Urządzenia Peryferyjne

**Ćwiczenie 14 – Bluetooth -> komunikacja z telefonem komórkowym**

Prowadzący: Dr Inż. Jan Nikodem

Grupa: Poniedziałek tydzień parzysty godz. 10:15

Data wykonania ćwiczenia: 13.11.2017r.

Wykonali : Paweł Biel 225949

Oskar Szubert 213624

1. Zagadnienia
   1. Mechanizm rejestracji funkcji callback’owych

Funkcje callback’owe inaczej wywołania zwrotne są odwrotnością normalnych funkcji, które wywołujemy podczas programowania.W tym przypadku użytkownik jedynie rejestruje funkcję do późniejszego wywołania, natomiast funkcje biblioteki wywołają ją w stosownym dla siebie czasie.  Kiedy następuje wywołanie zwrotne, funkcja, która je wykonuje, nie wie nic o tym, co się zdarzy. To zależy od tego, co zostało zarejestrowane do wywołania.  Rejestrowanie funkcji polega na stworzeniu własnej klasy na bazie innej klasy oraz przedefiniowaniu w niej odpowiedniej metody tak, aby wykonywała ona inne czynności, niż to jest założone w jej klasie bazowej. Wywołanie zwrotne odbywa się wtedy w ten sposób, że inna funkcja woła metodę na podanym obiekcie, ale faktyczna procedura, która zostanie wywołana, może być różna, w zależności od tego, jakiej klasy jest podany obiekt.

* 1. Komunikacja poprzez Bluetooth

Standard komunikacji bezprzewodowej opisany normami IEEE 802.15.1. Określane są trzy klasy mocy nadawczej:

* Klasa 1 – 100mW do 100m
* Klasa 2 – 2,5 MW do 10m
* Klasa 3 – 1mW do 1m

Standard korzysta z [fal radiowych](https://pl.wikipedia.org/wiki/Fale_radiowe) w paśmie [ISM](https://pl.wikipedia.org/wiki/Pasmo_ISM) 2,4 [GHz](https://pl.wikipedia.org/wiki/Herc). Na terenie Polski akceptowalne przedział częstotliwości wynosi 2,4 - 24,25 GHz .Bluetooth jest standardem cały czas rozwijanym:

* Bluetooth 1.0 – 21 [kb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99)
* Bluetooth 1.1 – 124 [kb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99)
* Bluetooth 1.2 – 328 [kb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99)
* Bluetooth 2.0 + EDR – wprowadzenie Enhanced Data Rate zwiększyło transfer teoretyczny do 2,1 [Mb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99) (około 3 [Mb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99) wliczając narzut protokołu)
* Bluetooth 2.1 + EDR - uproszczenie i ujednolicenie procesu parowania urządzeń BT, wsparcie dla przyszłych implementacji [NFC](https://pl.wikipedia.org/wiki/Near_Field_Communication), zmniejszenie zużycia energii[[4]](https://pl.wikipedia.org/wiki/Bluetooth#cite_note-4)
* Bluetooth 3.0 + HS (High Speed) – 24 [Mb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99) (3 MB/s)
* Bluetooth 3.1 + HS (High Speed) – 40 [Mb/s](https://pl.wikipedia.org/wiki/Bit_na_sekund%C4%99) (5 MB/s)
* Bluetooth 4.0 + LE (Low Energy) – 1 Mb/s znacząco ograniczono pobór energii, kosztem obniżonego transferu oraz zwiększono realny zasięg działania do 100 m
* Bluetooth 4.1 - standard opracowany do zastosowania  
  w tzw. "intrenecie rzeczy", umożliwiający bezpośrednią łączność przedmiotów z internetem
* Bluetooth 4.2 - w stosunku do poprzednich wersji: szybszy transfer, wyższy poziom bezpieczeństwa, nawiązanie  
  łączności z przedmiotami - łatwiejsze
* Bluetooth 5.0 - ujednolicenie wersji, szybszy transfer – 2 Mb/s   
  dla przedmiotów łączących się z „internetem rzeczy” i 50 Mb/s   
  do normalnych, realny zasięg działania do 140m

Każde urządzenie ma 48 bitowy adres IEEE MAC (Bluetooth Device Address, BD\_ADDR) i jest on używany do inicjowania pewnych operacji oraz obliczania kodu dostępu.

Moduł Bluetooth wyposażony jest w 28-bitowy wewnętrzny zegar, który determinuje synchronizację i skakanie po częstotliwościach. Nigdy nie jest on dostrajany, ani wyłączany. Do synchronizacji z innym modułem Bluetooth wykorzystywana jest różnica (offset) pomiędzy zegarami jednostek chcących  
się komunikować. Częstotliwość zegara wynosi 3,2 kHz.

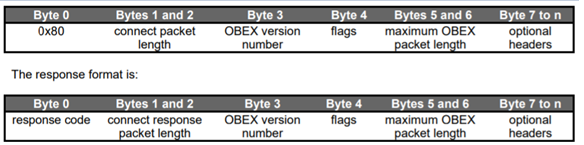
* 1. Protokół transferu plików OBEX i przygotowane polecenia

**OBEX - Object Exchange** - protokół komunikacyjny, określający procedury wymiany danych binarnych między urządzeniami.

Specyfikacja protokołu OBEX opiera się architekturze klient - serwer. Klient wykorzystuje zaufane medium transportowe do połączenia z serwerem w celu zażądania transmisji obiektów. Przesyłane obiekty są zapisane w formacie binarnym.

* Polecenie CONNECT

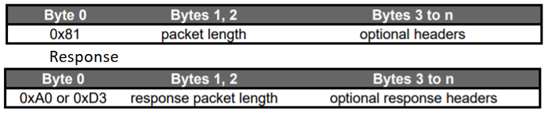
Ta operacja inicjuje połączenie i określa podstawowe oczekiwania każdej strony łącza.



Żądanie i odpowiedź CONNECT muszą się mieścić w jednym pakiecie.

* Polecenie DISCONNECT

Ten kod operacyjny sygnalizuje koniec sesji OBEX. Może zawierać dodatkowy nagłówek Description- czyli informacje czytelne dla użytkownika. Żądanie i odpowiedź DISCONNECT muszą się mieścić w

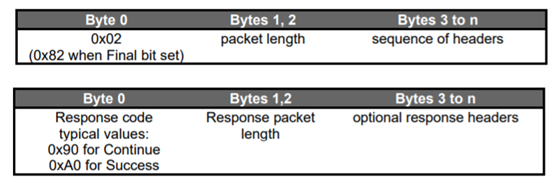


-0xA0 Operacja udana

- 0xD3 Service Unavailable

* PUT

Operacja ta wysyła obiekt z klienta do serwera. Request standardowo powinien zawierać nagłówki: nazwa i długość. Może występować także nagłówek Data/Time. Jednakże istnieje możliwość pominięcia wszystkich headerów- jeżeli urządzenie docelowe jest bardzo proste i może odpierać/obsługiwać tylko jeden typ obiektu a komunikacja z niezaufanych źródeł jest niemożliwa.



1. Kod programu
2. using System;
3. usingSystem.Collections.Generic;
4. usingSystem.ComponentModel;
5. usingSystem.Data;
6. usingSystem.Drawing;
7. usingSystem.Linq;
8. usingSystem.Text;
9. usingSystem.Threading.Tasks;
10. usingSystem.Windows.Forms;
11. using InTheHand.Net;
12. usingInTheHand.Net.Bluetooth;
13. usingInTheHand.Net.Sockets;
14. usingSystem.Threading;
15. namespace Bluetooth
16. {
17. publicpartialclassForm1 : Form
18. {
19. publicconststring DEVICE\_PIN = "000000";
20. privateboolcontinueScanning = false;
21. privateboolisPaired = false;
22. privateboolisUnpaired = false;
23. private List<BluetoothDeviceInfo> devices = new List<BluetoothDeviceInfo>();
24. BluetoothDeviceInfodeviceToPair = null;
25. public Form1()
26. {
27. InitializeComponent();
28. textBoxConsole.ReadOnly = true;
29. }
30. // Funkcja wyszukująca urządzenia oraz ich typ i adres
31. privatevoidScanDevices()
32. {
33. try
34. {
35. while(continueScanning)
36. {
37. varbluetoothClient = newBluetoothClient();
38. varbluetoothDevices = bluetoothClient.DiscoverDevices();
39. foreach(varbluetoothDeviceinbluetoothDevices)
40. {
41. varbluetoothInfo = string.Format("Nazwa urządzenia: {0}\r\nTyp urządzenia: {1} Adres: {2}",
42. bluetoothDevice.DeviceName, bluetoothDevice.ClassOfDevice.Device,bluetoothDevice.DeviceAddress);
43. textBoxConsole.Text += bluetoothInfo + "\r\n";
44. boolonList = false;
45. if(bluetoothDevice.DeviceName!=null)
46. {
47. foreach (stringiteminlistBoxDevices.Items)
48. if (item == bluetoothDevice.DeviceName)
49. onList = true;
50. }
51. if(!onList)
52. {
53. listBoxDevices.Items.Add(bluetoothDevice.DeviceName);
54. devices.Add(bluetoothDevice);
55. }
56. }
57. }
58. }
59. catch(Exception error)
60. {
61. textBoxConsole.Text += "Wystąpił błąd!\r\n" + error.Message + "\r\n";
62. }
63. }
64. // funkcja która wywołuje szukanie urządzeń po kliknięciu przycisku
65. privatevoidbuttonFindDevices\_Click(objectsender, EventArgs e)
66. {
67. ThreadfindDevices = newThread(ScanDevices);
68. findDevices.Priority = ThreadPriority.Highest;
69. findDevices.IsBackground = true;
70. findDevices.Start();
71. if(buttonFindDevices.Text == "Szukaj urządzeń")
72. {
73. buttonFindDevices.Text = "Zatrzymaj wyszukiwanie";
74. continueScanning = true;
75. textBoxConsole.Text += ("Szukam urządzeń.\r\n");
76. }
77. else
78. {
79. buttonFindDevices.Text = "Szukaj urządzeń";
80. continueScanning = false;
81. textBoxConsole.Text += "Wyszukiwanie zakończone.\r\n";
82. }
83. }
84. // Funkcja pozwalająca na odbieranie plików
85. privatevoidReceiveFiles()
86. {
87. while (isPaired)
88. {
89. varlistener = newObexListener(ObexTransport.Bluetooth);
90. listener.Start();
91. ObexListenerContextctx = listener.GetContext();
92. ObexListenerRequestreq = ctx.Request;
93. String[] pathSplits = req.RawUrl.Split('/');
94. String file = pathSplits[pathSplits.Length - 1];
95. req.WriteFile(file);
96. textBoxConsole.Text += "Odebrano plik.\r\n";
97. listener.Stop();
98. }
99. }
100. //Funkcja parująca urządzenia
101. privatevoidPairWithDevice()
102. {
103. foreach(vardevicein devices)
104. {
105. if (device.DeviceName == (string)listBoxDevices.SelectedItem)
106. deviceToPair = device;
107. }
108. if(deviceToPair==null)
109. {
110. textBoxConsole.Text += "Nie zaznaczyłeś z którym urządzeniem chcesz się połączyć!\r\n";
111. }
112. else
113. {
114. textBoxConsole.Text += "Łączenie z: " + (string)deviceToPair.DeviceName + "\r\n";
115. deviceToPair.Update();
116. deviceToPair.Refresh();
117. deviceToPair.SetServiceState(BluetoothService.ObexObjectPush, true);
118. isPaired = BluetoothSecurity.PairRequest(deviceToPair.DeviceAddress, DEVICE\_PIN);
119. if(isPaired)
120. {
121. textBoxConsole.Text += "Sparowano urządzenie.\r\n";
122. listBoxConnected.Items.Add(deviceToPair.DeviceName);
123. buttonPairWithDevice.Enabled = false;
124. buttonUnpair.Enabled = true;
125. ThreadreceiveFiles = newThread(ReceiveFiles);
126. receiveFiles.Priority = ThreadPriority.Highest;
127. receiveFiles.IsBackground = true;
128. receiveFiles.Start();
129. }
130. else
131. {
132. textBoxConsole.Text += "Nie sparowano.\r\n";
133. }
134. }
135. }
136. //Funkcja rozparowująca urządzenia
137. privatevoidUnpair()
138. {
139. if (deviceToPair.DeviceName == listBoxConnected.SelectedItem.ToString())
140. isUnpaired = BluetoothSecurity.RemoveDevice(deviceToPair.DeviceAddress);
141. if(isUnpaired)
142. {
143. textBoxConsole.Text += "Rozłączono.\r\n";
144. listBoxConnected.Items.Remove(deviceToPair.DeviceName);
145. isPaired = false;
146. }
147. else
148. {
149. textBoxConsole.Text += "Nie udało się rozłączyć urządzenia.\r\n";
150. }
151. }
152. privatevoidbuttonPairWithDevice\_Click(objectsender, EventArgs e)
153. {
154. ThreadpairWithDevice = newThread(PairWithDevice);
155. pairWithDevice.Priority = ThreadPriority.Highest;
156. pairWithDevice.IsBackground = true;
157. pairWithDevice.Start();
158. }
159. privatevoidbuttonUnpair\_Click(objectsender, EventArgs e)
160. {
161. Unpair();
162. buttonPairWithDevice.Enabled = true;
163. buttonUnpair.Enabled = false;
164. }
165. privatevoidbuttonAddFile\_Click(objectsender, EventArgs e)
166. {
167. openFile.ShowDialog();
168. }
169. //Funkcja pozwalająca na wybranie pliku który chcemy wysłać
170. privatevoidopenFile\_FileOk(objectsender, CancelEventArgs e)
171. {
172. boolexist = false;
173. foreach (varfileNameinopenFile.FileNames)
174. {
175. if (fileName != null)
176. {
177. foreach (stringiteminlistBoxFiles.Items)
178. if (item == fileName)
179. exist = true;
180. }
181. if (!exist)
182. {
183. listBoxFiles.Items.Add(fileName);
184. }
185. else
186. {
187. MessageBox.Show("");
188. }
189. }
190. }
191. // Funkcja wysyłająca plik do sparowanego urządzenia
192. privatevoidSendFiles()
193. {
194. foreach(string file inlistBoxFiles.Items)
195. {
196. try
197. {
198. if(deviceToPair!=null)
199. textBoxConsole.Text += ("Wysyłanie pliku\r\n");
200. varuri = newUri("obex://" + deviceToPair.DeviceAddress + "/" + file);
201. varrequest = newObexWebRequest(uri);
202. request.ReadFile(file);
203. varresponse = (ObexWebResponse)request.GetResponse();
204. textBoxConsole.Text += ("Pomyślnie wysłano plik:" + file + "\r\n");
205. }
206. catch(Exception error)
207. {
208. textBoxConsole.Text += "Wystapił błąd! " + error.Message + "\r\n";
209. }
210. }
211. listBoxFiles.Items.Clear();
212. }
213. privatevoidbuttonSendFile\_Click(objectsender, EventArgs e)
214. {
215. ThreadsendFiles = newThread(SendFiles);
216. sendFiles.Priority = ThreadPriority.Highest;
217. sendFiles.IsBackground = true;
218. sendFiles.Start();
219. }
220. privatevoidtextBoxConsole\_TextChanged(objectsender, EventArgs e)
221. {
222. textBoxConsole.SelectionStart = textBoxConsole.Text.Length;
223. textBoxConsole.ScrollToCaret();
224. }
225. Uwagi

Nasz program niestety nie ma możliwości wyboru z jakiego adaptera chcemy skorzystać.