БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий

и робототехники

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

О Т Ч Ё Т

о лабораторной работе № 3

Дисциплина

«Системное программирование»

Тема

«Программирование внешних устройств»

Выполнил: студент гр. 10702217 Храмков Д. С.

Проверил: Разорёнов Н. А.

Минск 2019

***Лабораторная работа № 3***

**ПРОГРАММИРОВАНИЕ ВНЕШНИХ УСТРОЙСТВ**

**Цель работы:** изучить основы программирования аппаратных устройств: клавиатуры, мыши, таймера.

**Изучаемые вопросы**

1. Клавиатура.
   1. Механизм сообщений от клавиатуры.
   2. Коды ОЕМ, ANSI, ASCII, виртуальные коды.
   3. Виртуальные клавиши.
   4. Символьные сообщения.
   5. Системные сообщения.
   6. Преобразование кодов.
   7. Как отличить нажатия совпадающих клавиш.
   8. Как узнать ввод строчных и прописных букв.
   9. Фокус ввода.
2. Мышь.
   1. Захват и освобождение окном мыши.
   2. Обработка сообщений от мыши.
   3. Координаты x, y.
   4. Курсор мыши. Управление курсором.
   5. Сообщения мыши в клиентской и неклиентской области окна.
3. Таймер.
   1. Установка и снятие таймера.
   2. Функция таймера.

***Постановка задачи***

На базе архитектуры WIN 32 Application создать многооконное приложение, где первое окно отображает информацию по клавиатуре, второе окно отображает информацию по мыши и третье окно отображает информацию по таймеру.

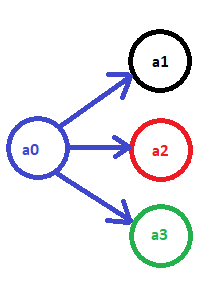
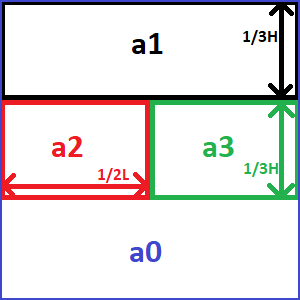


Рис1. Схема положения окон Рис2. Схема зависимости окон

1. **Клавиатура.**
   1. **Механизм сообщений от клавиатуры.**

**Когда пользователь нажимает или отпускает клавишу, Windows посылает в программу сообщеие ввода с клавиатуры. Ниже приводится перечень сообщений ввода с клавиатуры и события, их вызывающие:**

case WM\_KEYDOWN:

{

symbol[12] = wParam;

a[0] = (int)lParam >> 16;

a[1] = '\0';

CharToOem((LPCWSTR)a, (LPSTR)a);

swprintf(&vCode[18], L"%d", VkKeyScan(wParam));

swprintf(&OEMCode[14], L"%d", a[0]);

swprintf(&ANSICode[15], L"%d", wParam);

swprintf(&ASCIICode[16], L"%d", wParam);

if ((int)lParam >> 16 == 42) {

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: LShift");

}

else if ((int)lParam >> 16 == 54) {

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: RShift");

}

symbol[12] = wParam;

SendMessage(keyState, WM\_SETTEXT, 0, (LPARAM)L"Key State: Pressed");

SendMessage(keyCodeVirtual, WM\_SETTEXT, 0, (LPARAM)vCode);

SendMessage(keyCodeOEM, WM\_SETTEXT, 0, (LPARAM)OEMCode);

SendMessage(keyCodeANSI, WM\_SETTEXT, 0, (LPARAM)ANSICode);

SendMessage(keyCodeASCII, WM\_SETTEXT, 0, (LPARAM)ASCIICode);

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

break;

}

case WM\_KEYUP:

{

SendMessage(keyState, WM\_SETTEXT, 0, (LPARAM)L"Key State: None");

SendMessage(keyCodeVirtual, WM\_SETTEXT, 0, (LPARAM)L"Key Virtual Code: None");

SendMessage(keyCodeOEM, WM\_SETTEXT, 0, (LPARAM)L"Key OEM Code: None");

SendMessage(keyCodeANSI, WM\_SETTEXT, 0, (LPARAM)L"key ANSI Code: None");

SendMessage(keyCodeASCII, WM\_SETTEXT, 0, (LPARAM)L"Key ASCII Code: None");

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: None");

break;

}

case WM\_CHAR:

{

symbol[12] = wParam;

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

break;

}

* 1. **Коды ОЕМ, ANSI, ASCII, виртуальные коды.**

**OEM код хранится в lParam с 16 по 23 бит.**

**Виртуальный код клавиши – это независимое от устройства назначение клавиши. Windows использует виртуальные коды клавиш для того, чтобы не зависеть от используемой персональной ЭВМ. Виртуальный код передаётся в параметре wParam.**

* 1. **Виртуальные клавиши.**

**Когда пользователь нажимает или отпускает клавишу, Windows посылает в программу сообщение ввода с клавиатуры. Параметр wParam определяет виртуальный код данной клавиши.**

swprintf(&vCode[18], L"%d", VkKeyScan(wParam));

* 1. **Символьные сообщения.**

**Прикладная программа, которая читает символы, вводимые с клавиатуры, должна использовать функцию TranslateMessage() в своём цикле обработки сообщений. Эта функция преобразует сообщение ввода с клавиатуры в соответствующее сообщение ввода символа в коде ANSI – WM\_CHAR или WM\_SYSCHAR. Эти сообщения содержат ANSI-коды символа для данной клавиши в параметре wParam. Параметр lParam аналогичен таковому в сообщении ввода с клавиатуры.**

case WM\_SYSCHAR:

{

symbol[12] = wParam;

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

break;

}

* 1. **Системные сообщения.**

**Для системных клавиш Windows генерирует соответствующие сообщения WM\_SYSKEYUP и WM\_SYSKEYDOWN. Это специальные клавиши, например, Alt и F10, которые принадлежат интерфейсу пользователя и не могут использоваться прикладной программой другим способом.**

case WM\_SYSKEYDOWN:

{

symbol[12] = wParam;

a[0] = wParam;

a[1] = '\0';

CharToOem((LPCWSTR)a, (LPSTR)a);

swprintf(&vCode[18], L"%d", VkKeyScan(wParam));

swprintf(&OEMCode[14], L"%d", a[0]);

swprintf(&ANSICode[15], L"%d", wParam);

if (wParam > 0 && wParam <= 127) {

swprintf(&ASCIICode[16], L"%d", wParam);

SendMessage(keyCodeASCII, WM\_SETTEXT, 0, (LPARAM)ASCIICode);

}

symbol[12] = wParam;

SendMessage(keyState, WM\_SETTEXT, 0, (LPARAM)L"Key State: Pressed");

SendMessage(keyCodeVirtual, WM\_SETTEXT, 0, (LPARAM)vCode);

SendMessage(keyCodeOEM, WM\_SETTEXT, 0, (LPARAM)OEMCode);

SendMessage(keyCodeANSI, WM\_SETTEXT, 0, (LPARAM)ANSICode);

SendMessage(keyCodeASCII, WM\_SETTEXT, 0, (LPARAM)ASCIICode);

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

switch (wParam) {

case 0x31:

//if (GetAsyncKeyState(VK\_MENU)) {

DestroyWindow(window2);

//}

break;

case '2':

DestroyWindow(window3);

break;

}

break;

}

case WM\_SYSKEYUP:

{

SendMessage(keyState, WM\_SETTEXT, 0, (LPARAM)L"Key State: None");

SendMessage(keyCodeVirtual, WM\_SETTEXT, 0, (LPARAM)L"Key Virtual Code: None");

SendMessage(keyCodeOEM, WM\_SETTEXT, 0, (LPARAM)L"Key OEM Code: None");

SendMessage(keyCodeANSI, WM\_SETTEXT, 0, (LPARAM)L"Key ANSI Code: None");

SendMessage(keyCodeASCII, WM\_SETTEXT, 0, (LPARAM)L"Key ASCII Code: None");

SendMessage(keySymbol , WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: None");

break;

}

* 1. **Преобразование кодов.**

**Преобразование кодов нужно для корректного отображения вводимых символов. Например, функция CharToOem преобразовывает строку из кодировки ANSI в кодировку OEM.**

a[0] = wParam;

a[1] = '\0';

CharToOem((LPCWSTR)a, (LPSTR)a);

* 1. **Как отличить нажатия совпадающих клавиш.**

**На клавиатуре дублируются некоторые клавиши. Например, левый и правый Shift, Ctrl и др. Чтобы их отличить проверяют состояние 24 бита в lParam.**

if ((int)lParam >> 16 == 42) {

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: LShift");

}

else if ((int)lParam >> 16 == 54) {

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)L"Key Symbol: RShift");

}

* 1. **Как узнать ввод строчных и прописных букв.**

**Можно проверить была ли нажата до этого клавиша Shift.**

case WM\_CHAR:

{

symbol[12] = wParam;

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

break;

}

case WM\_SYSCHAR:

{

symbol[12] = wParam;

SendMessage(keySymbol, WM\_SETTEXT, 0, (LPARAM)symbol);

break;

}

* 1. **Фокус ввода.**

**Фокус ввода – состояние окна, в котором оно получает сообщения от клавиатуры. Такое окно может быть только одно.**

SetFocus(timerState3);

1. **Мышь.**
   1. **Захват и освобождение окном мыши.**

**Windows посылает окну сообщение мыши только в том случае, если курсор мыши находится в окне, или ввод от мыши захвачен с помощью функции SetCapture(). Эта функция предписывает Windows послать все сообщения ввода мыши (вне зависимости от того, где находится курсор мыши) в указанное окно. Прикладная программа обычно использует эту функцию для того, чтобы взять на себя управление при выполнении некоторых критических операций, таких как выборка с помощью мыши некоторого объекта в области пользователя. Захват ввода от мыши предотвращает получение управления другой прикладной программой до завершения операции.**

**Поскольку мышь является разделяемым ресурсом, важно освободить её сразу по завершении операции. Освобождение мыши делается функцией ReleaseCapture(). Для определения окна, захватившего мышь, используется функция GetCapture().**

SetCapture(window2);

GetCapture();

ReleaseCapture();

* 1. **Обработка сообщений от мыши.**

**При вводе с помощью мыши Windows посылает прикладной программе сообщение от мыши, когда пользователь перемещает курсор мыши в/через окно или нажимает/освобождает кнопку мыши, находясь в окне.**

case WM\_LBUTTONDOWN:

{

SendMessage(mouseState1, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 1: Left Button Pressed");

break;

}

case WM\_LBUTTONUP:

{

SendMessage(mouseState1, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 1: None");

break;

}

case WM\_RBUTTONDOWN:

{

SendMessage(mouseState1, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 1: Right Button Pressed");

break;

}

case WM\_RBUTTONUP:

{

SendMessage(mouseState1, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 1: None");

break;

}

case WM\_MOUSEMOVE:

{

x = LOWORD(lParam);

y = HIWORD(lParam);

swprintf(&mX[9], L"%d", x);

swprintf(&mY[9], L"%d", y);

SendMessage(mouseX, WM\_SETTEXT, 0, (LPARAM)mX);

SendMessage(mouseY, WM\_SETTEXT, 0, (LPARAM)mY);

SendMessage(mouseState2, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 2: In client widnow");

break;

}

case WM\_NCMOUSEMOVE:

{

SendMessage(mouseState2, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 2: Not in client window");

break;

}

* 1. **Координаты x, y.**

**Параметр lParam содержит координаты x и у курсора мыши.**

x = LOWORD(lParam);

y = HIWORD(lParam);

* 1. **Курсор мыши. Управление курсором.**

**Для того, чтобы использовать курсор определённой формы, вы должны получить его дескриптор с помощью функции LoadCursor() (необходимо использовать в обработчике WM\_MOUSEMOVE). После того, как ваша программа загрузила курсор, она может использовать его при необходимости.**

LoadCursor(hInst, IDC\_HELP);

* 1. **Сообщения мыши в клиентской и неклиентской области окна.**

**Windows посылает окну сообщение мыши только в том случае, если курсор мыши находится в окне, или ввод от мыши захвачен с помощью функции SetCapture().**

case WM\_MOUSEMOVE:

{

x = LOWORD(lParam);

y = HIWORD(lParam);

swprintf(&mX[9], L"%d", x);

swprintf(&mY[9], L"%d", y);

SendMessage(mouseX, WM\_SETTEXT, 0, (LPARAM)mX);

SendMessage(mouseY, WM\_SETTEXT, 0, (LPARAM)mY);

SendMessage(mouseState2, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 2: In client widnow");

break;

}

case WM\_NCMOUSEMOVE:

{

SendMessage(mouseState2, WM\_SETTEXT, 0, (LPARAM)L"Mouse State 2: Not in client window");

break;

}

1. **Таймер.**
   1. **Установка и снятие таймера.**

**Информация от таймера поступает в виде сообщения WM\_TIMER через очередь прикладной программы или от функции многократного вызова, которая определяется при вызове функции SetTimer. Чтобы снять таймер нужно прописать функцию KillTimer.**

case WM\_CREATE:

{

SetTimer(hWnd, timerID1, delta1, NULL);

SetTimer(hWnd, timerID2, delta2, NULL);

SetTimer(hWnd, timerID3, delta3, NULL);

break;

}

case WM\_TIMER:

{

switch (wParam) {

case timerID1:

{

//some code

KillTimer(hWnd, timerID1);

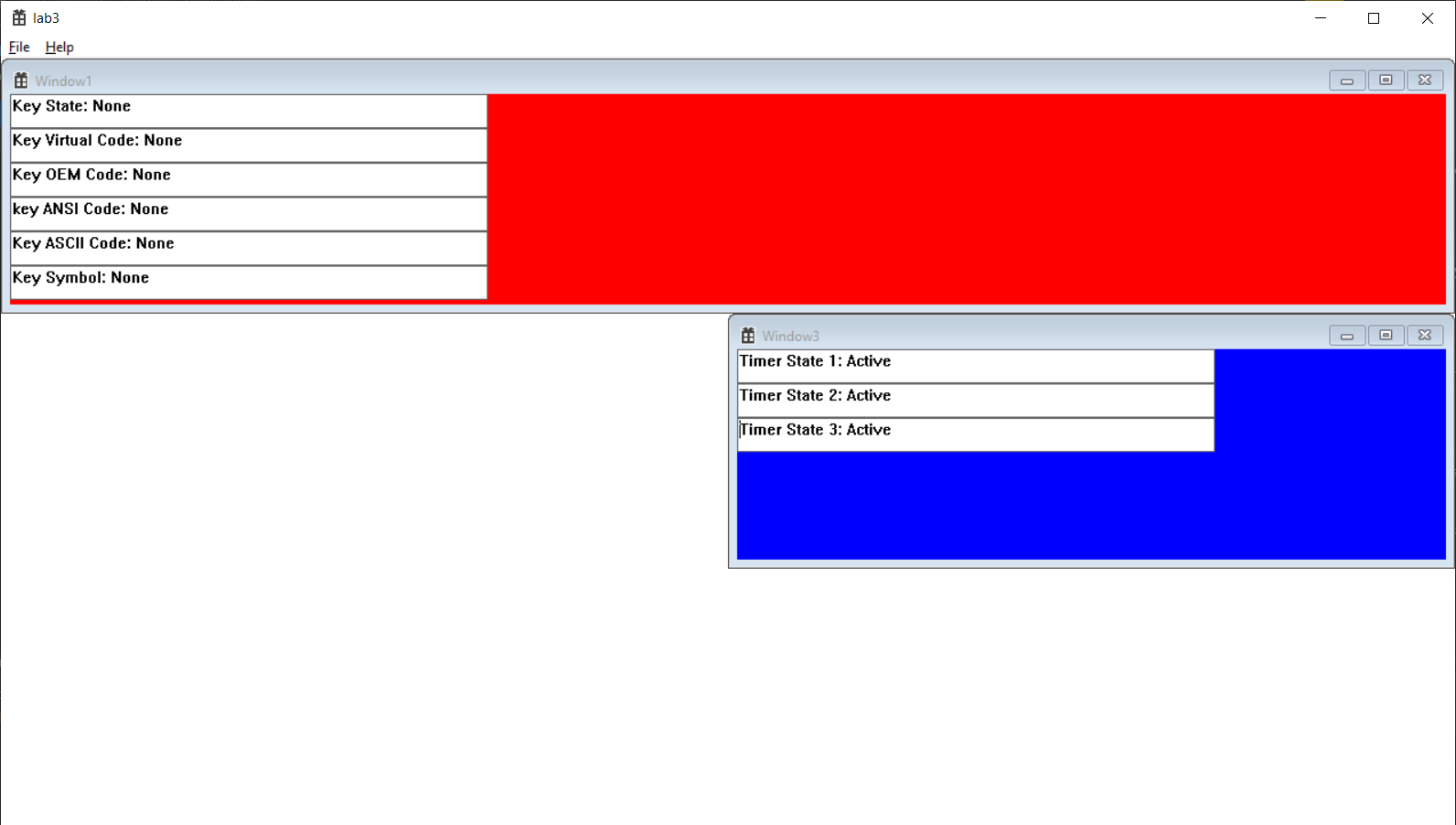
}

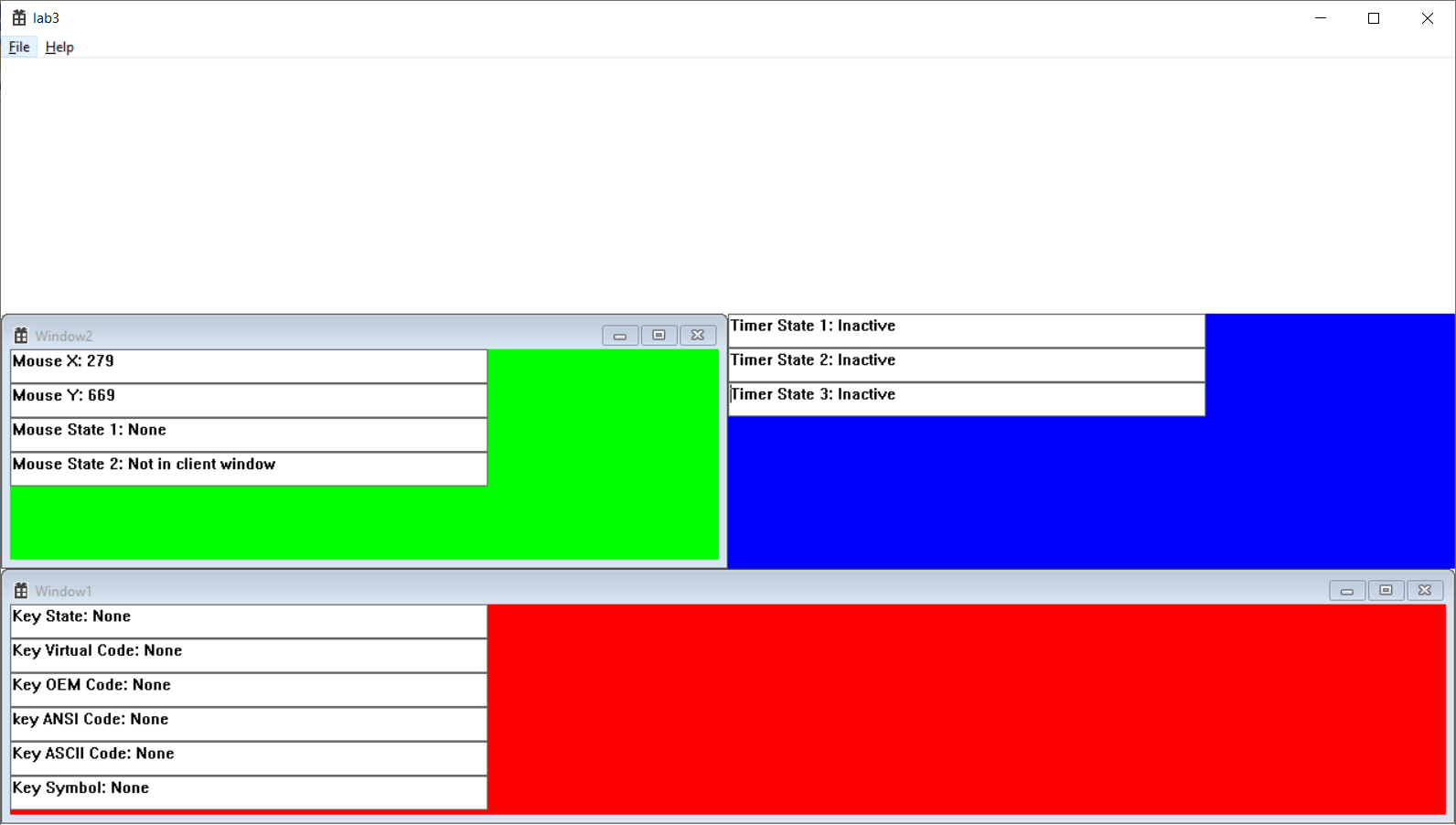
* 1. **Функция таймера.**

При установке таймера функцией SetTimer можно определить функцию, которая будет вызываться при каждом событии от таймера. Для этого последним параметром передаётся указатель на функцию обработчик. Если он есть, то просто происходит вызов функции, если нет, то общение происходит через очередь прикладной программы.

SetTimer(hWnd, timerID3, delta3, (TIMERPROC)GetProcessTimes);

**Результаты работы программы:**





**Выводы:**

1. **Научились обрабатывать сообщения, посылаемые мышью;**
2. **Задали команды на выполнение при нажатии на клавиши.**
3. **Разобрались в механизме работы таймера;**
4. **Установили обработку комбинации клавиш.**