VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MĚŘENÍ ZATÍŽENÍ SYSTÉMU V EMBEDDED LINUXU

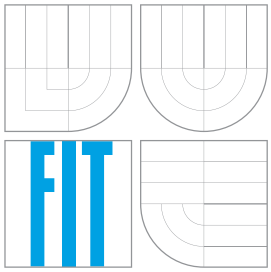
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB SKOPAL

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MĚŘENÍ ZATÍŽENÍ SYSTÉMU V EMBEDDED LINUXU

MEASURING OF SYSTEM LOAD IN EMBEDDED LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB SKOPAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN VIKTORIN
SUPERVISOR

BRNO 2015

Abstrakt

Práce popisuje veličiny, které charakterizují zatížení vestavěných systémů postaveném na Linuxovém jádře. Pro sledování zatížení systému byla navržena a implementována knihovna, která usnadňuje a sjednocuje sledování jednotlivých veličin a jejich vzájemných souvislostí. Knihovna nemá žádné externí závislosti, díky čemuž má nízkou paměťovou náročnost. Pro demonstraci využití implementované knihovny byl připraven program, který vykresluje grafy měřených veličin v reálném čase na vzdáleném PC.

Abstract

The bachelor thesis describes the quantities which characterize the load of the embedded systems based on the Linux kernel. The library was designed and implemented to observe the load of the system in order to facilitate and unify the observation of individual quantities and their reciprocal coherence. Due to the lack of external dependencies of the library the memory requirements are low. To demonstrate the use of the implemented library, a program was designed which depicts the graphs of measured quantities in real time on a remote computer.

Klíčová slova

Linux, vestavěné systémy, meření zatížení, měření spotřeby

Keywords

Linux, embedded systems, load measurement, power consumption measurement

Citace

Jakub Skopal: Měření zatížení systému v Embedded Linuxu, bakalářská práce, Brno, FIT VUT v Brně, 2015

Měření zatížení systému v Embedded Linuxu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Viktorina. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Skopal
20. května 2015

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Janu Viktorinovi za trpělivost a cenné rady.

Obsah

Kapitola 1

Úvod

Vestavěné systémy jsou nejrozšířenějšími počítačovými systémy na světě. Tyto systémy jsou například v bankomatech, kalkulačkách, mobilních telefonech atd. U těchto zařízení je nutné řešit otázku spotřeby, protože často pracují nepřetržitě a většinou jsou napájeny bateriemi.

Základem snižování spotřeby je vhodné zacházení s dostupnými výpočetními prostředky, například vypínání nepotřebných částí systému, dynamická změna frekvence procesoru, optimalizace operačního systému a jádra, snížení napětí na některých částech systému (například procesor, Wi-Fi čip), optimální rozložení úloh atd.

Aby bylo možné přizpůsobovat tyto systémy k těmto účelům, potřebujeme měřit různé parametry zatížení systému, na základě kterých lze rozhodovat o využití výpočetních prostředků. U systémů osazených čipem FPGA lze navíc dynamicky přesouvat některé výpočty z CPU do FPGA.

Cílem této práce je prostudování a seznámení se s možnostmi, které lze využít k monitorování zátěže těchto systémů. Dále tyto znalosti využít k navrhnutí a implementování knihovny, která svým rozhraním umožňuje uživatelským aplikacím monitorovat zátěž systému pomocí měřicích modulů. Knihovnu je možné rozšiřovat pomocí měřicích a výstupních modulů.

Kapitola 2

Vestavěné systémy

Vestavěné systémy, neboli embedded systémy, jsou zařízení, která obsahují software a hardware ve velmi těsném spojení. Většinou jsou tyto systémy navrženy k řešení konkrétních úloh tak, aby interakce s uživatelem byla minimální nebo žádná.

Systémy tohoto typu interagují s procesy nebo prostředím. Rozhodnutí, která jsou vytvářená v reálném čase za chodu systému, jsou závislá na vstupních informacích systému. Jsou reaktivní a zpracovávají vstupní informace v reálném čase.

Oproti osobním počítačům mají tyto systémy značně omezenou kapacitu paměti a diskového prostoru, výpočetní sílu a také se u těchto systémů předpokládá nízká spotřeba i nízká cena. [29]

2.1 Procesor ARM

První procesor ARM, který byl vyvinut v roce 1983 až 1985, využíval architekturu typu RISC, která vznikla přibližně v roce 1980. Tento procesor byl prvním procesorem tohoto typu určený pro komerční účely. Po vzniku společnosti ARM Limited byly procesory ARM považovány za lídra na trhu pro vestavěné systémy.

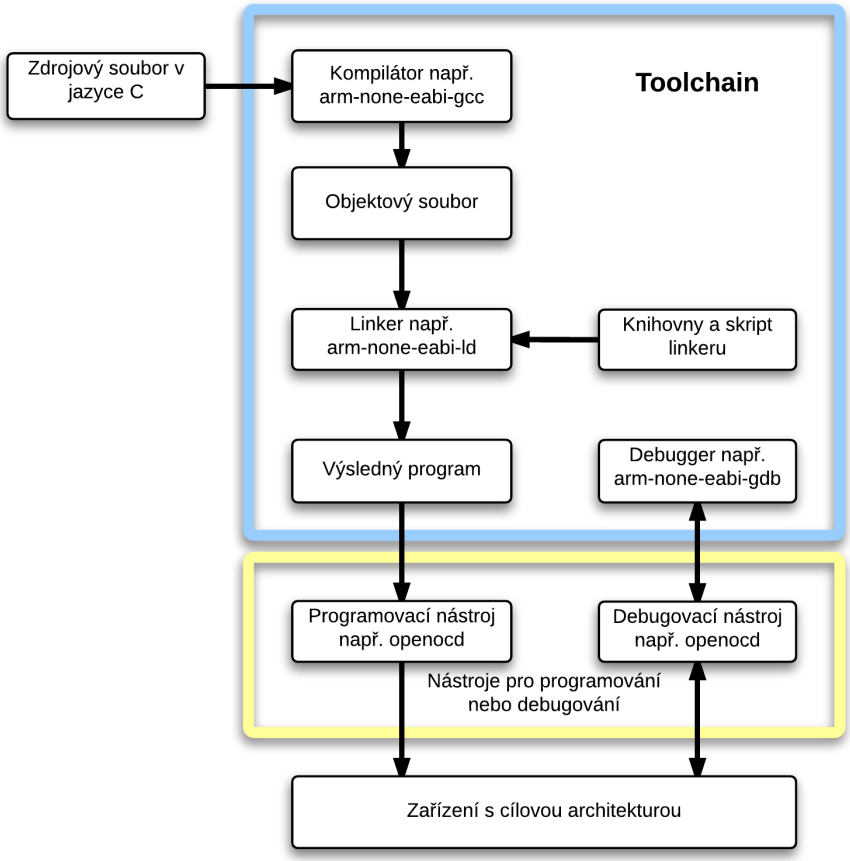
RISC I, II a Stanford MIPS byly v době vývoje prvního procesoru ARM jedinými příklady architektury pro procesory s redukovanou instrukční sadou Berkeley. Do architektury ARM je tedy zahrnuto několik vlastností architektury Berkeley RISC, zejména:

- Load and Store architektura
- 32 bitové instrukce pevné délky
- Tříadresní formát instrukcí [25]

Pro vývoj softwaru může být použito vývojové prostředí, které je vyvíjeno firmou ARM Limited (DS-5) nebo mnoho nástrojů třetích stran. Jelikož mnoho vestavěných systémů má omezené výpočetní zdroje, nelze na nich vytvořit prostředí pro vývoj softwaru. Aplikace na architekturu ARM jsou tedy vyvíjeny na jiné architektuře, například na osobním počítači x64 s operačním systém typu Unix nebo Windows. Tento proces se nazývá křížový vývoj (cross development). Poté se napsaný program zkompiluje na cílovou architekturu. [25]

2.2 Překladový systém aplikace na architekturu ARM

Kompilaci aplikace na jinou architekturu je možné provést pomocí toolchainů, které obsahují kompilátor, assembler, linker, debugger, knihovny a pomocné nástroje. Jeho úkolem je zkompilování kódu tak, aby výsledný program byl spustitelný na cílové architektuře. K toolchainu je možno použít externí nástroje například nástroj, který zajistí přesun zkompilované aplikace do cílového zařízení nebo nástroj pro debuggování cílové aplikace. [14] Struktura toolchainu je zobrazena na obrázku 2.1.



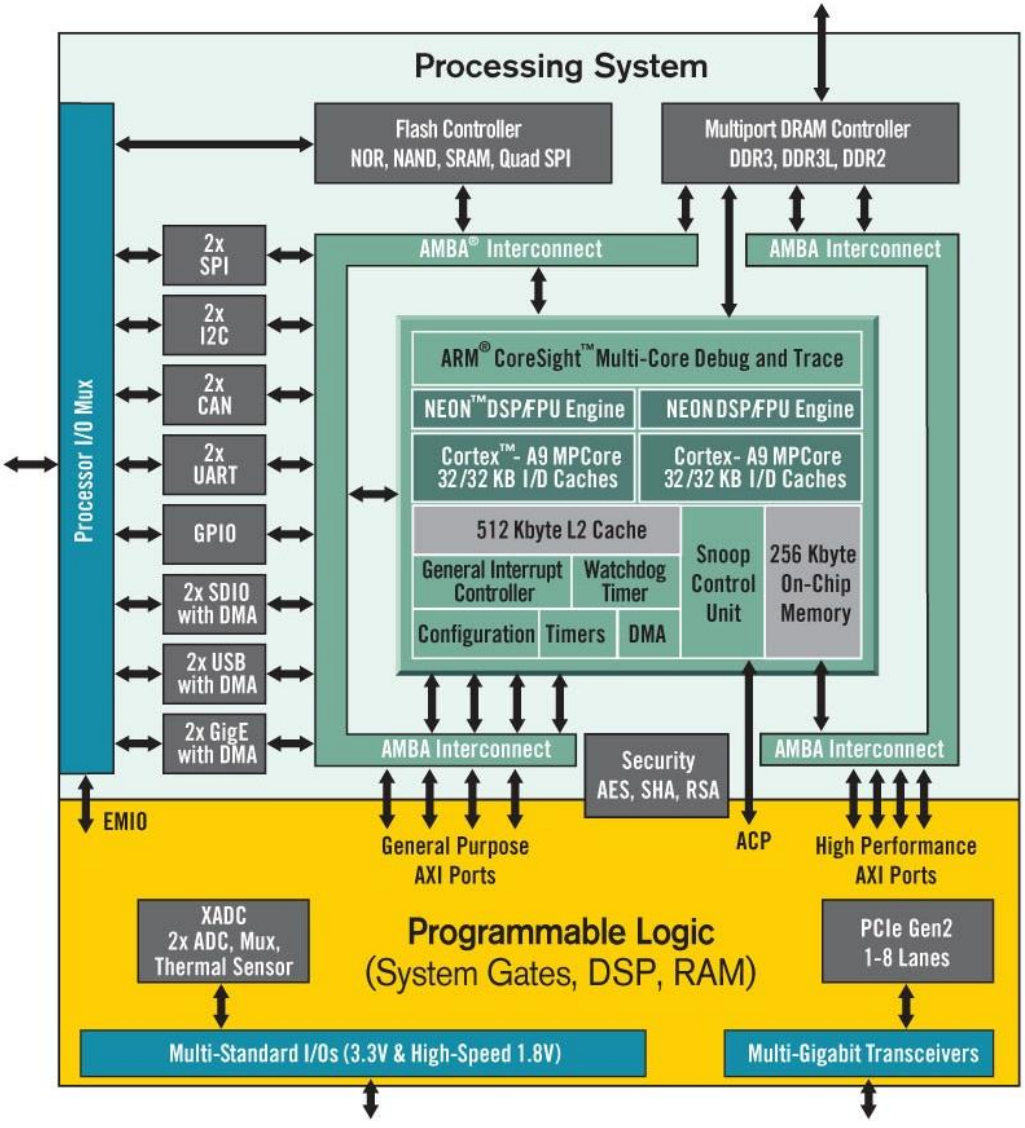
Obrázek 2.1: Struktura toolchainu [14]

Faktory, na kterých závisí výběr toolchainu:

- Architektura cílového zařízení (ARM, x86, PowerPC).
- Zda má cílové zařízení operační systém.
- Pro jaký operační systém bude kód určen (Linux, BSD, Windows).
- Osobní preference (grafické uživatelské prostředí toolchainu, integrované vývojové prostředí v toolchainu atd.). [14]

2.3 Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit

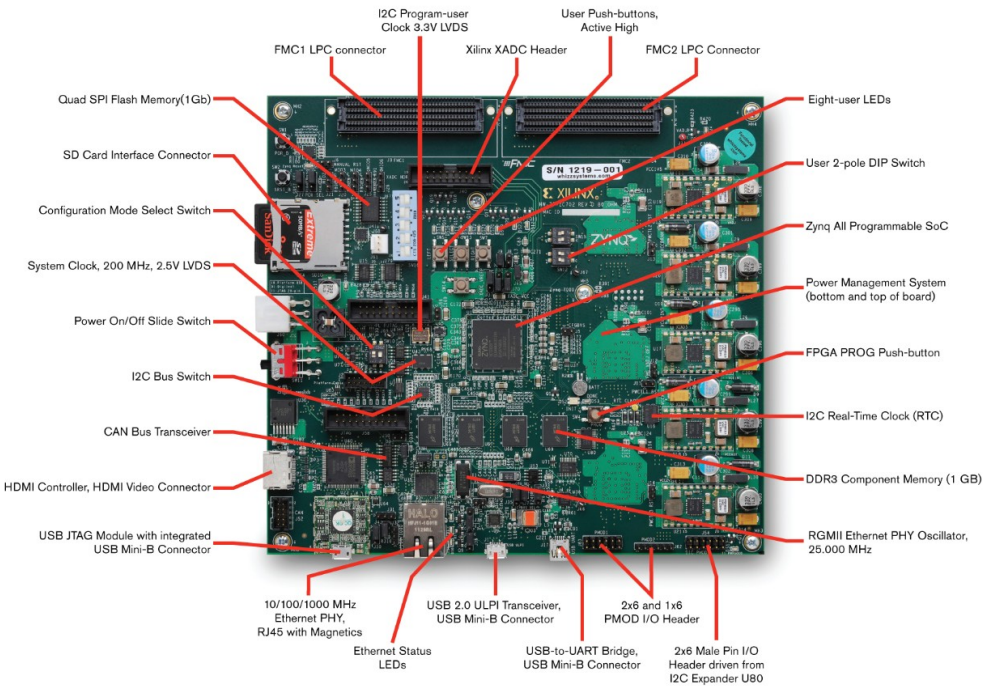
ZC702 je jedna z vývojových desek, které jsou založeny na SoC Zynq-7000. SoC Zynq-7000 se skládá ze dvou částí: z PS (procesorového systému) obsahující dvou jádrový mikroprocesor ARM Cortex-A9 MPCore, který má maximální taktovací frekvenci stanovenou na 1 GHz, a vnitřního komunikačního systému, který je založen na AMBA AXI. Vnitřní komunikační systém umožnuje spolupráci pamětí na čipu s externími SDRAM nebo FLASH kontroléry. Tento procesorový systém je připojen ke Zynq-7000 All Programmable SoC Programmable Logic (PL) pomocí devíti AMBA AXI portů a jednoho vstupně výstupního přepínatelného portu. [15]



Obrázek 2.2: Diagram procesorového systému Zynq-7000 [15]

2.4 Periferie vývojového kitu ZC702

Vývojový kit disponuje 1 KB IIC EEPROM, 1 GB Quad SPI flash pamětí a DDR 3 pamětí o velikosti 1 GB, která podporuje 32bitovou datovou sběrnici. Dále obsahuje gigabitový ethernet, rozhraní pro připojení OTG zařízení a rozhraní pro komunikaci pomocí UART přes USB. Pro grafický výstup lze využít HDMI port nebo 8 LED diod. ZC702 také obsahuje 3 mechanická tlačítka a 2 uživatelské přepínače. [11]



Obrázek 2.3: Náhled na vývojovou desku ZC702 [11]

Deska ZC702 obsahuje celkem 5 spínaných zdrojů a 1 lineární zdroj, které generují napětí potřebné pro procesorový systém Zynq-7000 a všechny ostatní komponenty, které jsou na desce ZC702. Napětí a proud dodávaný těmito zdroji jsou měřeny a monitorovány třemi čipy UCD9248 dostupnými na desce ZC702. [8]

Čipy UCD9248 vyhovují standardu PMBus specifikaci 1.1. Změřené proudy a napětí lze monitorovat pomocí připojení PMBus propojovacího kabelu. [8] PMBus je standard pro komunikaci s napěťovými a proudovými zdroji přes digitální komunikační sběrnici. [1]

2.5 Linux

Linux patří mezi Unix-like operační systémy. Linux byl původně vyvinut Linusem Torvaldem v roce 1991 jako operační systém pro personální počítače firmy IBM založené na mikroprocesoru Intel 80386. Linus zůstává zapojen do vývoje Linuxu. Vývojáři během několika let zpřístupnili Linux na ostatní architektury jako například MIPS, PowerPC a SPARC. Zdrojový kód Linuxu je publikován pod licencí GNU GPL a je tedy možné získat kompletní plnohodnotný operační systém postavený nad Linuxem zdarma. [23] Další výhodou Linuxu je jeho schopnost běžet s grafickým prostředím nebo i bez něj. [24]

V průběhu let se objevilo mnoho distribucí Linuxu. [24] Distribuce jsou sestavovány jednotlivci, týmy dobrovolníků, ale i komerčními firmami. Distribuce zahrnuje jádro, další systémový a aplikační software, grafické uživatelské rozhraní (X.org, KDE, GNOME atd.). Distribuce mají různá zaměření, například výběr obsažených programů, podpora určité počítačové architektury, použití ve vestavěných systémech atd. Mezi nejznámější distribuce patří Debian, Gentoo, Red Hat, Slackware a SUSE. [34]

2.6 Vestavěný Linux

Vestavěný Linux obvykle označuje kompletní linuxovou distribuci, která je cílená na vestavěná zařízení. Existují i linuxová jádra, která jsou určena pro speciální typy vestavěných zařízení například uCLinux (you-see-Linux). uCLinux dokáže na rozdíl od neupraveného Linuxového jádra běžet na procesorech bez MMU jednotky. Linuxové jádro může být zkompilováno pro velkou škálu zařízení, pro které umožňuje konfiguraci různých volitelných vlastností linuxového jádra. [40]

Vestavěné systémy s linuxovým jádrem se liší zejména použitou knihovnou jazyka C (musl, uClibc, glibc, dietlibc), což může mít značný vliv na velikost a výkon výsledného softwaru.

2.6.1 Distribuce pro vestavěná zařízení

Níže jsou uvedeny známé linuxové distribuce, které jsou vytvořeny pro vestavěná zařízení.

- Embedded Debian je výrazně minimalizovaná hlavní distribuce Debian. [16]
- Buildroot je sada Makefilů a záplat, které usnadňují generování kompletního vestavěného linuxu. Buildroot může generovat toolchainy, které jsou určené pro křížové kompilování, souborový systém root, obraz linuxového jádra a obraz bootloaderu. [28]
- Windriver je komerční Linux, který je optimalizovaný pro vestavěná zařízení. Tento Linux poskytuje out-of-the-box konfiguraci (po instalaci Linuxu není potřeba větších zásahů do konfigurace operačního systému), která usnadňuje programátorům tvořit a sestavovat jejich první projekty a aplikace. [12]

2.7 Služby operačního systému Linux

Služba operačního systému je proces operačního systému Linux, který běží na pozadí systému (daemon) a není pod přímou kontrolou uživatele. Linuxové operační systémy typicky používají několik služeb, většinou k reagování na události v systému, naslouchání na TCP/IP portech, monitorování stavu systému atd. [2]

Služby operačního systému Linux lze rozpoznat tak, že jejich rodičovský proces je proces init, který má PID 1. Init je vždy první proces, který je spuštěn, při nastartování počítače s linuxovým operačním systémem. Init zůstává zapnutý po celou dobu používání operačního systému. Init adoptuje každý proces, jehož otcovský proces byl ukončen. [2]

Vytvoření systémové služby se skládá z několika kroků:

- Vytvoření procesu běžícího na pozadí.
- Nastavení umask (file mode mask) pro povolení zápisu a čtení dat do souborů, které jsou vytvořeny touto službou.
- Nastavení logování pro zaznamenávání chyb, které vznikly při běhu služby.
- Vytvoření unikátního SID (session ID), aby linuxové jádro neoznačilo službu jako proces, který nemá žádného rodiče.
- Nastavení pracovního adresáře.
- Uzavření STDIN STDOUT a STDERR. Jelikož služba nemůže využívat terminál, jsou tyto výstupy redundantní a mohou být uzavřeny. [20]

Ukázka kódu pro vytvoření systémové služby

```
1   pid = fork();
2   if (pid < 0)
3   {
4       printf("Fork Error\n");
5       exit(EXIT_FAILURE);
6   }
7   if (pid > 0)
8   {
9       exit(EXIT_SUCCESS);
10  }
11
12  umask(0);
13  sid = setsid();
14  if (sid < 0)
15  {
16      printf("SETSID fail\n"); //
17      exit(EXIT_FAILURE);
18  }
19
20  if ((chdir(PATH)) < 0)
21  {
22      printf("CHDIR fail\n");
23      exit(EXIT_FAILURE);
24  }
25
26  close(STDIN_FILENO);
27  close(STDOUT_FILENO);
28  close(STDERR_FILENO);
29  // Kód, který bude vykonávat služba
```

Ukázka kódu 2.1: Vytvoření systémové služby

Kapitola 3

Možnosti sběru informací

Ke sběru informací o vestavěném zařízení nebo o operačním systému je možné využít virtuální souborové systémy procfs a sysfs a hardwarové senzory, které umožňují monitorování spotřeby a teploty komponent vestavěného zařízení. Z výsledků měření lze zjistit mnoho nejrůznějších informací, jako například množství nevyužité paměti RAM, počet přenesených paketů a dat atd.

3.1 Souborový systém procfs

Souborový systém procfs je virtuální souborový systém, který umožňuje získávat informace o vnitřním stavu systému. Tyto informace mohou být získány ze souborů umístěných v souborovém systému procfs, který je typicky namapován do adresáře */proc*, aniž by uživatel potřeboval komplikované nástroje, například nástroj cat je plně dostačující. Tyto informace mohou být přečteny z linuxového jádra, ale také mohou být do linuxového jádra zapsány pomocí zápisu informací do požadovaného souboru v souborovém systému procfs, například *echo hodnota >/proc/file*. Informace v požadovaném souboru jsou vygenerovány jen v případě, že je na tento soubor vytvořen požadavek pro čtení jeho obsahu. Celý souborový systém procfs je umístěn v paměti operačního systému. [31]

Používáním souborového systému procfs mohou být získány informace o nastavení linuxového jádra, ale také můžou být tyto hodnoty změněny, aniž by bylo nutné překompilovávat linuxové jádro, znovu načítat moduly nebo restartovat zařízení. [31]

Souborový systém procfs obsahuje informace o:

- Správě paměti.
- Informacích o procesech.
- Souborových systémech.
- Systémové sběrnici.
- Správě napájení.
- Terminálech. [31]

3.1.1 Diskstats

Soubor */proc/diskstats* poskytuje vstupně výstupní statistiky blokových zařízení. Každý záznam obsahuje 14 hodnot. [30]

```
> cat /proc/diskstats
  31        0 mtdblock0 0 0 0 0 0 0 0 0 0 0 0
  179       0 mmcblk0 420 3 2462 1380 453 17 9098 45490 0 1060 46860
```

Sloupec	Obsahuje
1	Hlavní číslo
2	Vedlejší číslo
3	Jméno zařízení
4	Počet úspěšně dokončených čtecích operací
5	Počet spojených čtení
6	Počet přečtených sektorů
7	Čas strávený čtením dat [ms]
8	Počet dokončených zápisů dat
9	Počet spojených zápisů
10	Počet zapsaných sektorů
11	Čas strávený zápisem dat [ms]
12	Počet probíhajících I/O operací
13	Čas strávený prováděním I/O operací
14	Časově vážený průměr strávený prováděním I/O operací

Tabulka 3.1: Hodnoty diskstats [30]

Z těchto hodnot lze zjistit například následující údaje:

- Množství přečtených a zapsaných dat za jednotku času, a to odečtením hodnoty 6. (pro množství přečtených dat) nebo 10. (pro množství zapsaných dat) sloupce v čase t a v čase $t + k$, kde k je doba měření. Výsledkem rozdílu těchto hodnot je počet zapsaných nebo přečtených sektoru za čas k. Výsledný počet zapsaných nebo přečtených sektorů vynásobíme velikostí jednoho sektoru (obvykle 512 B) a získáme množství zapsaných nebo přečtených dat v jednotkách B.
- Čas strávený prováděním vstupně výstupních operací, čtením dat a zápisem dat.
- Typ zařízení, který se určí podle hlavního a vedlejšího čísla. Hlavní číslo určuje typ zařízení, například 8 je označení pro bloková SCSI zařízení a vedlejší číslo určuje konkrétní zařízení, například 0 je označení pro */dev/sda* a 1 pro */dev/sdb*. [22]
- Potenciální problémy s rychlostí zpracovávání vstupně výstupních operací, pozorováním hodnoty 7. a 11. sloupce. Jestliže je alespoň jedna z těchto hodnot rovna času měření, je využití blokového zařízení maximální a pravděpodobně existuje fronta vstupně výstupních operací, které čekají na zpracování.

3.1.2 Meminfo

Soubor */proc/meminfo* je jeden z nejvíce používaných v procfs. Poskytuje mnoho důležitých informací o operační paměti. Mnoho linuxových nástrojů (free, top a ps atd.) zpracovává informace právě z tohoto souboru. [36]

Tabulka 3.3 obsahuje pouze hodnoty relevantní pro tuto práci.

MemTotal	Celková velikost paměti RAM
MemFree	Velikost celkové nevyužité paměti RAM.
Buffers	Velikost paměti RAM, která je využita souborovými buffery
Cached	Velikost paměti RAM, která je využita jako vyrovnávací paměť
SwapCached	Velikost paměti swap, která je použita jako vyrovnávací paměť
Active	Celková velikost paměti, která je aktivně používána. Tato paměť byla nedávno použita a obvykle není alokována pro jiné účely
Inactive	Celková velikost paměti, která je volná a dostupná. Tato paměť nebyla nedávno použita a je obvykle alokována pro jiné účely
Active(file)	Velikost paměti, která obsahuje data souborového systému a jejich metadata [6]
Inactive(file)	Velikost paměti, která je k dispozici pro data souborového systému a jejich metadata [6]
SwapTotal	Celková velikost paměti swap
SwapFree	Celková velikost volné paměti swap
Dirty	Celková velikost paměti čekající na zapsání na disk
AnonPages	Celková velikost anonymous paměti [6]
Mapped	Celková velikost paměti, která byla použita k mapování souborů nebo knihoven pomocí mmap příkazu
Shmem	Celková velikost sdílené paměti [6]
Slab	Celková velikost paměti, která je alokována alokátorem slab
SReclaimable	Celková velikost paměti, která je alokována linuxovým jádrem a může být znova použita
KernelStack	Velikost paměti, která je použita jako zásobník linuxového jádra [6]
PageTables	Celková velikost paměti, která je potřebná pro udržení informací o všech stránkách paměti v systému
VmallocUsed	Celková velikost virtuální paměti, která je používána

Tabulka 3.2: Vysvětlení hodnot meminfo [36]

Z výše uvedených hodnot lze zjistit například:

- Jak velkou pamětí zařízení disponuje.
- Množství využité a nevyužité paměti RAM.
- Množství paměti, která je využívána jako vyrovnávací paměť.
- Možné úniky paměti (memory leaks). Úniky paměti linuxového jádra se dají detekovat pozorováním hodnot Slab a KernelStack. Pokud se tyto hodnoty s přibývajícím časem zvětšují, je pravděpodobné, že linuxové jádro způsobuje úniky paměti. Úniky paměti se dají pozorovat i u uživatelských aplikací a to pozorováním hodnot AnonPages, Mapped a Shmem.
- Množství dostupné paměti memAvailable. MemAvailable, narozdíl od MemFree, nepočítá s pamětí, která je využitá jako vyrovnávací paměť nebo využitá souborovými buffery. Jelikož je tento údaj přístupný pouze na zařízeních s jádrem vyšším než verze

3.14, je třeba tento údaj zvlášť dopočítat pro starší jádra. Nejprve se zjistí velikost jedné stránky PageSize. Poté se pomocí souboru */proc/zoneinfo* odečtou informace o zónách, konkrétně hranice minimálního počtu stránek alokovaných linuxovým jádrem v jednotlivých zónách. Počet stránek alokovaných linuxovým jádrem musí být tedy vyšší než stanovená hranice. Pokud je nižší, linuxové jádro začne alokovávat stránky, dokud počet alokovaných stránek není vyšší než stanovená hranice. [32]

Samotný memAvailable se vypočítá pomocí následujících vztahů:

$$FileCache = meminfo\_Active(file) + meminfo\_InActive(file) \tag{3.1}$$

$$PageCache = FileCache - MinPages \tag{3.2}$$

$$\begin{aligned} memAvailable = PageCache + (meminfo\_MemFree - MinPages) \\ +(meminfo\_SMemReclaimable - MinPages) \end{aligned} \tag{3.3}$$

FileCache je množství paměti RAM, která je k dispozici pro data souborového systému a jejich metadata.
MinPages označuje součet hranic jednotlivých zón.

3.1.3 Loadavg

Soubor */proc/loadavg* poskytuje informace o aktuálním průměrném zatížení systému, které je měřeno pomocí počtu běžících procesů a je vypočítáváno linuxovým jádrem periodicky každých 5 sekund. Běžící proces je proces, který je aktuálně vykonáván nebo je zařazen do fronty procesů, které jsou připraveny k běhu. [27]

Každé jádro procesoru může být obsazeno pouze jediným procesem a nacházet se v jednom ze dvou stavů: jádro je buď využíváno (obsazeno procesem) nebo nevyužíváno (není obsazeno procesem). [27]

```
> cat /proc/loadavg
  0.14 0.08 0.03 1/33 691
```

Soubor */proc/loadavg* poskytuje následující údaje:

- Průměrný počet běžících procesů za poslední minutu.
- Průměrný počet běžících procesů za posledních 5 minut.
- Průměrný počet běžících procesů za posledních 15 minut.
- První hodnota čtvrtého sloupce je okamžité zatížení systému, která nemůže být menší než 1, protože proces, který čte ze souboru */proc/loadavg* je považován za běžící proces. Okamžité zatížení systému je poměr počtu běžících procesů a počtu procesorů. Pokud je počet běžících úloh menší než počet dostupných jader procesorů, lze usoudit, že systém nevyužívá celý jeho výpočetní výkon a může být vykonáno více úloh k efektivnějšímu využití výpočetního výkonu. Pokud je počet aktivních úloh roven počtu procesorů, systém využívá svůj výpočetní výkon optimálně, tzn. každý proces má přiřazeno jedno jádro procesoru, na kterém běží po celý čas vykonávání úlohy. Pokud je počet běžících procesů větší než počet jader procesorů, systém nevyužívá optimálně svůj výpočetní výkon, tzn. úlohy potřebují více času k dokončení.
- Poslední PID, které bylo přiřazeno poslednímu vytvořenému procesu. [27]

3.1.4 Stat

Soubor */proc/stat* obsahuje informace o aktivitě linuxového jádra, o běžících a blokovaných procesech a o změnách kontextu. Dále lze z informací poskytnutých souborem */proc/stat* vypočítat celkové zatížení procesorů nebo jeho jader a detekovat možné problémy s rychlostí zpracovávaní vstupně výstupních operací pomocí hodnoty *iowait*. Informace v tomto souboru jsou sbírány od zapnutí systému. [37]

```
> cat /proc/stat
  cpu  161 0 1284 10381 0 0 0 0
  cpu0 161 0 1284 10381 0 0 0 0
  intr 106120 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11820 0 0
  0 0 6 0 0 71695 0 0 0 2340 20151 0 8 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  ctxt 7834
  procs_running 1
  procs_blocked 0
```

Řádky začínající slovem cpu až cpuN, kde N značí celkový počet procesorů, obsahují informace o tom, kolik času procesor vynaložil na řešení úloh. Používaná jednotka času je USER_HZ. [37] Ve většině architektur je obvykle USER_HZ setina sekundy. Pro získání hodnoty USER_HZ je možné zavolat funkci sysconf s parametrem _SC_CLK_TCK, jejíž návratová hodnota je velikost jednotky USER_HZ.

Vysvětlení sloupců řádku týkajícího se procesoru:

- Druhý sloupec je procesorový čas využitý uživatelskými procesy - user.
- Třetí sloupec je procesorový čas využitý uživatelskými procesy se zvýšenou prioritou - nice.
- Čtvrtý sloupec je procesorový čas využitý procesy linuxového jádra - system.
- Pátý sloupec je procesorový čas strávený v klidovém stavu - idle.
- Šestý sloupec je procesorový čas strávený čekáním na dokončení vstupně výstupních operací - iowait.
- Sedmý sloupec je procesorový čas strávený zpracováváním přerušení - irq.
- Osmý sloupec je procesorový čas strávený zpracováváním softwarových přerušení - softirq.
- Devátý sloupec je procesorový čas strávený vynuceným čekáním - steal. [39]

Další řádky poskytují následující údaje:

- intr udává celkový počet přijatých přerušení pro každé systémové přerušení. První hodnota řádku intr je počet všech přerušení.
- ctxt udává celkový počet změn kontextu všemi procesory.
- procs_running udává počet běžících procesů.
- procs_blocked udává počet procesů, které čekají na dokončení vstupně výstupních operací. [37]

Výpočet zatížení procesorů

Zatížení procesorů nebo jeho jader je vypočítáno na základě využití procesorového času. Výsledná hodnota nemůže být vyšší než 100%. Zatímco zatížení systému, které je poskytováno souborem */proc/loadavg*, je vypočítáváno na základě počtu procesů, které jsou buď běžící, nebo čekající na přidělení procesorového času.

Celkové zatížení procesoru nebo jeho jader lze vypočítat následujícími vztahy:

$$rozdiluser = user(t+k) - user(t) \tag{3.4}$$
$$rozdilnice = nice(t+k) - nice(t) \tag{3.5}$$
$$rozdilsystem = system(t+k) - system(t) \tag{3.6}$$
$$rozdilidle = idle(t+k) - idle(t) \tag{3.7}$$
$$rozdilusn = rozdiluser + rozdilnice + rozdilsystem \tag{3.8}$$
$$CPULoad = \frac{rozdilusn}{rozdilusn + rozdilidle} * 100 \tag{3.9}$$

Kde t představuje čas začátku měření a k čas měření.

3.1.5 Zoneinfo

Linuxové jádro rozdělí operační paměť do několika zón z důvodu adresovatelnosti paměti. Tyto zóny a jejich složitost jsou závislé na tom, zda je systém 32bitový nebo 64bitový. Existují 4 typy zón. Tyto zóny jsou popsány v tabulce 3.3. [38]

DMA	Obsahuje nejnižších 16 MB paměti RAM. Tato zóna existuje spíše pro historické účely. Dříve existovala zařízení, které mohla přímo přistupovat pouze do paměti v této zóně.
DMA32	Zóna DMA32 je dostupná pouze na 64bitových systémech. Obsahuje nejnižší 4 GB paměti RAM. Je dostupná pro zachování kompatibility s komponenty, které mohou přímo přistupovat pouze do nejnižších 4GB paměti RAM.
Normal	Zóna Normal je vytvořena na 32 i 64bitových systémech. Na 64 bitových systémech obsahuje veškerou paměť, která přesahuje nejnižší 4 GB paměti. Na 32bitových systémech je to všechna nejnižší RAM v rozmezí 16 - 896 MB a je určena pro komplexní úlohy a historické účely.
HighMem	Dostupná pouze na 32bitových systémech. Obsahuje veškerou paměť, která přesahuje nejnižších 896 MB.

Tabulka 3.3: Vysvětlení typů jednotlivých zón [38]

3.2 Souborový systém sysfs

Sysfs je virtuální souborový systém určený pro zpřístupnění objektů linuxového jádra do uživatelského prostoru. Pomocí souborového systému sysfs lze nejen pozorovat, ale i měnit vnitřní datové struktury linuxového jádra. Záznamy souborového systému sysfs vznikly pomocí objektů linuxového jádra tzv. *kobjects*. Tato struktura je definována v linuxovém

jádře a je použita pro počítání referencí a správu objektů linuxového jádra. Jelikož jsou všechna zařízení a sběrnice spravovány pomocí struktury kobjects, sysfs reprezentuje mj. topologii hardwaru. [31]

Pokud je souborový systém sysfs nakonfigurován tak, aby byl dostupný, je vždy součástí linuxového jádra. Obvyklý přípojný bod je */sys*. [31]

3.2.1 Síťová rozhraní

Všechna síťová rozhraní mají svůj záznam v souborovém systému sysfs a nachází se v */sys/class/net*. Obsahem těchto adresářů jsou informace o stavu dostupných síťových rozhraní. Obsahy těchto souborů jsou generovány od nahrání ovladače daného síťového rozhraní.

Soubory souborového systému sysfs týkající se síťových rozhraní jsou popsány v tabulkách 3.4 a 3.5.

Soubor	Popis
address	MAC adresa síťového rozhraní
duplex	V jakém duplexním módu (half duplex, full duplex) síťové zařízení pracuje
mtu	Hodnota maximální přenositelné jednotky
operstate	Stav síťového rozhraní.
speed	Rychlost sítě (10 Mb/s, 100 Mb/s, 1 Gb/s atd.)

Tabulka 3.4: Popis souborů síťového rozhraní v */sys/class/net/rozhraní/*

Soubor	Obsahuje
rx a tx_bytes	Počet přijatých a odeslaných bajtů dat
rx a tx_dropped	Počet odhozených paketů
rx a tx_packets	Počet přijatých a odeslaných paketů

Tabulka 3.5: Popis souborů síťového rozhraní v */sys/class/net/rozhraní/statistics/*

3.3 Měření spotřeby vestavěných zařízení

Jelikož je mnoho vestavěných zařízení neustále v provozu, sledováním spotřeby u těchto zařízení lze navrhnout řešení, které dokáže spotřebovanou energii zařízení snížit. Sledování spotřeby je realizováno pomocí senzorů dostupných ve vestavěném zařízení.

3.3.1 Softwarové řešení

Měření spotřeby na vestavěných zařízeních, které nejsou napájeny bateriemi, pouze pomocí softwaru není možné. U vestavěných zařízení, které jsou napájeny baterií, lze nejen pozorovat spotřebu pomocí senzorů, ale i pomocí softwaru, který vypočítává například předpokládanou výdrž na baterii. Předpokládanou výdrž lze vypočítat pomocí údajů, které poskytuje souborový systém sysfs v umístění */sys/class/power_supply/IDBaterie/*. Obsah nejdůležitějších souborů je popsán v tabulce 3.6.

Soubor	Obsahuje
model_name	Identifikátor baterie
voltage_min_design	Minimální napěťová úroveň baterie [μV]
voltage_now	Aktuální napětí baterie[μV]
current_now	Aktuální proud baterie [μA]
charge_full	Maximální náboj baterie [μAh]
charge_now	Aktuální náboj baterie [μAh]
capacity_level	Aktuální kapacita baterie [%]

Tabulka 3.6: Vysvětlení obsahu souborů týkajících se baterie v souborovém systému sysfs

Předpokládanou výdrž baterie lze vypočítat následujícími vztahy:

$$k\_naboj = charge\_now(t) - charge\_now(t+k) \tag{3.10}$$

$$hodinovynaboj = \frac{3600 * k\_naboj}{k} \tag{3.11}$$

$$vydrz = \frac{charge\_now(t+k)}{hodinovynaboj} \tag{3.12}$$

Kde t je čas začátku měření, k je čas měření v sekundách, *k_naboj* označuje náboj baterie, který byl spotřebován za čas k. *hodinovynaboj* je předpokládaný náboj, který bude potřeba na hodinu provozu a *vydrz* je předpokládaná výdrž baterie v hodinách.

3.3.2 Hardwarové řešení

Pro měření spotřeby a teploty je nutné využít senzorů, které se nacházejí ve vestavěném zařízení. Pro přečtení těchto senzorů může být využit například open source program lm-sensors. Tato aplikace využívá subsystému HWMON linuxového jádra. Linuxové jádro obsahuje mnoho ovladačů pro řadu senzorů, které se využívají pro měření teploty, napětí, proudů a spotřeby. Ovladače umožňují aplikacím (například lm-sensors) přečíst údaje z teplotních, napěťových a proudových senzorů. Po detekci všech senzorů ve vestavěném zařízení se následně vloží moduly do linuxového jádra. Po vložení těchto modulů se zpřístupní informace o teplotách, proudech, napětí a spotřebě v souborovém systému sysfs (*/sys/class/hwmon*).

3.4 Optimalizace spotřeby vestavěných zařízení

Optimalizace spotřeby vestavěného zařízení může být dosáhnuto pomocí dynamického řízení taktovacích frekvencí procesoru. Frekvence procesoru jsou přednastaveny v linuxovém jádře pomocí tzv. governorů. Governory jsou moduly linuxového jádra pro řízení procesoru, které umožňují dynamickou změnu taktovacích frekvencí v závislosti na využití procesoru nebo například na využití grafického adaptéru. Governory jsou implementovány v jazyce C a mohou být upraveny tak, aby byly více efektivní, případně vyvažovaly zatížení na jednotlivých jádrech procesoru.

Dalšími možnostmi pro snížení spotřeby je podtaktování a podvoltování periferií, například procesoru, úprava argumentů při připojování diskových oddílů, úprava nastavení síťové karty atd.

Kapitola 4

Existující řešení

Existuje mnoho nástrojů určených k monitorování stavu systému. Většina z těchto monitorovacích nástrojů je určena pro monitorování jedné specifické oblasti systému.

4.1 free

Nástroj free poskytuje základní informace o tom, jak je nakládáno s pamětí systému Linux. [19]

```
# free
             total         used         free       shared      buffers
Mem:        123828         7460       116368            0          100
-/+ buffers:               7360       116468
Swap:            0            0            0
```

Popis jednotlivých řádků výstupu nástroje free:

- První řádek se skládá ze sedmi hodnot a obsahuje detaily o operační paměti RAM. Hodnota total značí velikost celkové paměti v jednotkách KB. Hodnota used udává množství operační paměti RAM, která je využita operačním systémem. Hodnota free udává množství operační paměti RAM, která je dostupná pro použití systémovými procesy. Hodnota shared značí velikost sdílené paměti a poslední hodnota buffers celkovou velikost paměti RAM, která je uložena do bufferu jinými aplikacemi.
- Druhý řádek buffers obsahuje detaily o paměti, která je použita jako vyrovnávací paměť. Z toho tedy vyplývá, že celková hodnota použité paměti je součtem použité paměti RAM a použité vyrovnávací paměti, tzn. celková velikost volné paměti RAM je rozdílem hodnot celkové velikosti paměti RAM a hodnoty použité paměti.
- Třetí řádek poskytuje informace o paměti swap. [19]

Hodnoty poskytované nástrojem free jsou vypočteny ze souboru */proc/meminfo*, který byl popsán na straně 11. Pro řešení této bakalářské práce není vhodný, právě z důvodu jeho zaměřenosti pouze na operační paměť.

4.2 perf

Nástroj perf je založen na rozhraní perf_events, které je součástí linuxových jader verze 2.6 nebo vyšší. Nástroj perf lze použít ke sběru a analýze informací, které jsou poskytovány podporovanými softwarovými a hardwarovými událostmi. Seznam podporovaných událostí lze zjistit pomocí příkazu *perf list*. [10]

```
> perf-list
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                                [Hardware event]
  instructions                                        [Hardware event]
  cache-references                                    [Hardware event]
  cache-misses                                        [Hardware event]
  branch-instructions OR branches                     [Hardware event]
  branch-misses                                       [Hardware event]
  bus-cycles                                          [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend     [Hardware event]
  ref-cycles                                          [Hardware event]

  cpu-clock                                           [Software event]
  task-clock                                          [Software event]
  page-faults OR faults                               [Software event]
  context-switches OR cs                              [Software event]
  cpu-migrations OR migrations                        [Software event]
  minor-faults                                        [Software event]
  major-faults                                        [Software event]
  alignment-faults                                    [Software event]
  emulation-faults                                    [Software event]
```

Softwarové události jsou poskytovány linuxovým jádrem, například počet změn kontextu procesoru, chyby stránkování atd. zatímco hardwarové události jednotkou PMU (performance monitoring unit). [10]

K získání počtu instrukčních cyklů procesoru při kopírování dat z */dev/zero* do */dev/null* o velikosti 512MB je třeba spustit nástroj perf s dvěma argumenty. První parametr *-e* umožňuje vybrání událostí, v tomto případě cpu-cycles, a druhý parametr je operace, která realizuje kopírování dat.

```
perf stat -e cycles "dd if=/dev/zero of=/dev/null count=1000000"
 Performance counter stats for 'dd if=/dev/zero of=/dev/null count=1000000':
     1 331 384 091      cycles
       0,486173385 seconds time elapsed
```

4.2.1 Performance monitoring unit

Procesor Cortex-A9 obsahuje jednotku performance monitoring unit (PMU), která poskytuje šest čítačů ke sběru statistik o operacích procesoru a paměťového systému. Každý čítač může čítat některou z 58 událostí (například počet externích přerušení vykonávaných procesorem, počet instrukcí, které vykonala Load and Store jednotka atd.) v procesoru Cortex-A9. Čítače PMU a jejich kontrolní registry jsou dostupné pomocí vnitřního rozhraní CP15 a rozhraní Debug ADB. [17]

4.3 top

Nástroj top poskytuje dynamické informace o zatížení operačního systému v reálném čase. Poskytuje minimální interaktivní grafické prostředí. Pomocí grafického prostředí lze zobrazit jak souhrnné informace o operačním systému, ale i seznam procesů, které jsou spravovány linuxovým jádrem. Nástroj top obsahuje mnoho nastavení týkajících se zobrazení dat. Tyto změny mohou být zapsány do souboru a lze je tak udělat permanentními. [21]

```
> top
  Mem: 7460K used, 116368K free, 0K shrd, 100K buff, 1056K cached
  CPU:   0% usr   0% sys   0% nic  99% idle   0% io   0% irq   0% sirq
  Load average: 0.06 0.03 0.02 1/31 644
    PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
    644   628 root     R     1104   1%   1% top
    619     1 root     S     1116   1%   0% -sh
      1     0 root     S     1112   1%   0% init
    595     1 root     S     1096   1%   0% /sbin/syslogd -n
```

Popis řádků nástroje top:

- První řádek obsahuje informace o operační paměti RAM. Tento řádek je totožný s výstupem nástroje free s přidanou hodnotou, která udává množství paměti RAM využité jako vyrovnávací paměť systému.
- Druhý řádek obsahuje informace o využití procesorového času. Hodnoty, které tento řádek obsahuje, byly popsány na straně 14.
- Třetí řádek je výstup ze souboru */proc/loadavg*. Tyto hodnoty byly popsány na straně 13. [21]

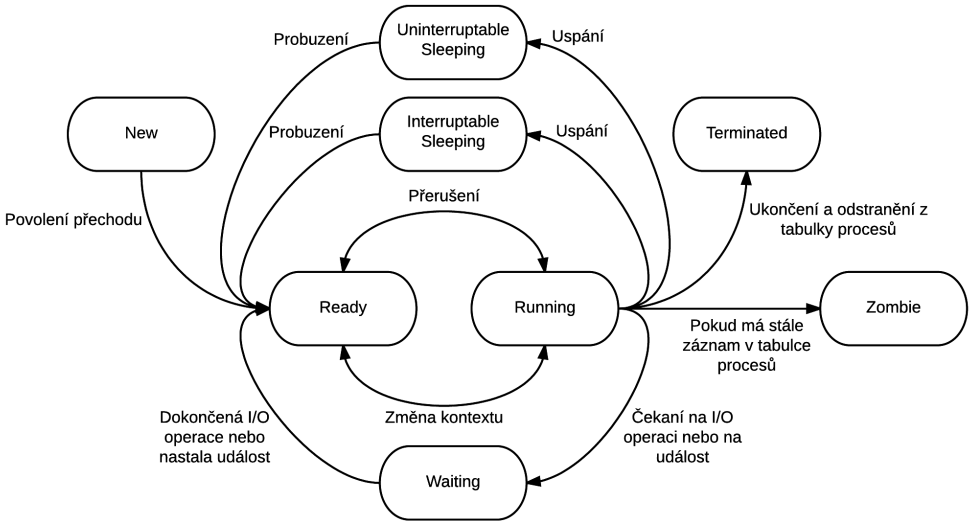
Poté následuje seznam procesů, které jsou popsány osmi sloupci.

Sloupec	Popis
PID	Process ID, unikátní identifikátor procesu
PPID	Parent Process ID, unikátní identifikátor rodičovského procesu
USER	User Name, jméno uživatele, který proces spustil
STAT	Process state, identifikátor stavu procesu
VSZ	Množství využité operační paměti procesem v KB
%VSZ	Poměr hodnoty VSZ s celkovou pamětí RAM
%CPU	CPU Usage udává z kolika procent je procesor vytěžován daným procesem
COMMAND	Jméno programu nebo příkaz, pomocí kterého byl proces spuštěn

Tabulka 4.1: Popis sloupců nástroje top [21]

Sloupec STAT obsahuje jednu z osmi hodnot, které značí, zda má proces vyšší prioritu, znak < , nebo nižší prioritu, označený písmenem N, nebo v jakém stavu se proces nachází. Jednotlivé stavy jsou popsány níže.

- New - Vytvořený proces, který čeká na povolení přechodu do stavu Running od plánovače úloh.
- R, Running - Proces, který má přidělen procesor.
- Waiting - Proces, který čeká na nějaké události systému, například dokončení vstupně výstupní operace.
- Ready - Proces, který je uložen do paměti a čeká na přidělení procesoru. Procesy ve stavu ready se řadí do fronty.
- T, Terminated - Proces, který dokončil vykonávání nebo byl ukončen. [3]
- Z, Zombie - Proces, který dokončil vykonávání, ale má stále záznam v tabulce procesů. [33]
- S, Sleeping - Proces, který se vzdal procesorového času a změnil svůj stav na nespustitelný po celou dobu spánku. Jeho spánek lze vyrušit pomocí signálu SIGCONT.[5]
- D, Uninterruptible sleep - Proces, který se nachází v nepřerušitelném spánku. [4]



Obrázek 4.1: Diagram stavů procesu [3]

Nástroj top neumožňuje ukládání naměřených hodnot ve formátu, který by byl vhodný například pro vykreslení grafů.

4.4 vmstat

Nástroj vmstat poskytuje informace o procesech, paměti, stránkování, blokových zařízeních a aktivitě procesoru. Má mnoho argumentů, například argument delay nastaví vzorkovací interval nástroje vmstat atd. [26]

```
root@test:/ # vmstat
  procs  memory                    system           cpu
  r  b  free    mapped   slab     in   cs    us  ni  sy id  wa
  0  0  105264  93252   26488   118  430    8   0  14 79   0
  0  0  105264  93256   26492    94  440   10   0  10 82   0
  1  0  105264  93256   26500    88  392   10   0  10 82   0
```

Popis hodnot výstupu nástroje vmstat:

- Hodnota sloupce r indikuje počet procesů, které čekají na spuštění.
- Hodnota b indikuje počet procesů, které jsou v nepřerušitelném spánku. [26] Proces v nepřerušitelném spánku nemůže být ukončen (ani pomocí signálu SIGKILL) a obvykle čeká na dokončení vstupně výstupní operace. [4]
- Hodnota free indikuje velikost paměti, která je nevyužitá. Tato hodnota je hodnotou řádku *MemFree* v souboru */proc/meminfo*. [36]
- Hodnota mapped udává celkovou velikost paměti, která byla použita k mapování souborů nebo knihoven pomocí mmap příkazu. [36]
- Hodnota slab značí celkovou velikost paměti, která je alokována alokátorem slab [36]
- Hodnota in indikuje počet vygenerovaných přerušení za sekundu. Hodnotu in lze využít ke sledování počtu přerušení v souvislosti s interakcí s periferiemi. Příliš vysoký počet přerušení může značně ovlivnit propustnost systému.
- Hodnota cs udává počet přepnutí kontextu za sekundu.
- Hodnota us je celkový čas vytížení procesoru uživatelskými procesy.
- Hodnota ni je celkový čas vytížení procesoru uživatelskými procesy nízké priority.
- Hodnota sy je celkový čas vytížení procesoru systémovými procesy.
- Hodnota id je celkový čas nečinnosti procesoru.
- Hodnota wa je celkový čas čekání procesoru na dokončení vstupně výstupních operací. [26]

Pomocí hodnot wa a id lze zjistit, kolik procesorového času bylo využito k čekání na vstupně výstupní operace a detekovat, které části systému mohou ovlivňovat výkon a propustnost systému.

Nástroj vmstat slouží pouze k zobrazení dat. Neumožňuje tedy ukládání nebo další zpracování naměřených hodnot. Z tohoto důvodu není vhodný pro řešení této bakalářské práce.

4.5 powertop

Powertop je linuxový nástroj, který je určen pro detekci problémů se spotřebou a řízením spotřeby. Powertop obsahuje interaktivní mód, ve kterém může uživatel experimentovat s nejrůznějšími nastaveními týkajícími se řízením spotřeby, například vypnutí wake-on-lan stavu na síťových adaptérech, zapnutí automatického vypnutí u zařízení, které tuto

možnost umožňují atd. Powertop oznamuje, které komponenty v systému jsou nejpravděpodobněji zodpovědné za nadměrnou spotřebu energie počínaje softwarovými aplikacemi konče aktivními komponenty systému. [18] Powertop pracuje s rozhraními cpufreq, cpuidle, power_supply class a pmu. Tyto rozhraní musí být implementovány v linuxovém jádře. Jestliže není nějaké rozhraní implementováno, dojde k vynechání informací, které by poskytovalo neimplementované rozhraní. [9] Nástroj powertop je vydaný pod licencí GPLv2, tudíž jeho zdrojový kód je otevřený.

Požadovaná rozhraní jsou:

- Cpuidle - Je využíváno ke správě idle stavů procesoru, které jsou závislé na zpoždění probuzení.
- Cpufreq - Je využíváno ke škálování napětí a frekvence, když je systém zaneprázdněn.
- Power_supply class nebo PMU - Je využíváno k exportování informací o napájení. [9]

Hlavní výhodou powertopu je jeho dostupnost pro mnoho architektur (x86, x86_64, ARM, MIPS). Vestavěné systémy, které využívají nástroj busybox, mohou využít applet powertop, který je již obsažen v nástroji busybox. Jelikož změny provedené nástrojem powertop nejsou permanentní, je potřeba tyto změny aplikovat po každém nastartování systému. Powertop vyžaduje pro svoji práci práva administrátora.

Jelikož je powertop primárně určen pro přenosná zařízení, nedostačuje k řešení této bakalářské práce. Mnoho vestavěných zařízení není napájeno baterií, a proto powertop na těchto zařízeních nebude poskytovat informace o předpokládané spotřebě zařízení. V případě vestavěných zařízení napájených baterií powertop poskytne informace o různých nastaveních, které ovlivňují spotřebu a jsou dostupné skrze souborový systém sysfs.

4.6 lm_sensors

Hlavním účelem tohoto nástroje je poskytnout nejvhodnější a nejkompletnější řešení pro monitorování hardwaru pro operační systémy s linuxovým jádrem. [13]
lm-sensors podporuje následující senzory a čipy:

- Monitorovací senzory pro zařízení připojená na sběrnici ISA.
- Monitorovací senzory pro zařízení připojená na sběrnici I2C/SMBus.
- Monitorovací senzory pro zařízení připojená na sběrnici SPI.
- Monitorovací vlastnosti, které jsou integrované v Super-I/O čipech.
- Monitorovací vlastnosti, které jsou integrované v jižním můstku (south bridge).
- senzory pro snímání teploty v procesoru.
- senzory pro snímání teploty v paměťových modulech. [13]

Aplikace lm-sensors dokáže monitorovat následující vlastnosti:

- Rychlost ventilátorů, které disponují rychlostními senzory a jsou zapojeny pomocí 3 nebo 4 pinového konektoru na základní desce.
- Proudy a napětí komponent.
- Teploty čidel tepelných senzorů.
- Voltage Identification (VID) - každá frekvence procesoru má přidělené napětí. Toto napětí se nazývá VID. [13]

Aplikace lm-sensors se skládá ze tří částí a to ze sensors, sensors-detect a sensorsd.

4.6.1 sensors-detect

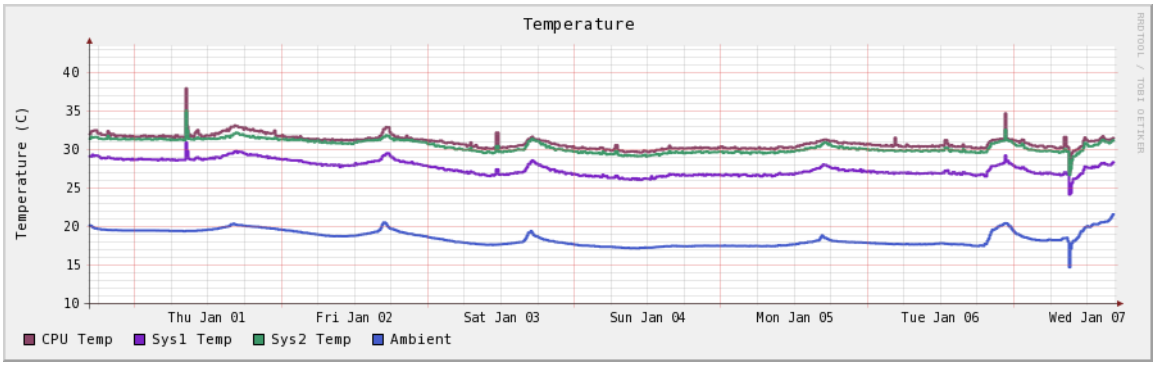
Tento skript je napsán v jazyce perl a zajišťuje detekci všech senzorů, kterými zařízení disponuje. Na základě detekovaných senzorů vytvoří seznam modulů linuxového jádra, které jsou potřeba pro zpřístupnění a přečtení informací ve všech senzorech, ke kterým existuje modul v linuxovém jádře. Tento seznam využívá služba sensors. [7]

4.6.2 sensors

Služba sensors zajistí vložení všech potřebných modulů do jádra systému. Aplikace sensors slouží k interpretaci dat, které jsou poskytovány senzory, jejichž moduly jsou vloženy do linuxového jádra. [13]

4.6.3 sensord

Služba sensord sbírá data a ukládá je do RRD (round robin database) databáze, pomocí které mohou být sesbíraná data zobrazena pomocí grafu. [13]



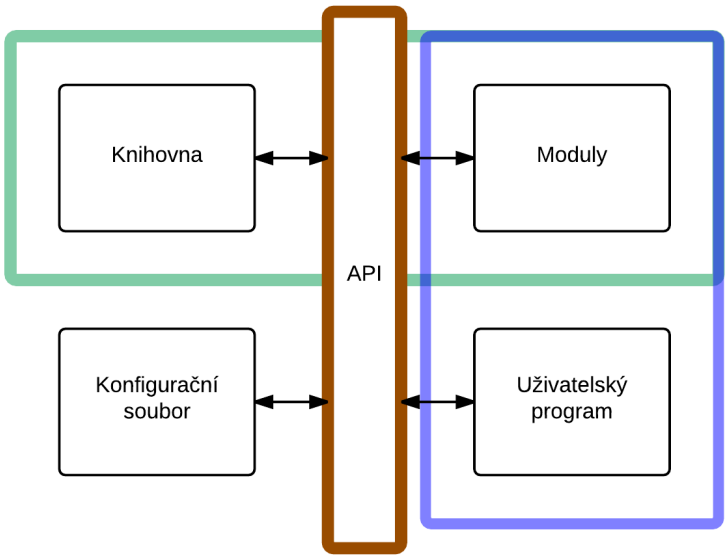
Obrázek 4.2: Graf vykreslený aplikací RRDTool [13]

Nástroj lm-sensors nedostačuje k řešení bakalářské práce právě z důvodu jeho zaměřenosti pouze na čtení dat ze senzorů.

Kapitola 5

Návrh a implementace knihovny pro sledování zatížení systému

Cílem této bakalářské práce je vytvoření knihovny, která bude poskytovat rozhraní (API) pro uživatelské aplikace a bude umožňovat sledování a monitorování operačního systému v reálném čase. Měření budou zajišťovat moduly, které mohou být napsány uživatelem. Interpretaci dat budou zajišťovat výstupní moduly, které mohou být taktéž napsané uživatelem. Po inicializaci všech dostupných modulů je možné vybrat měření jednotlivých veličin, které se má provést. Měření samo o sobě by mělo zatěžovat systém co nejméně. Knihovna načítá nastavení z konfiguračního souboru. Analýzou výsledků měření lze zkoumat chování a spotřebu systému při různé zátěži. Na základě informací o chování systému je možné navrhnout různé optimalizace pro zlepšení výkonnosti, efektivity, propustnosti systému atd.



Obrázek 5.1: High Level diagram knihovny

5.1 Měření a ukládání naměřených dat

Samotná knihovna pouze umožňuje měřit a sledovat zatížení systému, tzn. nerealizuje samotné měření nebo ukládání dat. Měření je zajištěno měřicími moduly, které určují zdroj dat potřebný k naměření dané veličiny. Zdrojem dat může být soubor, například */proc/stat* nebo výstup nějakého nástroje, například *free*. Před samotným zavoláním měřicí funkce, knihovna připraví určený zdroj dat tak, aby byl připraven ke zpracování během vykonávání měřicí funkce. Měřicí modul zpracuje připravený zdroj dat a uschová výsledek měření tak, aby mohl být zpracován výstupními moduly.

Výstupní moduly realizují ukládání a interpretaci naměřených hodnot. Jak budou data ukládána nebo interpretována je určeno v měřicích modulech.

5.2 Sledované parametry

Měřicí moduly mohou sbírat a ukládat například následující informace:

- Blokové zařízení - Využití blokových zařízení, počet čekajících a probíhajících vstupně výstupních operací, počet celkových IO operací atd.
- Paměť RAM - Využití pamětí RAM.
- Procesor - Využití procesoru a kolik času procesor strávil v jednotlivých frekvencích.
- Senzory - Všechny dostupné informace poskytované senzory, které obsahuje vestavěný systém.
- Síť - Celkový počet příchozích a odchozích paketů a počet přenesených dat na jednotlivých rozhraních.
- Další statistiky a informace měřené moduly naprogramovanými uživateli.

5.3 Možnosti ukládání naměřených dat

Výstupní moduly mohou interpretovat nebo ukládat naměřené hodnoty například:

- Zasláním pomocí TCP/IP socketů na vzdálené úložiště.
- Sapsáním do souborů na lokální úložiště.
- Dalšími možnostmi, které jsou poskytovány výstupními moduly naprogramovanými uživateli.

5.4 Implementace

Knihovna implementuje rozhraní API pro měřicí moduly, výstupní moduly a uživatelské aplikace. API knihovny je popsáno na straně 31. Dále implementuje seznam všech možností měření, které jsou dostupné měřícími moduly. Ke knihovně mohou být dynamicky připojovány měřicí a výstupní moduly rozšiřující základní funkčnost knihovny.

Knihovna obsahuje následující struktury:

- Features.
- FeaturesOut.
- PrevValues.

5.5 Popis struktury Features a FeaturesOut

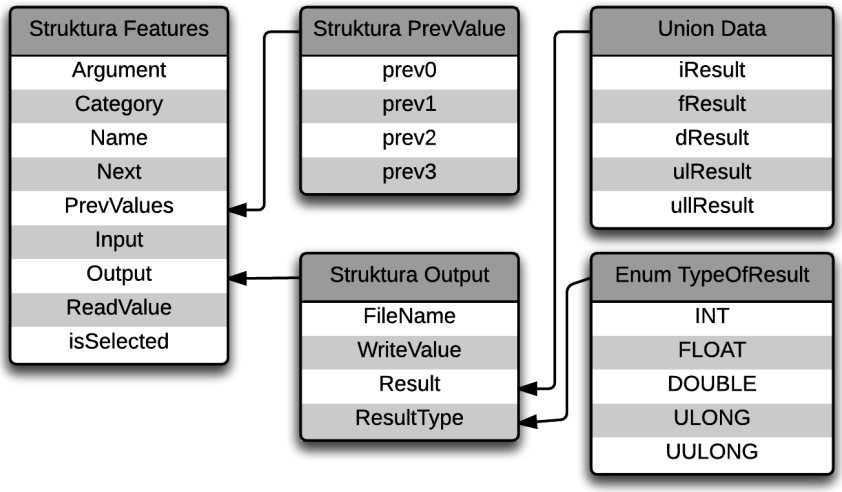
Struktura Features je stěžejní strukturou knihovny. Obsahuje všechny informace o měření veličin. Je plněna měřicími moduly. Uživatelský program inicializuje seznam strukur Features zavoláním inicializačních funkcí dostupných měřicích modulů. Po inicializaci jsou prvky seznamu přístupné uživateli přes API knihovny. Každý prvek seznamu struktur Features je určen pro měření jedné konkrétní hodnoty, například teploty, spotřeby, množství přerušení atd. Dále obsahuje strukturu FeaturesOut, která obsahuje informace o následném uchování nebo interpretaci naměřených hodnot.

Struktura Features obsahuje následující členy:

- Argument - Pomocná proměnná, která může být využita měřicími funkcemi v modulech.
- Category - Proměnná, pomocí které lze zjistit, do jaké oblasti měření patří.
- Name - Proměnná, jejíž hodnotou je název zařízení.
- Input - Proměnná, která určuje soubor nebo příkaz, který je zdrojem dat pro dané měření.
- isSelected - Proměnná typu bool, která indikuje, zdali je konkrétní prvek vybrán.
- Ukazatel na funkci ReadValue - Ukazatel na funkci modulu, která realizuje měření.
- Ukazatel na další prvek struktury.
- PrevValues - Struktura, která slouží pro uchování až 4 předešlých hodnot měření.
- Output - Struktura typu FeaturesOut.

Struktura FeaturesOut obsahuje následující členy:

- OutFile - Proměnná, jejíž hodnota je název výstupního souboru, do kterého se budou zapisovat naměřené hodnoty.
- Ukazatel na funkci WriteValue - Ukazatel na funkci výstupního modulu, která realizuje interpretaci dat.
- Result - Union, který uchovává výsledky měření.
- ResultType - Enum, který určuje jakého typu je výsledek.



Obrázek 5.2: Struktury Features a FeaturesOut

5.6 Popis struktury měřicích modulů

Modul se skládá ze dvou částí, první část zajištuje inicializaci a druhá část samotné měření. Inicializace zajistí naplnění seznamu struktur typu Features všemi dostupnými měřicími možnostmi. Tato inicializační funkce je volána uživatelskou aplikací. Druhá část měřicího modulu musí obsahovat funkci, která musí mít definici typu *int jménofunkce (struct Features **Pointer, FILE* file)*. Právě tato funkce bude uložena do ukazatele na funkci, který obsahuje struktura typu Features. Knihovnu lze rozšířit o další měřicí moduly napsané uživateli.

Základní měřicí moduly, které knihovna obsahuje, jsou:

- Měřicí modul operační paměti.
- Měřicí modul zatížení procesoru.
- Měřicí modul síťového provozu.
- Měřicí modul vstupně výstupních operací.
- Měřicí modul napětí, proudu, teplot a spotřeby (využívá nástroj lm-sensors, který je popsán na stránce 23).

5.6.1 Ukázka kódu měřicího modulu

Následující kód popisuje implementaci měřicího modulu, který měří celkové množství operační paměti. Nejprve je přidán prvek do seznamu struktur Features. Poté jsou proměnné vytvořeného prvku naplněny informacemi potřebnými ke změření hodnot nebo zpracování naměřených hodnot. Funkce MeasureMem se stará o samotné měření. Vyhledá v souboru */proc/meminfo* hodnotu řádku MemTotal a uloží ji do unionu result, Tato hodnota je poté uložena pomocí funkce, která je přiřazena do ukazatele na funkci na 15 řádku. V tomto případě se hodnota uloží do souboru na lokálním úložišti.

```
1  // Funkce, která inicializuje seznam struktur Features
2  void MemInit(struct Features **MyHead, struct Features **MyCurrent)
3  {
4    struct Features *Pointer;
5    char Out[BUFFERSIZE];
6          Pointer = add_to_list(true, MyHead, MyCurrent); // Vytvoření a
       přidání prvku typu Features do seznamu struktur
7    Pointer->Argument = MEM_TOTAL;
8    Pointer->Category = "MEMORY";
9    Pointer->isSelected = false;
10   strcpy(Pointer->Input, "/proc/meminfo");
11   Pointer->ReadValue = MeasureMem;
12   strcpy(Pointer->Name, "mem_total");
13   strcpy(Pointer->Output.OutFile, mem-Total);
14   Pointer->Output.WriteValue = WriteToNET;
15   Pointer->Output.ResultType = ULLONG;
16   Pointer->Output.Result.ullResult = 0;
17 }
18 // Funkce, která provádí měření a ukládá výsledek do unsigned long long
       proměnné ve struktuře FeaturesOut
19 int MeasureMem(struct Features **Pointer, FILE * file)
20 {
21   struct Features *tmp = *Pointer;
22   char buffer[BUFFERSIZE];
23
24   switch (tmp->Argument)
25   {
26   case MEM_TOTAL:
27     rewind(file);
28     while(fgets(buffer, sizeof(buffer), file) != NULL)
29     {
30       if(strstr(buffer, "MemTotal"))
31       {
32         strtok(buffer, " ");
33         tmp->Output.Result.ullResult = atoll(strtok(NULL, " "));
34         break;
35       }
36     }
37     break;
38   default:
39     printf("ERROR IN MEMORY MODULE\n");
40     break;
41   }
42
43   return 0;
44 }
```

Ukázka kódu 5.1: Struktura měřicího modulu paměti

5.7 Popis struktury výstupních modulů

Modul se skládá pouze z funkcí, které budou uloženy do ukazatele na funkci ve struktuře typu FeaturesOut. Tyto funkce musí mít definici typu *void názevfunkce (struct Features **Pointer)*. Knihovnu lze rozšířit o další výstupní moduly napsané uživateli.

Základní výstupní modul, který knihovna obsahuje, poskytuje funkce pro:

- Zapsání výsledků na lokální úložiště.
- Zaslaní výsledků pomocí TCP/IP socketů na vzdálené úložiště.

5.7.1 Ukázka kódu výstupního modulu

Funkce *WriteToTXT* zajišťuje uložení výsledků do souborů na lokálním úložišti. Hodnoty ve výsledném souboru jsou formátovány jako dvojice hodnot, z niž první hodnota je čas měření a druhá hodnota je výsledkem měření.

```
1  void WriteToTXT(struct Features **Pointer)
2  {
3    struct Features *tmp = *Pointer;
4
5    FILE *fd;
6
7    fd = libcore_fopen(tmp->Output.OutFile, "a+");
8  // Uložení hodnoty ve správném formátu
9    switch (tmp->Output.ResultType)
10   {
11   case INT:
12     fprintf(fd, "%lu %d\n", ITick, tmp->Output.Result.iResult);
13     break;
14   case DOUBLE:
15     fprintf(fd, "%lu %f\n", ITick, tmp->Output.Result.fResult);
16     break;
17   case FLOAT:
18     fprintf(fd, "%lu %f\n", ITick, tmp->Output.Result.dResult);
19     break;
20   case ULONG:
21     fprintf(fd, "%lu %lu\n", ITick, tmp->Output.Result.ulResult);
22     break;
23   case ULLONG:
24     fprintf(fd, "%lu %llu\n", ITick, tmp->Output.Result.ullResult);
25     break;
26   default:
27     break;
28   }
29
30   fclose(fd);
31 }
```

Ukázka kódu 5.2: Struktura výstupního modulu

5.8 Konfigurační soubor knihovny

Konfigurační soubor knihovny se nachází v */etc/libcore.conf*. Pomocí konfiguračního souboru lze nastavit pracovní adresář, ve kterém se budou ukládat výsledky měření, vzorkovací interval, prefix adresáře na vzdáleném úložišti, port, IP adresu a doménu vzdáleného úložiště.

```
#Konfigurační soubor pro libcore and daemon
#Formát hodnot tohoto souboru je: jménovlastnosti:hodnota
#Jednotkou sampling_value jsou mikrosekundy
```

```
sampling_value:5000000
working_directory:/home/howpathetic/library-for-measuring-system-load/
port:6789
server:localhost
folder_prefix:Socket-
```

5.9 API

API poskytuje soubor instrukcí a specifikuje, jak by klienti měli interagovat se softwarovými komponenty, aby dospěli k vyřešení daného problému. Tyto komponenty jsou typicky implementovány jako softwarové knihovny, které jim umožňují využití ve více aplikacích. [35]

5.9.1 API navrhnuté knihovny

API knihovny poskytuje všechny nezbytné funkce pro práci s pamětí, pro práci s dynamickým seznamem struktur a pro práci se soubory. Dále obsahuje funkce pro obsluhu měření a načtení nastavení z konfiguračního souboru. API knihovny čítá 19 funkcí, které jsou popsány níže.

Funkce pro práci se soubory

Jméno funkce: libcore_fopen
Parametry: Jméno souboru, mód otevření souboru.
Popis: Funkce pro otevření souboru. Vrací otevřený soubor v požadovaném módu.

Jméno funkce: skip_lines
Parametry: Ukazatel na soubor, počet řádků.
Popis: Funkce pro přeskočení n řádků v souboru. Vrací soubor.

Jméno funkce: read_init_settings
Parametry: Nejsou.
Popis: Inicializace nastavení knihovny z konfiguračního souboru. Tato funkce musí být volána na začátku práce s knihovnou.

Funkce pro práci s časem

Jméno funkce: get_time
Parametry: Ukazatel na pole hodnot typu char.
Popis: Funkce vrací aktuální čas ve formátu DD.MM.YY HH:MM:SS.

Funkce pro obsluhu měření

Jméno funkce: start_measure
Parametry: Ukazatel na začátek seznamu struktur.
Popis: Funkce provede měření u všech vybraných prvků struktury typu FeaturesOut.

Jméno funkce: end_measure
Parametry: Ukazatel na začátek seznamu struktur.

Popis: Funkce uloží nebo interpretuje data ze všech měření, které byly provedeny a uspí se na čas, který je nastaven v konfiguračním souboru.

Funkce pro práci se seznamem struktur

Jméno funkce: add_to_list
Parametry: Hodnota typu bool, která určuje, zda má být prvek přidán na konec nebo začátek seznamu, ukazatel na začátek seznamu struktur, ukazatel na aktuální prvek seznamu struktur.
Popis: Funkce přidá na začátek nebo konec prvek do seznamu struktur.

Jméno funkce: search_in_list
Parametry: Název zařízení, ukazatel na předešlý prvek, ukazatel na začátek seznamu struktur.
Popis: Funkce vyhledá odpovídající prvek seznamu struktur.

Jméno funkce: delete_from_list
Parametry: Název zařízení, ukazatel na začátek seznamu struktur, ukazatel na aktuální prvek seznamu struktur.
Popis: Funkce odstraní požadovaný prvek ze seznamu struktur.

Jméno funkce: print_list
Parametry: Ukazatel na začátek seznamu struktur.
Popis: Funkce vypíše všechny prvky seznamu struktur.

Jméno funkce: print_categories
Parametry: Ukazatel na začátek seznamu struktur.
Popis: Funkce vypíše všechny dostupné kategorie, ve kterých lze provádět měření.

Jméno funkce: list_all_features
Parametry: Ukazatel na začátek seznamu struktur.
Popis: Funkce vypíše všechny měřicí funkce poskytované měřicími moduly.

Jméno funkce: select_module
Parametry: Název zařízení, ukazatel na začátek seznamu struktur, argument.
Popis: Funkce vybere požadovaný prvek struktury.

Jméno funkce: select_all_modules
Parametry: Ukazatel na začátek seznamu struktur.
Popis: Funkce vybere všechny prvky v seznamu struktur.

Jméno funkce: select_all_name_modules
Parametry: Název zařízení, ukazatel na začátek seznamu struktur.
Popis: Funkce vybere všechny prvky, které se týkají požadovaného zařízení.

Jméno funkce: select_category_modules
Parametry: Název kategorie, ukazatel na začátek seznamu struktur.
Popis: Funkce vybere všechny prvky, které jsou zařazeny do požadované kategorie.

Funkce pro práci s pamětí

Jméno funkce: libcore_malloc
Parametry: Velikost alokované paměti
Popis: Funkce pro alokování paměti.

Jméno funkce: libcore_free
Parametry: Void
Popis: Funkce uvolní veškerou alokovanou paměť programem.

5.10 Použití knihovny

Použití knihovny se skládá z několika kroků:

- Načtení nastavení z konfiguračního souboru.
- Inicializace měřicích modulů.
- Vybrání prvků pro měření ze seznamu struktur Features.
- Spuštění měření zavoláním funkce start_measure.
- Ukončení a zpracování naměřených výsledků měření zavoláním funkce end_measure.

5.10.1 Ukázka použití knihovny

Následující kód popisuje použití knihovny pro měření informací o operační paměti, které zajišťuje měřicí modul pro operační paměť, jehož část kódu je vysvětlena v ukázce 5.1. Výsledky měření budou zapsány na lokální úložiště pomocí výstupního modulu, jehož část kódu je vysvětlena v ukázce 5.2.

Nejprve se zavolá knihovní funkce read_init_settings, která načte nastavení uživatelské aplikace a knihovny z konfiguračního souboru. Poté se zavolá inicializační funkce měřicího modulu pro operační paměť. Dále je nutné vybrat prvky seznamu struktur Features pro měření, v tomto případě všechny prvky, které se týkají měření operační paměti. Po vybrání prvků se volají obslužné funkce pro obsluhu měření na řádcích 8 a 9. Funkce na 8. řádku uskuteční všechna vybraná měření a následující funkce zapíše všechny naměřené hodnoty na lokální úložiště a uspí proces na interval, který je nastaven v konfiguračním souboru.

```
1  int main(void) {
2    read_init_settings();
3    MemInit(&head, &current);
4    select_category_modules("MEMORY", &head);
5
6    while (1)
7    {
8      start_measure(&head);
9      end_measure(&head);
10   }
11
12   libcore_free();
13   return 0;
14 }
```

Ukázka kódu 5.3: Ukázka použití knihovny

Kapitola 6

Měření na zařízeních

Následující měření jsou provedena na vývojové desce ZC702 nebo na 64bitovým osobním počítači. Obě zařízení mají nainstalovaný operační systém Linux. SoC Zynq-7000, který vývojová deska ZC702 obsahuje, byl popsán na straně 6. Při měření na vývojovém kitu ZC702 byly využity následující periferie: rozhraní pro UART komunikaci přes USB, síťové rozhraní a rozhraní pro připojení karty SD. Osobní počítač obsahuje procesor od společnosti Intel s označením i5-3230. Procesor i5-3230 obsahuje dvě fyzická jádra. Dále obsahuje disk typu SSD společnosti Samsung. Na osobním počítači byly provedeny testy, které se týkají zatížení operační paměti a blokových zařízení.

6.1 Vykreslení grafů

Vykreslení grafů je realizováno skriptem, který je v jazyce Python. Skript pro vykreslení grafů využívá knihovnu *matplotlib* umožňující vykreslení jednoduchých 2D grafů v okně, ve kterém lze měnit nastavení vykreslení grafů za běhu aplikace, například upravení maxima a minima osy x a y, změna barvy vykreslení atd. Dále lze manipulovat přímo se samotným grafem tzn. přiblížení a oddálení grafů, úprava velikosti grafu atd.

6.2 Měření operační paměti

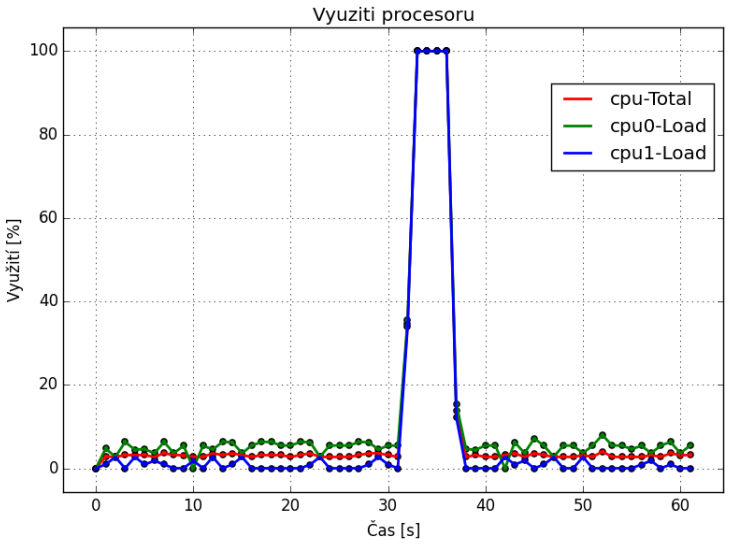
Měření operační paměti probíhalo na vývojovém kitu ZC702, který byl v klidovém stavu.

- Množství dostupné paměti - 1014788 KB
- Množství využité paměti souborovými systémy - 0 KB
- Množství využité paměti využité jako vyrovnávací - 10028 KB
- Množství paměti nevyužité systémem - 1015004 KB
- Množství celkové paměti - 1032660 KB
- Množství použité paměti - 17748 KB

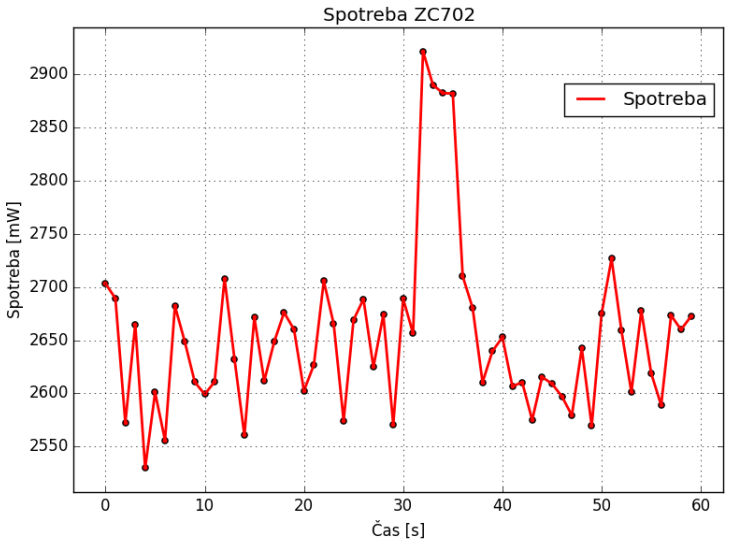
6.3 Závislost spotřeby vývojového kitu ZC702 na zatížení procesoru

6.3.1 Nízké a nárazové zatížení procesoru

Z grafů 6.1 a 6.2 lze určit, jaký vliv má zatížení procesoru na spotřebu vestavěného zařízení. Pokud se procesor nacházel v klidovém stavu, průměrná zátěž vestavěného zařízená byla 2625 mW. Při nárazovém zatížení obou jader procesoru na 100 % spotřeba okamžitě vzrostla o 10 % na hodnotu přibližně 2880 mW. Po navrácení procesoru do klidového stavu, byla průměrná spotřeba vestavěného zařízení opět 2625 mW.



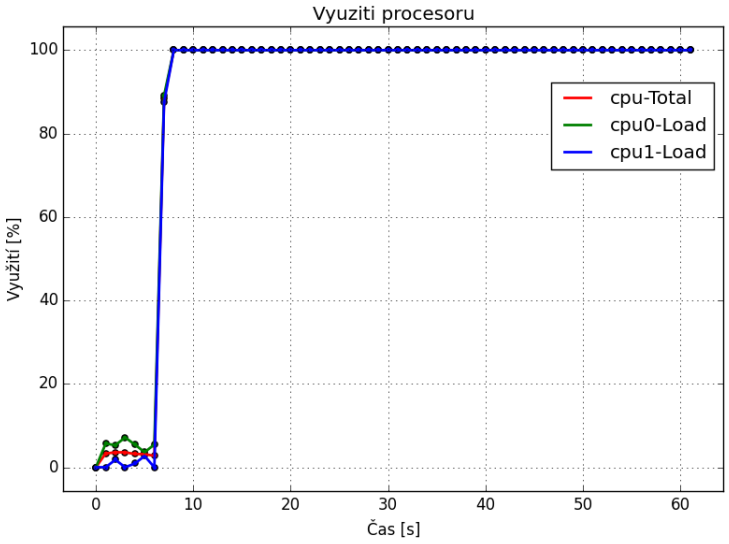
Obrázek 6.1: Graf nízkého a nárazového zatížení procesoru a jeho jader



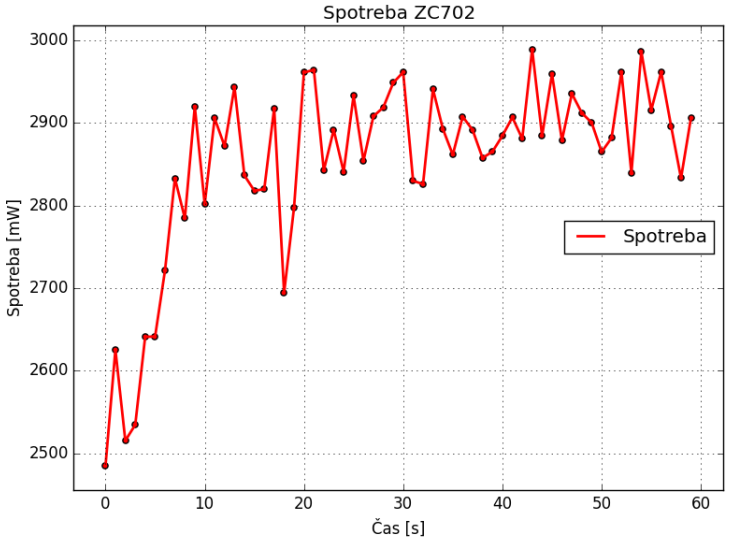
Obrázek 6.2: Průměrná spotřeba zařízení při klidovém a nárazovém zatížení

6.3.2 Vysoké zatížení

Při maximální zátěži obou jader procesoru byla průměrná spotřeba zařízení 2887 mW.



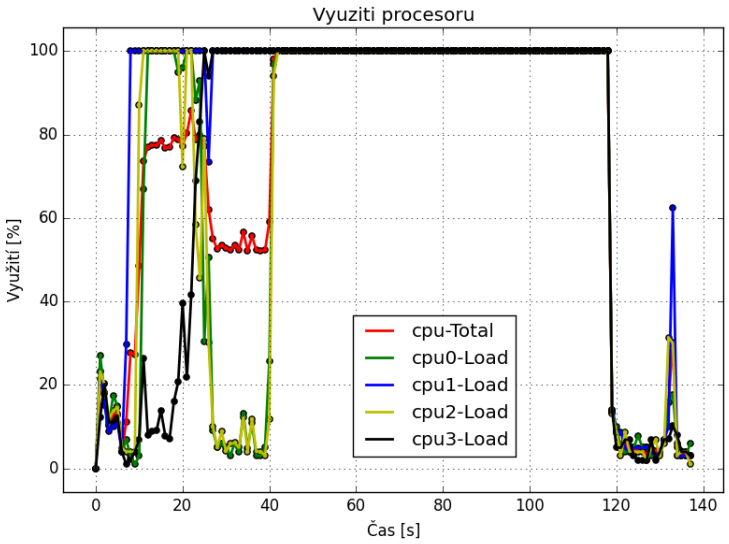
Obrázek 6.3: Graf vysokého zatížení procesoru a jeho jader



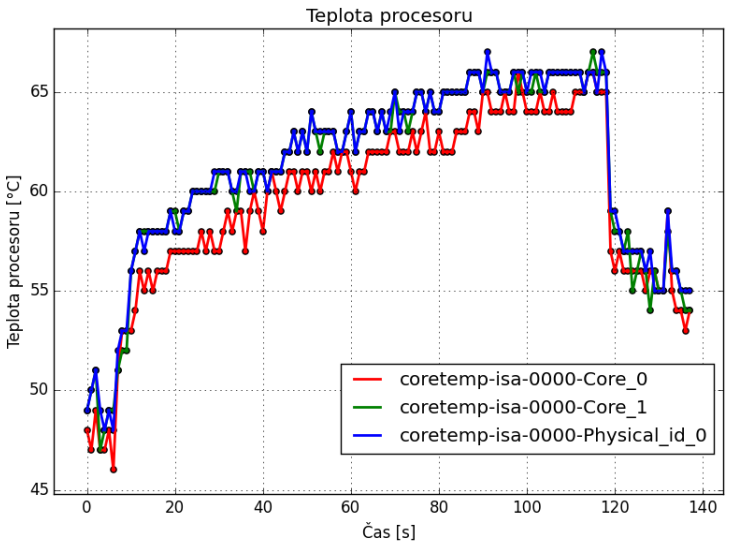
Obrázek 6.4: Průměrná spotřeba zařízení při vysoké zátěži procesoru

6.4 Závislost teploty a spotřeby osobního počítače na zatížení procesoru

V klidovém stavu měl procesor průměrnou teplotu 47,5 °C. Po zatížení všech jader procesoru teplota vzrostla o 38 % na 65 °C. Jelikož je procesor aktivně chlazen, nárůst teploty je postupný. Předpokládaná výdrž na baterii v klidovém stavu byla 3 hodiny a 2 minuty. Zužitkovaný elektrický náboj byl 59 mAh. Při maximálním využití procesoru byl potřebný elektrický náboj 100 mAh, což je nárůst o 70 % oproti klidovému stavu. Předpokládaná výdrž na baterii při maximálním využití se snížila na 1 hodinu a 25 minut.



Obrázek 6.5: Zatížení procesoru Intel i5-3230

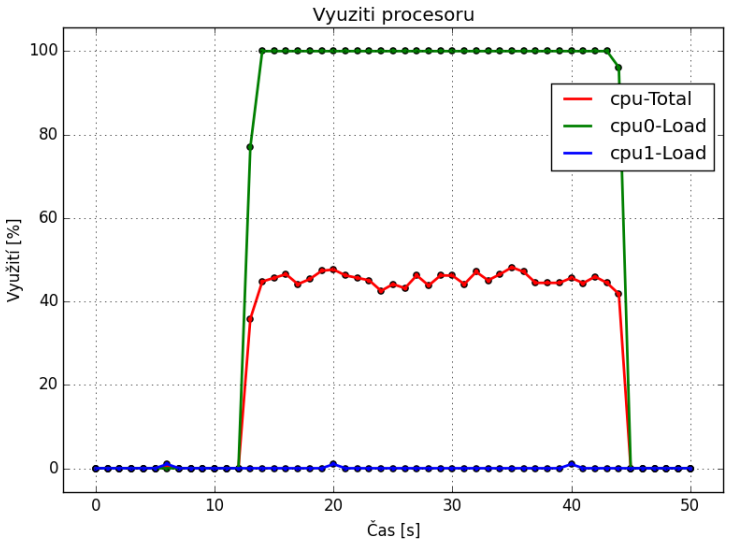


Obrázek 6.6: Teplota procesoru Intel i5-3230

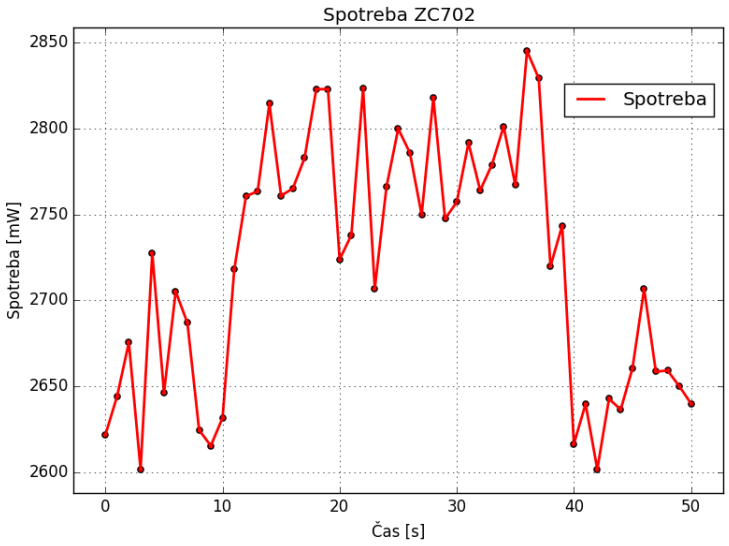
6.5 Závislost využití procesoru desky ZC702 na síťového provozu

6.5.1 Odesílání dat o velikosti 2 GB 1 Gbps linkou

Při odesílání souboru o velikosti 2 GB bylo vytíženo pouze jedno jádro procesoru. Průměrná přenosová rychlost byla 538 Mbps, což znamená, že linka o rychlosti 1000 Mbps byla využita z 53,8 %. Celkově bylo odesláno 262144 datagramů za 31,92 sekund. Z celkového počtu přenesených datagramů jich bylo 18 zahozeno. Z grafů lze předpokládat, že přenosová rychlost je omezena právě využitím pouze jednoho jádra procesoru. Průměrná spotřeba byla 2719,82 mW.



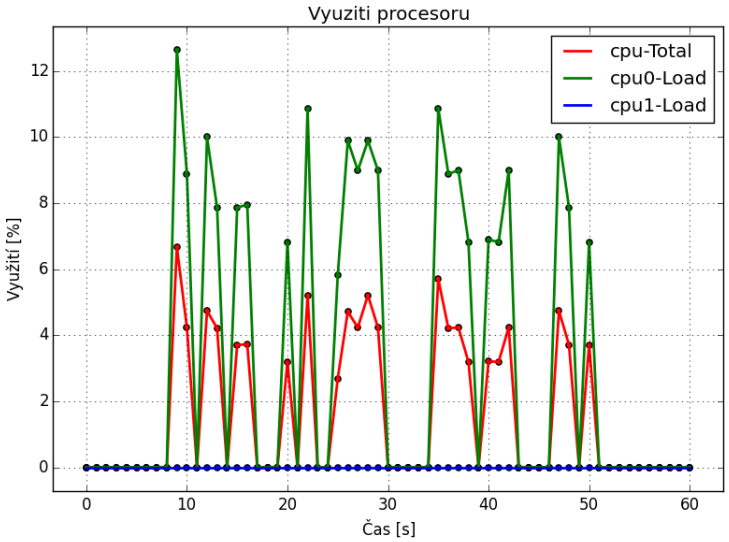
Obrázek 6.7: Využití procesoru při zasílání dat 1000 Mbps linkou



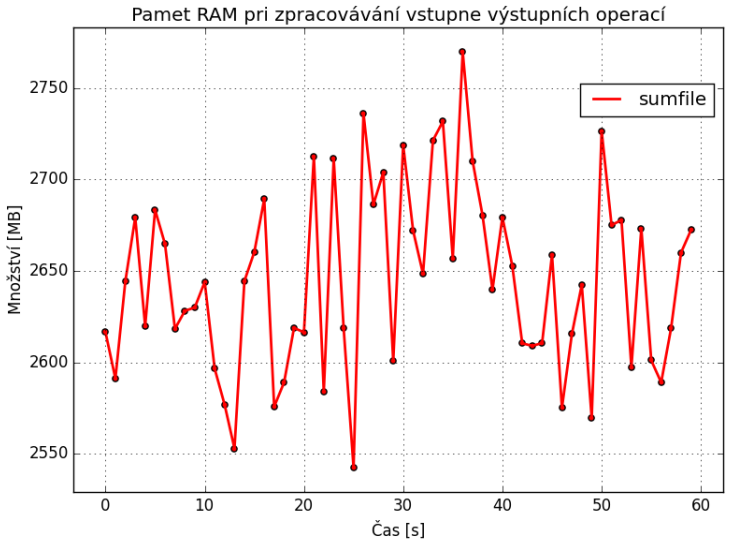
Obrázek 6.8: Spotřeba ZC702 při zasílání dat 1000 Mbps linkou

6.5.2 Odesílání dat o velikosti 500 MB 100 Mbps linkou

Při odesílání souboru o velikosti 500 MB bylo opět vytíženo pouze jedno jádro procesoru. Průměrná přenosová rychlost byla 99,8 Mbps, což znamená, že linka o rychlosti 100 Mbps byla využita z 99,8 %. Průměrné využití jádra procesoru bylo přitom pouze kolem 8 %. Celkově bylo odesláno 63999 datagramů za 42,04 sekund. Z celkového počtu přenesených datagramů nebyl žádný zahozen. Průměrná spotřeba byla 2644,62 mW. Z naměřených hodnot lze usoudit, že vývojový kit ZC702 zvládá přenos maximální rychlosti bez většího zatížení procesoru.



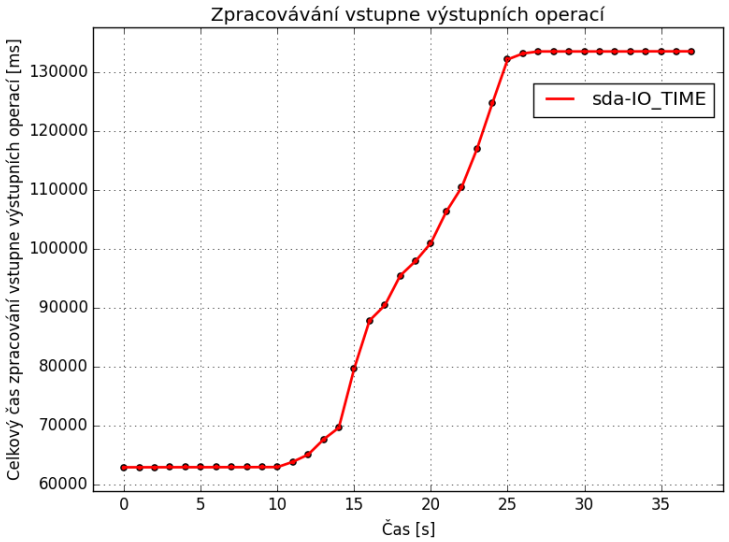
Obrázek 6.9: Využití procesoru při zasílání dat 100 Mbps linkou



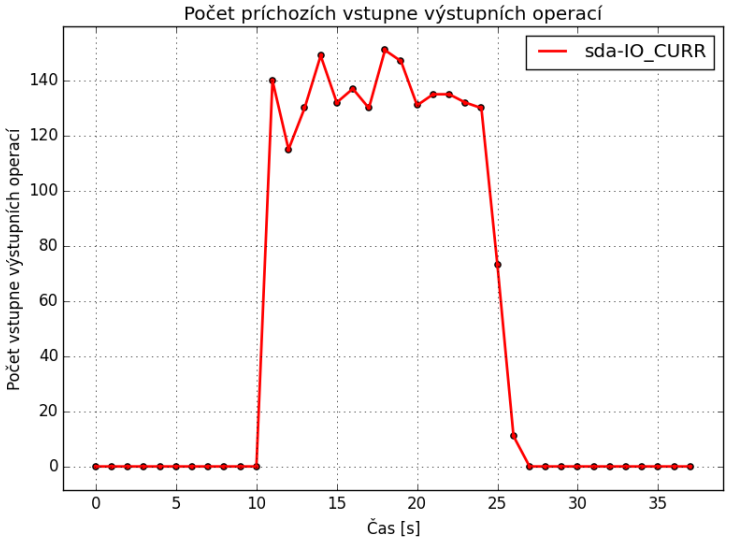
Obrázek 6.10: Spotřeba při zasílání dat 100 Mbps linkou

6.6 Měření statistik vstupně výstupních operací na osobním počítači

Generování vstupně výstupních operací začalo přibližně v 10. sekundě a trvalo 15 sekund. Za celou dobu běhu nástroje stress bylo vygenerováno 1846 vstupně výstupních operací, což je průměrně 123 operací za sekundu. Času potřebný na vykonávání přicházejících vstupně výstupních operací lze vyčíst z grafů. Za dobu měření bylo zapsáno 13 541 882 sektorů (6,933 GB) a přečteno 64 sektorů (32 KB).

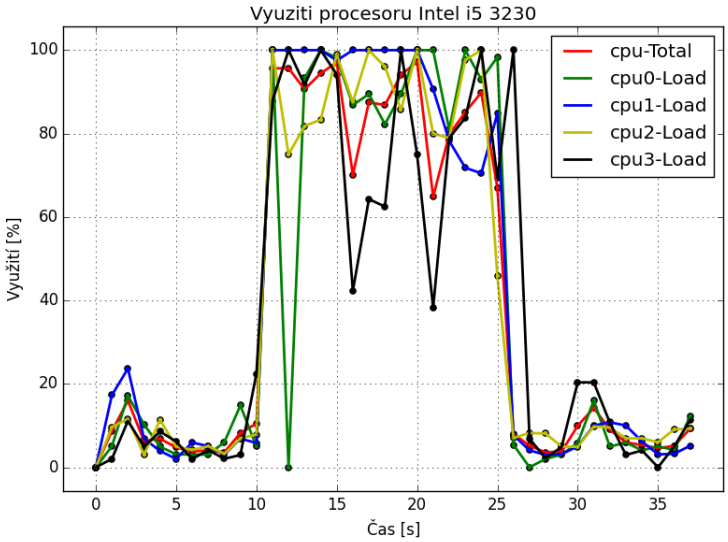


Obrázek 6.11: Celkový čas strávený vykonáváním vstupně výstupních operací

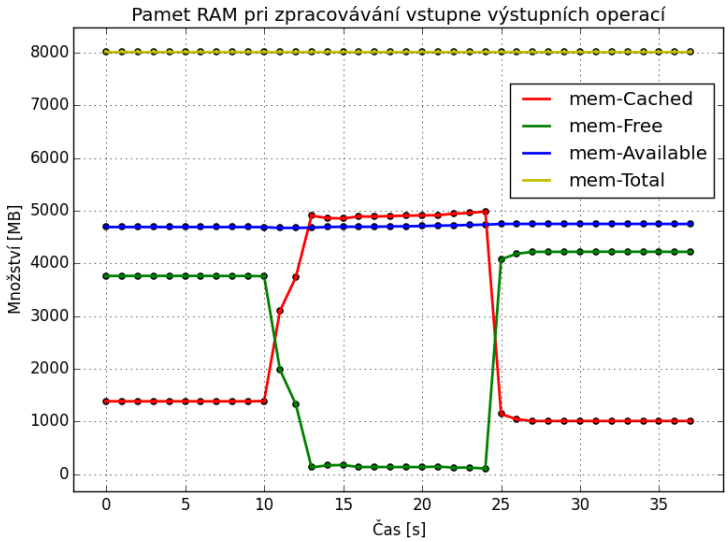


Obrázek 6.12: Počet příchozích vstupně výstupních operací

Jelikož množství dostupné paměti mem-Available nezohledňuje paměť, která je využívána jako vyrovnávací paměť mem-Cached, její hodnota se po dobu měření nemění. Naopak množství dostupné paměti mem-Free zohledňuje množství vyrovnávací paměti. Linux zapisuje data pomocí označení odpovídajících stránek ve stránkovací vyrovnávací paměti. Označené stránky ve stránkovací vyrovnávací paměti jsou poté periodicky zapisovány na disk. Z tohoto důvodu se množství vyrovnávací paměti během zapisování a dat postupně zvyšuje až na 4986,9 MB. Tento nárůst paměti způsobí vyčerpání volné paměti memFree, která je ve 25 sekundě 104,3 MB. Z grafů lze vyčíst, že průměrné vytížení procesoru v době zpracovávání vstupně výstupních operací bylo 86,38 %. Lze tedy předpokládat, že procesor byl využit průměrně z 86 % kvůli nedostatku paměti pro vyrovnávací paměť souborového systému.



Obrázek 6.13: Zatížení procesoru při zpracováváním vstupně výstupních požadavků



Obrázek 6.14: Využití paměti RAM při zpravování vstupně výstupních operací

Kapitola 7

Závěr

Tato bakalářská práce představuje možnosti sběru informací a jejich následné zpracování na systémech postavených na linuxovém jádře. První část práce se zabývá popisem vestavěného operačního systému Linux, který je důležitý pro následné řešení bakalářské práce. Dále je popsán vývojový kit ZC702, na které budou prováděny testy. Stěžejní částí této práce je prostudování možností měření zatížení systémů, na základě kterých je prováděno měření.

Praktická část práce využívá poznatků popsané teorie pro navržení knihovny umožňující sběr informací o zatížení systému. Naimplementovanou knihovnu lze dynamicky rozšiřovat pomocí měřících a výstupních modulů, které mohou být napsány uživateli. Z toho vyplývá, že je možné měřit veličiny, které nebyly v době psaní práce brány v úvahu. Pro interpretaci výsledků měření byla naimplementována aplikace pro vykreslování grafů. Pomocí vytvořené knihovny bylo provedeno několik měření, která názorně ukazují souvislosti mezi měřenými veličinami, například vliv síťového provozu na zatížení procesoru, vliv teploty a spotřeby zařízení na zatížení procesoru.

Výsledná knihovna umožňuje měření zatížení systému a mohla by tak být použita na mnoha vestavěných zařízeních, které jsou postaveny na linuxovém jádru. Při měření všech veličin s intervalem jedné sekundy bylo průměrné využití procesoru Zynq-7000 v rozmezí 2 až 4 procent.

Vývojem knihovny se budu nadále zabývat. V budoucnu bych chtěl rozšířit možnosti měření a zpracování dat základních modulů, vytvořit aplikaci pro sběr a zpracování výsledků měření s využitím této knihovny, snížit využití procesoru při měření, přidat moduly zajišťující detekování neoptimálních nastavení operačního systému Linux nebo linuxového jádra. Dále bych chtěl odstranit konfigurační soubor knihovny a upravit základní měřicí modul spotřeby, teploty, proudu a napětí tak, aby ke svému běhu nepotřeboval nástroj lm-sensors.

Literatura

[1] *Introduction To The PMBus*[online]. 2005 [cit. 13-05-2015].
URL http://www.pmbus.org/Assets/PDFS/Public/introduction_to_pmbus.pdf

[2] *Daemon Definition*[online]. 2005 [cit. 17-01-2015].
URL http://www.linfo.org/daemon.html

[3] *Process State*[online]. 2008 [cit. 14-05-2015].
URL
http://operatingsystemconcepts.blogspot.cz/2008/08/process-state.html

[4] *Processes in an Uninterruptible Sleep (D) State*[online]. 2012 [cit. 12-05-2015].
URL https://www.novell.com/support/kb/doc.php?id=7002725

[5] *Linux process states*[online]. 2012 [cit. 14-04-2015].
URL https://idea.popcount.org/2012-12-11-linux-process-states/

[6] *PageReplacementDesign* [online]. 2013 [cit. 18-01-2015].
URL http://linux-mm.org/PageReplacementDesign

[7] *lm sensors*[online]. 2014 [cit. 04-05-2015].
URL https://wiki.archlinux.org/index.php/Lm_sensors

[8] *Zynq-7000 AP SoC Low Power Techniques*[online]. 2014 [cit. 09-05-2015].
URL http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Low+Power+Techniques+part+2+-+Measuring+ZC702+Power+using+TI+Fusion+Power+Designer+Tech+Tip

[9] *PowerTOP*[online]. 2014 [cit. 27-04-2015].
URL https://wiki.linaro.org/WorkingGroups/PowerManagement/Resources/Tools/PowerTop

[10] *Linux kernel profiling with perf*[online]. 2015 [cit. 14-05-2015].
URL https://perf.wiki.kernel.org/index.php/Tutorial

[11] *Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit*[online]. 2015 [cit. 15-05-2015].
URL http://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html

[12] *Wind River Linux 7*[online]. 2015 [cit. 16-05-2015].
URL http://www.windriver.com/products/product-overviews/WR-Linux-7-Product-Overview.pdf

[13] *lm-sensors*[online]. [cit. 07-05-2015].
URL http://www.lm-sensors.org/wiki/ProjectInformation

[14] *The toolchain*[online]. [cit. 13-05-2015].
URL http://eleceng.dit.ie/frank/arm/BareMetalSTM32F0Discovery/toolchain.html

[15] *Enabling Smarter Systems*[online]. [cit. 15-05-2015].
URL http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/smarter-system.html

[16] *Embedded Debian Project*[online]. [cit. 16-05-2015].
URL http://www.emdebian.org/

[17] *Cortex-A9 Technical Reference Manual*[online]. [cit. 24-04-2015].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388f/Bcgddibf.html

[18] *PowerTOP*[online]. [cit. 25-04-2015].
URL https://01.org/powertop/

[19] Anne, S. K.: *Understanding free command in Linux/Unix*[online]. 2013 [cit. 15-04-2015].
URL http://www.linuxnix.com/2013/05/find-ram-size-in-linuxunix.html

[20] Bob Smith, G. P., John Hardin: *Linux Appliance Design: A Hands-on Guide to Building Linux Appliances*. No Starch Press, 2007, ISBN 1593271409, 9781593271404.

[21] Christias, P.: *top - display Linux tasks*[online]. 2002 [cit. 30-04-2015].
URL http://unixhelp.ed.ac.uk/CGI/man-cgi?top

[22] Cox, A.: *LINUX ALLOCATED DEVICES (2.6+ version)*[online]. 2009 [cit. 12-05-2015].
URL https://www.kernel.org/doc/Documentation/devices.txt

[23] Daniel Pierre Bovet, M. C.: *Understanding the Linux Kernel*. O'Reilly Media, Inc., 2002, ISBN 0596002130, 9780596002138.

[24] Eric Rosebrock, E. F.: *Linux, Apache, MySQL a PHP: instalace a konfigurace prostředí pro pokročilé webové aplikace*. Grada Publishing a.s., 2005, ISBN 8024712601, 9788024712604.

[25] Furber, S. B.: *ARM System-on-chip Architecture*. Pearson Education, 2000, ISBN 0201675196, 9780201675191.

[26] Henry Ware, F. F.: *vmstat - Report virtual memory statistics*[online]. 1994 [cit. 15-04-2015].
URL http://linuxcommand.org/man_pages/vmstat8.html

[27] Juliano: *Understanding the Linux load average*[online]. 2009 [cit. 23-04-2015].
URL http://juliano.info/en/Blog:Memory_Leak/Understanding_the_Linux_load_average

[28] Korsgaard, P.: *About Buildroot*[online]. [cit. 15-05-2015].
URL http://buildroot.uclibc.org/about.html

[29] Manuel Jiménez, I. C., Rogelio Palomera: *Introduction to Embedded Systems: Using Microcontrollers and the MSP430*. Springer Science & Business Media, 2013, ISBN 1461431433, 9781461431435.

[30] Marchand, J.: */proc/diskstats*[online]. 2008 [cit. 14-01-2015].
URL
https://www.kernel.org/doc/Documentation/ABI/testing/procfs-diskstats

[31] Mauerer, W.: *Professional Linux Kernel Architecture*. John Wiley & Sons, 2010, ISBN 1118079914, 9781118079911.

[32] Mehta, A.: *Optimizing Linux Memory Management for Low-latency / High-throughput Databases*[online]. 2013 [cit. 15-05-2015].
URL http://engineering.linkedin.com/performance/optimizing-linux-memory-management-low-latency-high-throughput-databases

[33] Napster: *Zombie Process*[online]. 2011 [cit. 28-04-2015].
URL http://www.geekride.com/zombie-process-defunct-linux/

[34] Polanka, J.: *Linuxové distribuce*[online]. [cit. 16-05-2015].
URL http://home.zcu.cz/~jpolanka/SemprZPS/web/distribuce.htm

[35] Reddy, M.: *API Design for C++*. Elsevier, 2011, ISBN 0123850045, 9780123850041.

[36] Redhat: */proc/meminfo*[online]. [cit. 18-01-2015].
URL https://www.centos.org/docs/5/html/5.1/Deployment_Guide/s2-proc-meminfo.html

[37] Seibold, T. B. B. B. J. N. S. F. S.: */proc/stat*[online]. 2009 [cit. 21-04-2015].
URL http://www.mjmwired.net/kernel/Documentation/filesystems/proc.txt

[38] Siebenmann, C.: *How the Linux kernel divides up your RAM*[online]. 2012 [cit. 09-05-2015].
URL https://utcc.utoronto.ca/~cks/space/blog/linux/KernelMemoryZones

[39] Terrehon Bowden, J. N., Bodo Bauer: *THE /proc FILESYSTEM*[online]. 2009 [cit. 9-04-2015].
URL https://www.kernel.org/doc/Documentation/filesystems/proc.txt

[40] Thakur, A.: *Embedded Linux : Understanding the embedded linux*[online]. [cit. 12-01-2015].
URL http://www.engineersgarage.com/articles/what-is-embedded-linux

Seznam použitých zkratek

API - Application Programming Interface
CPU - Procesor
DMA - Direct Memory Access
FPGA - Programovatelné hradlové pole
GNU GPL - GNU General Public License
I/O - Vstup/Výstup
I2C - Sériová sběrnice I2C
ISA - Paralelní sběrnice I2C
MIPS - Označení pro procesor bez automaticky organizovaného zřetězeného zpracování
MMU - Jednotka správy paměti
PID - Jednoznačný identifikátor procesu v systému
PL - Programmable Logic
RAM - Random Access Memory
ROM - Read Only Memory
RISC - Označení pro procesory s redukovanou instrukční sadou
RRD - Round Robin Database
SPI - Synchronní sériová sběrnice SPI
SoC - System On Chip
WiFi - Bezdrátová komunikace

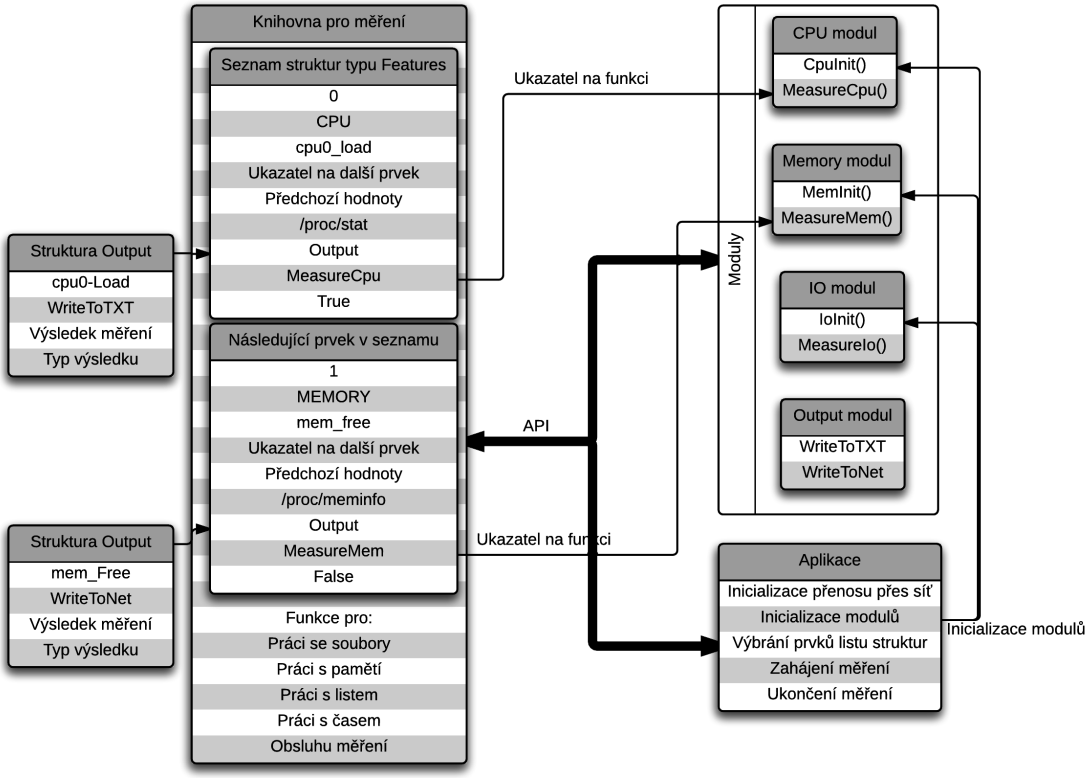
Seznam příloh

- Detailní struktura navrhnuté knihovny
- Obsah CD

Seznam obrázků

Příloha A

Struktura Knihovny



Obrázek A.1: Struktura knihovny

Příloha B

Obsah CD

Přiložené CD obsahuje náskedující soubory a adresáře:

- Složka Technicka zprava - Obsahuje zdrojové soubory a vygenerované PDF technické zprávy.
- Složka Daemon - Obsahuje zdrojový kód služby pro přijímání výsledků měření pomocí TCP/IP socketů.
- Složka Graf - Obsahuje skript k vykreslení grafů.
- Složka Moduly - Obsahuje zdrojové kódy základních měřicích modulů.
- Složka Uzivatelska aplikace - Obsahuje zdrojový kód jednoduché aplikace, pracující s knihovnou.
- Složka Spustitelne soubory - Obsahuje zkompilovanou knihovnu a uživatelský program.
- core.h - Hlavičkový soubor knihovny pro měření zatížení systému.
- core.c - Zdrojový soubor knihovny pro měření zatížení systému.
- libcore.conf - Konfigurační soubor knihovny.
- make.sh - Skript v bashi pro zkompilování knihovny a aplikace.
- OutputModule.h - Hlavičkový soubor základního výstupního modulu.