Bramka SMS na bazie Raspberry Pi z dołączonym modułem GSM

SMS-y jeszcze do niedawna były szalenie popularne. Są to krótkie wiadomości tekstowe przesyłane z telefonu na telefon poprzez sieć komórkową. Obecnie są wypierane przez łączność wykorzystującą Internet (komunikatory internetowe, chaty i inne), jednakże nadal pełnią istotną rolę w systemie komórkowym. Możliwość wysyłania tego rodzaju wiadomości z poziomu komputera, bez wykorzystania telefonu komórkowego, jest niezwykle istotna, szczególnie gdy osoba, z którą chcemy się skontaktować nie korzysta z innych systemów komunikacji tekstowej, które wykorzystują połączenie z Internetem.

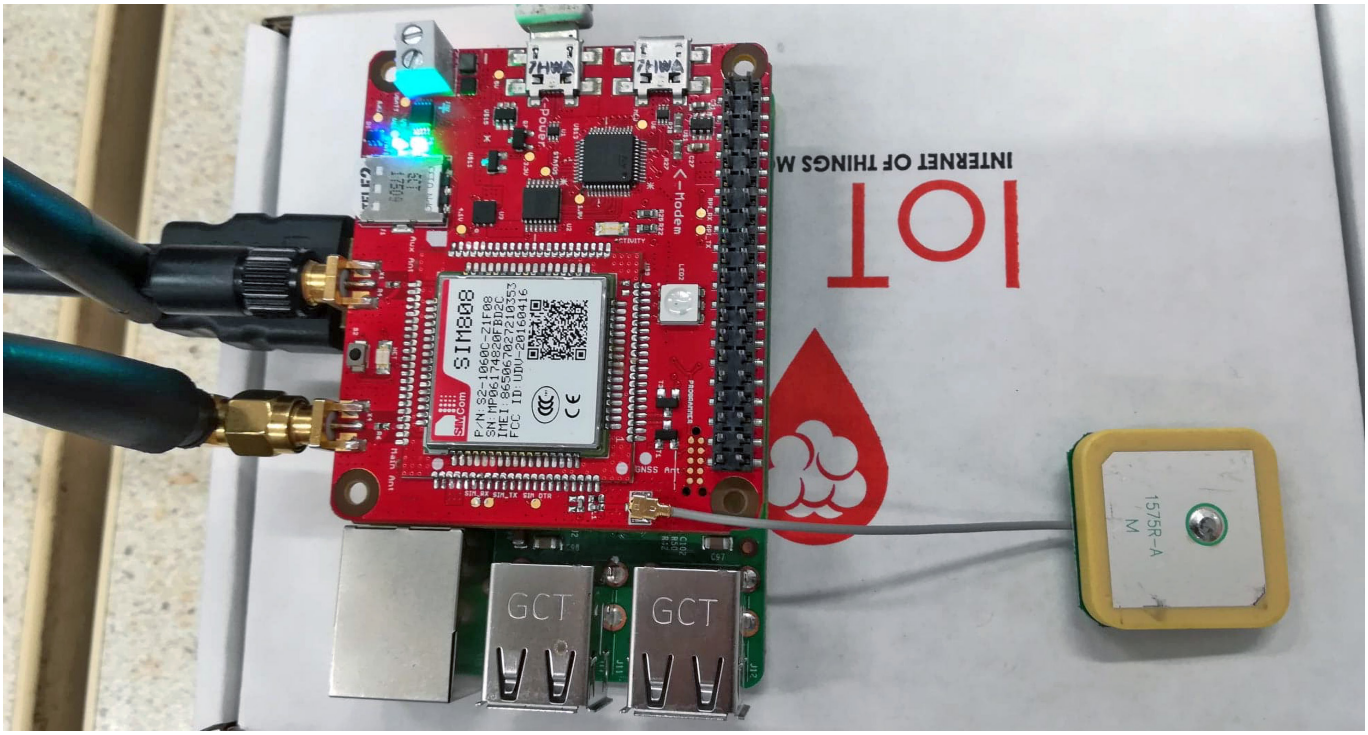
W poniższym artykule opowiemy, jak można wykorzystać komputer jednopłytkowy z serii Raspberry Pi z dołączonym modemem GSM do wysyłania krótkich wiadomości tekstowych – popularnych SMS-ów. Do tego popularnego minikomputera można dołączyć, na wiele sposobów, modem GSM, pozwalający na komunikowanie się z siecią telefonii komórkowej. W artykule opisany jest sposób podłączania popularnych modemów na USB do minikomputera Raspberry Pi oraz przedstawiona jest biblioteka, dedykowana do wysyłania i odbierania SMS-ów za pomocą tego minikomputera z modemem GSM.

Tego rodzaju system może mieć wiele zastosowań – samodzielnie, bądź w połączeniu z innymi urządzeniami, czy systemami, takimi jak na przykład centralka alarmowa czy system automatyki domowej. W podsumowaniu artykułu przedstawimy kilka potencjalnych zastosowań takiego systemu.

Modemy GSM

Aby dodać funkcjonalność GSM do komputera jednopłytkowego, takiego jak Raspberry Pi, musimy go wyposażyć w modem GSM. Na rynku dostępnych jest wiele urządzeń tego rodzaju – przyjrzyjmy się im, aby wiedzieć jaki mamy wybór, projektując naszą bramkę SMS. Modemy GSM podłączane są do minikomputerów na kilka sposobów. Typowo, poprzez interfejs szeregowy (UART), który jest wyprowadzony na 40 pinowym złączu GPIO, ale także modem może być podłączony poprzez mostek UART-USB lub jako 'dongle' – zintegrowany modem w postaci urządzenia podpinanego bezpośrednio do portu USB komputera.

Jeśli chodzi o modemy GSM podłączane do sprzętowego interfejsu UART na rynku dostępnych jest wiele zintegrowanych modułów – w postaci modułów HAT oraz samodzielnych elementów, do zabudowania w swoich projektach. Moduły typu HAT, czyli dołączane



Fotografia 1. Moduł typu HAT z modemem GSM oraz odbiornikiem GPS (czerwone PCB) na Raspberry Pi (zielone PCB).

do złącza GPIO minikomputera Raspberry Pi, są najprostsze do użycia w projektach hobbystycznych – wystarczy zamontować moduł na płytkę komputera. Przykład takiego rozwiązania pokazano na **fotografii 1.** Wszystkie elementy niezbędne dla działania modemu GSM znajdują się w takim module. Także antena lub gniazdo do podłączenia zewnętrznej anteny. Jeśli nie chcemy korzystać z modułu tego rodzaju, możemy zbudować własny w oparciu o zintegrowany modem GSM o odpowiadających nam parametrach (2G, 3G, 4G) i z potrzebnymi nam dodatkowymi funkcjami. Modem wymaga zainstalowania karty SIM lub tzw. *embedded SIM* (eSIM), czyli karty w postaci układu scalonego, lutowanego na płytce.

Osobną klasę sprzętów stanowią popularne modemy GSM na USB. Ich podłączanie do komputera z systemem operacyjnym Windows jest bardzo proste, niestety w przypadku Linuksa, który standardowo instalowany jest na komputerach jednopłytkowych, nie jest to już takie trywialne. Istnieje wiele przewodników dostępnych w sieci, wykorzystujących np. skrypt *sakis3g*, ale uruchomienie niektórych modemów wymaga większego nakładu pracy. Poniżej przedstawimy uniwersalny sposób na podłączenie tego rodzaju modemu do Raspberry Pi pracującego z systemem operacyjnym Raspbian.

Główny problem z podłączaniem tego rodzaju dongla do Raspberry Pi polega na tym, że większość modemów USB zgłasza się jako dwa urządzenia – urządzenie pamięci masowej USB oraz modem USB. Po podłączeniu do Raspberry PI urządzenie zwykle znajduje się w trybie pamięci USB. Istnieje specjalny program o nazwie *usb_modeswitch*, który może zostać wykorzystany do przełączania trybów zgłaszania się dongla USB. Druga brakująca część to sposób na połączenie się z siecią komórkową za pomocą modemu USB z poziomu minikomputera Raspberry Pi. W tym celu wykorzystane zostanie klasyczne oprogramowanie *ppp* i *wvdial*.

W pierwszej kolejności musimy zainstalować program do przełączania trybów urządzenia USB. W tym celu korzystamy z apt-get; w terminalu systemu wpisujemy:

```
sudo apt-get install usb-modeswitch
```

Po zainstalowaniu oprogramowania musimy ustalić numery VID oraz PID (odpowiednio Vendor ID i Product ID – ID producenta oraz ID produktu) naszego modemu GSM, aby móc odpowiednio skonfigurować *usb-modeswitch*. W tym celu podłączamy modem GSM do komputera i resetujemy Raspberry Pi. Najlepiej robić to bez podłączonego Internetu do minikomputera. Po uruchomieniu systemu operacyjnego w terminalu wpisujemy:

```
lsusb
```

Uruchomimy wbudowane w Linuksa narzędzie do wyświetlania informacji o systemowych szynach USB i podłączonych do nich urządzeniach. Komenda zwróci nam listę urządzeń USB podłączonych do komputera, tak jak pokazano na **rysunku 2**. System wykrywa poprawnie modem USB, który zgłasza się swoimi numerami VID oraz PID, rozdzielonymi dwukropkiem (podkreślone pomarańczową linią). Możemy teraz zresetować komputer, aby zainstalowany wcześniej usb_modeswitch zmienił tryb działania dołączonego urządzenia oraz ponownie wyświetlić podłączone urządzenia USB. Numer PID urządzenia zmienił się. Zapamiętane numery VID i PID urządzenia w obu trybach posłużą nam do konfiguracji usb_modeswitch, tak, aby urządzenie uruchamiało się za każdym razem jako modem GSM. W tym celu musimy wypakować, na przykład do folderu /tmp, odpowiedni plik konfiguracyjny. W tym celu w terminalu wpisujemy:

```
cd /tmp
tar -xzvf /usr/share/usb_modeswitch/configPack.tar.
gz 19d2\:2000
```

```
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 003: ID 0bda:0136 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 006: ID 19d2:2000 ZTE WCDMA Technologies MSM MF627/MF628/MF628+/MF636+ HSDPA/HSUPA
Bus 001 Device 004: ID 046d:c50c Logitech, Inc. Cordless Desktop S510
```

Rysunek 2. Przykładowa lista, drukowana na ekranie przez polecenie lsusb. Kolorem pomarańczowym podkreślono VID:PID modemu GSM w trybie pamięci USB

```
# ZTE devices

TargetVendor=   0x19d2
TargetProductList="0001,0002,0015,0016,0017,0031,0037,0052,0055,0063,0064,0066,0091,0108,0117,0128,0157,2002,2003"

MessageContent="5553424312345678000000000000061e000000000000000000000000000000"
MessageContent2="5553424312345679000000000000061b000000020000000000000000000000"
MessageContent3="555342431234567020000000800000c85010101180101010101000000000000"

NeedResponse=1
```

Rysunek 3. Przykładowa zawartość pliku konfiguracyjnego usb_modeswitch dla VID:PID 19d2:2000

gdzie VID i PID (19d3:2000 w powyższym przykładzie) zastępujemy numerami identyfikacyjnymi, jakie odpowiadają naszemu urządzeniu. Możemy teraz otworzyć wypakowany plik konfiguracyjny z pomocą naszego preferowanego edytora tekstowego (większość użytkowników Raspberry Pi używa vi albo nano). Na **rysunku 3** widzimy przykładową zawartość takiego pliki dla podanych powyżej VID i PID. Zapamiętajmy lub zapiszmy dane zaznaczone na niebiesko w na tym rysunku, będą nam potrzebne do konfiguracji programu.

Po zapoznaniu się z danymi dla naszego urządzenia możemy otworzyć plik konfiguracyjny */etc/usb_modeswitch.conf* i dodać do niego informacje, jakie uzyskaliśmy powyżej. Po edycji plik powinien wyglądać następująco (dla powyższych przykładowych danych):

```
DefaultVendor=0x19d2
DefaultProduct=0x2000

TargetVendor=0x19d2
TargetProduct=0x2002

MessageContent="5553424312345678000000000000061e000000000000000000000000000000"
MessageContent2="5553424312345679000000000000061b000000020000000000000000000000"
MessageContent3="555342431234567020000000800000c85010101180101010101000000000000"
```

W ten sposób określamy, jakie ma być docelowe VID i PID podłączonego urządzenia – ma to być modem GSM.

Po skonfigurowaniu urządzenia tak, aby uruchamiało się każdorazowo jako modem GSM, musimy skonfigurować łączenie się z siecią GSM. W tym celu instalujemy wymienione wcześniej w tekście programy ppp oraz wvdial. Do instalacji używamy klasycznie apt-get. W terminalu wpisujemy:

```
sudo apt-get install ppp wvdial
```

Po zainstalowaniu potrzebnego oprogramowania możemy przystąpić do konfiguracji *wvdial*. W tym celu otwieramy do edycji plik konfiguracyjny */etc/wvdial.conf*. W pliku tym wpisujemy następujący blok konfiguracji:

```
[Dialer 3gconnect]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","APN"
Stupid Mode = 1
Modem Type = Analog Modem
ISDN = 0
Phone = *99#
Modem = /dev/gsmmodem
Username = { }
Password = { }
Baud = 460800
```

W ten sposób skonfigurowaliśmy tzw. dialer, czyli moduł łączący nas z Internetem. W powyższej konfiguracji zamiast zaznaczonego na żółto *APN* podać musimy nazwę sieci pakietowej, z jakiej korzystamy do łączenia się z Internetem. Numer telefonu, zaznaczony na żółto – *99# – jest typowym numerem dostępowym w internecie

Listing 1. Biblioteka do połączenia z modemem GSM

```
import serial, time, logging
from curses import ascii

# logowanie działania aplikacji
logger = logging.getLogger(__name__)

def openModem(modem, time):
        """Otwarcie połączenia z modemem"""
        logger.info("Openning " + modem)
        serialPort = serial.Serial(modem, 460800, timeout=5)
        if serialPort.isOpen():
                logger.info("Modem opened.")
                return serialPort
        else:
                logger.error("Could not open modem")
                return -1

def closeModem(modem):
        """Zamknięcie połączenia z modemem"""
        if modem.isOpen():
                modem.close()
                logger.info("Modem closed.")
        else:
                logger.error("Modem is not opened.")

def checkModem(modem):
        """Sprawdzenie połączenia z modemem"""
        modem.write("AT\r")
        modem.readline()
        status = modem.readline()
        if status.startswith("OK"):
                logger.info("AT OK")
                return True
        else:
                logger.error("AT ERROR")
                return False

def setModemTextMode(modem):
        """Ustawienie modemu w trybie tekstowym
        (do wysyłania wiadomości SMS"""
        modem.write("AT+CMGF=1\r")
        modem.readline()
        status = modem.readline()
        if status.startswith("OK"):
                logger.info("Modem set in text mode")
                return True
        else:
                return False

def sendSMS(modem, phoneNumber, text):
        """Wysłanie wiadomości SMS"""
        modem.write('AT+CMGS="%s"\r' %phoneNumber)
        modem.write(text)
        modem.write(ascii.ctrl('z'))
        time.sleep(2)

        modem.readline()
        modem.readline()
        modem.readline()
        cmgi = modem.readline()
        modem.readline()
        modem.readline()
        status = modem.readline()
        if status.startswith("OK"):
                logger.info("SMS sent.")
                return True
        else:
                logger.error("Error sending SMS")
                return False

def deleteSMS(modem, messageIndex):
        """Skasowanie wiadomości SMS z pamięci"""
        modem.write("AT+CMGD=%s\r" %messageIndex)
        modem.readline()
        status = modem.readline()

        if status.startswith("OK"):
                logger.info("SMS deleted")
                return True
        else:
                return False

def readSMS(modem, messageIndex):
        """Odczytanie wiadomości SMS"""
        modem.write("AT+CMGR=%s\r" %messageIndex)
        time.sleep(2)

        modem.readline()
        header = modem.readline()
        logger.info(header)
        body = modem.readline()
        modem.readline()
        status = modem.readline()
```

mobilnym, ale istnieje możliwość jego edycji, gdyby nasz dostawca usług wykorzystywał inny. Jeśli nasze połączenie wymaga podania nazwy użytkownika oraz hasła (odpowiednio *Username* i *Password*), to wpisujemy je w nawiasy klamrowe w pliku konfiguracyjnym.

Po uzupełnieniu wszystkich danych możemy przystąpić do łączenia się z siecią. Aby to zrobić musimy w pierwszej kolejności skonfigurować dongle do pracy jako modem USB, a następnie wykorzystać skonfigurowany przed chwilą dialer do połączenia się z siecią. W tym celu w terminalu wpisujemy:

```
sudo usb_modeswitch -c /etc/usb_
modeswitch.conf
wvdial 3gconnect
```

Listing 1. cd

```
                logger.info("Status is: " + status.strip())
                if status.startswith("OK"):
                        logger.info("SMS read.")
                        headerParts = header.split(",")
                        if len(headerParts) > 1:
                                return "-".join([headerParts[1].strip('"'), body])
                        else:
                                return body
                else:
                        logger.error("Error reading SMS")

def readLineFromModem(modem):
        """Odczytanie jednej linii z interfejsu UART"""
        return modem.readline()

def flushBuffer(modem):
        """Wyczyszczenie buforu interfejsu UART"""
        modem.readlines()

def getMessageIndex(newMessageString):
        """Pobierz ilość SMSów (aktualny indeks w liście)"""
        newSMSIndicationParts = newMessageString.strip().split(":")
        logger.debug(newSMSIndicationParts[1])
        messageIndex = newSMSIndicationParts[1].split(",")[1]
        logger.debug(messageIndex)
        return messageIndex
```

Komendy AT

Do komunikacji z większością modemów wykorzystuje się tak zwane komendy AT, zwane czasami komendami Hayesa. Zarys tego języka poleceń opracowany został na początku lat '80 XX wieku przez Dennisa Hayesa, dla opracowanej przez jego firmę Hayes Microcomputer Products, rodziny modemów do przesyłu danych przez sieć telefoniczną, Smartmodem. Pierwsze urządzenie, które korzystało z tego zestawu komend, był Smartmodem 300 (pracujący z prędkością 300 bodów, czyli około 300 bajtów na sekundę) wypuszczony na rynek w 1981 roku. Od tego czasu komendy te ewoluowały i znalazły swoje zastosowanie w szerokiej gamie modemów, także GSM (zestaw komend AT dla modemów GSM opisany jest w specyfikacji 3GPP TS 27.007).

Nazwa komend AT pochodzi od słowa *attention*, czyli „uwaga". Te dwa znaki rozpoczynają każdą komendę wysyłaną do modemu. Komendy dla modemów GSM opisane są w specyfikacji 3GPP TS 27.007, a dodatkowo w 3GPP TS 27.005 znajduje się opis komend wykorzystywanych do wysyłania SMSów. Od typowych komend AT, używanych w przeszłości, odróżnia je wykorzystanie znaku „+" po AT w większości komend. Wykorzystanie tego znaku wprowadzone zostało do komend AT w połowie lat '90 XX wieku, wraz ze standardem komend dla modemów V.250.

Przykładowo, komenda AT+CPIN=1234 wprowadzi do karty SIM numer PIN – 1234, z kolei np. AT+CSQ pozwala na pomiar siły sygnału – modem zwraca siłę odbieranego sygnału (RSSI) oraz odsetek błędów bitowych (BER) w danym momencie. Dokładne informacje na temat zaimplementowanych w konkretnym modemie komend, znaleźć można w jego dokumentacje. Jakkolwiek specyfikacja 3GPP mówi o tym, jakie zestawy komend powinny być zaimplementowane, to producenci różnie je implementują, lub nie implementują ich wcale.

Biblioteka do połączenia z modemem GSM

Biblioteka, która zaprezentowana jest na **listingu 1** jest dedykowana do stosowania w skryptach języka Python. Korzysta z komend AT, aby kontrolować pracę modemu – wysyłać i odbierać wiadomości tekstowe. W bibliotece zdefiniowano szereg funkcji, które używane są do komunikacji z modemem GSM w celu wysłania czy odebrania wiadomości tekstowej. Komentarze w listingu 1 powinny dostatecznie tłumaczyć zastosowanie poszczególnych funkcji. Jako parametr *modem* podajemy adres sprzętowy portu szeregowego do którego dołączony jest nasz modem. W przypadku dongla na USB będzie to, na przykład, */dev/ttyUSB1*.

Stosując opisywaną bibliotekę do wysyłania wiadomości SMS w pierwszej kolejności należy połączyć się z modemem, wykorzystując funkcję *openModem*. Następnie trzeba przełączyć modem do trybu tekstowego korzystając z funkcji *setModemTextMode* i już można wysyłać SMS-y z pomocą funkcji *sendSMS*. Po zakończonej pracy z modemem nie zapominajmy zamknąć połączenia funkcją *closeModem*.

Przykłady wykorzystania bramki GSM

Podstawowym zastosowaniem opisanego wyżej systemu jest oczywiście wysyłanie i odbieranie SMS-ów. Taka funkcjonalność przydatna jest w szerokiej klasie programów. Bibliotekę można zastosować do stworzenia bramki SMS – urządzenia, które pozwoli nam wysyłać z poziomu komputera wiadomości tekstowe. Jeżeli uzupełnimy program napisany w Pythonie o interfejs webowy, na przykład napisany z wykorzystaniem frameworku *Flask*, możemy stworzyć prostą bramkę SMS, dostępną jako strona w naszej wewnętrznej domowej sieci.

Biblioteka może posłużyć też do rozbudowy innych aplikacji o możliwość komunikacji za pomocą wiadomości SMS. W szczególności w systemach automatyki domowej może to być wartościowe – pozwala z jednej strony np. sterować poszczególnymi urządzeniami w domu z pomocą wiadomości SMS lub wysyłać SMS-owe alerty

Z systemu SMS skorzystać można też w samochodzie. Raspberry Pi w naszym pojeździe może np. komunikować się z interfejsem diagnostycznym OBD-II, aby zbierać istotne parametry pracy silnika w naszym pojeździe, a następnie wysyłać nam SMS-y, jeżeli któreś z nich wychodzą poza normę, lub też auto zgłasza jakieś usterki z pomocą kodów błędów. Jeśli do takiego urządzenia dodamy np. moduł nawigacji satelitarnej (GPS), który często jest instalowany na wspólnych modułach HAT z modemami GSM, z łatwością stworzyć możemy system śledzenia pozycji naszego auta, przydatny w razie kradzieży pojazdu... lub gdy zapomnimy, gdzie go zaparkowaliśmy.

Nikodem Czechowski