ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ИНФОРМАТИКИ И МИКРОЭЛЕКТРОНИКИ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ОТЧЕТ

По лабораторной работе №6
по Програмированию для Windows
Тема: Таймер

Выполнила: ст. гр. TI-155 Зверкова К.

Проверил: Скроб С.

Кишинев 2017

1. Цель лабораторной работы

Изучение принципов работы с системным таймером.

2. Теоретические понятия

Таймер в Windows является устройством ввода информации, которое периодически извещает приложение о том, что истек заданный интервал времени. Ваша программа задает Windows интервал, как бы говоря системе: "Подталкивай меня каждые 10 секунд." Тогда Windows посылает вашей программе периодические сообщения WM_TIMER, сигнализируя об истечении интервала времени. Сначала таймер Windows может показаться менее важным устройством ввода, чем клавиатура или мышь, и, конечно, это верно для многих приложений Windows. Но таймер более полезен, чем вы можете подумать, и не только для программ, которые индицируют время, таких как программа Windows clock, появляющаяся на панели задач. Существуют также и другие применения таймера в Windows, некоторые из которых, может быть, не столь очевидны:

- Многозадачность — Хотя Windows 95 является вытесняющей многозадачной средой, иногда самое эффективное решение для программы — как можно быстрее вернуть управление Windows. Если программа должна выполнять большой объем работы, она может разделить задачу на части и отрабатывать каждую часть при получении сообщения WM_TIMER.
- Поддержка обновления информации о состоянии — Программа может использовать таймер для вывода на экран обновляемой в "реальном времени" (real-time), постоянно меняющейся информации, связанной либо с системными ресурсами, либо с процессом выполнения определенной задачи.
- Реализация "автосохранения" — Таймер может предложить программе для Windows сохранять работу пользователя на диске всегда, когда истекает заданный интервал времени.
- Завершение демонстрационных версий программ — Некоторые демонстрационные версии программ рассчитаны на свое завершение, скажем, через 30 минут после запуска. Таймер может сигнализировать таким приложениям, когда их время истекает.
- Задание темпа изменения — Графические объекты в играх или окна с результатами в обучающих программах могут нуждаться в задании установленного темпа изменения. Использование таймера устраняет неритмичность, которая могла бы возникнуть из-за разницы в скоростях работы различных микропроцессоров.
- Мультимедиа — Программы, которые проигрывают аудиодиски, звук или музыку, часто допускают воспроизведение звуковых данных в фоновом режиме. Программа может использовать таймер для периодической проверки.

Как уже упоминалось, программы под DOS, написанные для IBM PC и совместимых компьютеров, могут использовать аппаратные срабатывания таймера, перехватывая аппаратное прерывание. Когда происходит аппаратное прерывание, выполнение текущей программы приостанавливается и управление передается обработчику прерываний. Когда прерывание обработано, управление возвращается прерванной программе. Также как аппаратные прерывания клавиатуры и мыши, аппаратное прерывание таймера иногда называется асинхронным прерыванием, поскольку оно происходит случайно по отношению к

прерываемой программе. (Фактически, термин "асинхронные" не совсем точен, поскольку прерывания случаются через одинаковые промежутки времени. Но по отношению к другим процессам прерывания остаются асинхронными.) Хотя Windows тоже обрабатывает асинхронные таймерные прерывания, сообщения WM_TIMER, которые Windows посылает приложению, не являются асинхронными. Сообщения Windows ставятся в обычную очередь сообщений и обрабатываются как все остальные сообщения. Поэтому, если вы задаете функции *SetTimer* 1000 миллисекунд, то вашей программе не гарантируется получение сообщения WM_TIMER каждую секунду или даже (как уже упоминалось выше) каждые 989 миллисекунд. Если ваше приложение занято больше, чем секунду, то оно вообще не получит ни одного сообщения WM_TIMER в течение этого времени. Вы можете убедиться в этом с помощью представленных в этой главе программ. Фактически, Windows обрабатывает сообщения WM_TIMER во многом также, как сообщения WM_PAINT. Оба эти сообщения имеют низкий приоритет, и программа получит только их, если в очереди нет других сообщений.

Вызов функции *SetTimer* выглядит следующим образом:
SetTimer(hwnd, 1, iMsecInterval, NULL);
Первый параметр — это описатель того окна, чья оконная процедура будет получать сообщения WM_TIMER. Вторым параметром является идентификатор таймера, значение которого должно быть отличным от нуля. В этом примере он произвольно установлен в 1. Третий параметр — это 32-разрядное беззнаковое целое, которое задает интервал в миллисекундах. Значение 60000 задает генерацию сообщений WM_TIMER один раз в минуту.

3. Задание

Создать приложение, реализующее:

1) Циферблат часов (ЧЧ:ММ:СС), использующий таймер для обновления каждую секунду.
2) Аналоговые часы, изобразить циферблат и стрелки используя примитивы GDI. Использовать таймер для обновления каждую секунду.

4. Листинг программы

```
#include <windows.h>
#include <string.h>
#include <time.h>
#include <math.h>

#define ID_TIMER 1
#define TWOPI (2 * 3.14159)

RECT rect;

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int
iCmdShow)
{
   static char szAppName[] = "AnaClock";
```

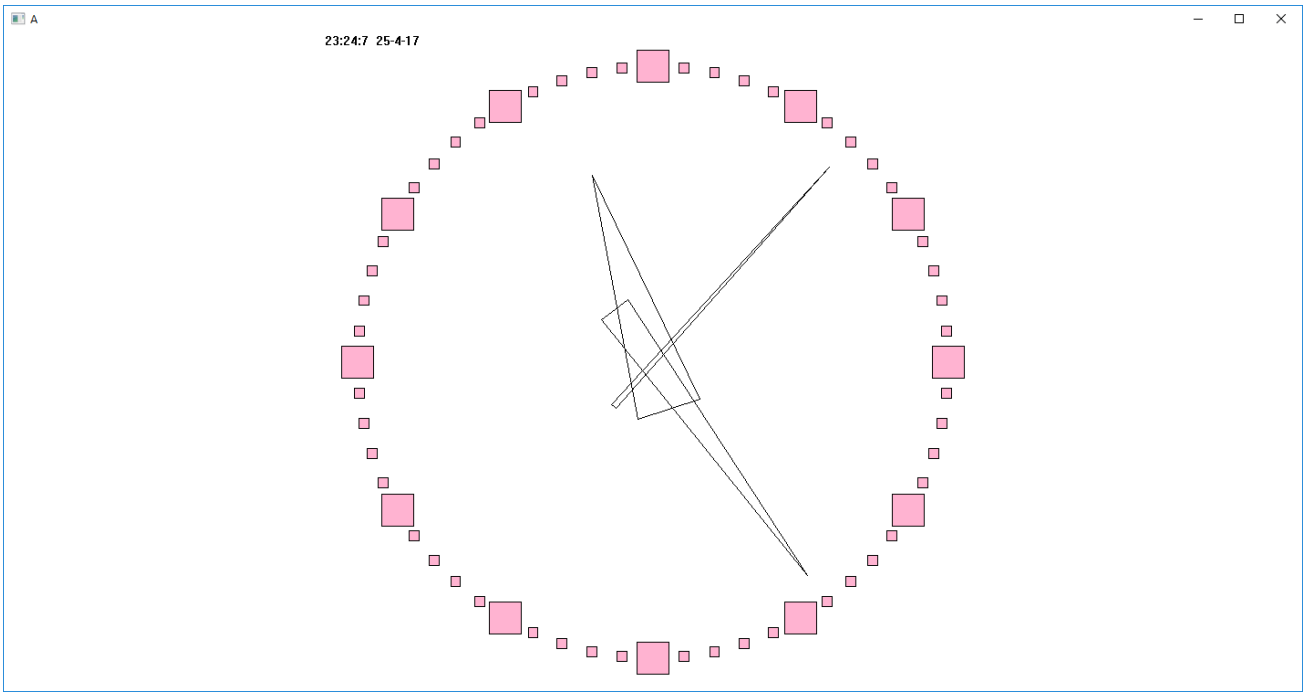
```
   HWND hwnd;
   MSG msg;
   WNDCLASSEX wndclass;
   wndclass.cbSize = sizeof(wndclass);
   wndclass.style = CS_HREDRAW | CS_VREDRAW;
   wndclass.lpfnWndProc = WndProc;
   wndclass.cbClsExtra = 0;
   wndclass.cbWndExtra = 0;
   wndclass.hInstance = hInstance;
   wndclass.hIcon = NULL;
   wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
   wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wndclass.lpszMenuName = NULL;
   wndclass.lpszClassName = (LPCWSTR)szAppName;
   wndclass.hIconSm = NULL;
   RegisterClassEx(&wndclass);
   hwnd = CreateWindow((LPCWSTR)szAppName,
      L"Analog Clock & Digital Clock",
      WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      NULL,
      NULL,
      hInstance,
      NULL);
   if (!SetTimer(hwnd, ID_TIMER, 1000, NULL))
   {
      MessageBox(hwnd, L"Too many clocks or timers!", (LPCWSTR)szAppName, MB_ICONEXCLAMATION
| MB_OK);
      return FALSE;
   }
   ShowWindow(hwnd, iCmdShow);
   UpdateWindow(hwnd);
```

```
   while (GetMessage(&msg, NULL, 0, 0))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }

   return msg.wParam;
}

void SetIsotropic(HDC hdc, int cxClient, int cyClient)
{
   SetMapMode(hdc, MM_ISOTROPIC);
   SetWindowExtEx(hdc, 1000, 1000, NULL);
   SetViewportExtEx(hdc, cxClient / 2, -cyClient / 2, NULL);
   SetViewportOrgEx(hdc, cxClient / 2, cyClient / 2, NULL);
}

void RotatePoint(POINT pt[], int iNum, int iAngle)
{
   int i;
   POINT ptTemp;

   for (i = 0; i < iNum; i++)
   {
      ptTemp.x = (int)(pt[i].x * cos(TWOPI * iAngle / 360) + pt[i].y * sin(TWOPI * iAngle /
360));
      ptTemp.y = (int)(pt[i].y * cos(TWOPI * iAngle / 360) - pt[i].x * sin(TWOPI * iAngle /
360));
      pt[i] = ptTemp;
   }
}

void DrawClock(HDC hdc)
{

   int iAngle;
   POINT pt[3];
   HBRUSH penni;
   penni = CreateSolidBrush(RGB(255, 179, 209));

   for (iAngle = 0; iAngle < 360; iAngle += 6)
   {
      pt[0].x = 0;
      pt[0].y = 900;
           RotatePoint(pt, 1, iAngle);
           pt[2].x = pt[2].y = iAngle % 5 ? 33 : 100;
           pt[0].x -= pt[2].x / 2;
           pt[0].y -= pt[2].y / 2;
           pt[1].x = pt[0].x + pt[2].x;
           pt[1].y = pt[0].y + pt[2].y;
           SelectObject(hdc, penni);
           Rectangle(hdc, pt[0].x, pt[0].y, pt[1].x, pt[1].y);
   }
}

void DrawHands(HDC hdc, struct tm *datetime, BOOL bChange)
{
   static POINT pt[3][5] = { 0, -150, 100, -150, 0, 600, -100, -150, 0, -150,
      0, -200, 50, -200, 0, 800, -50, -200, 0, -200,
      0, -180, 10, -180, 0, 800, -10, -180, 0, -180 };
   int i, iAngle[3];
   POINT ptTemp[3][5];
   iAngle[0] = (datetime->tm_hour * 30) % 360 + datetime->tm_min / 2;
   iAngle[1] = datetime->tm_min * 6;
```

```
   iAngle[2] = datetime->tm_sec * 6;
   memcpy(ptTemp, pt, sizeof(pt));

   for (i = bChange ? 0 : 2; i < 3; i++)
   {
      RotatePoint(ptTemp[i], 5, iAngle[i]);
      Polyline(hdc, ptTemp[i], 5);
   }

   HPEN hPen2;
   hPen2 = CreatePen(PS_SOLID, 10, RGB(255, 255, 255));
   SelectObject(hdc, hPen2);
   struct tm *Time;
   time_t ltime;
   time(&ltime);
   Time = localtime(&ltime);
   unsigned char szTime[40];
   SelectObject(hdc, GetStockObject(BLACK_PEN));
   int a = wsprintf((LPWSTR)szTime, "%d:%d:%d  %d-%d-%d", Time->tm_hour, Time->tm_min, Time-
>tm_sec, Time->tm_mday, Time->tm_mon + 1, Time->tm_year % 100);
   TextOut(hdc, -1000, 1000, (LPWSTR)szTime, a);
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
   HDC hDC, hCompatibleDC;
   PAINTSTRUCT PaintStruct;
   RECT Rect;
   static int nHDif = 0, nVDif = 0, nHPos = 0, nVPos = 0;

   static int cxClient, cyClient;
   static struct tm dtPrevious;
   BOOL bChange;
   HDC hdc;
   PAINTSTRUCT ps;

   time_t lTime;
   struct tm *datetime;

   time(&lTime);
   datetime = localtime(&lTime);

   switch (iMsg)
   {

   case WM_CREATE:

      time(&lTime);
      datetime = localtime(&lTime);
      dtPrevious = *datetime;
      return 0;

   case WM_SIZE:

      cxClient = LOWORD(lParam);
      cyClient = HIWORD(lParam);
      return 0;

   case WM_TIMER:

      time(&lTime);
      datetime = localtime(&lTime);
      bChange = datetime->tm_hour != dtPrevious.tm_hour ||datetime->tm_min !=
dtPrevious.tm_min;
```

```
            hdc = GetDC(hwnd);
            SetIsotropic(hdc, cxClient, cyClient);
            SelectObject(hdc, GetStockObject(WHITE_PEN));
            DrawHands(hdc, &dtPrevious, bChange);
            SelectObject(hdc, GetStockObject(BLACK_PEN));
            DrawHands(hdc, datetime, TRUE);
            ReleaseDC(hwnd, hdc);

            dtPrevious = *datetime;
            return 0;

        case WM_PAINT:

            hdc = BeginPaint(hwnd, &ps);
            SetIsotropic(hdc, cxClient, cyClient);
            DrawClock(hdc);
            DrawHands(hdc, &dtPrevious, TRUE);
            EndPaint(hwnd, &ps);
            return 0;

        case WM_DESTROY:

            KillTimer(hwnd, ID_TIMER);
            PostQuitMessage(0);
            return 0;
        }

        return DefWindowProc(hwnd, iMsg, wParam, lParam);
}
```

5. Результат работы прогрммы



Вывод:

В результате выполненной работы были изучены принципы работы с системным таймером. Таймер в Windows является устройством ввода информации, которое периодически извещает приложение о том, что истек заданный интервал времени. Была создана программа, реализующая интерфейс аналоговых и цифровых часов. Часы показывают локальное время системы. Благодаря таймеру время обновляется каждую секунду. Благодаря реализованной программе была возможность оценить удобство использования таймера.