Proyecto 22-ntp

Secciones

- Objetivo
- Programa
- platformio.ini
- Biblioteca
- Constantes

Objetivo

El objetivo primordial de este proyecto es demostrar la capacidad de acceder a un servidor de hora (*NTP Server*) en el Web para recabar datos de fecha y hora actuales.

Estos servidores se encuentran distribuídos a lo largo del mundo y, en el caso de Argentina, la dirección es *ar.pool.ntp.org*.

Programa

Para el acceso al protocolo correspodiente, debe usarse una biblioteca, tanto para *ESP8266* como para *ESP32*, que se denomina ***NTPClient***; el acceso a Internet se realiza sobre *UDP*, por lo cual deben crearse una objeto *ntpUDP*; además, y para poder acceder al servidor, se debe crear otro objeto denominado *timeClient*.

En las líneas 54 y 55 se encuentra el respectivo código:

```
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_SERVER, TZ(TIME_ZONE) );
```

En resumen, se deben definir las siguientes constantes, de acuerdo a lo que se muestra en las líneas 15 a 23

```
/*
 *  Definiciones incluidas en platformio.ini
 *
 *      MY_SSID     Nombre de la red WiFi
 *      MY_PASS     Password de la red
 *      TIME_ZONE   Zona de tiempo donde reside
 *      NTP_SERVER  Nombre del servidor de NTP
```

```
*      SERIAL_BAUD Baud rate del port serie
*/
```

Las constantes *TIME_ZONE* y *NTP_SERVER* son usadas en la creación de *timeClient*; *TIME_ZONE* indica la cantidad de husos horarios respecto del meridiano de Greenwich en que se encuentra el país en cuestión (Argentina está tres horas atrasado respecto de él, por lo cual corresponde -3).

MY_SSID y *MY_PASS* sirven para acceder a WiFi y se utilizan para inicializar la conexión al objeto *WiFi* como se muestra en la línea 62

```
    WiFi.begin(MY_SSID, MY_PASS);
```

A su vez, la inicialización del objeto *timeClient* se realiza en la línea 81:

```
    timeClient.begin();
```

Luego de la inicialización en la función *setup()*, la acción, como siempre, se desarrolla en la función *loop()*; en efecto, cada segundo se accede al servidor NTP y se recaba la fecha, mostrando posteriormente los resultados.

Para obtener los valores actuales de fecha y hora del servidor NTP, se llama a la actualización del *cliente* en la línea 90 mediante:

```
    timeClient.update();
```

Para obtener los resultados, se realiza de dos formas distintas.

La primera de ellas, es obteniendo el día de la semana, accediendo al método *getDay()* y a la hora local mediante el método *getFormattedTime()*; el primero de ellos se imprime accediendo primero al arreglo *daysOfWeek* que se encuentra en las líneas 44 a 48. El código de esta primera parte se encuentra entre las líneas 92 a 94 y se repite aquí para mayor comodidad:

```
    Serial.print( daysOfTheWeek[timeClient.getDay()] );
    Serial.print( ": ");
    Serial.println(timeClient.getFormattedTime());
```

La segunda forma, que es mucho más aconsejable, es leyendo del servidor lo que se denomina *epoch time* que no es otra cosa que la cantidad de segundos transcurridos desde las 0 horas del primero de Enero de 1970 en el meridiano de Greenwich pero corregidos de acuerdo a la zona horaria previamente declarada; este valor ocupa 4 bytes codificados como numero entero y de él, mediante cálculos, se pueden obtener todos los datos necesarios para la ubicación de la fecha y hora actuales, obviamente al segundo de precisión.

Si bien estos cálculos no son difíciles de realizarlos mediante un programa en C, hay bibliotecas *standard* de C (que provienen de Unix), que se pueden usar en Arduino y que convenientemente, solo hay que incluir *<time.h>*, como se ha realizado en el programa en la línea 39.

Sería conveniente que el lector consulte los siguientes *links* para tener la correcta especificación de esta biblioteca así como los detalles de la función de biblioteca *strftime*:

- time library
- strftime function

Se copian los códigos correspondientes a esta segunda parte, que se encuentran en las líneas 96 a 100:

```
      Serial.print( "Epoch = " );
      epoch_time = timeClient.getEpochTime();
      Serial.println( epoch_time );
      strftime(buf, sizeof(buf),
         "%A: %d %B %Y %H:%M:%S", localtime(&epoch_time));
      Serial.println(buf);
```

---

platformio.ini

Biblioteca

La sección de *platformio.ini* que comienza con *[env]* es común para los otros ambientes particulares, en este caso *[env:wemos_d1_mini32]* y *[env:d1_mini]*.

Como en este caso, la biblioteca *NTPClient* es común para ambos, se define en *[env]* agregando:

```
lib_deps =
    NTPCLient
```

Constantes

Como se habrá observado, hay cinco constantes que se definen en el archivo *platformio.ini* para mayor comodidad de mantenimiento; de esa manera, no es necesario entrar en los archivos de código para dicho mantenimiento.

Si bien el proyecto está armado para poder compilarse para dos procesadores (*ESP8266* y *ESP32*), cuatro de estas constantes son comunes a ambos, por lo cual están definidas en *build_flags* de la sección *[env]*; dichas definiciones están contenidas entre las líneas 28 y 22 y se copian más abajo

```
build_flags =
      -DTIME_ZONE=-3
      -DMY_SSID=\"EDUMEL51\"
      -DMY_PASS=\"0044415146\"
      -DNTP_SERVER=\"ar.pool.ntp.org\"
```

Obsérvese que, por definirse en tres de ellas constantes *string*, debe escaparse el caracter **"** mediante el caracter barra invertida.

La otra constante que falta es *SERIAL_BAUD*, lo cual lleva una disgresión: en este caso, lo importante es que esta constante, que se usa para inicializar por programa la interfase serie, esté en un todo de acuerdo con la definición de *hardware* de la interfase que se entrega por *monitor_speed* en este archivo.

Ello se logra como se muestra en la línea 29 para la inicialización de *hardware* mediante:

```
monitor_speed = 115200
```

Bastaría, por lo tanto, colocar bajo *build_flags*:

```
build_flags =
    -DSERIAL_BAUD=115200
```

pero eso significaría que se debería estar atento a que los números que se colocasen en las líneas 29 y 32 *fuesen exactamente iguales*.

Sería, entonces, mejor vincular automáticamente ambos lo cual se logra en la línea 32 recabando el valor de el ítem *monitor_speed* de la misma sección *env*; en definitiva, podemos aquí mostrar el *env* correspondiente a *ESP32* que se encuentra entre las líneas 26 y 32

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32
monitor_speed = 115200
build_flags =
    ${env.build_flags}
    -DSERIAL_BAUD=${env:wemos_d1_mini32.monitor_speed}
```

De esta forma, se pueden poner distintos *baud_rate* para *ESP32* y *ESP8266*, que es justamente lo que se ha hecho en este caso, ya que para este último dicho *baud_rate* es de **74880**

Claro, la pregunta que queda es ¿porqué se ha elegido tal *baud_rate* no normalizado para este caso? La respuesta es que, no sabemos porqué, cuando *nace* después de un *reset* este procesador, amanece con dicho *baud_rate*; de esta manera, con este poco común *baud_rate* se podrán observar todos los caracteres que, a partir de *reset*, se envían a la interfase serie.