

I/O moduly Quido a Linux

Prozatím jsem zkoumal jen ASCII formát Spinelu na sériovém portu. Nejjednodušším způsobem rozchození v Linuxu je vyrobit vlastní příkaz, který jako parametr bude mít příkazy Spinelu.

Nejprve je potřeba nastavit parametry portu. Ty stačí nastavit jen jednou, např. po spuštění počítače a nastavení tedy stačí dát do `rc.local`. To je startovací skript, který se spouští při bootu jako poslední s právy roota.

V Linuxu je potřeba si dát **pozor na práva**, běžný uživatel nemá právo port nastavovat, ani z něj číst nebo na něj zapisovat. Kdo má právo pracovat s portem zjistíme příkazem `ls -ls /dev/ttyS0`. Zde uvidíme uživatele i skupinu, které je port přiřazen. Nytí máme tři možnosti jak povolení provést.

1. Povolit všem přístup na port `chmod o+w /dev/ttyS0` a `chmod o+r /dev/ttyS0`, ale to může být nebezpečné.
2. Přidat vašeho uživatele do skupiny, která má na portu právo `usermod -G skupina_portu uzivatel`
3. Povolit spuštění příkazu pomocí `sudo`, a pak se program bude spouštět jako od roota.
4. Poslední možností je aplikaci spouštět jako root, ale to se s ostrými aplikacemi z bezpečnostních důvodů nedělá.

Tahle vložka ohledně práv se netýká jen nastavení portu, ale také samotného provozování skriptu, který s Quido modulem komunikuje. Aby uživatel, pod kterým skript spustíme, měl práva zápisu a čtení na port.

Nastavení sériového portu PC na defaultní nastavení modulu Quido provedeme příkazem `stty -F /dev/ttyS0 -crtcts cs8 -cstopb -parenb 9600`. Detailní informace jsou v manuálové stránce a zde je popis parametrů, které nás zajímají:

<code>-F /dev/ttyS0</code>	- určení portu, o který se nám jedná. Modul funguje i plnohodnotným převodníkem USB-RS232 a pak je port samozřejmě jiný, např. <code>/dev/ttyUSB0</code> .
<code>-crtcts</code>	- vypnutí hw kontroly toku, je to jen na 3 dráty tak je to jasné :-)
<code>cs8</code>	- 8mi bitové kódování
<code>-cstopb</code>	- stop bity na 1
<code>-parenb</code>	- vypnutí parity
<code>9600</code>	- komunikační rychlost 9600 Bd

Příkazem `stty -F /dev/ttyS0 -a` si nastavení můžete zkontrolovat. Je zde vypsáno mnohem víc parametrů jako nastavení bufferů atd.

Vlastní příkaz je **bashový skript**, který umístíme třeba do `/usr/local/bin` a pak ho můžeme volat buď ze shellu nebo `php`, `perl` atd.

```
#!/bin/bash
exec 4< /dev/ttyS0
echo -en "$1\r" > /dev/ttyS0
read -u 4 vystup
echo $vystup
```

A teď vysvětlení jednotlivých řádků:

<code>exec 4< /dev/ttyS0</code>	- Interní příkaz bashe, který vytvoří file descriptor. To způsobí, že následující příkaz <code>read</code> nebude číst standardně z klávesnice, ale z portu. Po provedení příkazu <code>read</code> se filedescriptor ruší a pokud by bylo potřeba mít v jednom skriptu více příkazů, je pro každý nutné vytvořit nový. Číslo 4 je jeho název, který jsem si zvolil.
------------------------------------	--

echo -	- Poslání příkazu na port. Parametr <code>-e</code> způsobí převod escape znaků na příkazy, to
en "\$1\r" > /dev/ttyS0	je kvůli poslání příkazu CR, který je převeden z napsaného <code>\r</code> . Parametr <code>-n</code> vypne automatické odřádkování, které je defaultní a quido by si s tím neporadilo. <code>\$1</code> je automaticky vytvořená proměnná z parametru napsaného za příkaz.
read -u 4 vystup	- Čtení odpovědi z portu pomocí filedescriptoru. Přečtená hodnota je uložena do proměnné <i>vystup</i> .
echo \$vystup	- Zobrazení výstupu při ukončení skriptu.

Podrobný manuál k příkazům `exec` a `read` je v *man bash* – pozor manuál je opravdu dlouhý.

Pokud je vše správně nastaveno a zapojeno a my napíšeme `quido.sh *B10S1H`, sepne se relé výstupu 1 a vrátí se nám výpis `*B10`. Tímto máme vytvořený systémový příkaz pro ovládání Quida. Vlastní využití může probíhat potom třeba přes PHP tak, že přes PHP funkci `system`, budeme volat tento skript a číst výsledky. Textové výsledky pak můžeme dál zpracovávat. Ovšem znovu upozorňuji, pozor na práva k portu, když to nechodí.

Jan Vondráček