Wprowadzenie do Linuksa embedded (6)

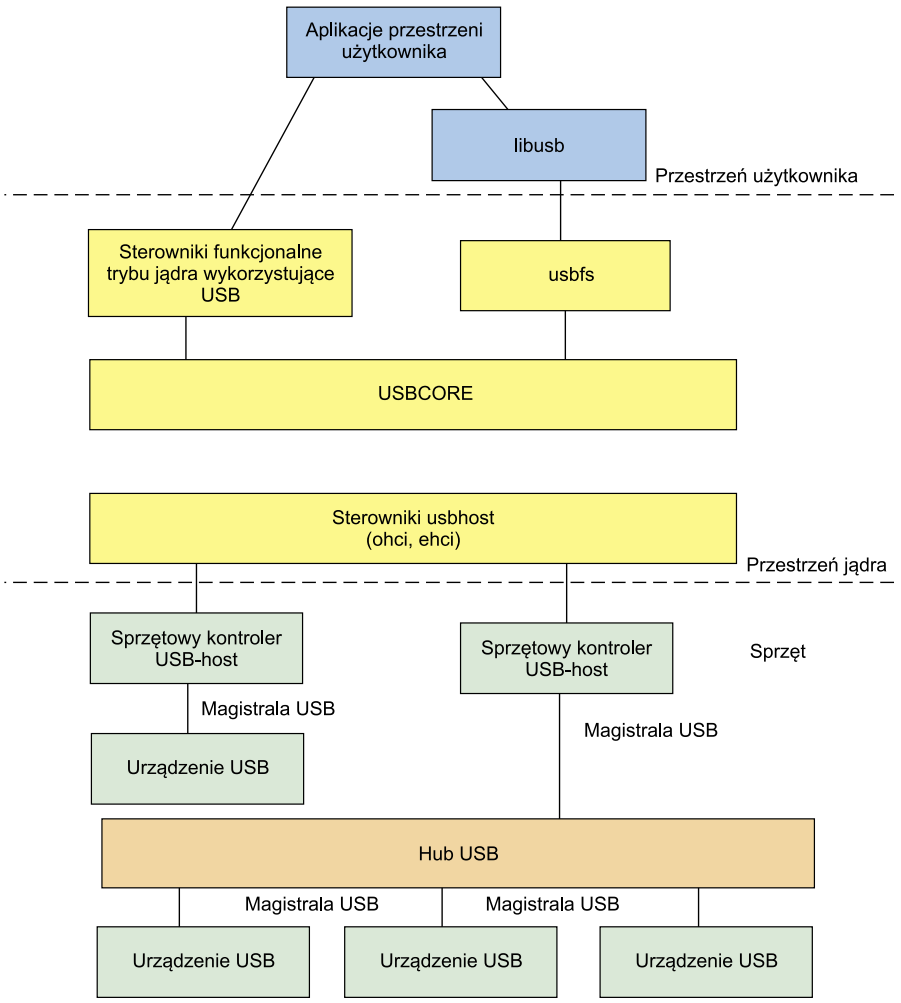
Obsługa interfejsu USB

W ostatnim części kursu poświęconej obsłudze interfejsów przyjrzymy się USB. Jest on od kilkunastu lat dostępny w komputerach PC i wyparł wiele interfejsów komputerowych, takich jak Centronics, RS232, PS/2. We współczesnych komputerach do dyspozycji mamy wiele gniazd USB, które są głównym interfejsem komunikacyjnym z układami peryferyjnymi. Zajmiemy się najbardziej rozpowszechnioną topologią magistrali USB typu master-slave.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 14464, pass: 87f371o5
• poprzednie części kursu

W tym artykule zajmiemy się sytuacją, w której do magistrali USB może być dołączony pojedynczy układ typu master nazywany hostem USB (*USB host*) oraz do 127 układów podrzędnych slave nazywanych urządzeniami USB (*USB device*). Kontrolerem nadrzędnym jest najczęściej komputer PC, natomiast urządzeniami USB są różne urządzenia mikroprocesorowe. Interfejs USB używa technologii *plug&play*, dzięki czemu są automatycznie wykrywane urządzenia oraz instalowane ich sterowniki.

Od dobrych kilkunastu lat możemy nabyć mikrokontrolery jednoukładowe, które mają sprzętowy kontroler USB-device, a ostatnimi czasy pojawiły się mikrokontrolery z interfejsem USB-OTG, który umożliwia pracę zarówno w trybie *USB host* jak i *USB device*. Dzięki swoim zaletom interfejs USB zastąpił większość interfejsów komputerowych, jednak z uwagi na dużą uniwersalność jego oprogramowanie jest skomplikowane. O ile oprogramowanie mikrokontrolera jednoukładowego pełniącego rolę urządzenia USB z użyciem bibliotek dostarczanych przez producenta reprezentuje średni poziom trudności, o tyle oprogramowanie hosta USB jest zadaniem dużo bardziej skomplikowanym. W wypadku oprogramowania hosta, oprócz oprogramowania samego układu kontrolera hosta, musimy oprogramować enumerację urządzeń i komunikację z urządzeniem podrzędnym. Jeżeli do tego uwzględnić fakt, że do USB możemy dołączyć wiele typów urządzeń oraz dodatkowo poszczególne urządzenia mogą być dołączane za pomocą hubów USB, które też wymagają obsługi programowej od strony hosta, wówczas może okazać się, że takie zadanie jest bardzo skomplikowane. Dużo lepszym rozwiązaniem jest użycie mikrokontrolera z systemem Linux, gdzie kosztem nieco większego skomplikowania sprzętowego jako wartość dodaną otrzymujemy obsługę USB (*host* i *device*!) wraz ze sterownikami do większości spotykanych na rynku urządzeń USB. Czytelnicy zapewne po tytule spodziewali się, że w dzisiejszym odcinku będziemy mieli do czynienia ze skomplikowanym programowaniem, jednak tym razem nie napiszemy ani jednej linijki kodu, ponieważ wykorzystamy gotowe sterowniki USB oraz konsolę szeregową. Dostęp do urządzeń USB z własnych programów sprowadza się w zasadzie do otwarcia pliku reprezentującego dane urządzenie i komunikacji za pomocą metod read/write.



Rysunek 1. Podsystem usb-host

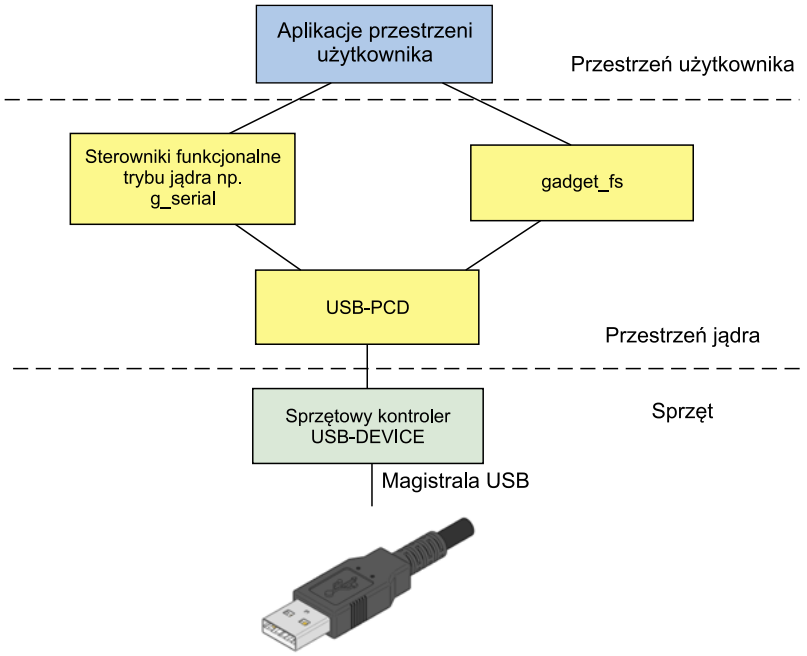
Zagadnienia teoretyczne – architektura USB

Obsługa USB w Linuksie jest realizowana jest przez dwa niezależne systemy, podsystem *usb-host*, który zapewnia infrastrukturę dla trybu nadrzędnego, umożliwiając dołączenie urządzeń peryferyjnych USB oraz *usb-device*, który zapewnia infrastrukturę umożliwiającą pracę Linuksa jako urządzenie podrzędne USB umożliwiając dołączanie płytki np. do komputera PC. Na **rysunku 1** przedstawiono podsystem *usb-host. Jego* sercem jest moduł *usb-core*, który z jednej strony komunikuje się ze sprzętowymi sterownikami hosta, a z drugiej udostępnia niezależne od sprzętu API dla pozostałych modułów jądra oraz implementuje główną logikę magistrali USB, na przykład: enumerację, obsługę endpointów, obsługę hubów itd. Sterownik hosta jest odpowiedzialny za komunikację ze sprzętowym kontrolerem USB zapewniając jednolite API dla jądra podsystemu *usb-core*.

W wypadku sprzętowych kontrolerów USB mamy tutaj porządek, ponieważ zostały one ustandaryzowane przez firmy Intel i Compaq, i w zasadzie większość sprzętowych implementacji kontrolerów USB jest zgodna z tą specyfikacją na poziomie rejestrów. Używane są dwa standardy: OHCI (*Open Host Controler Interface*) przeznaczony dla hostów *USB full speed* (12 Mb/s), oraz EHCI (*Enhaced Host Controller Interface*), który stanowi rozszerzony standard dla obsługi hostów *USB hi speed* (480 Mb/s). W większości wypadków producenci mikrokontrolerów stosują się do tych dwóch standardów, zatem w Linuksie w większości programów będziemy używać dla kontrolerów zgodnych z OHCI modułu *ohci_hcd*, natomiast dla kontrolerów zgodnych z EHCI modułu *ehci_hcd*. Moduły niezgodne z tymi specyfikacjami są rzadkością.

Jak już wielokrotnie wspominano, magistrala USB nie jest urządzeniem samym w sobie, dlatego sterowniki jądra wykorzystują moduł *usb-core* do obsługi magistrali USB i zależnie od rodzaju urządzenia udostępniają w przestrzeni użytkownika odpowiednie API. Na przykład, jeżeli do USB dołączymy konwerter USB na UART, to w przestrzeni użytkownika, za pomocą odpowiedniego sterownika wykorzystującego magistralę USB i podsystem TTY, zostanie udostępniony plik urządzenia /dev/ttyUSB0, który z poziomu aplikacji możemy obsługiwać jak zwykły port szeregowy. Jeżeli np. do USB dołączymy kartę sieciową, to sterownik tej karty udostępni interfejs sieciowy widoczny jako np. eth1. Można zatem powiedzieć, że sama magistrala USB jest „przeźroczysta" dla użytkownika.

Istnieje również możliwość bezpośredniego dostępu do podsystemu USB z trybu użytkownika. Jądro udostępnia specjalny interfejs przestrzeni użytkownika za pomocą *usbfs*, który jest używany przez bibliotekę *libusb* umożliwiającą bezpośrednią obsługę magistrali oraz urządzeń USB z programów użytkownika. Z uwagi na to, że programowanie sterowników dla USB jest zadaniem stosunkowo skomplikowanym, większość sterowników USB jest implementowanych w przestrzeni użytkownika z wykorzystaniem *libusb*. Dzięki temu można uniknąć konieczności pisania sterowników jądra, których uruchamianie jest trudne i skorzystać z dobrodziejstw, jakie niesie pisanie aplikacji w przestrzeni użytkownika.



Rysunek 2. Podsystem usb-device

Oddzielny podsystem stanowi obsługa trybu *USB device*, w którym system Linux może implementować określone urządzenia USB, tak aby była możliwość dołączania płytki z Linuksem do innego hosta USB, na przykład komputera PC. Podsystem *usb-device* (**rysunek 2**) w Linuksie nosi nazwę *usb gadget* i jego architektura jest znacznie mniej skomplikowana, niż w wypadku architektury hosta USB. Sercem podsystemu jest sterownik USB-PCD (*Peripheral Controller Device*), komunikujący się bezpośrednio ze sprzętowym układem *usb-device*, udostępniając jednocześnie jednolite, niezależne od sprzętu API dla sterowników funkcjonalnych.

Sterowniki funkcjonalne jądra wykorzystują API do implementacji poszczególnych funkcjonalności urządzeń USB. Ponieważ urządzenie USB w jednym czasie może pełnić tylko pojedynczą rolę, umożliwia ono współpracę tylko z jednym sterownikiem *gadget* w danym momencie. Na przykład, załadowanie modułu *g_ether* spowoduje że urządzenie *usb device* będzie pełniło rolę karty sieciowej Ethernet.

Istnieje również możliwość pisania sterowników w przestrzeni użytkownika z wykorzystaniem modułu *gadget_fs*. W Linuksie gotowe są w zasadzie wszystkie najpopularniejsze urządzenia usb, w postaci modułów *gadget*, dlatego w większości wypadków nie będziemy musieli ich pisać własnoręcznie.

Przykłady praktyczne

Mikrokontroler AT91RM9200 w obudowie *TQFP208* ma pojedynczy kontroler USB host pracujący z prędkością *full speed* (12 Mb/s), zgodny z OHCI, którego wyjście wyprowadzono na złączu J10 oraz pojedynczy kontroler USB device, którego wyjście wyprowadzono na złącze J11. Do wykonania ćwiczeń z części praktycznej potrzebować będziemy następujących komponentów:

- dowolny pendrive,
- kabel USB A-B
- dowolną przejściówkę USB Serial np. bazującą z układem FT232.

Przedstawione przykłady będą bazowały tylko na wykorzystaniu narzędzi konsolowych. Zostaną również podane krótkie komentarze umożliwiające wykorzystanie danego urządzenia z poziomu własnych aplikacji. Aby wykonać wszystkie zaprezentowane przykłady, należy rozpakować plik *example6.img* na kartę SD, tak jak opisano w pierwszym przykładzie. Następnie włożyć kartę SD do BF210, uruchomić system i zalogować się za pomocą konsoli szeregowej dostępnej na porcie DBG (J6).

Obsługa pamięci masowych typu pendrive, dyski USB itp. Obsługa pamięci masowych z wykorzystaniem mikrokontrolerów jednoukładowych wymaga – oprócz implementacji hosta USB i sterownika dla urządzenia USB komunikującego się z urządzeniem – napisania warstwy SCSI odpowiedzialnej za fizyczny zapis i odczyt sektorów oraz warstwy systemu plików FAT (ew. innego), umożliwiającego dostęp do h-plików, co

bez wsparcia systemu operacyjnego jest bardzo trudne. W Linuksie wszystko mamy już zapewnione przez system. Przeprowadźmy teraz proste ćwiczenie dołączając pendrive do BF210, co zaowocuje następującym ciągiem informacji wyświetlonym na konsoli szeregowej:

```
[ 157.664000] usb 1-1: new full
speed USB device using at91_ohci
and address 3
[ 157.832000] usb 1-1: New USB
device found, idVendor=3538,
idProduct=0059
[ 157.840000] usb 1-1: New USB
device strings: Mfr=1, Product=2,
SerialNumber=3
[ 157.852000] usb 1-1: Product:
PQI USB Flash Drive
[ 157.856000] usb 1-1:
Manufacturer: PQI
[ 157.864000] usb 1-1:
SerialNumber: 0000000000255E
[ 157.896000] scsi1 : usb-storage
1-1:1.0
[ 158.912000] scsi 1:0:0:0:
Direct-Access Generic USB Flash
Disk 0.00 PQ: 0 ANS
I: 2
[ 158.940000] sd 1:0:0:0: [sda]
3948544 512-byte logical blocks:
(2.02 GB/1.88 GiB)
[ 158.980000] sd 1:0:0:0: [sda]
Write Protect is off
[ 158.992000] sd 1:0:0:0: [sda]
Assuming drive cache: write
through
[ 159.044000] sd 1:0:0:0: [sda]
Assuming drive cache: write
through
[ 159.188000] sda: sda1
[ 159.344000] sd 1:0:0:0: [sda]
Assuming drive cache: write
through
[ 159.356000] sd 1:0:0:0: [sda]
Attached SCSI removable disk
```

Z konsoli możemy odczytać informację, że system automatycznie wykrył nowe urządzenie USB oraz załadował dla niego odpowiedni sterownik, dzięki czemu pendrive jest widoczny w systemie jako urządzenie */dev/sda*. Poszczególne partycje są widoczne jako kolejne cyfry. Na przykład, pendrive z tego przykładu ma pojedynczą partycję oznaczoną jako */dev/sda1*. Listę dostępnych partycji możemy sprawdzić odczytując plik */proc/partitions* za pomocą polecenia *cat /proc/partitions*, w wyniku czego otrzymamy listę dostępnych partycji. W naszym wypadku zostanie wyświetlony następujący komunikat:

```
root@bf210-at91:~# cat /proc/
partitions
major minor #blocks name
179 0 965120 mmcblk0
179 1 102400 mmcblk0p1
179 2 32768 mmcblk0p2
8 0 1974272 sda
8 1 1973248 sda1
```

Partycje *mmcblk0* są partycjami karty SD, natomiast partycje *sda* partycjami pendrive'a. Widzimy tu tylko jedną dostępna partycję *sda1*. Po włożeniu pendrive'a jeżeli system wykryje prawidłowy system plików, zostanie ona automatycznie zamontowana, o czym możemy się przekonać wydając polecenie *mount*:

```
root@bf210-at91:~# mount rootfs
on / type rootfs (rw) /dev/
root on / type ext3 (rw,sy
nc,noatime,errors=continue
,barrier=0,data=writeback)
devtmpfs on /dev type devtmpfs
(rw,relatime,size=14556k,nr_
inodes=3639,mode=755)
proc on /proc type proc
(rw,relatime) sysfs on /sys type
sysfs (rw,relatime) none on /dev
type tmpfs (rw,relatime,mode=755)
devpts on /dev/pts type devpts
(rw,relatime,gid=5,mode=620)
usbfs on /proc/bus/usb type
usbfs (rw,relatime) tmpfs
on /var/volatile type tmpfs
(rw,relatime) tmpfs on /dev/shm
type tmpfs (rw,relatime,mode=777)
tmpfs on /media/ram type tmpfs
(rw,relatime) /dev/sda1 on /
media/sda1 type vfat
(rw,relatime,fmask=0022
,dmask=0022,codepage=cp
437,io charset=iso8859-
1,shortname=mixed,errors=remount-
ro)
```

Polecenie *mount* umożliwia montowanie partycji w systemie oraz wyświetlenie aktualnie zamontowanych partycji. Możemy tutaj zobaczyć, że partycja */dev/sda* została zamontowana w katalogu */media/sda1* oraz ma system plików vfat. Wszystkie pliki, które znajdują się na pendrive będą widoczne w tym katalogu. Możemy teraz przejść do katalogu z plikami wydając polecenie *cd /media/sda1*, a następnie wyświetlić listę dostępnych plików wydając polecenie *ls –al*. Zostanie wyświetlony komunikat jak niżej:

```
root@bf210-at91:~# cd /media/sda1
root@bf210-at91:/media/sda1# ls
-al
drwxr-xr-x 2 root root 4096 Jan 1
1970 .
drwxr-xr-x 11 root root 1024 Apr
24 02:23 ..
-rwxr-xr-x 1 root root 1187001760
Mar 22 11:01 OmniaBack.7z
-rwxr-xr-x 1 root root 657575 Mar
25 07:14 isix_multiple_objects.
tar.bz2
root@bf210-at91:/media/sda1#
```

Na pendrive znajdują się dwa pliki. Dostęp do pendrive'a jest możliwy zarówno za pomocą narzędzi konsolowych (co przedstawiono powyżej), jak i z własnych programów. Dostęp z do plików z programów jest możliwy za pomocą biblioteki standardowej *stdio.h*, z użyciem funkcji f*open, fwrite, fread()* lub z poziomu C++ za pomocą biblioteki *iostream*. Wystarczy jedynie operować w podkatalogach znajdujących się w katalogu */media/sda1*. Jeżeli chcemy wyjąć pendrive'a, to musimy pamiętać o jego odmontowaniu, co możemy wykonać za pomocą polecenia konsoli *umount /dev/sda1*. Z poziomu aplikacji użytkownika odmontować pendrive można za pomocą wywołania systemowego *umount(const char *target)* znajdującego się pliku nagłówkowym *sys/mount.h*. Funkcja ta jako argument (podobnie jak konsolowe polecenie *mount*) przyjmuje ścieżkę do urządzenia, które chcemy odmontować. Brak odmontowania pendrive po dokonaniu zapisu, podobnie jak w systemie Windows, gdy wyjmiemy pendrive, bez kliknięcia w ikonę bezpiecznego usuwania sprzętu, może spowodować utratę zapisanych danych i/lub uszkodzenie systemu plików.

Obsługa urządzeń typu USB-Serial. Pod nazwą USB-Serial kryją się różne konwertery USB/RS232, jak i ostatnio bardzo popularne modemy 3G w postaci urządzeń USB wyglądających jak pendrive'y. Modemy 3G poprzez USB udostępniają interfejs portu szeregowego, do którego jest dołączony modem udostępniający polecenia AT. Interfejs szeregowy jest używany przez warstwę PPP w celu uzyskania połączenia internetowego TCP/IP. W Linuksie układy konwertery USB-Serial są obsługiwane głównie przez sterownik *usb_serial* lub inne, pokrewne, które za pomocą warstwy TTY udostępniają w systemie porty TTY, tak jak ma to miejsce w wypadku wbudowanych portów szeregowych. Wystarczy zatem włożyć konwerter lub modem 3G do złącza USB, a w systemie pojawi się plik urządzenia */dev/ttyUSBx* lub */dev/ttyACMx* (gdzie *x* oznacza kolejny numer porządkowy), który będzie reprezentował port szeregowy. Sterowanie portem szeregowym dołączonym za pomocą USB nie różni się niczym od obsługi portu wbudowanego. Należy jedynie otworzyć odpowiedni plik reprezentujący urządzenie i wykorzystać API umówione w poprzednim odcinku.

W tym ćwiczeniu do portu USB BF210 dołączymy konwerter z układem FT2232, która emuluje podwójny port szeregowy. W wyniku tego na konsoli będziemy mogli zaobserwować następujące informacje diagnostyczne:

```
[ 2666.316000] usb 1-1: new full
speed USB device using at91_ohci
and address 4
[ 2666.484000] usb 1-1: New USB
device found, idVendor=0403,
idProduct=6010
[ 2666.492000] usb 1-1: New USB
device strings: Mfr=1, Product=2,
SerialNumber=3
```

```
[ 2667.164000] usbcore:
registered new interface driver
usbserial
[ 2667.192000] USB Serial support
registered for generic
[ 2667.228000] usbcore:
registered new interface driver
usbserial_generic
[ 2667.252000] usbserial: USB
Serial Driver core
[ 2667.424000] USB Serial support
registered for FTDI USB Serial
Device
[ 2667.436000] ftdi_sio 1-1:1.0:
FTDI USB Serial Device converter
detected
[ 2667.448000] usb 1-1: Detected
FT2232C
[ 2667.456000] usb 1-1: Number of
endpoints 2
[ 2667.460000] usb 1-1: Endpoint
1 MaxPacketSize 64
[ 2667.468000] usb 1-1: Endpoint
2 MaxPacketSize 64
[ 2667.476000] usb 1-1: Setting
MaxPacketSize 64
[ 2667.488000] usb 1-1: FTDI
USB Serial Device converter now
attached to ttyUSB0
[ 2667.500000] ftdi_sio 1-1:1.1:
FTDI USB Serial Device converter
detected
[ 2667.512000] usb 1-1: Detected
FT2232C
[ 2667.516000] usb 1-1: Number of
endpoints 2
[ 2667.524000] usb 1-1: Endpoint
1 MaxPacketSize 64
[ 2667.532000] usb 1-1: Endpoint
2 MaxPacketSize 64
[ 2667.536000] usb 1-1: Setting
MaxPacketSize 64
[ 2667.552000] usb 1-1: FTDI
USB Serial Device converter now
attached to ttyUSB1
[ 2667.568000] usbcore:
registered new interface driver
ftdi_sio
[ 2667.576000] ftdi_sio:
v1.6.0:USB FTDI Serial Converters
Driver
```

Po dołączeniu konwertera do portu USB zostaje wykryty układ FT2232, w wyniku czego podsystem USB automatycznie załadował sterownik *ftdi_sio*. W tego czego zostały utworzone dwa pliki */dev/ttyUSB0* oraz */dev/ttyUSB1* reprezentujące porty szeregowe. Tak powstałe porty szeregowe działają identycznie jak wbudowane, systemowe o czym możemy przekonać się zmieniając w przykładzie z poprzedniego odcinka *port* */dev/ttyS1* na */dev/ttyUSB1*.

W wypadku modemów 3G, aby uzyskać połączenie z siecią należy przygotować i skonfigurować warstwę PPP, co wykracza poza łamy niniejszego artykułu. Konfiguracja PPP przebiega w sposób identyczny jak systemie Linux dla PC, dlatego zainteresowanych tą tematyką odsyłam do strony: *http://my-broadband.co.za/vb/showthread.php/21726-Linux-HOWTO-%28With-Stats%29*

Inne urządzenia USB-Host

W jadrze istnieją gotowe w zasadzie wszystkie sterowniki USB do najbardziej popularnych urządzeń USB spotykanych w handlu, dlatego konieczność pisania własnego sterownika, będzie zdarzeniem bardzo rzadkim. Wymienione powyżej przykłady obsługi wybranych urządzeń nie wyczerpują możliwości obsługi wszystkich urządzeń USB, i trudno w ramach jednego artykułu opisać wszystkie możliwe. Niemniej jednak sposób postępowania w zasadzie zawsze jest taki sam, wystarczy jedynie podłączyć urządzenie USB do portu Host, a system automatycznie wykryje i załaduje odpowiedni sterownik. W jaki sposób obsługiwać dalej dane urządzenie zależy od jego rodzaju, a odpowiedni opis można znaleźć w internecie w postaci wielu artykułów, oraz wpisów na forach. Obsługa USB dla systemów wbudowanych jest taka sama jak dla komputerów PC, dlatego wszystkie opisy odnoszące się do PC, będą także odpowiednie dla systemów embedded.

Emulacja portu szeregowego, w trybie usb-device. Omówiliśmy jak podłączać urządzenia peryferyjne do portu host, przejdziemy teraz do sytuacji odwrotnej i opiszemy kilka najbardziej popularnych przypadków dołączenia BF210 do komputera PC z wykorzystaniem kontrolera USB-DEVICE. Jako pierwszy przykład pokażemy w jaki sposób spowodować, aby nasza płytka BF210 od strony komputera PC była widziana jako port szeregowy. Aby to zrobić wystarczy w konsoli szeregowej BF210 załadować moduł g_serial:

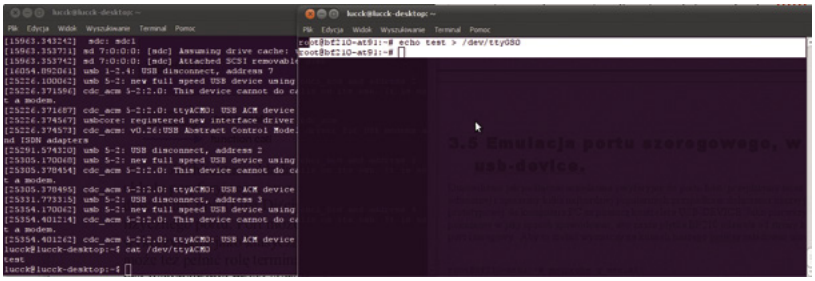
```
root@bf210-at91:~# modprobe g_
serial
[ 1471.676000] g_serial gadget:
Gadget Serial v2.4
[ 1471.688000] g_serial gadget:
g_serial ready
```

W wyniku tego w BF210 zostanie utworzony plik reprezentujący port szeregowy /dev/ttyGS0. Następnie należy podłączyć złącze USB-DEVICE BF210 do komputera PC z wykorzystaniem kabla USB A-B. Po włożeniu wtyczki w systemie na PC pojawi się urządzenie /dev/ttyACM0, stanowiące wirtualny port szeregowy, za pomocą którego możemy się komunikować z BF210. Aby sprawdzić działanie sterownika w konsoli komputera PC należy wpisać polecenie cat /dev/ttyACM0, natomiast w konsoli BF210 polecenie echo test > /dev/ttyGS0. W wyniku wydania polecenia echo na BF210 które wysyła znaki „test" do urządzenia ttyGS0, z wirtualnego portu szeregowego komputera PC będziemy mogli odebrać napis „test" (**rysunek 3**). Obsługa obu portów szeregowych, zarówno po stronie PC jak i BF210, jest taka sama jak w przypadku klasycznych portów szeregowych omówionych w poprzedniej części kursu.

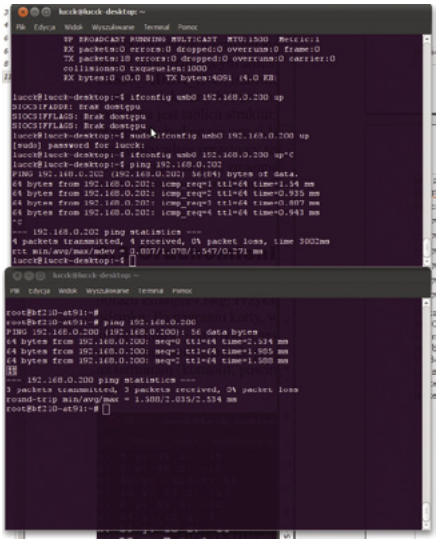
Emulacja karty sieciowej Ethernet w trybie usb-device. Kolejnym interesującym trybem pracy, jest moduł umożliwiający emulowanie karty sieciowej Ethernet, która od strony komputera PC i BF210 jest widziana jako interfejs sieciowy umożliwiający komunikację pomiędzy hostem, a BF210, tak jakby były one połączone za pomocą Ethernetu. Aby udostępnić tą funkcjonalność należy załadować moduł *g_ether*:

```
root@bf210-at91:~# modprobe g_
ether
[ 2702.632000] g_ether gadget:
using random self ethernet
address
[ 2702.644000] g_ether gadget:
using random host ethernet
address
[ 2702.664000] usb0: MAC
7a:52:95:59:0d:e4
[ 2702.676000] usb0: HOST MAC
2a:59:2c:a5:ae:32
[ 2702.684000] g_ether gadget:
Ethernet Gadget, version:
Memorial Day 2008
[ 2702.696000] g_ether gadget:
g_ether ready
```

Po załadowaniu modułu wypisane zostały informację diagnostyczne o wybraniu losowych adresów MAC dla wirtualnych kart sieciowych. Po tej czynności urządzenie *usb-device* jest gotowe do pracy. Wystarczy zatem połączyć komputer PC z BF210 za pomocą kabla USB w wyniku czego na obu maszynach zostanie utworzony wirtualny interfejs sieciowy *usb0*. Na BF210 adres IP został automatycznie wybrany losowo, natomiast na komputerze PC należy wydać polecenie *sudo ifconfig usb0 192.168.0.200 up* w celu jego ręcznego ustanowienia. Od tego momentu



Rysunek 3. Komunikat „test" wysyłany przez port wirtualny /dev/ttyGS0

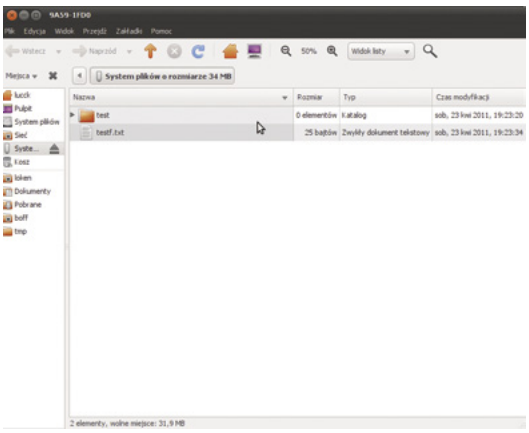


Rysunek 4. Efekt działania polecenia ping w trybie emulacji połączenia Ethernet

komputer PC łączy się z BF210 za pomocą kabla USB, o czym możemy się przekonać za pomocą polecenia ping (**rysunek 4**).

Emulacja urządzenia storage (pendrive), w trybie usb-device. Interesującym trybem jest emulacja urządzeń typu storage, np. popularnych pendrive'ów. Umożliwia to implementacje bardzo popularnego w aparatach czy odtwarzaczach MP3 trybu umożliwiającego przegranie danych multimedialnych na nośnik urządzenia docelowego, bez konieczności posiadania specjalistycznego sterownika. Możemy udostępnić zarówno partycję nośnika, jak i pojedynczy plik znajdujący się na partycji systemowej, który będzie systemem plików dla urządzenia zewnętrznego.

System plików udostępniony przez USB możemy również zamontować w BF210. Musimy pamiętać o tym, aby nie robić tego równocześnie z montowaniem partycji. Dlatego albo udostępniamy plik/partycję za pomocą modułu *usb_storage*, albo montujemy ją w systemie. W przeciwnym razie możemy doprowadzić do uszkodzenia danych, ponieważ dostęp do systemu plików odbywa się za pomocą zapisu sektorów na dysk



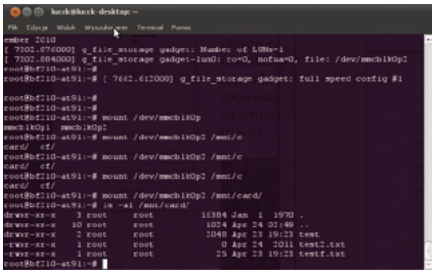
Rysunek 5. Rezultat emulowania urządzenia klasy storage

i systemy operacyjne nic nie „wiedzą" o sobie wzajemnie. W przykładzie na karcie SD utworzono dodatkową partycję z systemem plików FAT16. Aby udostępnić tę partycję w trybie emulowania pamięci pendrive, musimy załadować moduł *g_file_storage*:

```
modprobe g_file_storage file=/dev/
mmcblk0p2 removable=1 stall =0
[ 7302.852000] g_file_storage
gadget: No serial-number string
provided!
[ 7302.864000] g_file_storage
gadget: File-backed Storage
Gadget, version: 1 Sept ember
2010
[ 7302.876000] g_file_storage
gadget: Number of LUNs=1
[ 7302.884000] g_file_storage
gadget-lun0: ro=0, nofua=0, file:
/dev/mmcblk0p2
```

Jako parametr *file* należy przekazać plik lub partycję, które chcemy udostępnić. Jako parametr *removable* przekazujemy, że mamy do czynienia z nośnikiem wymiennym, natomiast parametr *stall* umożliwia uniknięcie generowania zatrzymania przesyłania danych poprzez rozkaz *STALL*, co jest potrzebne systemom Windows. Jako dane wirtualnego dysku udostępniliśmy całą partycję drugiej karty o wielkości 32 MB, sformatowaną w systemie plików FAT16 (*/dev/mmcplk0p2*). Teraz wystarczy włożyć wtyczkę USB do portu USB komputera, i po chwili zobaczymy, że w systemie pojawi się nowy dysk, a na nim dwa pliki (**rysunek 5**).

Teraz możemy dowolnie modyfikować pliki w urządzeniu np. utworzyć dowolny plik tekstowy na test2.txt,a następnie odmontować (bezpiecznie usunąć) urządzenie. Aby teraz zobaczyć w systemie BF210 pliki, które utworzyliśmy na dysku, należy wyładować moduł *g_file_storage* (polecenie *rmmod g_file_storage*), a następnie zamontować system plików w systemie (polecenie mount */dev/mmcblk0p2 /mnt/card*). Po wykonaniu tych czynności za pomocą polecenia *ls -al /mnt/card* możemy zobaczyć, że plik test2.txt, który wcześniej utworzyliśmy na PC istnieje (**rysunek 6**). Przed ponownym udostępnieniem partycji za pomocą sterownika *g_file_storage* należy za pomocą polecenia *umount /dev/mmcblk0p2* odmontować z systemu partycję.



Rysunek 6. Lista plików zapamiętanych na karcie SD podczas emulowania urządzenia klasy storage

A co z Plug&Play?

W opisanych przykładach sami dołączaliśmy urządzenia USB, więc można było bezpośrednio odwoływać się do określonych plików urządzeń. W wielu wypadkach, gdy zakładamy że jakieś urządzenie dołączone jest na stałe do USB bez możliwości odłączenia (np. we wnętrzu urządzenia) takie rozwiązanie jest prawidłowe. Zupełnie inaczej wygląda sytuacja, gdy na zewnątrz wyprowadzone jest gniazdo USB i nie można zakładać dostępności urządzenia w systemie. Ponieważ magistrala USB umożliwia dołączenie i odłączenie urządzeń w dowolnym momencie, musimy mieć możliwość powiadamiania o dołączanych lub odłączanych urządzeniach. W systemie Linuks takie zadanie realizują programy przestrzeni użytkownika *udev* w połączeniu z *devicekit*. Korzystając z usług *devicekit* mamy możliwość otrzymywania notyfikacji o zmianach urządzeń w systemie. Aby zobaczyć jak działa *devicekit* wystarczy wydać polecenie *devkit -m*, a następnie włożyć np. konwerter z FT2232. Wtedy zobaczymy na terminalu szereg notyfikacji, od różnych podsystemów, z których najistotniejsza dla nas jest ostatnia pochodząca od podsystemu TTY:

```
--------------------------------
--------------------------------
-------
action=add @ 05:37:24.032427
device: /sys/devices/platform/
at91_ohci/usb1/1-1/1-1:1.1/
ttyUSB1/tty/ttyUSB1
subsystem: tty
device file: /dev/ttyUSB1
symlink: /dev/serial/by-path/
platform-at91_ohci-usb-0:1:1.1-
port0
symlink: /dev/serial/by-id/usb-
BoFF_OOCDLink_FTQYA8VN-if01-
port0
properties:
ID_IFACE=01
ID_VENDOR_ENC=BoFF
ID_VENDOR_ID=0403
ID_SERIAL=BoFF_OOCDLink_FTQYA8VN
UDEV_LOG=3
ID_MODEL=OOCDLink
ID_TYPE=generic
ID_USB_DRIVER=ftdi_sio
ID_VENDOR=BoFF
ID_PATH=platform-at91_ohci-
usb-0:1:1.1
DEVNAME=/dev/ttyUSB1
SEQNUM=827
ACTION=add
```

```
DEVPATH=/devices/platform/at91_
ohci/usb1/1-1/1-1:1.1/ttyUSB1/
tty/ttyUSB1
ID_SERIAL_SHORT=FTQYA8VN
ID_MODEL_ENC=OOCDLink
MAJOR=188
ID_BUS=usb
ID_REVISION=0500
ID_MODEL_ID=6010
ID_PORT=0
SUBSYSTEM=tty
ID_USB_INTERFACES=:ffffff:
ID_USB_INTERFACE_NUM=01
DEVLINKS=/dev/serial/by-path/
platform-at91_ohci-usb-0:1:1.1-
port0 /dev/serial/by-id/usb-
BoFF_OO
CDLink_FTQYA8VN-if01-port0
MINOR=1
```

Dla nas najistotniejsze są tutaj pola *action @add* stanowiące notyfikację o dołączeniu nowego urządzenia, nazwę pliku reprezentującego urządzenie oraz *VendorId* umożliwiający identyfikację urządzenia. Podobnie po wyjęciu urządzenia dostajemy następujące informacje:

```
action=remove @ 05:40:39.411938
device: /sys/devices/platform/
at91_ohci/usb1/1-1/1-1:1.0/
ttyUSB0/tty/ttyUSB0
subsystem: tty
device file: /dev/ttyUSB0
symlink: /dev/serial/by-path/
platform-at91_ohci-usb-0:1:1.0-
port0
symlink: /dev/serial/by-id/usb-
BoFF_OOCDLink_FTQYA8VN-if00-
port0
properties:
ID_IFACE=00
ID_VENDOR_ENC=BoFF
ID_VENDOR_ID=0403
ID_SERIAL=BoFF_OOCDLink_FTQYA8VN
UDEV_LOG=3
ID_MODEL=OOCDLink
ID_TYPE=generic
ID_USB_DRIVER=ftdi_sio
ID_VENDOR=BoFF
ID_PATH=platform-at91_ohci-
usb-0:1:1.0
DEVNAME=/dev/ttyUSB0
SEQNUM=829
ACTION=remove
DEVPATH=/devices/platform/at91_
ohci/usb1/1-1/1-1:1.0/ttyUSB0/
tty/ttyUSB0
ID_SERIAL_SHORT=FTQYA8VN
ID_MODEL_ENC=OOCDLink
MAJOR=188
ID_BUS=usb
ID_REVISION=0500
ID_MODEL_ID=6010
ID_PORT=0
SUBSYSTEM=tty
ID_USB_INTERFACES=:ffffff:
ID_USB_INTERFACE_NUM=00
DEVLINKS=/dev/serial/by-path/
platform-at91_ohci-usb-0:1:1.0-
port0 /dev/serial/by-id/usb-
BoFF_OO
CDLink_FTQYA8VN-if00-port0
MINOR=0
```

Podobnie jak poprzednio, najistotniejsze dla nas są: rodzaj zdarzenia (*remove*), plik urządzenia oraz product_id. Aby skorzystać z funkcjonalności notyfikacji przez *devkit*, należy dołączyć plik nagłówkowy *#include <devkit-gobject/devkit-gobject.h>*, utworzyć nową instancję klienta *devkit* oraz włączyć się do systemu notyfikacji devkita:

```
client = devkit_client_new
((const char **) opt_subsys);
if (!devkit_client_connect
(client, &error)) {
  g_warning („Cannot connect to
DeviceKit daemon: %s”, error-
>message);
  g_error_free (error);
  goto out;
}
```

Następnie należy zarejestrować handler zdarzenia odpowiedzialnego za sygnalizację zdarzeń od urządzeń: *g_signal_connect (client, „device-event”, G_CALLBACK (device_event_signal_handler), NULL);*

Teraz przy każdej zmianie urządzenia w systemie zostanie wywołana funkcja *device_event_signal_handler*. Szczegółowe informacje na temat sposobu funkcjonowania powiadomień można uzyskać analizując kod źródłowy zapisany w pliku *devkit.c*, który zawiera implementację wypisującą otrzymane notyfikację bezpośrednio na ekranie.

Wykorzystując powiadomienia przez *devicekit* możemy napisać aplikację, która odpowiednio zareaguje na zdarzenia wykrycia dołączenia urządzenia USB. Na przykład, gdybyśmy pisali aplikację odtwarzacza MP3, to po wykryciu urządzenia masowego, program może sprawdzić jego zawartość i zacząć automatycznie odtwarzać pliki muzyczne.

Zakończenie

Na tym zakończyliśmy pierwszą część kursu opisującą podstawowe zagadnienia wprowadzające do systemu Linux. Celowo, z uwagi na objętość, pomnieliśmy opis tworzenia wątków procesów, komunikacji międzyprocesowej obsługi sieci i wszystkich aspektów programowania systemowego. Szczegółowe opisy można znaleźć w wielu ogólnych publikacjach dotyczących systemu Linux dla komputerów PC. Aspekty systemowe nie różnią się w przypadku Linuksa w urządzeniach wbudowanych. Celowo skupiliśmy się tutaj na dołączaniu różnych urządzeń i układów spotykanych w naszej elektronicznej praktyce, ponieważ literatury na ten temat jest raczej niewiele.

W następnej części kursu przejdziemy do zagadnień nieco bardziej zaawansowanych. Czytelnicy mogą także zgłaszać swoje propozycje o czym chcieliby przeczytać w kolejnych częściach kursu na adres mailowy autora lub redakcji.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl