar, 0, &gstatus);
171 }
172
173 // смена разрешения окна OM
174 static void output_resolution(wlc_handle output, const struct wlc_size *from,
175     const struct wlc_size *to) {
176     (void)from, (void)to;
177     layout(output);
178 }
179
180 // обработка нового созданного приложения
181 static bool view_created(wlc_handle view) {
182     // устанавливаем флаги отображения окна поумолчанию(- - 1, те.. рисовать)
183     wlc_view_set_mask(view, wlc_output_get_mask(wlc_view_get_output(view)));
184     // переносим новое окно на первый план
185     wlc_view_bring_to_front(view);
186     // устанавливаем новое окно активным
187     wlc_view_focus(view);
188     // указываем, чтобы новое окно рисовалось во весь экран
189     wlc_view_set_state(view, WLC_BIT_FULLSCREEN, true);
190     // перерисовываем все окна
191     layout(wlc_view_get_output(view));
192     return true;
193 }
194
195 // обработка завершившегося приложения
196 static void view_destroyed(wlc_handle view) {
197     // получаем самое верхнее окно и выносим его на первый план
198     wlc_view_focus(get_topmost(wlc_view_get_output(view), 0));
199     // перерисовываем все окна
200     layout(wlc_view_get_output(view));
201 }
202
203 // установить окно активным
204 static void view_focus(wlc_handle view, bool focus) {
205     // устанавливаем окну флаг активности
206     wlc_view_set_state(view, WLC_BIT_ACTIVATED, focus);
207 }
208
209 // запрос на перемещение окна
210 static void view_request_move(wlc_handle view, const struct wlc_point *origin) {

```

```

210     start_interactive_move(view, origin);
211 }
212
213 // запрос на изменение размеров окна
214 static void view_request_resize(wlc_handle view, uint32_t edges, const struct
    wlc_point *origin) {
215     start_interactive_resize(view, edges, origin);
216 }
217
218 // запрос на изменение отображения окна
219 static void view_request_geometry(wlc_handle view, const struct wlc_geometry *g) {
220     (void)view, (void)g;
221     // заглушка, не позволяющая изменить окно
222 }
223
224 // обработка событий клавиатуры
225 static bool keyboard_key(wlc_handle view, uint32_t time,
226                         const struct wlc_modifiers *modifiers, uint32_t key, enum
    wlc_key_state state) {
227     (void)time, (void)key;
228     // получаем считанный с клавиатуры символ
229     const uint32_t sym = wlc_keyboard_get_keysym_for_key(key, NULL);
230
231     // если есть активное окно
232     if (view) {
233         // CTRL+q — закрытие окна
234         if (modifiers->mods & WLC_BIT_MOD_CTRL && sym == XKB_KEY_q) {
235             // close the window, if it's not system view
236             if (state == WLC_KEY_STATE_PRESSED && view != statusbar && view !=
                desktop) {
237                 wlc_view_close(view);
238             }
239             return true;
240             // CTRLстрелка+ вниз — перелистывание окон
241         } else if (modifiers->mods & WLC_BIT_MOD_CTRL && sym == XKB_KEY_Down) {
242             if (state == WLC_KEY_STATE_PRESSED) {
243                 wlc_view_send_to_back(view); // отправляем текущее окно на задний план
244                 wlc_view_focus(get_topmost(wlc_view_get_output(view), 0)); //
                устанавливаем новое верхнее окно активным
245             }
246             return true;
247         }
248     }
249
250     // CTRL+Escape — завершение работы
251     if (modifiers->mods & WLC_BIT_MOD_CTRL && sym == XKB_KEY_Escape) {
252         if (state == WLC_KEY_STATE_PRESSED) {
253             wlc_terminate();
254         }
255         return true;
256     // CTRL+Enter — запускаем терминал
257     } else if (modifiers->mods & WLC_BIT_MOD_CTRL && sym == XKB_KEY_Return) {
258         if (state == WLC_KEY_STATE_PRESSED) {
259             char *terminal = (getenv("TERMINAL") ? getenv("TERMINAL") : "weston-
                terminal");
260             wlc_exec(terminal, (char *const[]) { terminal, NULL });
261         }

```

```

262         return true;
263     }
264
265     return false;
266 }
267
268 // обработка нажатий кнопок мыши
269 static bool pointer_button(wlc_handle view, uint32_t time, const struct
    wlc_modifiers *modifiers,
270                          uint32_t button, enum wlc_button_state state, const
    struct wlc_point *position) {
271     (void)button, (void)time, (void)modifiers;
272
273     // если кнопка нажата, то начинаем интерактивное действие
274     if (state == WLC_BUTTON_STATE_PRESSED) {
275         wlc_view_focus(view);
276         if (view) {
277             // CTRLлевая+ кнопка — передвигаем окно
278             if (modifiers->mods & WLC_BIT_MOD_CTRL && button == BTN_LEFT)
279                 start_interactive_move(view, position);
280             // CTRLправая+ кнопка — изменяем размеры окна
281             if (modifiers->mods & WLC_BIT_MOD_CTRL && button == BTN_RIGHT)
282                 start_interactive_resize(view, 0, position);
283         }
284         // иначе завершаем интерактивное действие
285     } else {
286         stop_interactive_action();
287     }
288
289     return (compositor.action.view ? true : false);
290     //return false;
291 }
292
293 // обработка движения мыши
294 static bool pointer_motion(wlc_handle handle, uint32_t time, const struct
    wlc_point *position) {
295     (void)handle, (void)time;
296
297     // если есть активное окно
298     if (compositor.action.view) {
299         // определяем координаты мышки относительно окна
300         const int32_t dx = position->x - compositor.action.grab.x;
301         const int32_t dy = position->y - compositor.action.grab.y;
302         struct wlc_geometry g = *wlc_view_get_geometry(compositor.action.view);
303
304         // если есть запросы на перерисовку границ окна
305         if (compositor.action.edges) {
306             const struct wlc_size min = { 80, 40 }; // минимально допустимые размеры
окна
307
308             struct wlc_geometry n = g;
309             if (compositor.action.edges & WLC_RESIZE_EDGE_LEFT) {
310                 n.size.w -= dx;
311                 n.origin.x += dx;
312             } else if (compositor.action.edges & WLC_RESIZE_EDGE_RIGHT) {
313                 n.size.w += dx;
314             }

```



```

315
316     if (compositor.action.edges & WLC_RESIZE_EDGE_TOP) {
317         n.size.h -= dy;
318         n.origin.y += dy;
319     } else if (compositor.action.edges & WLC_RESIZE_EDGE_BOTTOM) {
320         n.size.h += dy;
321     }
322
323     if (n.size.w >= min.w) {
324         g.origin.x = n.origin.x;
325         g.size.w = n.size.w;
326     }
327
328     if (n.size.h >= min.h) {
329         g.origin.y = n.origin.y;
330         g.size.h = n.size.h;
331     }
332
333     // устанавливаем новые размеры окна
334     wlc_view_set_geometry(compositor.action.view, compositor.action.edges,
335 &g);
336     // если нет запросов на изменение размеров окна, значит мы его перемещаем
337     } else {
338         g.origin.x += dx;
339         g.origin.y += dy;
340         wlc_view_set_geometry(compositor.action.view, 0, &g);
341     }
342
343     // запоминаем текущую позицию мыши
344     compositor.action.grab = *position;
345 }
346
347 // Устанавливаем координаты мышки и возвращаем управление композитору
348 wlc_pointer_set_position(position);
349 return (compositor.action.view ? true : false);
350 }
351
352 // функция логирования
353 static void cb_log(enum wlc_log_type type, const char *str) {
354     (void)type;
355     printf("%s\n", str); // выводим все в стандартный поток вывода
356 }
357
358 int main(int argc, char** argv) {
359     // parse input arguments
360     char* config = DEFAULT_CONFIG;
361     if (argc < 2) {
362         printf("No config file specified, using default: %s\n", config);
363     } else {
364         config = argv[1];
365         printf("Using config file: %s\n", config);
366     }
367     // parse config
368     init_config(config);
369
370     // устанавливаем функцию логгер-

```

```

371 wlc_log_set_handler(cb_log);
372
373 // устанавливаем функцию, которая отвечает за перерисовку всех окон при смене разрешения
374 wlc_set_output_resolution_cb(output_resolution);
375 // устанавливаем функцию, которая отвечает за обработку запущенный приложений
376 wlc_set_view_created_cb(view_created);
377 // устанавливаем функцию, которая отвечает за обработку завершившихся приложений
378 wlc_set_view_destroyed_cb(view_destroyed);
379 // устанавливаем функцию, которая отвечает за смену фокуса между окнами выбирает(
активное приложение)
380 wlc_set_view_focus_cb(view_focus);
381 // устанавливаем функцию, которая отвечает за передвижение окна ОМ по экрану
382 wlc_set_view_request_move_cb(view_request_move);
383 // устанавливаем функцию, которая отвечает за смену размеров окна ОМ
384 wlc_set_view_request_resize_cb(view_request_resize);
385 // устанавливаем функцию, которая отвечает за смену изменения отображения окна ОМ
386 wlc_set_view_request_geometry_cb(view_request_geometry);
387 // устанавливаем функциюобработчик– нажатий на клавиатуру
388 wlc_set_keyboard_key_cb(keyboard_key);
389 // устанавливаем функцию, которая отвечает за обработку нажатия кнопок мыши
390 wlc_set_pointer_button_cb(pointer_button);
391 // устанавливаем функцию, которая отвечает за передвижение мыши
392 wlc_set_pointer_motion_cb(pointer_motion);
393
394 // инициализируем композитор
395 if (!wlc_init())
396     return EXIT_FAILURE;
397
398 // spawning process that starts statusbar
399 statusbar_pid = fork();
400 if (statusbar_pid < 0) {
401     printf("Startup_launch_failure\n");
402     return EXIT_FAILURE;
403
404 } else if (statusbar_pid == 0) {
405     const char *statusbar = get_statusbar();
406     printf("Running_statusbar_%s\n", statusbar);
407     execv(statusbar, (char *const[]) {statusbar, NULL});
408
409 } else {
410     // spawning process that starts desktop
411     desktop_pid = fork();
412     if (desktop_pid < 0) {
413         printf("Startup_launch_failure\n");
414         return EXIT_FAILURE;
415
416     } else if (desktop_pid == 0) {
417         // sleep for some time, wait until the WM actually starts
418         usleep(250000);
419         const char *desktop = get_desktop();
420         printf("Running_desktop_%s\n", desktop);
421         execv(desktop, (char *const[]) {desktop, NULL});
422
423     }
424
425     printf("Running_WM\n");
426     wlc_run();

```

```

427     }
428
429     return EXIT_SUCCESS;
430 }

```

Листинг 2: Файл config.h

```

1  //
2  // Created by kivi on 17.05.17.
3  //
4
5  #ifndef XXONWM_CONFIG_H
6  #define XXONWM_CONFIG_H
7
8  void init_config(const char* config_file);
9  const char* get_statusbar();
10 const char* get_desktop();
11
12 #endif //XXONWM_CONFIG_H

```

Листинг 3: Файл config.c

```

1  //
2  // Created by kivi on 17.05.17.
3  //
4  #include <string.h>
5  #include <stdlib.h>
6
7  #include "ini.h"
8
9  typedef struct {
10     char* statusbar;
11     char* desktop;
12 } configuration;
13
14 configuration config = { NULL, NULL };
15
16 static int handler(void* user, const char* section, const char* name,
17                    const char* value)
18 {
19     configuration* pconfig = (configuration*)user;
20
21     #define MATCH(s, n) strcmp(section, s) == 0 && strcmp(name, n) == 0
22     if (MATCH("statusbar", "exe")) {
23         pconfig->statusbar = strdup(value);
24     } else if (MATCH("desktop", "exe")) {
25         pconfig->desktop = strdup(value);
26     } else {
27         return 0; /* unknown section/name, error */
28     }
29     return 1;
30 }
31
32 void init_config(const char* config_file) {
33     if (ini_parse(config_file, handler, &config) < 0) {
34         printf("Can't load %s\n", config_file);
35         exit(1);

```

```

36     }
37 }
38
39 const char* get_statusbar() {
40     return config.statusbar;
41 }
42
43 const char* get_desktop() {
44     return config.desktop;
45 }

```

Листинг 4: Файл сборки CMakeLists.txt

```

1  cmake_minimum_required(VERSION 3.7)
2  project(xxonwm)
3  set(CMAKE_C_STANDARD 99)
4
5  # includes for wlc
6  include_directories(/usr/include)
7  include_directories(/usr/local/include)
8  include_directories(/usr/local/include/lib/chck)
9  link_directories(/usr/local/lib64/)
10
11 # include inih
12 add_subdirectory(inih)
13 include_directories(inih)
14 link_directories(inih)
15
16
17 set(SOURCE_FILES main.c config.c)
18 add_executable(xxonwm ${SOURCE_FILES})
19
20 target_link_libraries(xxonwm wlc inih)

```

Листинг 5: Стандартный файл конфигурации config.ini

```

1  [statusbar]
2  exe=/home/kivi/workspace/Phone/src/status_bar/status
3
4  [desktop]
5  exe=/home/kivi/workspace/Phone/src/desktop/desktop

```

Листинг 6: Файл ярлыка для экранного менеджера xxon.desktop

```

1  [Desktop Entry]
2  Name=XXwm
3  Comment=Mobile Wayland window manager
4  Exec=/home/kivi/workspace/XXwm/xxonwm
5  Type=Application

```