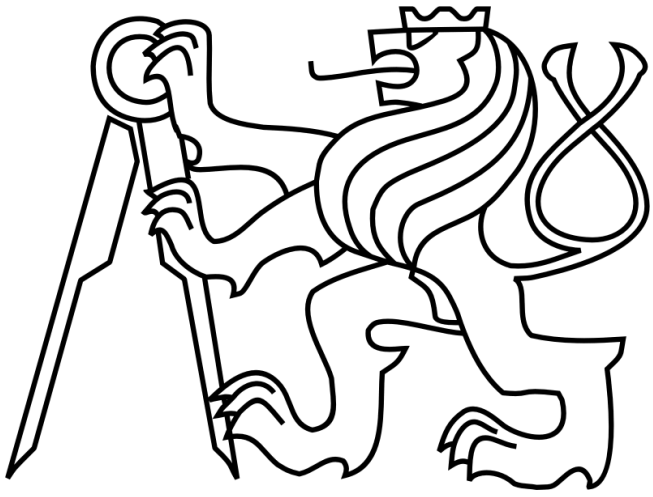
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ
V PRAZE

Fakulta elektrotechnická

Katedra řídící techniky



Bakalářská práce

Vedoucí práce: Ing. Michal Sojka

2009

Jiří Balach

Zadaní práce

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, atd…) uvedené v přiloženém seznamu.

V Praze dne …… …….

Podpis

Chtěl bych poděkovat především vedoucímu mé bakalářské práce Ing. Michalovi Sojkovi za vstřícnost a cenné připomínky k práci. Mé díky patří také Ing. Pavlovi Píšovi za úpravu programu vca_canping. Děkuji také rodině a blízkým za podporu po dobu studia.

Anotace

Práce se zabývá otestováním a porovnáním dvou ovladačů sběrnice CAN pro operační systém Linux. Testovaní klade důraz na real-time aplikace. Součástí práce je také program pro testování obou ovladačů.

Klíčová slova: CAN, Lincan, SocketCAN, vca_canping, srovnávací testy

Annotation

The thesis is focused on the testing and comaparation of two CAN drivers for operation systém Linux. Tests insist on real-time aplication . Also a software tool for testing both drivers is a part of this thesis.

Key words: CAN, Lican, SocketCAN, vca_canping, benchmark

1 Úvod

Cílem této bakalářské práce je porovnání vlastností dvou Linuxových ovladačů sběrnice CAN, a to ovladače Lincan a SocketCAN. Vzhledem k tomu, že SocketCAN využívá socketů, stejně jako TCP/IP komunikace, bude nás zajímat, jak tato síťová komunikace ovlivní jeho vlastnosti.

Bakalářská práce je rozdělena do několika částí, z nichž první popisuje sběrnici CAN, její realizaci a funkci. V dalších částech jsou popsány ovladače, postup jejich kompilace a instalace. Dále je popsán program vca_canping, kterým byly oba ovladače testovány. Na závěr jsou vypsány parametry a výsledky všech testů a jejich zhodnocení.

2 Sběrnice CAN (Controller Area Network)

Jedná se o sériový komunikační protokol, který byl vyvíjen firmou Robert Bosch od roku 1983 jako projekt komunikační sítě pro motorová vozidla. Oficiálně byl vydán v roce 1986. Dnes se sběrnice CAN využívá v různých průmyslových aplikacích. CAN sám o sobě definuje pouze protokol síťové vrstvy. Parametry fyzické vrstvy jsou definovány několika protokoly, z kterých je nejrozšířenější realizace podle normy ISO 11898.

CAN umožňuje distribuované řízení v reálném čase s přenosovou rychlostí 1Mbit/s při maximální délce 40m a velmi dobrým zabezpečením proti chybám v přenosu. Zprávy jsou přenášeny v 13 (standardní formát) nebo 33 (přídavný formát) bitových rámcích. Jedná se o multi-master protokol, to znamená, že více uzlů připojených k sběrnici má právo vysílat. Kolize na sběrnici jsou řešeny na základě prioritního rozhodování. Priorita zprávy je určena jejím identifikátorem. V případě kolize jsou pak zprávy s vyšší prioritou posílány přednostně. Jednotlivé zprávy neobsahují informace o cílovém uzlu a jsou přijímány všemi uzly připojenými na sběrnici. Jednotlivé uzly si vybírají jim určené zprávy pomocí filtrování příchozích zpráv.

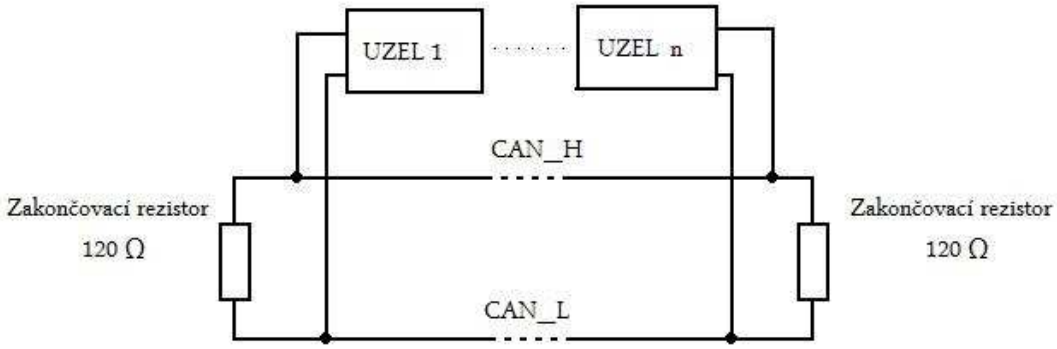
2.1 Fyzická vrstva

Hlavním požadavkem na fyzické přenosové médium protokolu CAN je aby realizovalo funkci logického součinu (Tabulka 1).

	0	**1**
0	0	0
1	0	1

Tabulka 1 Logický součin (AND)

Pro realizaci fyzického přenosového média se nejčastěji využívá rozdílová sběrnice definovaná podle normy ISO 11898 (Obrázek 2.1). Sběrnice se skládá ze dvou vodičů označených jako CAN_H a CAN_L. Dominantní a recesivní úroveň je definována rozdílovým napětím těchto vodičů. Velikost rozdílového napětí podle normy je V_{diff} = 0V pro recesivní úroveň a V_{diff} = 2V pro dominantní úroveň. Sběrnice je zakončena rezitory 120 Ω, které zabraňují odrazům na konci vedení.



Obrázek 2.1 Struktura CAN podle ISO 11898

2.2 Linková vrstva

Linková vrstva CAN protokolu je tvořena dvěma podvrstvami (MAC a LLC). MAC (Medium Access Control) se stará o řízení přístupu k médiu, provádí kódování dat, vkládá doplňkové bity do komunikace (Stuffing/ destuffing). Řídí přistup všech uzlů k médiu s rozlišením priorit zpráv, detekuje a hlásí chyby a hlásí správně přijaté zprávy. Druhá podrvstva LLC (Logica Link Control) se stará o filtrování příchozích zpráv a o hlášení o přetížení. Je-li sběrnice volná, může libovolný uzel zahájit komunikaci. Jakmile jeden uzel zahájí komunikaci, ostatní uzly musí počkat až jeho komunikace skončí. Poté mohou vysílat i ostatní uzly. Jednou výjimkou jsou chybové rámce, které může vysílat libovolný uzel, detekuje-li chybu v právě přenášené zprávě. Pokud se o vysílání bude snažit více uzlů najednou, získá přístup k sběrnici uzel s nejvyšší prioritou (nejmenší hodnotou identifikátoru). Identifikátor se vysílá jako první část zprávy. Každý vysílač porovná hodnotu právě vysílaného bit se stavem na sběrnici a zjistí, zda-li není na sběrnici jiná hodnota než ta, kterou vysílá. Pokud jeden uzel vysílá recesivní bit a

druhý dominantní (Tabulka 2), uzel, který vysílal recesivní bit, detekuje kolizi a na chvíli přestane vysílat.

	Dominantní	**Recesivní**
Dominantní	Dominantní	Dominantní
Recesivní	Dominantní	Recesivní

Tabulka 2 Stav sběrnice při vysílání dvou uzlů

Přenos zpráv po sběrnici CAN je zabezpečen několika způsoby, které pracují současně. Jsou to:

2.2.1 CRC kód (Cyclic Redindancy Check CRC):

CRC chrání informace v rámci a přidává kontrolní bity na konec přenosu. Na konci příjmu jsou tyto bity přepočítány a testovány vůči přeneseným bitům. Jestliže nejsou v pořádku, pak je hlášena CRC chyba. Zabezpečení je provedeno pomocí CRC-15 polynomu ($x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$).

2.2.2 Kontrola rámce (Message Frame Check):

Tento mechanismus prověřuje strukturu přeneseného rámce dle kontroly bitového pole vůči pevnému formátu a velikosti rámce. Chyba detekovaná kontrolou rámce je označena jako “format errors”.

2.2.3 Potvrzení přijetí (Acknowledge)

Je-li zpráva v pořádku přijata libovolným uzlem, je to potvrzeno změnou jednoho bitu zprávy (ACK). Vysílač na tomto místě vždy vysílá recesivní bit. Detekuje-li dominantní bit, je vše v pořádku. Potvrzení přijetí zprávy je detekováno všemi uzly bez ohledu na filtrování zpráv. V případě detekce chyby kterýmkoli

uzlem, vyšle daný uzel na sběrnici chybový rámec, který se skládá z šesti dominantních bitů. Tím je porušeno pravidlo o vkládání bitů (viz. níže) a je detekována chyba i ostatními uzly. Aby nemohl vadný uzel blokovat komunikaci, existuje dodatečný mechanismus, který odpojí uzel, který generuje příliš velké množství chyb.

CAN protokol také užívá dva mechanismy pro detekci chyb na bitové úrovni:

2.2.4 Pozorování (Monitoring)

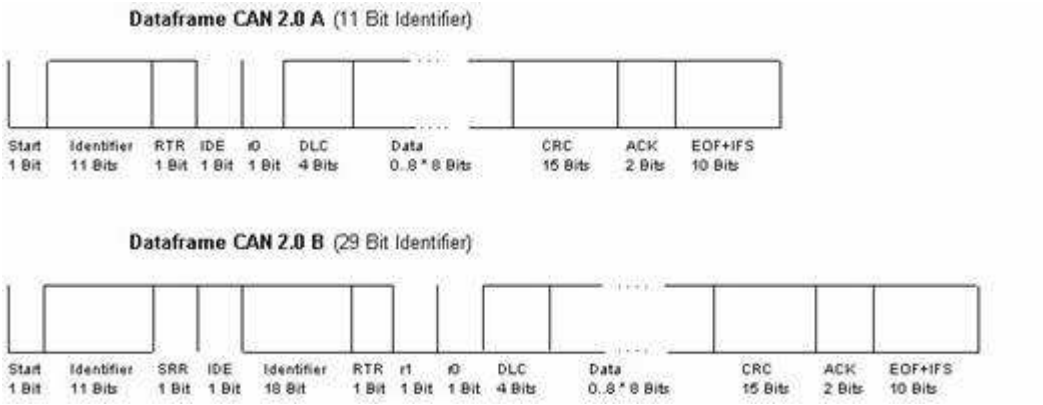
Schopnost vysílače detekovat chyby je základem pro pozorování sběrnicových signálů: každý uzel, který vysílá, také pozoruje sběrnicovou část, a tak detekuje rozdíly mezi bity poslanými a bity přijatými. Toto dovoluje spolehlivou detekci globálních a lokálních chyb na straně vysílače.

2.2.5 Vkládání bitů (Bit stuffing)

Kódování samostatných bitů je testováno na bitové úrovni. Bitová reprezentace CAN je NRZ kód (non-return-to-zero), který garantuje maximální efektivitu v bitovém kódování. Synchronizaci tvoří vkládání pěti po sobě jdoucích stejných bitů. Odesílatel přidá do bitového toku vložený bit s doplňkovou hodnotou, která je odstraněna příjemcem. Kódová kontrola je omezena kontrolou správnosti vloženého pravidla.

Protokol CAN definuje čtyři typy zpráv. První dvě se týkají přenosu dat. Jsou to zprávy „Data Frame“ (Datová zpráva), což je základní prvek datové komunikace po sběrnici a „Remote Request Frame“ (Žádost o data), kterou uzel žádá ostatní účastníky o požadovaná data. Další dva typy zpráv se týkají řídící komunikace po sběrnici. Jsou to zprávy „Error Frame“ (Chybová zpráva) a „Overload Frame“ (Zpráva o přetížení).

Formáty datových zpráv:



Obrázek 2.2 Datové zprávy podle specifikace CAN 2.0

Význam jednotlivých datových polí:

Začátek zprávy (*Start of frame= SOF*) – začátek zprávy, 1 dominantní bit

Řízení přístupu na sběrnici (*Arbitration Field*) – určení priority zprávy

- Identifikátor zprávy (*Identifier*) – 11 bitů, udává význam přenášené zprávy

- RTR (*Repote Request*) – 1 bit, určuje typ zprávy (Datová zpráva / Žádost o zprávu). Dominantní bit znamená datovou zprávu, žádost o data bit recesivní.

Řídící informace (*Control Field*)

- r0, r1 – rezervované bity

- DLC (*Délka Dat*) – 4 bity, určují počet datových bitů ve zprávě (0-8)

Datová oblast (*Data Field*) – datové byty zprávy. Max. 8 bytů, vysíláno od MSB

CRC (*CRC Field*) – 16 bitů

- CRC kód – 15 bitů

- ERC (*CRC Delimiter*) – 1 dominantní bit

Potvrzení (*ACK Field*) – 2 bity

- ACK (*Acknowledge bit*) – 1 dominantní bit

- ACD (*Acknowledge delimiter*) – 1 recesivní bit

Konec zprávy (*End of frame)* – 7 recesivnich bitů

Mezera mezi zprávami (*Interframe Space*) – 3 recesivní bity, odděluje dvě zprávy

3 Ovladače a programy

CAN ovladače dodávané výrobci jsou tvořeny pro dané zařízení. Obvykle slouží jen k odesílání a přijímání zpráv. Implementace pro dané zařízení umožňuje pouze jednomu procesu komunikovat, to znamená že ostatní procesy jsou blokovány. Lincan i SocketCAN ovšem podporují více CAN zařízení a jsou schopné komunikovat pomocí stejného objektu s několika aplikacemi nebo vlákny najednou.

3.1 Lincan

Jedná se o open source modul jádra Linuxu, který implementuje ovladače CAN sběrnice. Byl vytvořen Ing. Pavlem Píšou v rámci projektu OCERA (Open Components for Embedded Real-time Applications). Je kompatibilní s jádry Linux řad 2.4.x a 2.6.x, jádry 2.4.x rozšířené o RT-Linux a jádry 2.6 s podporou real-time preemptivity. Podporováno je více než 20 karet na sběrnicích ISA/PC-104, VME, PCI i embedded HW. Interně je ovladač řešen jako systém vyrovnávacích front (FIFO).

Každý ovladač je podsystém, který nemá přímé aplikační API. Operační systém je odpovědný za to, aby byly uživatelské zprávy převedeny na funkce ovladače. CAN ovladač je implementován jako znakové zařízení se standardními jmény uzlů /dev/can0, /dev/can1,… Aplikace komunikuje s ovladačem přes standardní nízko úrovňové I/O primitiva (open, close, read, write a ioctl).

3.1.1 Kompilace a Instalace

Díky tomu, že kompilace a instalace Lincanu probíhá pomocí OMK Make-Systému, je velice jednoduchá. Stačí stáhnout soubory z

[cvs] ocera.cvs.sourceforge.net/viewvc/ocera/ocera/components/comm/can/

Pokud není stažen celý adresářový OCERA strom, spustíme ve staženém adresáři (/can) skript ./switch2standalone.

V Makefile.omk nastavíme adresáře, které chceme zkompilovat a příkazem

make && sudo make install se zkompilují a nainstalují požadované adresáře.

Lincan lze zkompilovat klasicky bez použití OMK systému po spuštění skriptu v adresáři Lincanu ./switch-omk2std a pak klasickým *make*.

Samotný hardware přidáme příkazem insmod:

insmod can.o hw='your hardware' irq='irq number' io='io address' <options>

Po přidání hardwaru by se měly objevit zařízení v adresáři /dev (/dev/can0, /dev/can1, …)

3.2 SocketCAN

Socketcan je open source modul jádra, který implementuje sadu CAN ovladačů pro jádro linuxu původně vytvořených firmou Volkswagen Research. Je znám také jako nízkoúrovňový CAN Framework (LLCF).

Koncept SocketCANu rozšiřuje Berkley sockets API v Linuxu přidáním nové „protocol family“ PF_CAN, která koexistuje s ostatními protokoly jako PF_INET (Internetový protokol). Komunikace s CAN sběrnicí pak probíhá stejně jako při komunikaci s TCP/IP a to pomocí socketů.

3.2.1 Kompilace a Instalace

Zdrojové soubory lze získat z

```
svn://svn.berlios.de/socketcan/trunk
```

Dostanem se do adresáře zdrojových kódů a spustíme make

cd /socketcan/kernel/<verze kernelu>

make KERNELDIR=<zdrojové soubory kernelu>

Ovladač je dostupný pro dvě verze kernelu a to 2.4 a 2.6. Zdrojové soubory kernelu jsou obvykle v adresáři /usr/src/linux

K instalaci modulů je potřeba vytvořit adresář socketcan v /lib/modules/… , zkopírovat do něj moduly a spustit depmod(8) aby se zjistili závislosti nových modulů.

```
mkdir /lib/modules/$(uname -r)/socketcan
find -name \*.ko | xargs install -t /lib/modules/$(uname -r)/socketcan
depmod $(uname -r)
```

3.3 VCA_CANPING

VCA CANPING je program na měření času odezvy zpráv na CAN sběrnici. Program byl vytvořen Ing. Michalem Sojkou. Využívá VCA (Virtual CAN API) knihovnu, která mu umožňuje pracovat s více CAN ovladači najednou. Původně byl vytvořen pro testování ovladače Lincan, ale pan Ing. Píša doprogramoval i podporu pro SocketCAN.

Po spuštění programu se zadanými parametry (viz. 3.3.2) se vytvoří příslušný počet master a slave vláken, která spolu komunikují. Master vlákno vyšle zprávu (PING) s přiděleným identifikátorem (ID) a čeká na odpověď. Na tuto zprávu čeká slave vlákno, které po přijetí této zprávy vyšle zprávu (PONG) zpět s navýšeným ID o jedna. Čas mezi odesláním zprávy a přijetí odpovědi je zaznamenán a zpracován v programu. V závěru se vypíší pro každé ID minimální časy, maximální časy, průměr všech časů, směrodatná odchylka a počet nedoručených zpráv.

3.3.1 Kompilace programu

Program lze získat z git repozitářů:

http://rtime.felk.cvut.cz/gitweb/canping.git

Následná kompilace je pak velmi prostá. Vzhledem k tomu, že se program kompiluje pomocí OMK systému jako Lincan, stačí jen adresář s campingem zkopírovat do kořenového adresáře can od Lincanu (viz. výše). Poté se přidá adresář do Makefile.omk a config.omk se doplní o řádky:

CONFIG_OC_CANVCA=y

CONFIG_OC_SOLIBS=y

CONFIG_OC_CANVCA_IFC=socketcan (lincan nebo multi)

CONFIG_OC_CANVCA_IFC_lincan=y

CONFIG_OC_CANVCA_IFC_socketcan=y

U CONFIG_OC_CANVCA=… záleží na požadovaném ovladači.

Pokud chceme multi (podpora obou ovladačů najednou), necháme nastavení jak bylo popsáno. Při volbě pro jeden ovladač zakomentujeme řádek ovladače který nechceme. Poté stačí spustit *make* a výsledný program je v adresáři _compiled

Při změně podpory ovladačů je nutné smazat adresáře _build a _compiled nebo zavolat *make distclean*. Problém spočívá v tom, že v OMK není řešené vyjmutí objektových souborů z knihovny po tom, co je ze seznamu jejích zdrojových souborů podmínkou vyjmutý určitý zdrojový soubor.

3.3.2 Používání programu

Program se spouští s parametry:

-m *počet* vytvoří zadaný počet master vláken

-s *počet* vytvoří zadaný počet slave vláken

-c *počet* určuje kolik zpráv bude posláno každým master vlákenm

-d *zařízení* určuje pomocí jakého zařízení se bude komunikovat. Pokud se parametr nezadá bude program komunikovat pomocí /dev/can0

Lincan zařízení jsou */dev/can0-4* a SocketCAN zařízení *socketcan:can0-4*

-h vypíše help

-i *id* nastaví id první zprávy

-l *délka* nastaví délku zpráv (od 0 do 8)

-t *čas* čas vypršení zprávy (bez určení 4s)

-v *výpis* stavu programu na obrazovku. Pro detailnější výpis lze opakovaně napsat „v“ (-vv nebo -vvv)

-w *ms* kolik milisekund má vlákno počkat než vyšle další zprávu

-y synchronizuje vlákna před začátkem

-r spustí s real-time prioritou

-R *P:prio* spustí s real-time prioritou (RR nebo FF)

Příklad programu a jeho výpisu:

./vca_canping -m 10 -d /dev/can1 -c 1000 -t 1 -v -w 2

(souběžně s příkazem výše byl spuštěn i příkaz ./vca_canping -s 10 -d /dev/can0)

Výpis programu:

Total count: 10000, Timeouts: 0

Summary statistics:

Id 1000: count = 1000 mean = 590.69 stddev = 133.36 min = 367 max = 3671 [us] loss = 0% (0)

Id 1002: count = 1000 mean = 589.82 stddev = 124.26 min = 371 max = 3269 [us] loss = 0% (0)

Id 1004: count = 1000 mean = 589.81 stddev = 120.71 min = 391 max = 2914 [us] loss = 0% (0)

Id 1006: count = 1000 mean = 603.53 stddev = 424.20 min = 393 max = 13694 [us] loss = 0% (0)

Id 1008: count = 1000 mean = 590.58 stddev = 118.32 min = 386 max = 3147 [us] loss = 0% (0)

Id 1010: count = 1000 mean = 593.64 stddev = 188.18 min = 378 max = 4622 [us] loss = 0% (0)

Id 1012: count = 1000 mean = 598.28 stddev = 258.14 min = 378 max = 7700 [us] loss = 0% (0)

Id 1014: count = 1000 mean = 609.78 stddev = 522.32 min = 391 max = 15276 [us] loss = 0% (0)

Id 1016: count = 1000 mean = 590.24 stddev = 133.34 min = 387 max = 3898 [us] loss = 0% (0)

Id 1018: count = 1000 mean = 603.39 stddev = 428.33 min = 365 max = 13827 [us] loss = 0% (0)

Vypsané položky z leva: Id, počet zpráv, průměrná hodnota, směrodatná odchylka, minimální čas, maximální čas, počet nepřijatých zpráv.

4 Testování ovladačů

4.1 Použitý hardware a software

4.1.1 Hardware

Parametry počítače, na kterém probíhaly testy, jsou: procesor Intel P4 2000MHz, 512 MB RAM, Intel 845G Chipset, integrovaná síťová karta Intel PRO/100 VE.

Testování probíhalo na CAN kartě Kvaser PCIcan. Karta má rozhraní PCI, je vybavena jedním D-SUB 25 konektorem a podporou čtyř CAN rozhraní Philips SJA1000. Podporuje rámce CAN 2.0 A i CAN 2.0 B. CAN oscilátor má frekvenci 16MHz.

4.1.2 Software

Pro testování byl použit Linux Ubuntu 8.10 Inerpid Ibex verze kernelu 2.6.27-9-generic a 2.6.27-3-rt a ovladače Lincan 0.3.4. a SocketCAN . Testy byly prováděny pomocí programu vca_canping.

4.2 Příprava měření

Nejdříve je za potřebí nahrát moduly pro používanou CAN kartu pro oba ovladače. Pro Lincan přidáme zařízení příkazem insmod

insmod lincan.ko hw=pcican-q io=1 baudrate=1000,1000,1000,1000

Virtuální zařízení Lincanu je přidáno příkazem

insmod lincan.ko hw=virtual io=0 baudrate=0

Po zadání jednoho z příkazů se zařízení nastaví a mělo by se objevit v adresáři /dev/

Pro SocketCAN je potřeba pomocí *modprobe* přidat moduly *can, can-dev, can-raw, sja1000* a *kvaser_pci*. Pro přidání virtuálního zařízeni je potřeba přidat modul *vcan*. V tuto chvíli by se měla zařízení can0-can3 popř. vcan objevit v adresáři /sys/class/net/. Ještě je potřeba nastavit bitrate. To uděláme příkazem

echo 1000000 > sys/class/net/canX/can_bittiming/bitrate a pak příkazem

ip link set canX up zařízení aktivujeme.

Virtuální zařízení aktivujeme příkazy:

ip link add dev vcan0 type vcan

ip link set up dev vcan0

4.3 Testy

Testování proběhlo z několika hledisek, a to doba odezvy na poslání zprávy při nezatíženém operačním systému, odezva při různých dobách mezi vysláním zprávy a odezva při zatíženém systému síťovou komunikací. Zatížení bylo způsobeno přenosem dat přes FTP, příkazem *ping -f* a *ping -fs 64000*. Testována byla také virtuální CAN zařízení, která oba ovladače nabízí. K testování byl použit jednoduchý skript.

Ukázka skriptu na měření odezvy na zprávu.

vca_canping -s 1 -R FF:60 -d /dev/can1 & *Spustí slave vlákno na pozadí*

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0 *Měřící master vlákno*

killall vca_canping *Ukončí běh slave vlákna v pozadí*

4.4 Výsledky testů

Byly zpracovány celkové výsledky jednotlivých testů (Summary statistics) a jednotlivé časy všech odezev k vytvoření kumulativního histogramu. Z hlediska Real-Time aplikací jsou důležité hlavně maximální hodnoty odezvy a jejich četnost.

V histogramu je zaznamenán celkový počet přijatých zpráv v logaritmickém měřítku, od kterého se postupně odečítá počet odezev přijatých v daný čas. Výsledkem je, že jsou dobře patrné maximální odezvy a jejich počet.

4.4.1 Odezva na odeslání zprávy bez zatížení systému

Byla změřena odezva na odeslání zprávy bez jakéhokoli zatížení systému.Všechny nedůležité části včetně síťového připojení byly vypnuty.

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 365.92 stddev = 2.12 min = 360 max = 446 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 374.94 stddev = 2.40 min = 369 max = 500 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

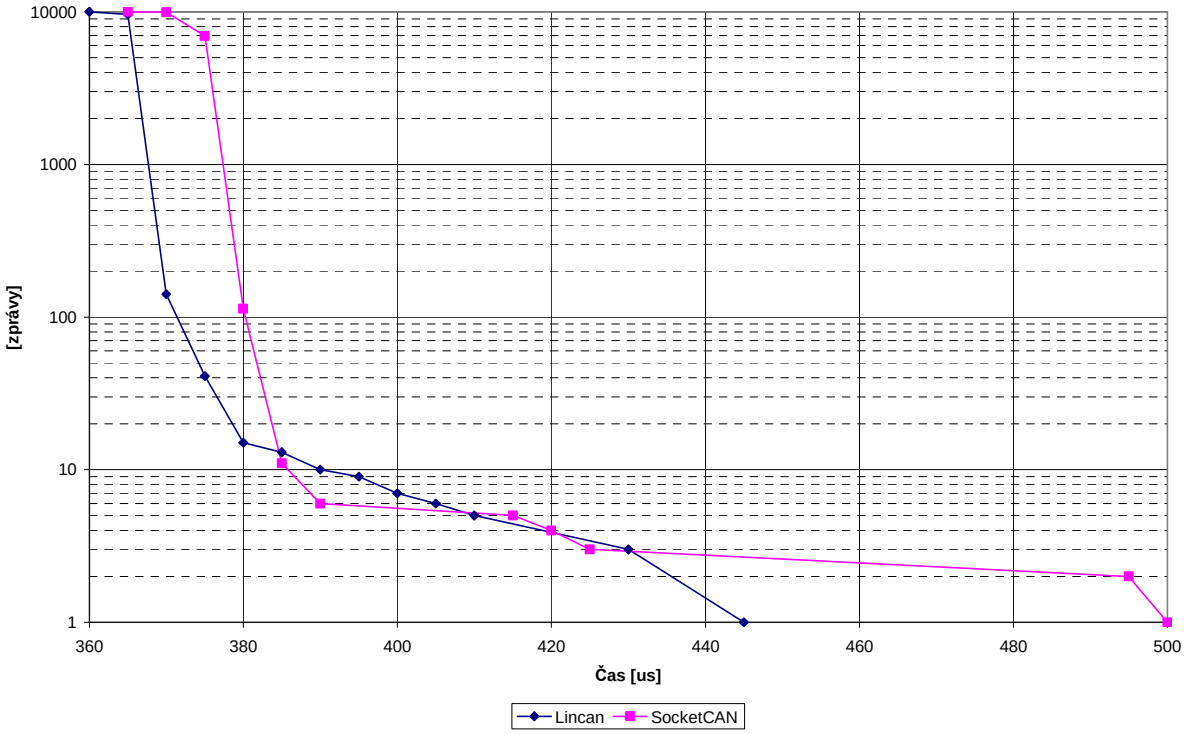
měření na LinCANu

Id 1000: count = 10000 mean = 354.02 stddev = 7.33 min = 350 max = 585 [us] loss = 0% (0)

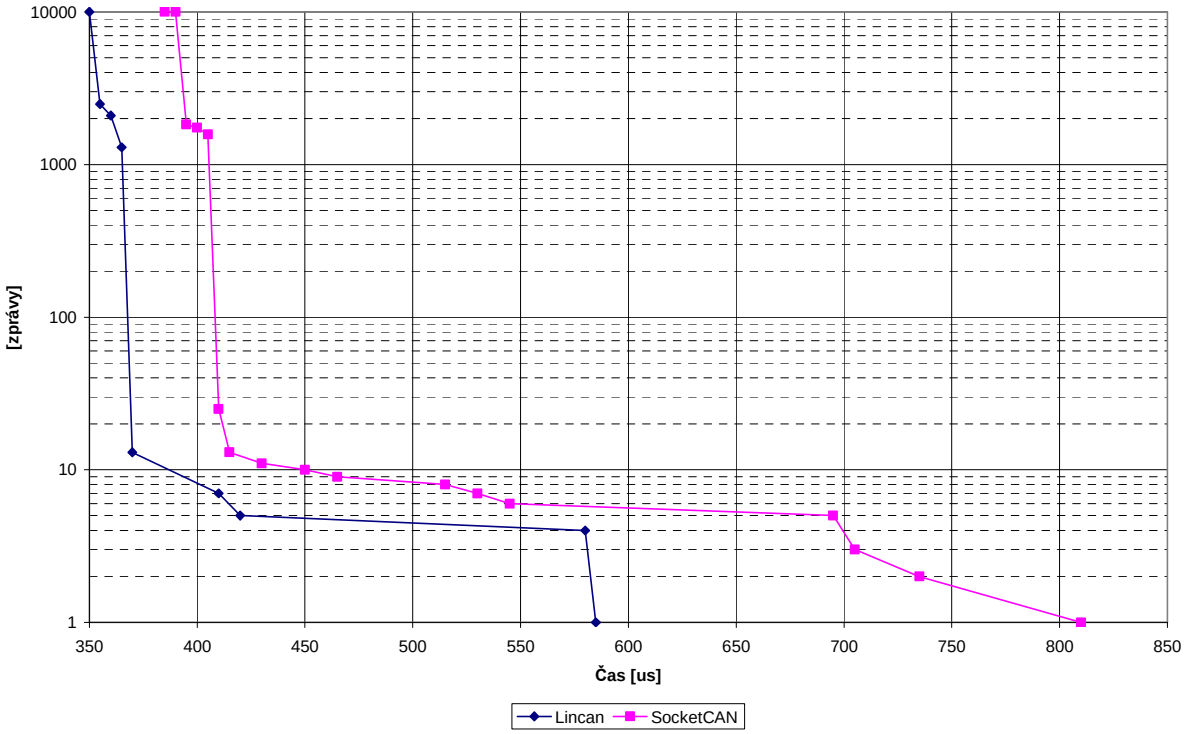
měření na SocketCANu

Id 1000: count = 10000 mean = 394.27 stddev = 9.91 min = 389 max = 811 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.1 Odezva na odeslání zprávy bez zatížení



Obrázek 4.2 Odezva na odeslání zprávy bez zatížení (Linux-RT)

Rozložení zpráv jsou u obou měření podobná. V obou měřeních má rychlejší odezvu Lincan.

4.4.2 Odezva na odeslání zprávy s rozdílnými časy mezi odesláním zprávy

Opět byly vypnuty nedůležité části systému a měřila se odezva s rozdílnými časy mezi odesíláním další zprávy po přijetí odezvy a to 0, 1 a 2 ms.

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 0/1/2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 366.40 stddev = 5.10 min = 360 max = 742 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 366.30 stddev = 5.02 min = 360 max = 700 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 366.49 stddev = 5.48 min = 360 max = 769 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 375.55 stddev = 5.52 min = 369 max = 781 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 373.95 stddev = 6.44 min = 367 max = 772 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 374.49 stddev = 3.84 min = 368 max = 680 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

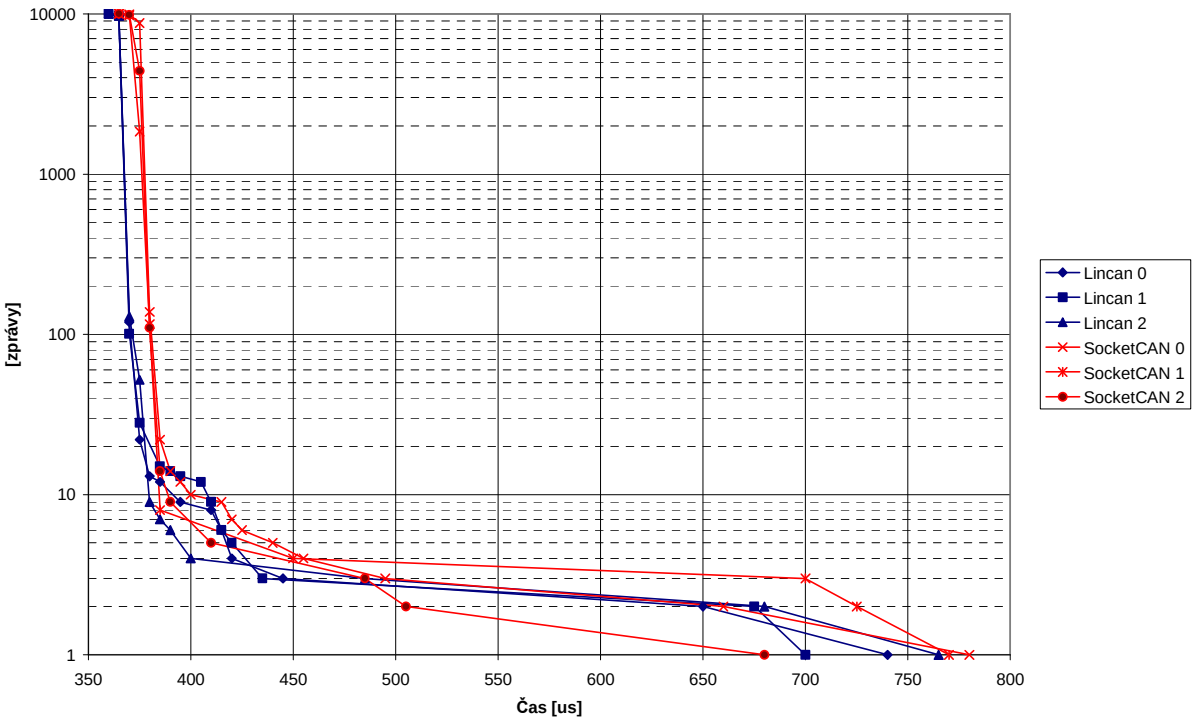
měření na LinCANu

Id 1000: count = 10000 mean = 351.55 stddev = 5.80 min = 348 max = 670 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 353.59 stddev = 6.26 min = 348 max = 586 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 354.11 stddev = 11.05 min = 349 max = 766 [us] loss = 0% (0)

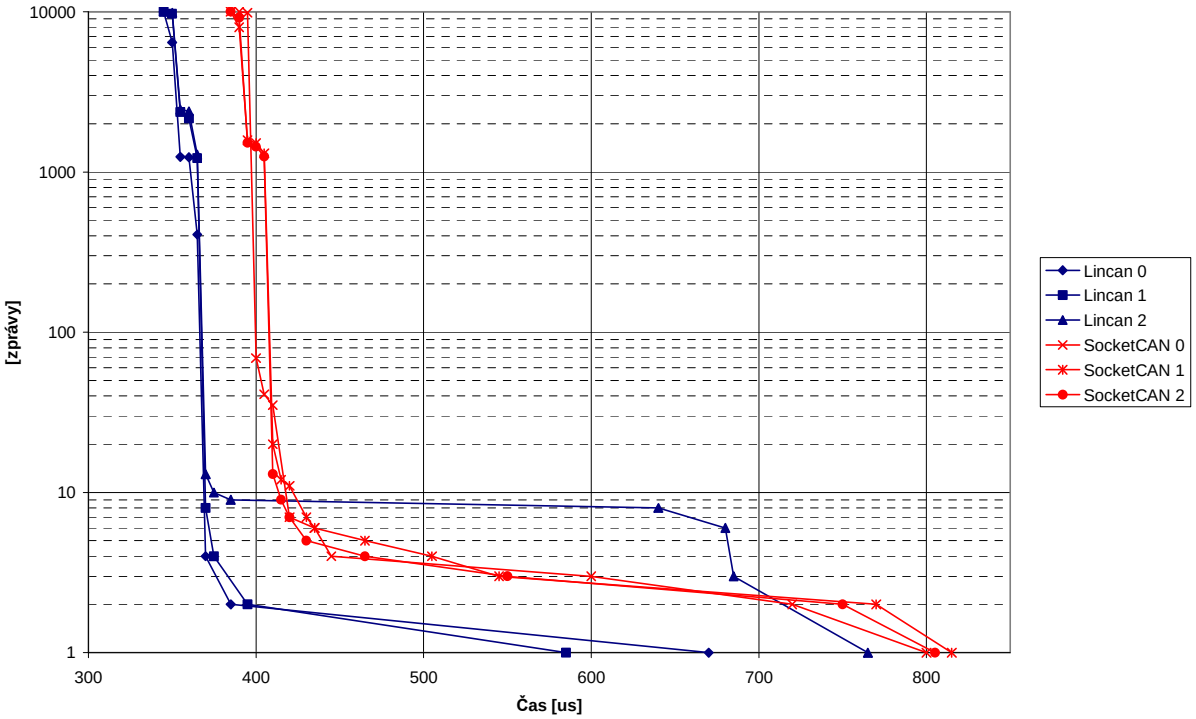
měření na SocketCANu

Id 1000: count = 10000 mean = 396.32 stddev = 5.82 min = 394 max = 802 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 392.82 stddev = 8.38 min = 388 max = 818 [us] loss = 0% (0)
Id 1000: count = 10000 mean = 392.84 stddev = 8.03 min = 388 max = 809 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.3 Odezva na odeslání zprávy s různými časy mezi odesláním zprávy



Obrázek 4.4 Odezva na odeslání zprávy s různými časy mezi odesláním zprávy (Linux-RT)

Rozložení zpráv je opět velmi podobné pro obě měření. Z grafů je patrné, že doba mezi odesláním zpráv nemá na odezvu žádný vliv, protože sběrnice není při žádném z případů přetížená. V obou měřeních byli časy Lincanu lepší.

4.4.3 Odezva na odeslání zprávy se síťovým zatížením

Odezvy byly měřeny se třemi různými způsoby síťového zatížení a to:

4.4.3.1 Zatížení příkazem „ping -f“

Parametr *-f* u příkazu *ping* znamená, že se vysílá jeden ping za druhým, hned jak je to možné, nebo 100x za vteřinu, podle toho co je pomalejší.

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 372.68 stddev = 10.31 min = 361 max = 593 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 382.20 stddev = 9.86 min = 368 max = 613 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

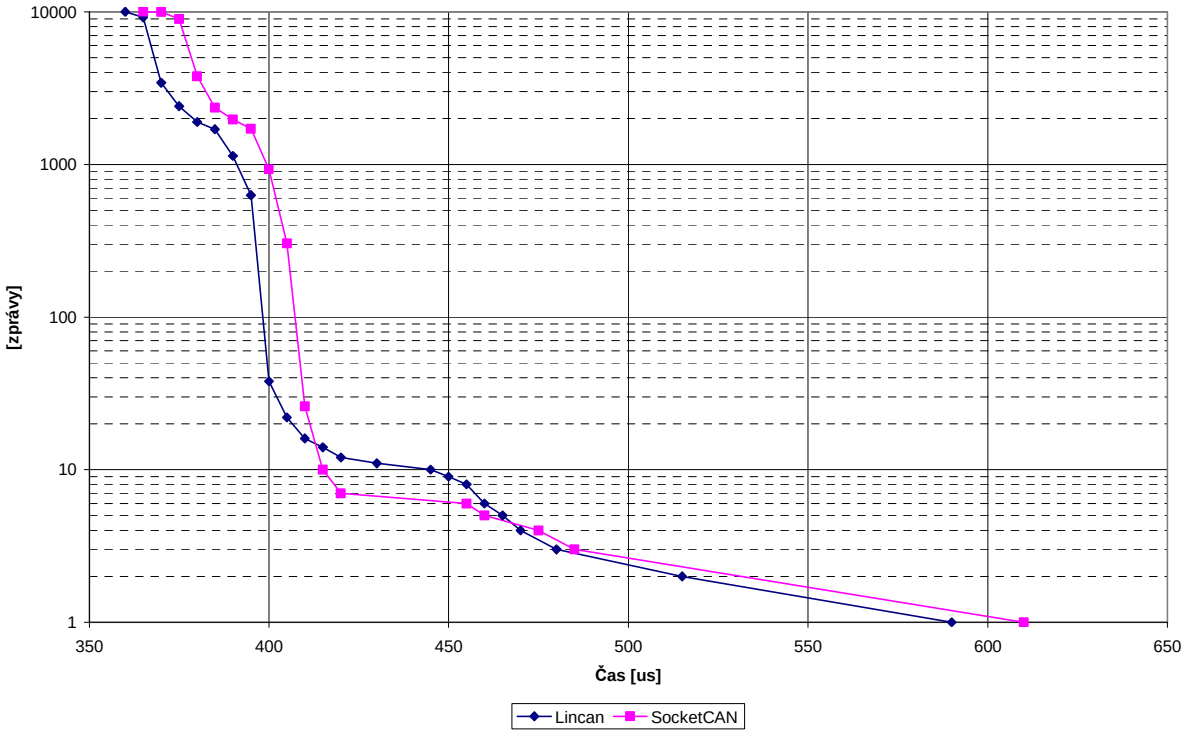
měření na LinCANu

Id 1000: count = 10000 mean = 355.15 stddev = 6.90 min = 349 max = 571 [us] loss = 0% (0)

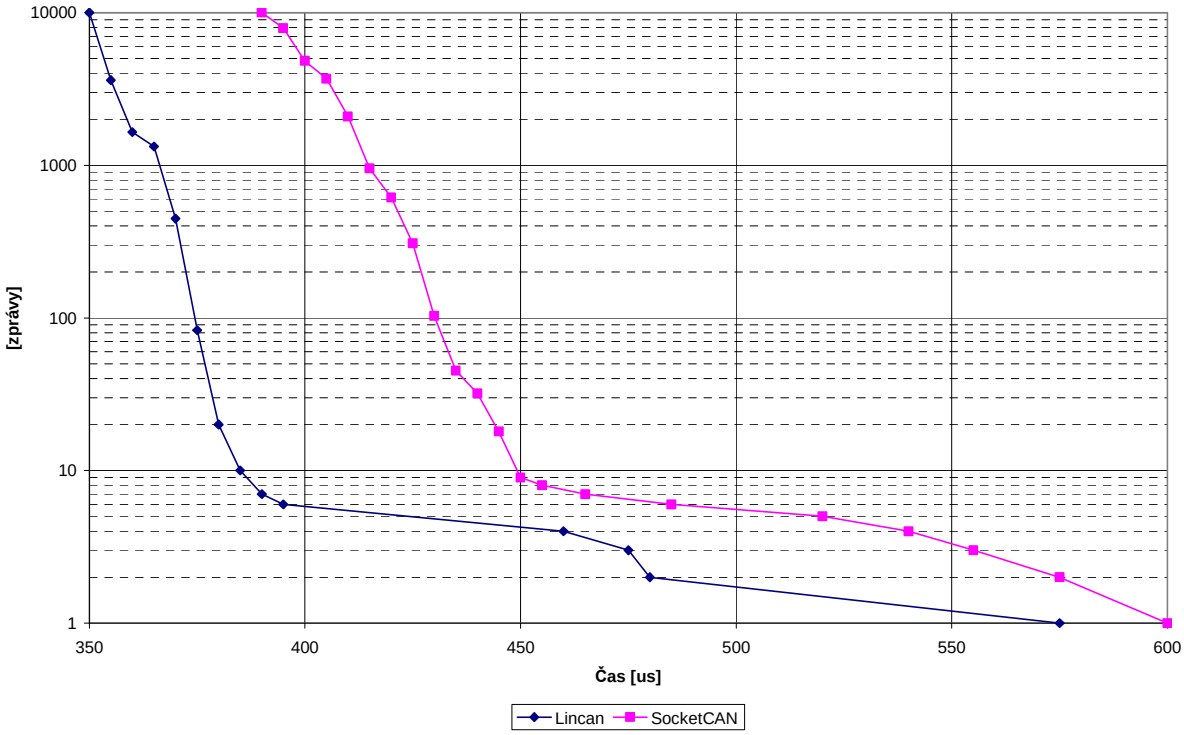
měření na SocketCANu

Id 1000: count = 10000 mean = 399.87 stddev = 10.33 min = 389 max = 655 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.5 Odezva na odeslání zprávy zatížená příkazem "ping -f"



Obrázek 4.6 Odezva na odeslání zprávy zatížená příkazem "ping -f" (Linux-RT)

Při měření na normálním jádru Linuxu jsou oba drivery zatíženy stejně, ale při měření na Linux-RT je patrné zhoršení časů u SocketCANu. Je to způsobeno tím, že SocketCAN pracuje se sockety stejně jako TCP/IP komunikace.

4.4.3.2 Zatížení příkazem „ping -fs 64000“

Parametr -*s* u příkazu *ping* určuje velikost odesílaného paketu. Normálně je velikost nastavena na 56 (64) data bytů.

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 393.05 stddev = 57.76 min = 361 max = 1532 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 404.88 stddev = 58.41 min = 369 max = 832 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

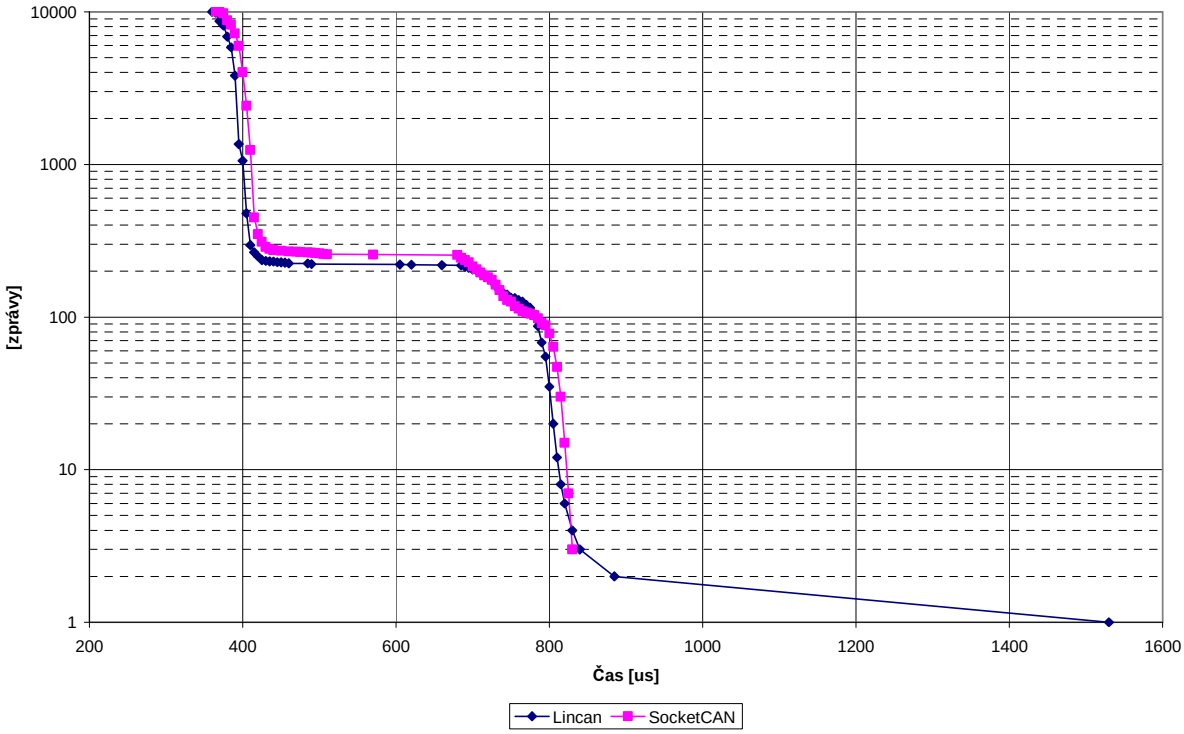
měření na LinCANu

Id 1000: count = 10000 mean = 381.35 stddev = 65.74 min = 349 max = 1082 [us] loss = 0% (0)

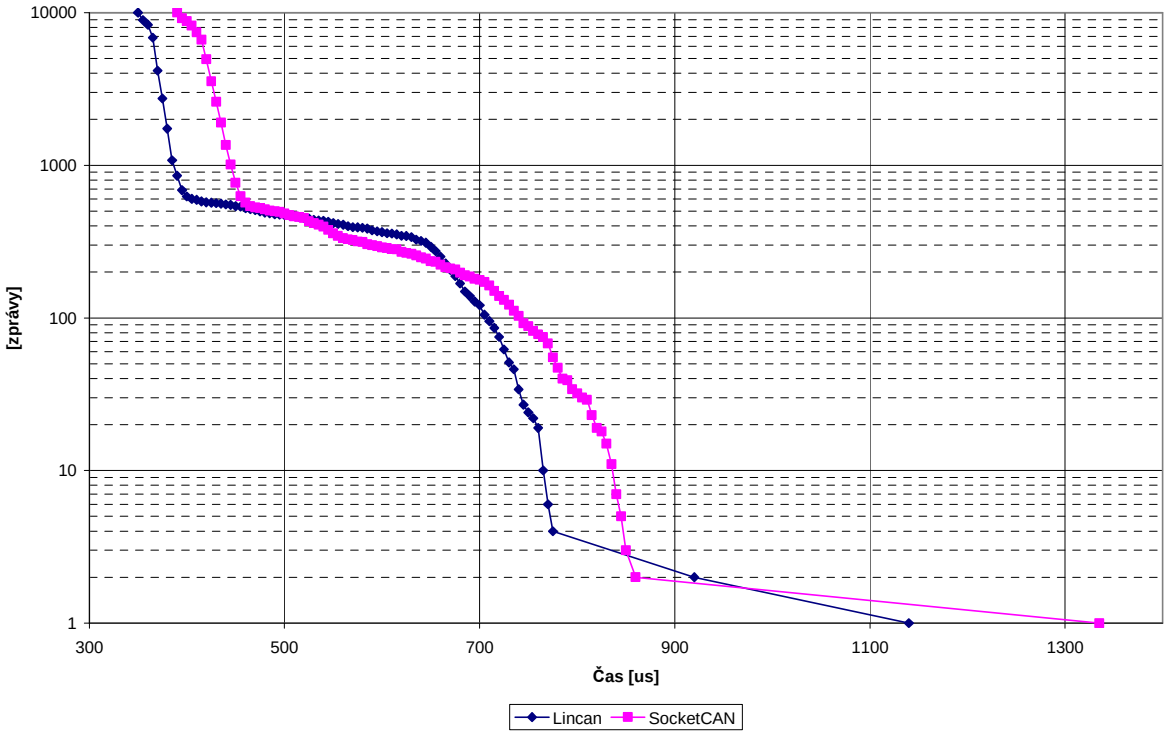
měření na SocketCANu

Id 1000: count = 10000 mean = 431.19 stddev = 63.68 min = 388 max = 1325 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.7 Odezva na odeslání zprávy se zatížením příkazem "ping -fs 64000"



Obrázek 4.8 Odezva na odeslání zprávy se zatížením příkazem "ping -fs 64000" (Linux-RT)

Při tomto měření měly oba ovladače podobné odezvy. Je to způsobeno velikostí odesílaných zpráv, které zatížily celý systém. Naměřené časy Lincanu byly opět lepší.

4.4.3.3 Zatížení stahováním dat z FTP

Z FTP byl stahován jeden 2GB soubor rychlostí 7-10 MB/s

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 404.50 stddev = 27.33 min = 361 max = 664 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 421.02 stddev = 27.83 min = 369 max = 843 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

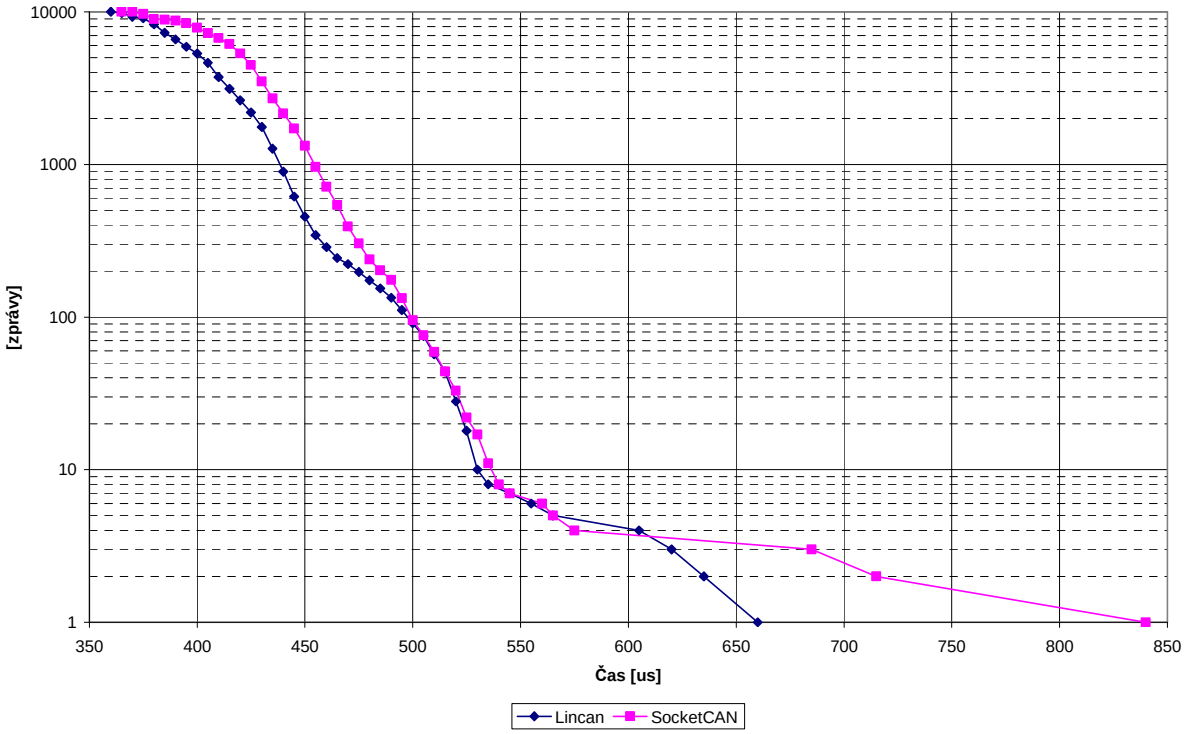
měření na LinCANu

Id 1000: count = 10000 mean = 376.70 stddev = 10.77 min = 350 max = 611 [us] loss = 0% (0)

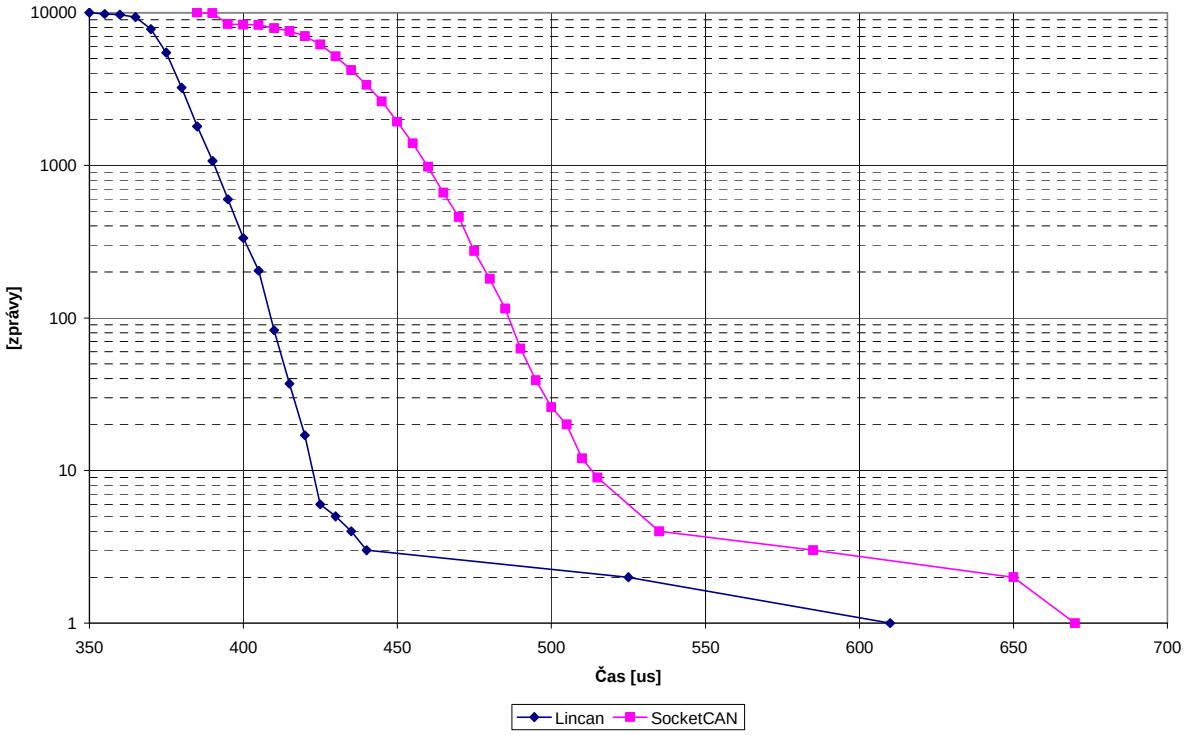
měření na SocketCANu

Id 1000: count = 10000 mean = 429.46 stddev = 24.15 min = 388 max = 674 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.9 Odezva na odeslání zprávy zatížená FTP přenosem



Obrázek 4.10 Odezva na odeslání zprávy zatížená FTP přenosem (Linux-RT)

Zatížení FTP přenosem mělo vliv na SocketCAN jako při zatížení příkazem *ping -f*. Důvod je opět stejný, je to způsobeno využíváním socketů jak u SocketCANu tak u FTP přenosu. Přenášení dat ale nebylo tak zásadní jako u příkazu *ping -fs 64000* a nezatížilo celý systém.

4.4.4 Odezva na odeslání zprávy na virtuálním zařízení

Byly změřeny odezvy pro obě virtuální zařízení.

Odezvy byly měřeny s parametry:

vca_canping -m 1 -R FF:60 -w 2 -c 10000 -t 1 -d /dev/can0

Celkové výsledky pro kernel **2.6.27-9-generic**:

měření na LinCANu

Id 1000: count = 10000 mean = 27.17 stddev = 1.19 min = 26 max = 96 [us] loss = 0% (0)

měření na SocketCANu

Id 1000: count = 10000 mean = 6908.07 stddev = 676.12 min = 122 max = 7805 [us] loss = 0% (0)

pro kernel **2.6.27-3-rt**:

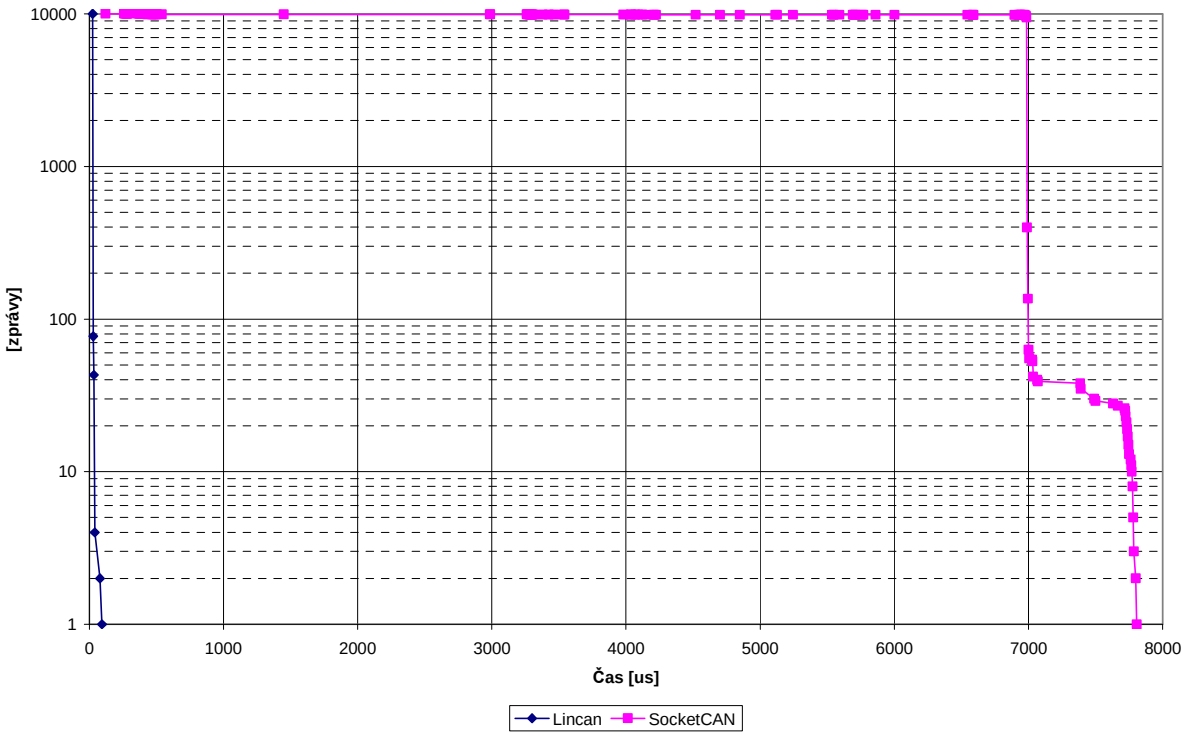
měření na LinCANu

Id 1000: count = 10000 mean = 19.90 stddev = 2.15 min = 19 max = 89 [us] loss = 0% (0)

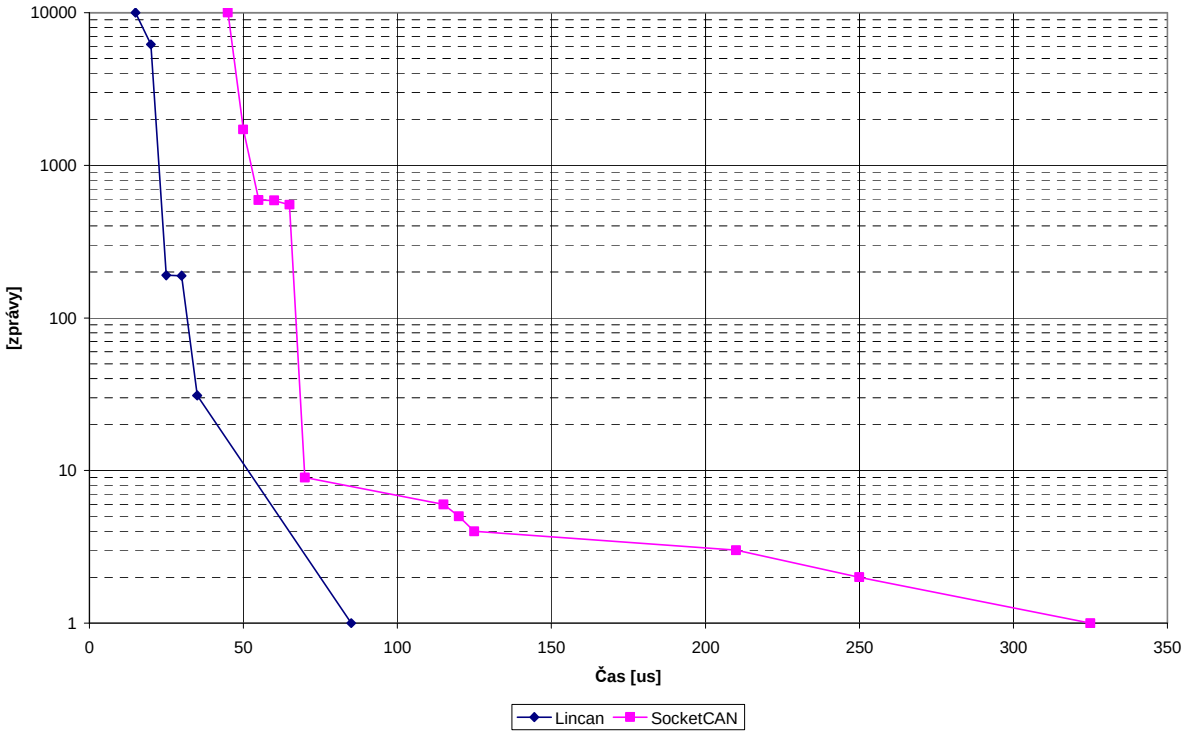
měření na SocketCANu

Id 1000: count = 10000 mean = 50.07 stddev = 5.84 min = 47 max = 329 [us] loss = 0% (0)

Kumulativní histogramy:



Obrázek 4.11 Odezva na odeslání zprávy na virtuálním zařízení



Obrázek 4.12 Odezva na odeslání zprávy na virtuálním zařízení (Linux-RT)

Měření na virtuálním zařízení SocketCANu na klasickém jádru Linuxu bylo zatíženo neznámou chybou. Při měření na Linux-RT vypadají výsledky podobně jako při měření při nezatíženém systému. U ovladače Lincan vypadala obě měření stejně.

5 Závěr

Výsledkem této práce je porovnání dvou ovladačů CAN sběrnice, a to ovladače Lincan a SocketCAN pomocí programu vca_canping. Tvorba programu pro testování ovladačů byla obtížnější a časově náročnější než jsem očekával. Pomohl mi Ing. Píša, který upravil program vca_canping i pro testování ovladače SocketCAN. Výsledky testů obou ovladačů byly naměřeny a zdokumentovány. Z celkového hlediska má lepší výsledky ovladač Lincan. Odezva při použití Lincanu se zhoršila jen při zatížení celého systému a i tak dosahoval lepších časů než SocketCAN. U SocketCANu byly výsledky výrazně ovlivněny zatížením systému síťovou komunikací, což bylo očekávané. Ovladače by se jistě ještě daly testovat i mnoha jinými způsoby, jako je například přetížení sběrnice. O tento test jsem se pokoušel, ale při spuštění více jak cca 60 vláken (30 master a 30 slave) docházelo k potížím s nedostatečnou pamětí pro běh programu.

6 Bibliografie

1. **Doc. Ing. Petr Kocourek, CSc., Ing. Jiří Novák, Ph.D.** *Přenos Informace,* ČVUT v Praze 2006
2. **Ing. Pavel Píša,** *Linux / RT-Linux CAN Driver (LinCAN)*

(http://www.ocera.org/archive/ctu/public/components/lincan/lincan-0.3.3/doc)

3. **OCERA Project,** *http://www.ocera.org*
4. **SocketCAN documentation**

(https://lists.berlios.de/pipemail/socketcan-core/2009-February/002253.html)

5. **Berlios.de,** *http://berlios.de*
6. **CAN a SocketCAN** *en.wikipedia.org*, *cs.wikipedia.org*

(http://en.wikipedia.org/wiki/Controller-area_network)

(http://en.wikipedia.org/wiki/Socketcan)

(http://cs.wikipedia.org/wiki/CAN-BUS)

7. **vca_canping** *rtime.felk.cvut.cz*

http://rtime.felk.cvut.cz/gitweb/canping.git

7 Seznam obrázků