Балтийский государственный технический университет «ВОЕНМЕХ»
им. Д. Ф. Устинова

Кафедра И5
«Информационные системы и программная инженерия»

Практическое задание № 2
по дисциплине «Программирование на ЯВУ»
на тему «События в SDL 2.0»

Выполнил:
Студент Дубровский В.И.
Группа И582

Преподаватель:
Спирин Д.О.

Санкт-Петербург
2019

Цель работы:

Изучить основные события библиотеки SDL2.0, разобраться с проектированием простейших приложений с графическим интерфейсом с использованием классов.

Задание:

1. В файле l2.c приведен текст программы: разобраться и устранить возможные ошибки, проанализировать виды событий и причины их срабатывания.
2. В программе перевести все сообщения на русский язык.
3. Изменить логику построения и структуру программы взяв за основу ЛР 1.
4. На основе полученных знаний написать программу, создающую два окна. В первом фиксируются движение мыши по экрану, во втором выводятся координаты положения курсора и наоборот. В каждом окне имеется кнопка, включающая/отключающая слежение за мышью с выводом соответствующего сообщения.
5. Реализовать возможность перемещения кнопки в пределах окна.

Выполнение:

В программе были обнаружены и устранены следующие ошибки:

1) 164. event.window.windowID = SDL_GetWindowID(param);, param — void* явно не приводится к SDL_Window*
2) 177. SDL_HideWindow(param);, param — void* явно не приводится к SDL_Window*

Каждое из событий было проанализировано, каждое событие было связано с определенным событием SDL2. Все сообщения были переведены на русский.

Взяв за основу 1 лабораторную работу, программа была написана в объектно-ориентированном стиле.

Текст программы:

main.cpp

```
#include <iostream>
#include "Application.h"

constexpr auto TITLE = "Laboratornay rabota 2. Remake.";
constexpr auto WIDTH = 500;
constexpr auto HEIGHT = 500;

Application application(TITLE, WIDTH, HEIGHT);

int main(int argc, char* argv[])
{
	try {
		return application.run();
	}
	catch (std::exception& e)
	{
		std::cerr << e.what() << std::endl;
		application.destroy();
		return 1;
	}
```

```
}
```

EventFilter.h

```
#pragma once
#include <SDL.h>
#include <stdlib.h>

void fillScreen(SDL_Window* window);
int asmFunction(void);
int eventFilter(void* userdata, SDL_Event* event);
Uint32 repeatOnceFunction(Uint32 interval, void* param);
Uint32 customEventFunction(Uint32 interval, void* param);
```

EventFilter.cpp

```
#include "EventFilter.h"


void fillScreen(SDL_Window* window) {
	SDL_Surface* screen = SDL_GetWindowSurface(window);

	SDL_FillRect(screen, NULL, SDL_MapRGB(screen->format, rand() % 255, rand() % 255,
rand() % 255));

	SDL_UpdateWindowSurface(window);
}

int asmFunction(void) {
	static int internalValue = 1;

#ifdef __GNUC__
	__asm__("movl %0, %%eax\n\t"
		"add %%eax, %0"
		: "=r" (internalValue)
		: "r" (internalValue));
#elif _MSC_VER
	_asm {
		mov eax, internalValue
		add internalValue, eax
	};
#endif

	return internalValue;
}

int eventFilter(void* userdata, SDL_Event* event) {
	switch (event->type) {
	case SDL_KEYDOWN:
		if (event->key.keysym.sym == SDLK_q && event->key.keysym.mod == KMOD_LCTRL)
{
			SDL_Event exitEvent = { SDL_QUIT };
			SDL_PushEvent(&exitEvent);
		}

		SDL_Log("Кнопочка вниз/key Down %d", event->key.keysym.sym);
		break;
	case SDL_KEYUP:
		SDL_Log("Кнопочка вверх/key Up %d", event->key.keysym.sym);
		break;
	case SDL_TEXTEDITING:
		SDL_Log("Клавиатурка редактирует текст/Keyboard text editing (composition).
Composition is '%s', cursor start from %d and selection lenght is %d", event->edit.text,
event->edit.start, event->edit.length);
		break;
```

```
	case SDL_TEXTINPUT:
		SDL_Log("Ввод текста с клавиатуры/Keyboard text input. Text is '%s'",
event->text.text);
		break;
	case SDL_FINGERMOTION:
		SDL_Log("Пальчик/Finger: %lld, x: %f, y: %f", event->tfinger.fingerId,
event->tfinger.x, event->tfinger.y);
		break;
	case SDL_FINGERDOWN:
		SDL_Log("Пальчик вниз/Finger: %lld down - x: %f, y: %f",
			event->tfinger.fingerId, event->tfinger.x, event->tfinger.y);
		return 1;
	case SDL_FINGERUP:
		SDL_Log("Пальчик вверх/Finger: %lld up - x: %f, y: %f", event-
>tfinger.fingerId, event->tfinger.x, event->tfinger.y);
		break;
	case SDL_MULTIGESTURE:
		SDL_Log("Множественный жест/Multi Gesture: x = %f, y = %f, dAng = %f, dR =
%f", event->mgesture.x, event->mgesture.y, event->mgesture.dTheta, event-
>mgesture.dDist);
		SDL_Log("Множественнный жест: циферкаснизукоснуться/ Multi Gesture:
numDownTouch = %i", event->mgesture.numFingers);
		break;
	case SDL_DOLLARGESTURE:
		SDL_Log("Жест номер исполнен, ошибка/Gesture %lld performed, error: %f",
event->dgesture.gestureId, event->dgesture.error);
		break;
	case SDL_DOLLARRECORD:
		SDL_Log("Записанный жест/Recorded gesture: %lld", event-
>dgesture.gestureId);
		break;
	case SDL_MOUSEMOTION:
		SDL_Log("Мышку подвигали/Mouse Move. X=%d, Y=%d, Относительно/RelativeX=%d,
Относительно/RelativeY=%d", event->motion.x, event->motion.y, event->motion.xrel, event-
>motion.yrel);
		break;
	case SDL_MOUSEBUTTONDOWN:
		if (event->button.button == SDL_BUTTON_LEFT)
			asmFunction();

		SDL_Log("Мышечка кнопка вниз/Mouse Button Down %u", event->button.button);
		break;
	case SDL_MOUSEBUTTONUP:
		SDL_Log("Мышечка кнопка вверх/Mouse Button Up %u", event->button.button);
		break;
	case SDL_MOUSEWHEEL:
		SDL_Log("Мышечка колёсико/Mouse Wheel X=%d, Y=%d", event->wheel.x, event-
>wheel.y);
		break;
	case SDL_QUIT:
		SDL_Log("Пользовательский-запрос выйти/User-requested quit");
		return 1;
	case SDL_WINDOWEVENT:
		switch (event->window.event) {
		case SDL_WINDOWEVENT_SHOWN:
			SDL_Log("Окошко номер показано/Window %d shown", event-
>window.windowID);
			break;
		case SDL_WINDOWEVENT_HIDDEN:
			SDL_Log("Окошко номер спрятано/Window %d hidden", event-
>window.windowID);
			break;
		case SDL_WINDOWEVENT_EXPOSED:
			fillScreen(SDL_GetWindowFromID(event->window.windowID));
```

```
                        SDL_Log("Окошко номер обнаружено/Window %d exposed", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_MOVED:
                        SDL_Log("Окошко номер передвинуто ->/Window %d moved to %d,%d", event->window.windowID, event->window.data1, event->window.data2);
                        break;
                case SDL_WINDOWEVENT_RESIZED:
                        SDL_Log("Окошко номер изменено ->/Window %d resized to %dx%d", event->window.windowID, event->window.data1, event->window.data2);
                        break;
                case SDL_WINDOWEVENT_SIZE_CHANGED:
                        SDL_Log("Окошко номер размер изменён ->/Window %d size changed to %dx%d", event->window.windowID, event->window.data1, event->window.data2);
                        break;
                case SDL_WINDOWEVENT_MINIMIZED:
                        SDL_Log("Окошко номер минимизировано/Window %d minimized", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_MAXIMIZED:
                        SDL_Log("Окошко номер максимизировано/Window %d maximized", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_RESTORED:
                        SDL_Log("Окошко номер Window %d restored", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_ENTER:
                        SDL_Log("Мышка вошла в окно/Mouse entered window %d", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_LEAVE:
                        SDL_Log("Мышка покинуло окно/Mouse left window %d", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_FOCUS_GAINED:
                        SDL_Log("Окно номер было под фокусом клавиатуры/Window %d gained keyboard focus", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_FOCUS_LOST:
                        SDL_Log("Окно номер потеряла фокус клавиатуры/Window %d lost keyboard focus", event->window.windowID);
                        break;
                case SDL_WINDOWEVENT_CLOSE:
                        SDL_Log("Окно номер закрыто/Window %d closed", event->window.windowID);
                        break;
                default:
                        SDL_Log("Окно номер получило неизвестный эвент/Window %d got unknown event %d", event->window.windowID, event->window.event);
                        break;
                }
                break;
        default:
                SDL_Log("Получили неизвестный эвент/Got unknown event %d", event->type);
                break;
        }

        return 0;
}

Uint32 customEventFunction(Uint32 interval, void* param) {
        SDL_Event event = { SDL_WINDOWEVENT };

        SDL_Log("Timer signaled with interval %d ms", interval);
```

```
	event.window.windowID = SDL_GetWindowID((SDL_Window*)param);
	event.window.event = SDL_WINDOWEVENT_EXPOSED;

	SDL_PushEvent(&event);
	return(interval);
}

Uint32 repeatOnceFunction(Uint32 interval, void* param) {
	SDL_Event exitEvent = { SDL_QUIT };

	SDL_Log("Timer signaled with interval %d ms", interval);

	if (asmFunction() != 0) {
		SDL_HideWindow((SDL_Window*)param);

		SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_INFORMATION, "Something going
wrong", "Find me! I'm scared", NULL);

		SDL_Delay(15000); /* 15 sec */
		SDL_LogWarn(SDL_LOG_CATEGORY_APPLICATION, "You didn't find me! You
disappointed me... I'm leaving.");

		SDL_PushEvent(&exitEvent);
	}

	return 0;
}
```

Application.h

```
#pragma once
#include<SDL.h>
#include<memory>
#include<vector>


class Application
{
private:

	bool _isRunning = false;
	SDL_Window* _window = nullptr;
	SDL_Renderer* _renderer = nullptr;
	SDL_Event _events;

	SDL_TimerID repeatOnceFunctionTimer;
	SDL_TimerID customEventFunctionTimer;

	void init();
	void render();
	void setTimer(SDL_TimerID timer, SDL_TimerCallback callback, void* param);
public:
	Application() = delete;
	explicit Application(const char* title, int w, int h);
	~Application();
	void destroy();
	int run();

};
```

Application.cpp

```cpp
#include "Application.h"
#include <iostream>
#include <SDL_ttf.h>
#include "EventFilter.h"
#include <exception>

Application::Application(const char* title, int w, int h)
{
	if(SDL_Init(SDL_INIT_EVERYTHING) == -1) std::exception(static_cast<const
char*>(SDL_GetError()));
	if (TTF_Init() == -1) throw std::exception(static_cast<const
char*>(TTF_GetError()));

	_window = SDL_CreateWindow(title, SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
w, h, SDL_WINDOW_SHOWN);
	if (_window) std::cerr << "Ok Window" << std::endl;
	else std::exception(static_cast<const char*>(SDL_GetError()));

	_renderer = SDL_CreateRenderer(_window, -1, NULL);
	if (_renderer) std::cerr << "Ok Renderer" << std::endl;
	else std::exception(static_cast<const char*>(SDL_GetError()));


}

Application::~Application()
{

}

void Application::destroy()
{
	SDL_RemoveTimer(repeatOnceFunctionTimer);
	SDL_RemoveTimer(customEventFunctionTimer);
	SDL_DestroyRenderer(_renderer);
	SDL_DestroyWindow(_window);
	SDL_Quit();
	TTF_Quit();
}



void Application::init()
{
	_isRunning = true;
}



void Application::render()
{
	SDL_SetRenderDrawColor(_renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
	SDL_RenderClear(_renderer);

	SDL_RenderPresent(_renderer);
}




int Application::run()
{
	SDL_SetEventFilter(eventFilter, nullptr);
```

```
	if (!(customEventFunctionTimer = SDL_AddTimer(2000 /* 2 sec */,
customEventFunction, _window))) {
		SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR, "Error", "Unable to create
custom event timer. See the log for more info.", _window);
		SDL_LogCritical(SDL_LOG_CATEGORY_APPLICATION, "Unable to create custom
event timer, error: %s", SDL_GetError());
	}

	if (!(repeatOnceFunctionTimer = SDL_AddTimer(10000 /* 10 sec */,
repeatOnceFunction, _window))) {
		SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR, "Error", "Unable to create
repeat once timer. See the log for more info.", _window);
		SDL_LogCritical(SDL_LOG_CATEGORY_APPLICATION, "Unable to create repeat once
timer, error: %s", SDL_GetError());
	}

	SDL_WaitEvent(NULL);

	destroy();
	return 0;
}
```

Программа работает и выполняет те же функции, как и приведенная в вложеном файле l2.c

На основе полученных знаний была написана программа для пункта 4.

Основные классы приложения:

1) BaseApp — базовый класс для всех графических элементов приложения. Задаёт единый интерфейс и поля.
2) Window — класс реализующий абстракцию окна. От него наследуются окна, специализацией которых занимается разработчик.
3) Signal — класс содержащий в себе массив указателей на функции и метод их вызова.
4) CallBack — класс реализующий статические методы связывания обратного вызова методов
5) ErrorLoger — класс, который наследуется от std::exception. Основной класс исключений приложения.
6) Fonts — класс отвечающий за работу с файлами, в конкретном случае с шрифтами.
7) Widget — базовый класс для виджетов приложения.
8) Label — класс реализующий надписи.
9) Button — класс реализующий кнопку.
10) Application – класс реализующий основную логику приложения. От него наследуются производные классы приложений, специализацией которых занимается разработчик.

По заданию было сделана возможность передвижения кнопки по нажатию правой кнопки мыши. Результат работы программы представлен на рисунках 1-2.

Рисунок 1 — Демонстрация работы программы

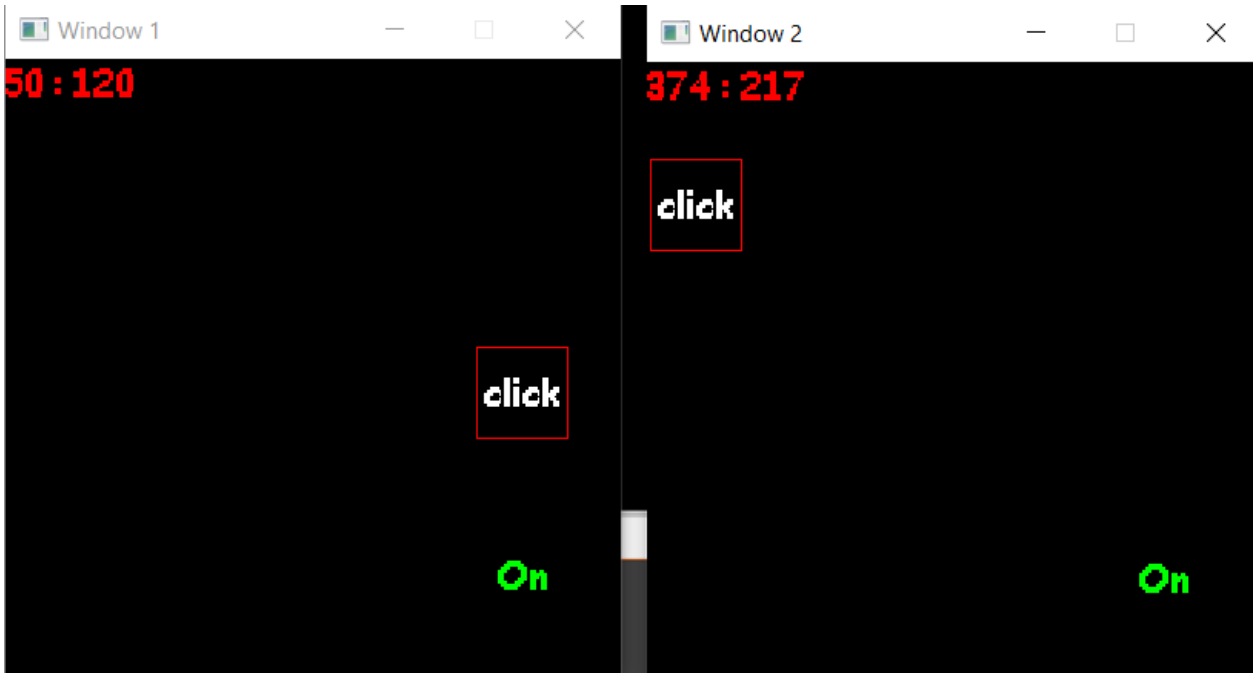

Рисунок 2 — Демонстрация передвинутых кнопок