Linux z node.js dla Raspberry Pi

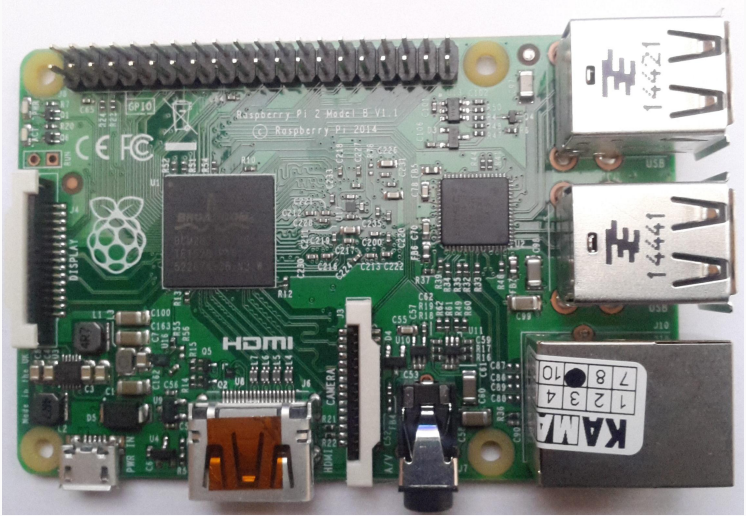
Adam Stolcenburg

10 stycznia 2018

Akademia ADB

Raspberry Pi

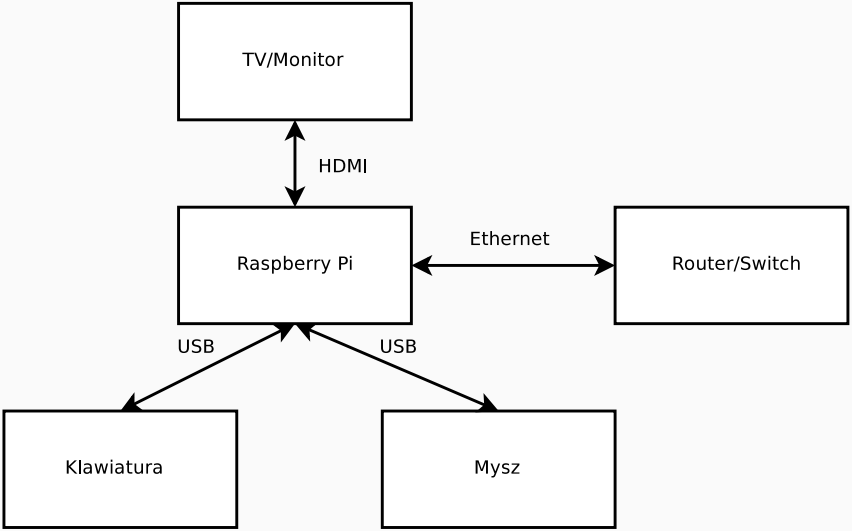
Raspberry Pi 2 Model B V1.1 i



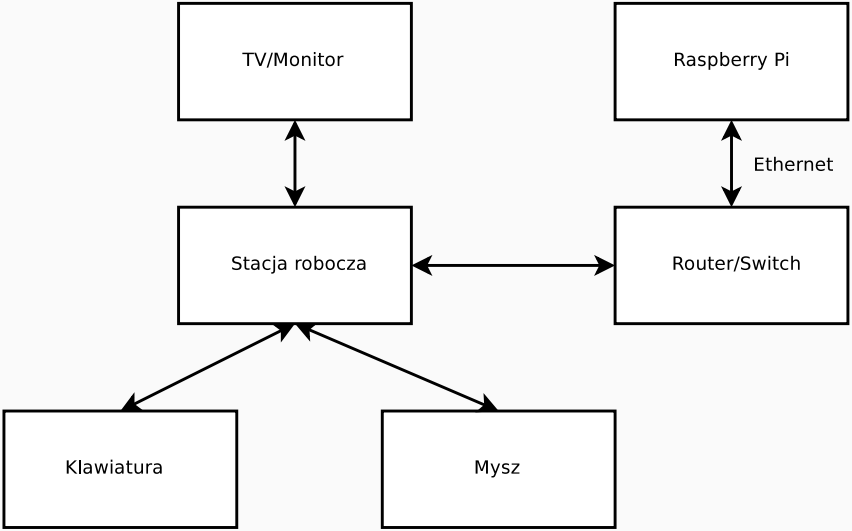
Raspberry Pi 2 Model B V1.1 ii

- BCM2836 (rodzina BCM2709)
  - czterordzeniowy procesor ARM Cortex-A7 900 MHz (ARMv7-A)
  - rdzeń graficzny 3D VideoCore IV
  - slot na kartę micro SD
  - 40 pinów ogólnego przeznaczenia
  - port HDMI
  - 3.5mm port audio i analogowego video
  - interfejs do podłączenia kamery
  - interfejs do podłączenia wyświetlacza
- SMSC LAN9514
  - port Ethernet
  - 4 porty USB
- Elpida EDB8132B4PB-8D-F
  - 1 GB DDR2 SDRAM

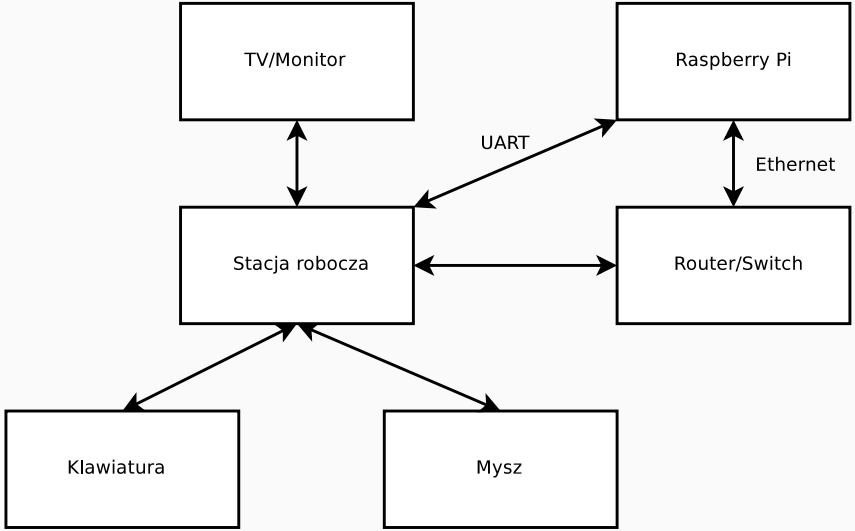
Model pracy - stacja robocza



Model pracy - dostęp zdalny



Model pracy - system wbudowany



Oprogramowanie stacji roboczej

- zazwyczaj Linux
- oprogramowanie umożliwiające komunikacje przez port szeregowy
- oprogramowanie sieciowe
  - Telnet, SSH (Secure Shell) – zdalny terminal
  - NFS (Network File System) – dzielenie plików
  - TFTP (Trivial File Transfer Protocol), HTTP (Hypertext Transfer Protocol) – przesyłanie obrazu systemu
- zestaw narzędzi do kompilacji skrośnej (ang. cross compiling toolchain)

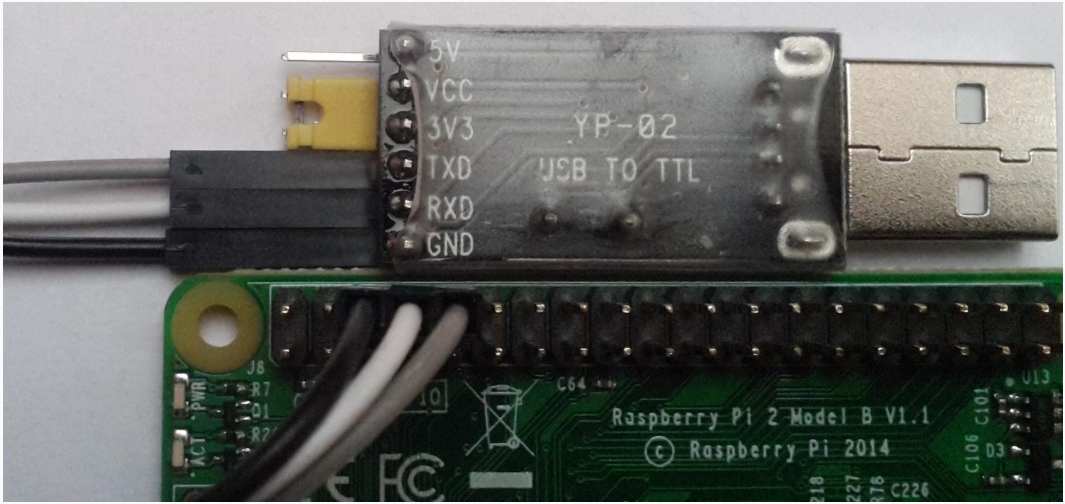
Kompilacja skrośna

- proces kompilacji wykonywany na innej architekturze procesora niż ta, dla której kod jest generowany
- wyróżnia się następujące platformy
  - platforma na której budowany jest kompilator (ang. build platform)
  - platforma na której kompilator będzie uruchamiany (ang. host platform)
  - platforma na której będzie wykonywany kod wygenerowany przez budowany kompilator (ang. target platform)
- zalety
  - brak konieczności posiadania działającego systemu na urządzeniu docelowym
  - brak konieczności posiadania kompilatorów na systemie docelowym
  - znacznie krótszy czas budowania

Komunikacja przez UART i

- UART używa dwóch sygnałów TxD (transmisja danych) – pin 8, RxD (odbiór danych) – pin 10
- aby skomunikować urządzenia należy połączyć skrośnie ich sygnały RxD z sygnałami TxD oraz ich masę – pin 6 (na przykład)
- komunikacja odbywa się na poziomie napięć TTL 3,3V
- w przypadku komunikacji ze stacją roboczą wymagany jest konwerter
  - poziomów napięć – MAX3232
  - USB-UART – FT232RL, CH340G
- domyślne parametry transmisji danych: 115200 8N1 (115200 bps, 8 bitów, bez kontroli parzystości, 1 bit stopu) bez kontroli przepływu danych

Komunikacja przez UART ii



Systemy operacyjne dla Raspberry Pi

- bazujące na Linuxie
  - Raspbian
  - Android Things
  - Gentoo Linux
  - własne dystrybucje
  - zbudowane z wykorzystaniem systemów wpierających tworzenie dystrybucji wbudowanych (ang. embedded), np. Yocto Project
  - (...)
- nie bazujące na Linuxie
  - Windows 10 IoT Core
  - FreeBSD
  - NetBSD
  - Haiku
  - (...)

https://en.wikipedia.org/wiki/Raspberry_Pi#Operating_systems

Własny system bazujący na Linux

+ idealnie skrojony do potrzeb
+ doskonały by zdobyć dogłębną wiedzę na temat Linuxa
− wymaga własnego systemu budowania
− integracja dodatkowych bibliotek wymaga ręcznego dbania o zależności
− wymaga dużej wiedzy i nakładu pracy

Dystrybucja Linux przygotowana za pomocą Yocto Project

+ w razie potrzeby umożliwia idealne dopasowanie do potrzeb
+ umożliwia łatwą integracje tysięcy pakietów wraz z ich zależnościami
+ wspierany przez społeczność
– stosunkowo wysoki próg wejścia
– długi czas budowania
– wymaga ogromnych ilości miejsca na stacji roboczej

Dystrybucja Raspbian

+ gotowa do natychmiastowego użycia
+ umożliwia łatwą instalację tysięcy pakietów wraz z ich zależnościami
+ proces instalacji pakietów taki sam jak dla dystrybucji na PC
+ wspierany przez społeczność
– duży rozmiar minimalnej instalacji (2GB dla wersji Stretch Lite)
– dostępne pakiety zazwyczaj nie są najświeższe

Raspbian

Instalacja Raspbian Strech Lite

- strona Raspbian https://www.raspberrypi.org/downloads/raspbian/
- pobranie i rozpakowanie obrazu

```
~$ wget https://downloads.raspberrypi.org/raspbian_lite/images/raspbian_lite
    -2017-12-01/2017-11-29-raspbian-stretch-lite.zip
~$ unzip 2017-11-29-raspbian-stretch-lite.zip
```

- wykrycie nazwy urządzenia przypisanego do czytnika kart SD za pomocą dmesg

```
sd 6:0:0:1: [sdc] 30318592 512-byte logical blocks: (15.5 GB/14.5 GiB)
sd 6:0:0:1: [sdc] Write Protect is off
 sdc: sdc1 sdc2
sd 6:0:0:1: [sdc] Attached SCSI removable disk
```

- zapisanie obrazu na karcie SD – wartość <sdx> należy zastąpić odpowiednią nazwą urządzenia

```
~$ sudo dd bs=4M if=2017-11-29-raspbian-stretch-lite.img of=/dev/<sdx> status=
    progress conv=fsync
```

Partycje obrazu Raspbian

```
~$ echo p | fdisk ./2017-11-29-raspbian-stretch-lite.img
Disk ./2017-11-29-raspbian-stretch-lite.img: 1,7 GiB, 1858076672 bytes, 3629056
    sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x37665771

Device                                Boot Start    End Sectors Size Id Type
./2017-11-29-raspbian-stretch-lite.img1   8192  93236  85045 41,5M c W95 FAT
./2017-11-29-raspbian-stretch-lite.img2  94208 3629055 3534848 1,7G 83 Linux
```

Początkowe etapy startu Raspbian na RPi2

- GPU wykonuje kod bootera pierwszego poziomu z ROM
- booter 1-go poziomu czyta z partycji FAT karty SD kod bootera drugiego poziomu (bootcode.bin), ładuje go do cache i uruchamia
- booter 2-go poziomu włącza SDRAM i ładuje firmware GPU (start.elf)
- firmware czyta pliki konfiguracyjne (config.txt, cmdline.txt) i na ich podstawie ładuje z karty SD odpowiedni kod jądra systemu Linux do pamięci RAM, a następnie resetuje rdzeń ARM
- rdzeń ARM uruchamia jądro systemu Linux podając jako parametr wywołania jądra zawartość (cmdline.txt)
- jądro systemu Linux montuje partycję ext4 i uruchamia z niej init

Włączenie serwisu SSH na Raspbian z poziomu stacji roboczej

```
~$ mkdir rpiboot
~$ sudo losetup -o $((8192*512)) /dev/loop0 ./2017-11-29-raspbian-stretch-lite.img
~$ sudo mount /dev/loop0 ./rpiboot
~$ sudo touch ./rpiboot/ssh
~$ ls ./rpiboot
bcm2708-rpi-0-w.dtb   bcm2710-rpi-cm3.dtb fixup.dat   LICENCE.broadcom start.elf
bcm2708-rpi-b.dtb     bootcode.bin        fixup_db.dat LICENSE.oracle  start_x.elf
bcm2708-rpi-b-plus.dtb cmdline.txt        fixup_x.dat  overlays
bcm2708-rpi-cm.dtb    config.txt          issue.txt    ssh
bcm2709-rpi-2-b.dtb   COPYING.linux       kernel7.img  start_cd.elf
bcm2710-rpi-3-b.dtb   fixup_cd.dat        kernel.img   start_db.elf
~$ sudo umount ./rpiboot
~$ sudo losetup -d /dev/loop0
~$ rmdir rpiboot
```

Stacja robocza na Ubuntu

Minicom

- instalacja

```
~$ sudo apt-get install minicom
```

- dodanie uprawnień do terminala (po tej zmianie należy się ponownie zalogować)

```
~$ sudo adduser $USER dialout
```

- uruchomienie (nazwę `ttyUSB0` należy zastąpić odpowiednią nazwą terminala do którego został przypisany konwerter USB-UART informacja dostępna przez `dmesg`)

```
~$ minicom -con -D /dev/ttyUSB0 ttyUSB0
```

- wyłączenie sprzętowej kontroli transmisji danych (ang. Hardware Flow Control) – należy wcisnąć `CTRL + a`, następnie `o`, wybrać `Serial port setup` i wcisnąć `f` aby przełączyć ustawienie `Hardware Flow Control` na `No`
- zapisanie ustawień – należy wyjść z poprzedniego menu za pomocą `ESC` i wybrać `Save setup as ttyUSB0`

SSH

- instalacja

```
~$ sudo apt-get install openssh-client
```

- logowanie do Raspbian (domyślne hasło: raspberry)

```
~$ ssh pi@192.168.0.180
pi@192.168.0.180's password:
```

NFS

- instalacja

```
~$ sudo apt-get install nfs-kernel-server nfs-common
```

- konfiguracja

```
~$ sudo mkdir /mnt/nfs
~$ sudo chmod 777 /mnt/nfs
~$ sudo sh -c 'cat << EOF >> /etc/exports
/mnt/nfs    *(rw,sync,no_subtree_check)
EOF'
```

- uruchomienie

```
~$ sudo service nfs-kernel-server restart
```

- montowanie katalogu z poziomu Raspbian

```
pi@raspberrypi:~$ sudo mkdir /mnt/nfs
pi@raspberrypi:~$ sudo mount -t nfs -o nolock 192.168.0.100:/mnt/nfs /mnt/nfs
```

Zestaw narzędzi do kompilacji skrośnej dla Raspberry Pi

- instalacja (najnowsza dostępna wersja znajduje się w katalogu arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf)

```
~$ git clone https://github.com/raspberrypi/tools.git
```

- elementy zestawu narzędzi GNU
  - narzędzia generujące i operujące na plikach obiektowych
    - gcc, g++ – kompilatory dla języków C i C++
    - gdb – debugger
    - nm, objdump, readelf – analiza plików obiektowych
    - ld – linker
    - strip – usuwanie symboli z plików obiektowych
    - ar, ranlib – zarządzanie archiwami
    - as – generowanie kodu binarnego z assemblera
  - podstawowe biblioteki platformy docelowej (libc, libm, libstdc++, libpthread, itd.)

Node.js dla Raspbian

Instalacja z repozytorium Raspbian

- instalacja domyślnej wersji

```
pi@raspberrypi:~$ sudo apt-get install nodejs
pi@raspberrypi:~$ nodejs -v
v4.8.2
```

- instalacja najnowszej wersji LTS

```
pi@raspberrypi:~$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash
    -
pi@raspberrypi:~$ sudo apt-get install nodejs
pi@raspberrypi:~$ node -v
v8.9.4
```

Instalacja z oficjalnej strony node.js

- wersje LTS dla architektury ARM do pobrania ze strony https://nodejs.org/en/download/
  - ARMv6
  - ARMv7
  - ARMv8
- pobranie i instalacja z poziomu Raspberry Pi

```
pi@raspberrypi:~$ wget https://nodejs.org/dist/v8.9.4/node-v8.9.4-linux-armv7l.
    tar.xz
pi@raspberrypi:~$ sudo tar xJvf node-v8.9.4-linux-armv7l.tar.xz -C /usr/local
    --strip-components 1
pi@raspberrypi:~$ node -v
v8.9.4
```

Kompilacja na Raspberry Pi

- pobranie i wypakowanie źródeł z poziomu Raspberry Pi

```
pi@raspberrypi:~$ wget https://nodejs.org/dist/v8.9.4/node-v8.9.4.tar.gz
pi@raspberrypi:~$ tar xzvf node-v8.9.4.tar.gz
pi@raspberrypi:~$ cd node-v8.9.4
```

- kompilacja i instalacja

```
pi@raspberrypi:~/node-v8.9.4 $ ./configure
pi@raspberrypi:~/node-v8.9.4 $ make -j 3
pi@raspberrypi:~/node-v8.9.4 $ sudo make install
```

Kompilacja skrośna

- pobranie i wypakowanie źródeł

```
wget https://nodejs.org/dist/v8.9.4/node-v8.9.4.tar.gz
tar xzvf node-v8.9.4.tar.gz
cd node-v8.9.4
```

- kompilacja i instalacja w katalogu stacji roboczej

```
export PATH=~/tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/bin:$PATH
export AR=arm-linux-gnueabihf-ar
export CC=arm-linux-gnueabihf-gcc
export CXX=arm-linux-gnueabihf-g++
./configure --without-snapshot --without-intl --dest-cpu=arm --dest-os=linux
make -j 4
make install DESTDIR=~/node-v8.9.4-arm
```

Rozwój aplikacji node.js dla Raspberry Pi

Wskazówki dotyczące procesu tworzenia aplikacji

- o wiele łatwiej pisać kod na stacji roboczej niż na Raspberry Pi
- katalog z projektem może być dzielony za pomocą NFS pomiędzy stacją robocza a Raspberry Pi
- pomimo różnic w architekturze, ten sam kod JavaScript może być uruchomiony zarówno na stacji roboczej jak i na Raspberry Pi
- transpilację TypeScript do JavaScript najlepiej wykonywać na stacji roboczej
- warto utrzymywać możliwość uruchamiania aplikacji na stacji roboczej, nawet jeśli pełen zakres funkcjonalności nie jest możliwy do osiągnięcia

Pakiety wykorzystujące kod natywny

- odpowiedni kod binarny jest ściągany w trakcje instalacji pakietu, a w przypadku jego braku następuje próba kompilacji
- w przypadku Raspbian z zainstalowanym kompilatorem możliwa jest instalacja za pomocą `npm` bezpośrednio na urządzeniu
- zalecana jest instalacja pakietów na stacji roboczej z użyciem kompilacji skrośnej

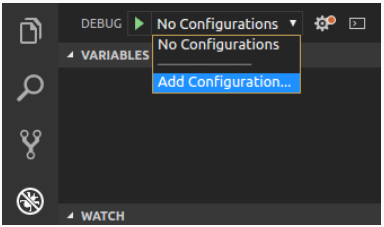
```
export PATH=~/tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/bin:$PATH
export AR=arm-linux-gnueabihf-ar
export CC=arm-linux-gnueabihf-gcc
export CXX=arm-linux-gnueabihf-g++

npm install --arch=arm --target_arch=arm
```

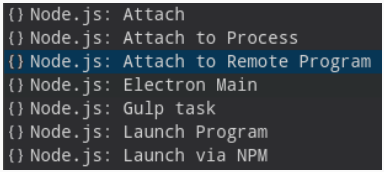
Debugowanie z poziomu Visual Studio Code i

- uruchomienie node.js z włączonym inspektorem na Raspberry Pi

```
pi@raspberrypi:/mnt/nfs/project $ node --inspect=0.0.0.0:7777 --nolazy ./dist/
    index.js
```

- dodanie konfiguracji umożliwiającej podłączenie do node.js działającego na Raspberry Pi z poziomu Visual Studio Code działającego na stacji roboczej



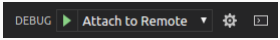


Debugowanie z poziomu Visual Studio Code ii

- dodanie dodatkowego wpisu do pliku `launch.json`

```
        "type": "node",
        "request": "attach",
        "name": "Attach to Remote",
        "address": "192.168.0.180",
        "port": 7777,
        "localRoot": "${workspaceFolder}",
        "remoteRoot": "${workspaceFolder}"
```

- uruchomienie sesji debugowej



- informacje na temat debugowania z poziomu Visual Studio Code dostępne są pod adresem: https://code.visualstudio.com/docs/nodejs/nodejs-debugging

Dziękuję