Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Bakalářská práce

Zařízení pro dokumentaci a analýzu forenzních stop z oblasti informačních technologií

František Doule

Školitel: Ing. Jaroslav Kothánek, Ph.D.

České Budějovice 2013

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě – v úpravě vzniklé vypuštěním vyznačených částí archivovaných Přírodovědeckou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

22. dubna 2013 **František Doule**

Bibliografické údaje

DOULE, F.: [Zařízení pro dokumentaci a analýzu forenzních stop z oblasti informačních technologií, bakalářská práce] – 32 stran, České Budějovice, 2013: Jihočeská univerzita, Přírodovědecká fakulta, Ústav fyziky a biofyziky. Školitel a vedoucí bakalářské práce: Ing. Jaroslav Kothánek, Ph.D.

DOULE, F.: [Device for the documentation and analysis of forensic traces of Information Technology, Bc. Thesis, in Czech] – 32 p, České Budějovice, Czech Republic, 2013: The University of South Bohemia, Faculty of Science, Department of Physics and Biophysics. Tutor and Supervisor : Ing. Jaroslav Kothánek, Ph.D.

Abstrakt

Cílem této bakalářské práce je seznámit se s problematikou zásad prvotních úkonů při zajišťování digitálních stop pro účely forenzního zkoumání a navrhnout jednoduché elektronické zařízení, které bude schopno provádět základní operace při provádění bitové kopie laickou veřejností z řad policie tak, aby byla eliminována možnost poškození dat (přepsání zájmových dat atd.)

Abstract

The aim of this bachelor thesis is to study the issues of primary actions in securing digital traces for forensic investigation and design simple electronic device that will be able to perform basic operations on the implementation image by the general public of the police, so as to eliminate the possibility of data corruption (overwriting the data of interest, etc.)

Poděkování

Děkuji vedoucímu bakalářské práce panu Ing. Jaroslavu Kothánkovi, Ph.D. za cenné rady a připomínky. Dále děkuji panu Jaroslavu Královi za pomoc při kompletaci zařízení na duplikaci dat.

Obsah

Úvod

V dnešní době se člověk žijící v civilizovaném světě prakticky dnes a denně setkává s digitálními technologiemi, jako jsou platební karty a terminály, osobní počítače (PC, Notebook), „chytré" telefony (smart phone), „chytré" televize (smart tv), nebo datová média, jako například USB flash disk, paměťové karty atd. Čím častěji se tyto technologie využívají, tím častěji se pak stávají terčem zločinů a jsou samozřejmě také zneužívány jako nástroj zločinu – takový zločin je pak nazýván cybercrime. Proto bylo potřeba vytvořit vědu (nebo metody), jež toto prostředí digitálních dat bude zkoumat. [12]

Rekonstrukce zločinů v oblasti digitálních, nebo chcete-li informačních a komunikačních technologiích probíhá obdobně, jako když dojde například k vraždě nebo k vloupání. Orgány činné v trestním řízení budou zkoumat a ohledávat místo činu, budou se snažit nalézt viníky a sbírat důkazy. Prostředkem k tomuto sběru informací bude (u vraždy nebo vloupání) například fotografování místa činu, snímání otisků prstů, sběr genetického i jiného materiálu a shromažďování důkazů. V případech, kde je potřeba zkoumat počítače a jiná digitální zařízení či pouze data, je pak prostředkem ke sběru důkazů vědní obor, který se jmenuje forenzní analýza digitálních dat.

1 Forenzní analýza digitálních dat

Forenzní analýza (anglicky forensics[1]) je investigativní postup používaný k vyšetřování různých subjektů [12], tento postup by pak měl vést k prokázání identity osob, pravosti listin a podobně, aby takto prokázané skutečnosti mohli být použity jako důkazní materiál například v trestním řízení. Při těchto analýzách se pak aplikují takzvané forenzní vědy [2], jako jsou například:

- Daktyloskopie - metoda určování identity pomocí otisků prstů
- Forenzní antropologie - zabývá se zejména identifikací koster, lebek a chrupu
- Forenzní balistika - snaží se identifikovat zbraň, hlavně pomocí stop na projektilu
- Forenzní chemie - identifikuje různé látky, například jedy
- Forenzní medicína čili soudní lékařství - snaží se stanovit čas a příčiny smrti nebo poškození
- Forenzní psychologie - posuzuje psychologii podezřelých osob
- Forenzní genetika - identifikuje osoby či stopy z místa činu na základě analýzy DNA
- Forenzní fotografie - identifikuje osoby podle jejich ručního písma

Mezi tyto vědy pak také patří forenzní analýza digitálních dat. Pojem forenzní analýza digitálních dat nelze, jak uvádí Ken Zatyko[24], jednoduše definovat. Mnoho nadšených akademiků se o to v minulosti pokoušelo, ale například po prvním otevřeném workshopu pořádaném organizací DFRWS[2] v roce 2001 vznikla kolekce návrhů na definici. Tu později publikoval ve svém článku Brian Carrier[5] a jeho výsledný návrh definice čítal v angličtině 54 slov. Ken Zatyko[24] došel k závěru že by bylo možné tuto definici zkrátit na cituji:

> „*Digital Forensics Science: The application of computer science and investigative procedures for a legal purpose involving the analysis of digital evidence (information of probative value that is stored or transmitted in binary form) after proper search authority, chain of custody, validation with mathematics (hash function), use of validated tools, repeatability, reporting, and possible expert presentation.*"

Což volně přeloženo znamená, že forenzní analýza digitálních dat je věda, která se zabývá aplikací počítačových věd a vyšetřovacích procedur na získávání a analyzování důkazů, které jsou uloženy, nebo se vyskytují v digitální (binární) podobě.

[1] z latinského jazyka - soudní

[2] DFRWS je organizace která se věnuje sdílení informací o forenzních analýzách digitálních dat (http://www.dfrws.org/)

1.1 Zločiny v digitálním světě

Někdy se zločiny ve světě digitálních technologií nazývají jako cybercrime. Obecné povědomí je takové, že pod tímto pojmem se skrývají zločiny, které se odehrávají jen v prostoru počítačů, počítačových a mobilních sítích (tzv. cyberspace), ať už je to šíření malware[3], nebo krádež cenných firemních dat, nebo třeba kyberšikana[4]. Pojem cybercrime ale zahrnuje více než tento prostor. Výzkum digitálních dat je často potřeba v případech, ve kterých to není tak očekávatelné. Když například policie ohledává byt, ve kterém pobýval drogový dealer a ve kterém se našel kontraband, měl by policii zajímat i počítač, který v této situaci může vypadat relativně nevinně. Došlo k případu, kdy byly v takovém počítači nalezeny kompletní informace o zákaznících, dodávkách a dalších zajímavých věcech (v současnosti asi běžná situace). Takže vidíme, že počítač nebo jiná zařízení s obsahem digitálních dat nemusejí být v roli oběti a také nemusejí mít nic společného se zločiny páchanými na takovýchto zařízeních nebo pro dosahování vyšších cílů skrze tato zařízení [12].

1.2 Vyšetřování

Vyšetřování tedy probíhá stejně jako u jakéhokoliv jiného zločinu, nebo trestného činu, to znamená že se musí nalézt a zajistit co nejvíce důkazů, které by mohly usvědčit pachatele trestného činu, nebo viníka nějakého incidentu. V oblasti digitálních zařízení je to ale o to složitější, že se v těchto zařízeních často takové důkazy vyskytují ve formě dočasných dat, které se po vypnutí takového zařízení ztrácejí. Jedná se zejména o dočasná data v paměti RAM. Tato data se musí pokud je to možné zajistit ještě za běhu systému - musí se udělat takzvaný *dump* paměti RAM, nebo pokud se jedná o paměť typu DRAM[5], tak je možné data z paměti získat ještě krátkou dobu po vypnutí zařízení (zhruba 30 sekund). Jestliže se krátce po vypnutí tento typ paměti ochladí na -50^{o}C (například tekutým dusíkem), vydrží v ní data ještě o něco déle - několik minut a to i po vyjmutí ze slotu pro paměť [11].

Dále je nutné zajistit důkazy z pevných paměťových médií (HDD, SSD, SD karta, USB Flash disk). U těchto paměťových medií se postupuje tak, že se nejprve provede duplikace tohoto média a případné důkazy se pak vyhledávají v pořízené kopii. Je to kvůli ochraně všech dat (a tedy důkazů) na originálním médiu proti nechtěnému smazaní nebo přepsání [12].

Je také nutné počítat s tím, že soubory (nebo celé paměťové médium) mohou být nějakým způsobem zašifrované, potom jsou důkazy z těchto dat vyšetřovatelům skryty (pokud se jim je nepodaří rozšifrovat). Většinou to platí pouze pro případy, kdy je počítač nástrojem zločinu. Pokud organizace šifruje data, kvůli jejich ochraně před narušiteli, nemá důvod, proč by vyšetřovatelům šifrovací klíč neposkytla.

Samotnou duplikaci systému lze uskutečnit několika způsoby. Jednou z možností je odpojit zkoumaný disk, který se bude duplikovat, a připojit ho k rozhraní jiného stroje, nebo k nějakému zařízení které tuto duplikaci provede - takovéto zařízení je právě

[3]Malware je počítačový program určený ke zneužití počítačového systému. Pod souhrnné označení malware se zahrnují počítačové viry, trojské koně, spyware a adware.

[4]Kyberšikana je druh šikany, který využívá elektronické prostředky, jako jsou mobilní telefony, sociální sítě a podobně.

[5]DRAM (Dynamic Random Access Memory) je druh počítačové paměti, která uchovává data v podobě elektrického náboje v kondenzátoru který odpovídá parazitní kapacitě řídící elektrody (Gate) tranzistoru typu MOS.

předmětem této bakalářské práce. Další možností může být například zkopírování dat ze zkoumaného počítače přes síť [13].

2 Softwarové prostředky pro forenzní analýzu digitálních dat

I přesto že forenzní analýza digitálních dat je pořád poměrně mladý obor, již existuje velké množství softwaru, který lze v tomto oboru použít. Jako příklad můžu uvést **EnCase Forensic** od firmy Guidance Software[6], nebo **Evidence Center** od firmy Belkasoft[7]. Takovéto produkty můžou ale dosahovat značně vysoké ceny. Krajské ředitelství policie Moravskoslezského kraje, jak je uveřejněno ve smlouvě na serveru www.policie.cz[8], například zaplatilo 178.560Kč za 2 licence softwaru EnCase Forensic v. 7 s roční podporou.

Já se v této bakalářské práci budu zabývat jen softwarem svobodným, čili dostupným zdarma včetně zdrojových kódů. Většina těchto volně dostupných programů je naprogramovaná pro systémy UNIXového typu, jakým je například Gnu/Linux (viz. 3.2.1 a 3.2.2).

Jak je napsáno výše (1.2) u paměťových médií se postupuje tak, že se nejprve provede duplikace tohoto média (většinou do souboru). Programy, které toto dokáží a přitom splňují podmínku, že jsou svobodné a zdarma jsou například[10]:

- GNU dd
- GNU ddrescue
- d3dd
- dcfldd
- aimage
- libewf

Obraz zájmového média, který jsme získali jedním z těchto programů, pak můžeme analyzovat balíkem nástrojů **The Sleuthkit**[9], jehož autorem je Brian Carrier. The Sleuthkit umožňuje analyzovat různé typy souborových systémů, bez ohledu na platformu. Nástroje v tomto balíku je možné rozdělit do několika kategorií:

- Řízení média - **mmls, mmcat, mmstat**
- Souborový systém - **fsstat**
- Jména souborů("Human Interface") - **fls, ffind**
- Meta data (inode) - **icat, ils, ifind, istat**
- Obsah (data) - **blkcalc, blkcat, blkls, blkstat**

[6] http://www.guidancesoftware.com/encase-forensic.htm
[7] http://forensic.belkasoft.com/en/bec/en/evidence_center_for_law_enforcement.asp
[8] www.policie.cz/soubor/smlouva-sw-pro-forenzni-analyzu-pdf.aspx
[9] http://www.sleuthkit.org

- Nástroje žurnálovacího systému - **jcat, jls**
- Nástroje pro fyzický disk - **disk_stat, disk_reset**

Sleuthkit ještě obsahuje celou řadu utilit, které není možné takto rozdělit do kategorií. Jsou to:

- **sorter** - roztřídí alokované a nealokované soubory podle typu (spustitelný, textový, obrázek, atd.)
- **img_cat** - umí oddělit meta data od originálních dat v získaném obrazu média
- **img_stat** - poskytuje informace o obrazu média
- **hfing**
- **sigfind**
- **mactime** - vytvoří časovou osu změn souboru
- **srch_strings**

2.1 Program GNU dd

Program dd čte vstupní data (ve výchozím nastavení ze standardního vstupu) blok po bloku se stanovenou velikostí bloku (výchozí velikost bloku je 512 bajtů) a zapisuje data opět blok po bloku na standardní výstup. Někteří lidé míní že „dd" znamená „Destroy Disk" nebo „Delete Data" (česky zničit disk nebo smazat data), protože dd je nástroj který se používá pro čtení a zápis v celém prostoru paměťového média, včetně systémových částí jako je MBR[10], nebo novější GPT[11] a nebo souborový systém v jednotlivých oddílech pevného disku. Při špatném použití příkazu dd může tedy velmi rychle dojít ke ztrátě dat.

2.2 Program dcfldd

Program dcfldd je rozšířená verze programu GNU dd o funkce pro forenzní analýzu digitálních dat, umožňuje tedy provádět stejné operace jako GNU dd a zároveň rozšiřuje schopnosti GNU dd, o tyto funkce:

- hašování vstupních dat už při probíhajícím kopírování
- informování uživatele o průběhu kopírování
- možnost rychle „ovzorkovat" médium
- schopnost rozdělit výstup do více souborů
- možnost kopírovat data do více výstupů souběžně

[10] Master Boot Record

[11] GUID Partition Table (GPT) je v informatice standard pro popis členění pevného disku na oddíly. Nahrazuje starší tabulku MBR, která neumožňuje použít disk větší, než 2 TiB. GPT je součástí standardu EFI od firmy Intel, který by měl nahradit v IBM PC kompatibilních počítačích klasický BIOS[3].

Nejdůležitější vlastností programu dcfldd pro forenzní zkoumání je schopnost provádět hašování vstupních dat při probíhajícím kopírování. Hašování dat se provádí pomocí hašovací funkce, použít lze algoritmy MD5, SHA-1, SHA-256, SHA-384 nebo SHA-512.

2.2.1 Hašovací funkce

Hašovací funkce jsou základem pro moderní kryptografii. Používají se všude tam kde, je potřeba ověřit integritu dat, nebo zajistit autenticitu dokumentu (digitální podpis).

Je to matematický algoritmus, pomocí kterého se ze vstupního řetězce libovolné délky vygeneruje výstupní krátký řetězec tzv. otisk nebo haš[12], který jednoznačně odpovídá vstupnímu řetězci. Pravděpodobnost že haš náhodně vybraného řetězce se bude shodovat s nějakou konkrétně zadanou haší je $\frac{1}{2^n}$ kde n je délka haše[15], u haše délky 128 bitů by to tedy byla pravděpodobnost přibližně $2.39 * 10^{-39}$.

Hašovací funkce se dělí na *„bezklíčové"*, které mají na vstupu pouze řetězec k zahašování a na *„klíčové"*, které mají na vstupu kromě řetězce ještě druhý parametr, kterým je tajný klíč. U programu dcfldd se používá hašovací funkce jen ke kontrole integrity dat a jedná se tedy o bezklíčovou hašovací funkci.

2.2.2 Příklady použití programu dcfldd

Používání programu je stejné jako GNU dd, jedná se o program s CLI[13] rozhraním, který se ovládá pomocí různých přepínačů.

Standardně čte program data ze standardního vstupu (/dev/stdin) a zapisuje na standardní výstup (/dev/stdout). Přesměrování vstupu a výstupu se provádí pomocí parametrů *if* (input file) a *of* (output file). Jednoduché zkopírování jakéhokoliv souboru se pak tedy provede příkazem:

```
dcfldd if=vstupnisoubor of=vystupnisoubor
```

Jelikož v Linuxu (viz. 3.2.2) se všechno chová jako soubor, můžeme stejným příkazem zkopírovat celý pevný disk, nebo jakékoliv jiné paměťové médium, pro který existuje linuxový ovladač (modul jádra). Takový příkaz by tedy vypadal například takto:

```
dcfldd if=/dev/sdb of=/mnt/obraz.img
```

Tím jsme zkopírovali celý obsah pevného disku /dev/sdb do souboru /mnt/obraz.img (za předpokladu, že /mnt je přípojný bod pro nějaké jiné paměťové médium). Kopírování by v tomto případě probíhalo po blocích o velikosti 512B. Pokud bychom chtěli ještě vygenerovat haš disku /dev/sdb, vypadal by příkaz takto:

```
dcfldd if=/dev/sdb of=/mnt/obraz.img hash=SHA1 hashlog
   =/mnt/hash.txt
```

Argument *hash* říká programu dcfldd, že má vygenerovat haš vstupního souboru, přičemž standardním algoritmem je MD5. V příkazu výše je zvolen algoritmus SHA1. Parametr *hashlog* pak přesměruje výsledný haš do souboru (u tohoto příkladu je to soubor /mnt/hash.txt).

Program dcfldd má celou řadu dalších parametrů, z nichž za zmínku, z hlediska forenzní analýzy, stojí ještě *noerror*, kterým programu říkáme, že pokud narazí na nečitelný blok dat, tak tento blok má ignorovat a pokračovat dalším blokem, místo toho

[12] Z anglického hash

[13] Command Line Interface - textové uživatelské rozhraní

aby skončil s chybou (návratový kód „1”). Ostatní parametry pak lze nalézt v manuálových stránkách programu.

2.3 LibEWF

Jedním z nejpoužívanějších souborových formátů v oboru digitálních forenzních analýz je takzvaný „Expert Witness format” **EWF**, který používají komerční produkty jako například EnCase. Knihovna LibEWF vydaná pod licencí LGPL[14] obsahuje nástroje pro práci s tímto typem souborů.[10]

- **ewfinfo** - zobrazuje metadata obsažená v EWF souborech
- **ewfverify** - kontroluje integritu dat v EWF souborech
- **ewfexport** - exportuje data z EWF souboru do surového bitového souboru a odstraní všechna metadata
- **ewfacquire** - zapisuje data z paměťových médií nebo souborů do EWF souboru
- **ewfacquire** - zapisuje data ze standardního vstupu do EWF souboru

2.4 The Autopsy Forensic Browser

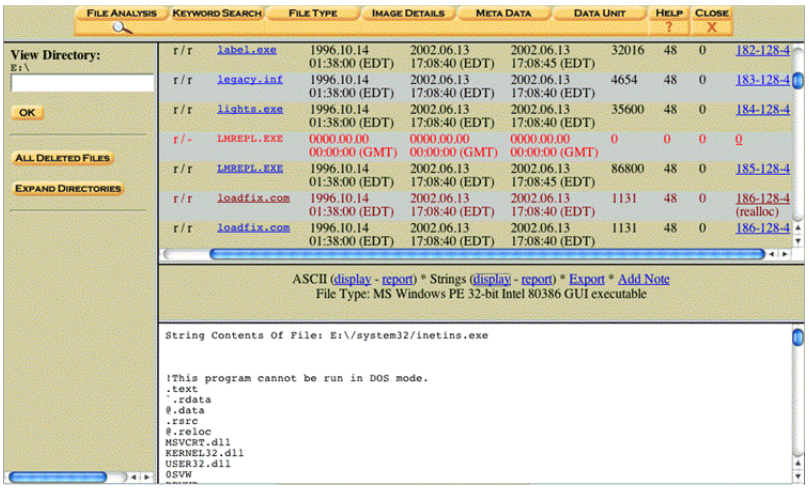
Jedná se o grafickou nadstavbu pro balík Sleuthkit. To znamená že umí provést hloubkovou analýzu různých druhů souborových formátů (NTFS, FAT, HFS+, Ext3 a UFS) a některých jiných diskových svazků (například ISO9660). The Autopsy Forensic Browser je v současnosti dostupný ve dvou verzích:

- Verze 2 je napsaná v Perlu[15] a je primárně určená pro UNIXové systémy. Jedná se vlastně o webový server, takže uživatelské rozhraní se ovládá pomocí webového prohlížeče (Firefox, Internet Explorer, Midori, atd.). Verze 2 je šířena pod licencí GNU GPL v.2[16].
- Verze 3 vychází z verze 2 a je kompletně přepsaná do jazyka Java. Tato verze je dostupná pro Microsoft Windows. Uživatelské rozhraní je podobné souborovému manažeru, jakým je v Microsoft Windows Průzkumník, snaží se tak být intuitivnější na používání, takže by analýzu disků a souborů měl zvládnout i méně technicky zdatný uživatel (vyšetřovatel). Tato verze je šířena pod licencí Apache v.2[17].[6]

2.5 BEViewer a bulk_extractor

Program **bulk_extractor** je napsaný v C++ a umí analyzovat různé informace o souborech a adresářích, které se vyskytují v obrazu disku (ale i přímo na disku, nebo na CD), bez toho aniž by k tomu potřeboval údaje z tabulky souborového systému. Tyto

[14] http://www.gnu.org/licenses/lgpl.html
[15] interpretovaný programovací jazyk
[16] http://www.gnu.org/licenses/gpl.html
[17] http://www.apache.org/licenses/LICENSE-2.0



Obrázek 2.1: The Autopsy Forensic Browser verze 2, převzato z [6]

informace pak zapisuje do jednotlivých souborů (tzv. features), nazvaných podle toho jaké informace se v nich vyskytují.

Tak například v souboru **domain.txt** budou vypsány všechny názvy internetových domén, které bulk_extractor v harddisku objeví. Dalším příkladem může být soubor s názvem **email.txt**, který bude obsahovat všechny emailové adresy a na něj navazující **rfc822.txt**, jenž obsahuje hlavičky emailových zpráv (datum, předmět, ID). Dále tento program umí extrahovat z disku slova použitelná pro lámání hesel hrubou silou. Tato slova jsou uložena v souboru příznačně nazvaném **wordlist.txt** .

Protože bulk_extractor ignoruje souborový systém, může jednotlivé části disku analyzovat paralelně. V praxi to probíhá tak, že je disk rozdělen do malých částí (16MB), které jsou pak zkoumány jedna po druhé, tak že pro každou část je spuštěn nový proces na dostupném jádře procesoru. To znamená, že pokud analýza bude probíhat na počítači s osmi-jádrovým procesorem, tak celý průběh bude 8 krát rychlejší než na počítači s jedno-jádrovým procesorem. Díky této vlastnosti je rychlost analyzování ve srovnání s jinými forenzními nástroji poměrně vysoká a to při zachování velké důkladnosti.

BEViewer je program napsaný v jazyce Java, má grafické uživatelské rozhraní a je určen na prohlížení informací, které do jednotlivých souborů vyextrahoval bulk_extractor. Také obsahuje rozhraní pro spouštění skenování pomocí bulk_extractoru.[4]

Tento program jsem testoval na systému MS Windows. Nejprve jsem pomocí programu GNU dd přepsal USB flashdisk nulami, poté jsem udělal image tohoto disku (pomocí programu dcfldd) a ten zanalyzoval pomocí bulk_extractor. Podle očekávání nebylo nalezeno nic. Poté jsem USB flashdisk naformátoval a nakopíroval na něj nějaké soubory (emailové zprávy, obrázky atd.) ty jsem zase smazal a znovu naformátoval USB flashdisk (FAT32). Opět jsem udělal image a tentokrát analýza proběhla úspěšně a informace o souborech, které ještě stále byli fyzicky přítomny na flashdisku (byla pouze přepsána tabulka FAT), se objevily v jednotlivých „features" souborech.

Tyto dva programy jsou vydávány pod záštitou *Naval Postgraduate School*, což je jedna ze součástí *U.S. Department of Navy*, tedy amerického námořnictva. Jediné licenční omezení, jak jsem pochopil, se týká toho, že tato organizace nenese žádnou odpovědnost za používání tohoto softwaru a výsledky tímto softwarem získané.

Na základě toho by bylo možné v budoucnu mnou navrhované zařízení na duplikaci médií rozšířit o funkce tohoto programu.

Obrázek 3.1: Forenzic Duplicator, převzato z [8]

3 Návrh jednoduchého elektronického zařízení

Zařízení, které bude provádět bitovou kopii paměťových médií, by mělo mít uživatelské rozhraní maximálně pochopitelné běžnými uživateli, kteří by pomocí něj měli být schopni vytvořit bitovou kopii - obraz disku (disk image). Jako inspiraci jsem si vzal Forensic Duplicator (obrázek 3.1) od firmy Digital Intelligence, jehož cena je $1.249 (při současném kurzu dolaru je to zhruba 23.000Kč).

Jako základ pro návrh takového zařízení jsem zvolil desku s procesorem Intel Atom, který je cenově dostupný a přitom má dostačující výkon. Původně jsem se domníval že použiji počítač Raspberry pi,[18] což je počítač o velikosti kreditní karty, který pohání procesor s architekturou ARM o frekvenci 700 MHz, ale má zásadní nedostatek - chybí mu SATA konektor (SATA revize 2.0 má maximální rychlost přenosu dat 3Gbit/s), má pouze dva USB 2.0 porty (maximální rychlost přenosu dat 480 Mbit/s). Po připojení dvou medií na porty USB je pořizování obrazu disku příliš pomalé. Je potřeba, aby alespoň médium na které se bude ukládat obraz (image) zájmového média, bylo připojeno přes nějakou rychlejší sběrnici než je USB 2.0

Jako uživatelský interface jsem se rozhodl použít obyčejný dvouřádkový LCD displej, který se mi zdá na toto jednoúčelové zařízení plně dostačující. Tento displej bude připojen na paralelní port základní desky. Pro tento port jsem se rozhodl proto, že zapojení se mi zdálo nejjednodušší (LCD má rozhraní určené pro paralelní komunikaci) a tím pádem nejlevnější - nejsou potřeba žádné převodníky na sériovou komunikaci (RS232 nebo USB). K paralelnímu portu jsou ještě připojena čtyři ovládací tlačítka.

3.1 Základní deska

Jako základ pro kopírovací zařízení jsem použil základní desku Intel D2500HN (obr.3.2) formátu Mini-ITX[19] s dvoujádrovým procesorem Intel Atom D2500 (1.86 GHz, 1MB cache), čipsetem Intel NM10, s 2x SO-DIMM DDR3 1066/800 MHz (až 4GB) a integrovanou grafickou kartou INTEL 3600. Deska dále obsahuje 1x PCI, 1x COM (Serial), 1x LPT, 2x SATA 3Gb/s, 4x USB 2.0, VGA, GLAN, pasivní chlazení pro úsporu energie. Procesor na této desce má udávanou spotřebu 10W, což oproti spotřebě výše zmiňovaného Raspberry Pi (3W), není takový rozdíl. Obrovskou výhodou oproti Raspberry Pi je ale přítomnost dvou SATA konektorů a to, že tato deska podporuje na těchto konektorech takzvaný „Hot Plug", tedy připojování a odpojování periférií za běhu. To je možné díky rozhraní AHCI (Advanced Host Controller interface).

[18] http://www.raspberrypi.org/
[19] Základní deska počítače o rozměru 17x17cm

Obrázek 3.2: Základní deska Intel D2500HN

3.1.1 AHCI

AHCI je patentováno firmou Intel. Jedná se abstrakční vrstvu mezi SATA řadičem a sběrnicí procesoru (PCI nebo PCIe). Před příchodem AHCI nebylo možné využívat všech možností (a nebo pouze jen ve specifických případech) rozhraní SATA, z nichž pro mě je nejdůležitější možnost za běhu systému odpojovat a připojovat pevné disky (Hot Plug), ať už se jedná o HDD nebo SSD [23]. V BIOSu základní desky D2500HN je *AHCI Hot Plug* standardně vypnuto. Pokud se tato možnost povolí funguje pak připojování a odpojování disků za běhu systému naprosto bez problémů.

3.1.2 Zdroj pico PSU

Zdroj pico PSU vyrábí firma Mini-Box[20], která se specializuje na zařízení s mini-ITX základními deskami. Pico PSU je v současnosti nejmenší mini-ITX zdroj na světě (je jen o málo větší než 20 nebo 24 pinový konektor ATX). Není to ale zdroj v pravém slova smyslu, jedná se v podstatě o DC-DC převodník, který mění vstupní stejnosměrné napětí 12 V na výstupních 5 a 12V. Potřebujeme tedy k němu mít ještě externí transformátor (AC-DC), který převede střídavé napětí 230V na stejnosměrných 12V (viz. obrázek 3.3).

V současnosti je nejvýkonnějším zástupcem těchto zdrojů **picoPSU-160XT**, který může být ve špičce zatížen až 200W. Já jsem ve svém návrhu použil nejlevnější(1100Kč včetně externího adaptéru) pouze **60W** picoPSU. Tento zdroj jsem testoval při běžícím systému se dvěma harddisky 3,5" a vše fungovalo bez problémů.

[20] http://www.mini-box.com/

Obrázek 3.3: ATX zdroj pico PSU a napájecí adaptér

3.2 Operační systém

Už od začátku jsem věděl, že jako operační systém budu používat GNU/Linux, protože všechny programy, které jsou svobodně dostupné (zdarma včetně zdrojových kódů) pro účely forenzního zkoumání digitálních dat jsou určeny pro systémy Unixového typu, kterým je i GNU/Linux. Konkrétně jsem se rozhodl pro distribuci Debian GNU/Linux, který je vyvíjen čistě komunitně, to znamená, že nemá v pozadí žádnou firmu, která by platila jeho vývoj.

Debian je svobodný operační systém, respektive distribuce operačního systému, postavený na svobodných programech z rodiny GNU (viz. 3.2.1) a na jádře Linux, jak se píše níže Linux by byl jako samostatné jádro k „ničemu", kdyby neobsahoval základní programy a nástroje. V naprosté většině takzvaných „Linuxových distribucí" (Ubuntu, RedHat, ArchLinux...) jsou těmito základními nástroji programy z projektu GNU. Podle serveru gnu.org[9] je to ve skutečnosti tak že například v roce 2008 zabíral v „hlavním" úložišti distribuce gNewSense Linux 1,5% a GNU balíčky 15%. Správné pojmenování takovýchto distribucí by tedy mělo být GNU/Linux.

Operační systém Debian GNU/Linux je jakožto svobodný nebo chcete-li open-source[21] software dostupný zdarma a je portován[22] na různé platformy (x86, ARM, Sparc...). Kromě těchto charakteristik má Debian ještě jednu vlastnost, která je pro mě zásadní - repozitář[23], který obsahuje obrovské množství již zkompilovaného softwaru. Programy jsou zde dostupné jako balíčky s příponou .deb (například firefox.deb), které pak lze velice jednoduše nainstalovat a aktualizovat pomocí „balíčkovacího" sytému APT (advanced package tool).

GNU/Linux je od základu navržený podle standardu POSIX[24], který určuje jak se mají operační systémy chovat například při správě procesů, nebo jak pracovat se soubory a také jaké základní programy má operační systém obsahovat. Všechny aplikace, které jsou pak vyvíjeny podle tohoto standardu, je možné zkompilovat a spustit na jakémkoliv operačním systému, který splňuje POSIX.

Celý systém je nainstalován na flashdisk a běží tedy z tohoto flashdisku. Je to jednak kvůli velmi malým rozměrům oproti pevným diskům, ale hlavně kvůli tomu aby zůstali volné oba dva SATA porty na základní desce (viz. obrázek 3.2).

[21] Software s otevřeným zdrojovým kódem, často šířený zdarma (http://opensource.org)

[22] Portace softwaru (též portování softwaru) jsou v informatice úpravy softwaru za účelem jeho fungování na jiné počítačové platformě (ať již hardwarové nebo softwarové).

[23] Zdroj softwaru na internetu dostupný přes http nebo ftp

[24] Portable Operating System Interface, přenositelné rozhraní pro operační systémy, standardizované jako IEEE 1003 a ISO/IEC 9945.

3.2.1 GNU

Na webu gnu.org[9] se píše:

> „*GNU je operační systém unixového typu, který je svobodným softwarem. Jméno GNU je rekurzivní akronym pro GNU's Not Unix! (GNU není Unix). Operační systém unixového typu je soubor softwaru tvořený jednak aplikacemi, knihovnami a vývojářskými nástroji a jednak programem pro přidělování zdrojů a komunikaci s hardwarem, kterému se říká jádro.*
>
> *Hurd, vlastní jádro GNU, čeká ještě nějaká cesta než bude připraven na běžné použití. Proto se dnes GNU obvykle používá s jádrem zvaným Linux. Této kombinaci říkáme operační systém GNU/Linux. GNU/Linux používají miliony lidí, ačkoli ho mnozí omylem nazývají Linux*".

Cílem projektu GNU bylo a je vytvořit operační systém podobný proprietárnímu[25] Unixu.

Jak dále uvádí server gnu.org[9] - Linux (viz. 3.2.2) jádro operačního systému (kernel), je program v systému, který se stará o to, aby aplikace (uživatelské programy) měly přístup k hardwarovým prostředkům (například k pevnému disku) bez toho, aby vývojáři těchto programů museli programovat své vlastní ovladače. Jádro je základ systému, ale samo o sobě by bylo k ničemu. Má smysl pouze jako součást kompletního operačního systému, ve kterém jsou alespoň základní programy pro ovládání systému, jako například:

- Bash - Bourne Again Shell, je kompatibilní s unixovým „sh" a obsahuje mnohá rozšíření známé z „csh" a „ksh".
- GNU coreutils - balík programů a utilit (ls, cat, dd, mv, cp, mkdir, a mnoho dalších)
- Nano - jednoduchý textový editor, klon editoru Pico. Je zaměřen na řešení několika „problémů" s Picem, přitom ale zachovává vzhled a snadnost použití originálu.

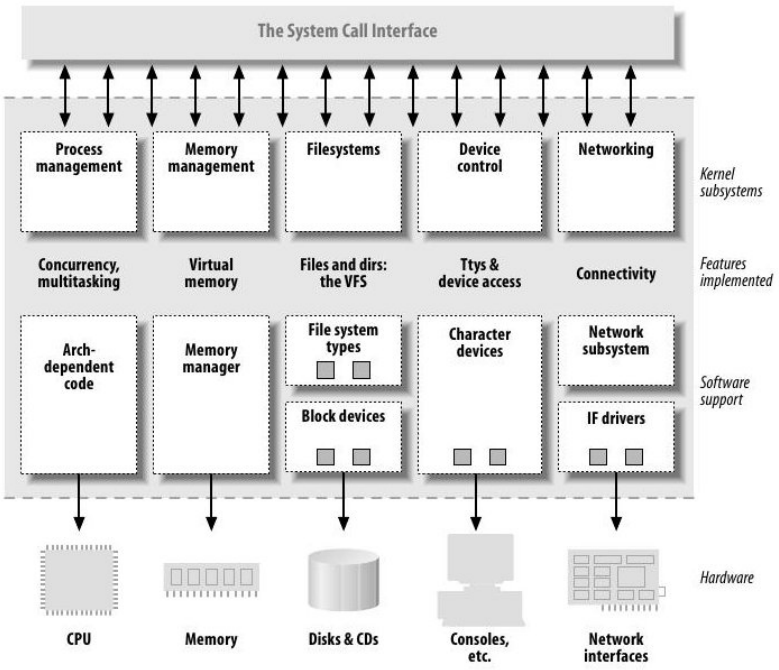
Jednou z mnoha výhod jádra, tedy Linuxu je fakt, že je šířené pod licencí GNU GPL (stejně jako všechen software z projektu GNU), která povoluje nejen jeho svobodné používání, ale také možnost studovat a upravovat jeho zdrojový kód.

3.2.2 Linux

Linux je jádro (kernel), které se stará o to, aby všechny aplikace z uživatelského prostoru měly přístup ke všem zdrojům (ať už je to výpočetní čas procesoru, paměť, síťové připojení, nebo přístup k souboru) prostřednictvím standardních systémových volání (na obrázku 3.4 The System Call Interface). Činnost kernelu můžeme zhruba rozdělit na tyto části[7]:

- *SPRÁVA PROCESŮ* - Jádro má za úkol spouštět a ukončovat jednotlivé procesy a obsluhovat jejich komunikaci s okolním světem (input/output). Komunikace mezi různými procesy (pomocí signálů, nebo pojmenovaných rour) je základem fungování celého systému a i to má na starosti jádro.

[25] Prorpietární sofware je software s uzavřeným kódem, který je dostupný pouze ve formě binárního kódu



Obrázek 3.4: Schématický model jádra, převzato z [7]

- *SPRÁVA PAMĚTI* - Správně nastavená pravidla pro používání operační paměti jsou naprosto zásadní věcí pro celkový výkon systému. Linux vytváří virtuální paměťový prostor pro všechny procesy. Jednotlivé části jádra pak tento prostor využívají, tak že interagují se subsystémem řízení paměti (na obrázku 3.4 Memory management).

- *SYSTÉM SOUBORŮ* - Linux je založen na konceptu souborového systému, všechno v Linuxu je reprezentováno souborem. Jádro vybuduje nad nestrukturovaným hardwarem strukturovaný souborový systém, vznikne tak abstrakční vrstva - stromová struktura souborů, jejíž vrchol se nazývá *root (kořen)* a označuje se znakem lomeno */*. Všechny další soubory a adresáře (což je jen speciální druh souboru) jsou pak zařazeny pod tímto kořenem - například */home/pepa* by mohl být domovskou složkou uživatele jménem „pepa". Linux podporuje mnoho typů souborových systémů (EXT2/3/4, btrfs, ReiserFS, XFS, VFAT, NTFS), každý z nich používá odlišné fyzické uspořádání dat na datovém médiu.

- *OVLÁDÁNÍ ZAŘÍZENÍ* - Téměř všechny systémové operace jsou nakonec napojeny na nějaké zařízení. Všechna zařízení, snad kromě procesoru a paměti, obsluhují malé kousky kódu (programy), kterým se říká ovladače (device driver). Jádro v sobě musí obsahovat tyto ovladače pro všechna zařízení, se kterými pracuje (pevný disk, klávesnice a myš, síťová karta, grafická karta, atd.)

- *SPRÁVA SÍTĚ* - Síťová komunikace musí být spravována operačním systémem (jádrem), protože pakety přicházejí asynchronně a jádro musí konkrétnímu procesu předat jen ty pakety, které mu patří. Kromě toho má jádro v sobě přímo implementováno směrování (routing) a address resolution protokol (ARP).

Velkou výhodou Linuxu je fakt, že je možné do jádra za běhu přidávat různé funkcionality. To je možné provést tak, že se přilinkuje do jádra potřebný kód, takzvaný

modul jádra, který požadovanou funkcionalitu implementuje. Modulů existuje několik typů. Kromě ovladačů různých zařízení (*ath5k*, *nv*, *battery*) a souborových systémů (ext2, ext3, fuse) ještě uvedu například paketový filtr *ip_tables*, nebo různé moduly určené pro virtualizaci procesů mezi hostem a hostovaným systémem (*kvm*, *vmnet*, *kqemu*). Všechny moduly, které jsou přítomny v běžícím jádře je možné vypsat pomocí příkazu **lsmod**. Nový modul se do jádra přidá příkazem **insmod** nebo **modprobe** (ten přidá modul včetně všech ostatních modulů na nichž závisí). Odstranit modul z jádra lze pomocí příkazu **rmmod** nebo **modprobe -r**.

Linux rozlišuje dvě hlavní třídy zařízení: **bloková zařízení** (například disky) vyznačující se možností přímého náhodného přístupu a obvykle bloky pevné délky a **znaková zařízení** (tiskárny, klávesnice, monitory). Každé zařízení je reprezentováno souborem v adresáři `/dev` a charakterizováno hlavním a vedlejším číslem.

```
$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 29. bře 08.27 /dev/sda
```

Na ukázce výše je zobrazeno blokové zařízení (**b** na začátku výpisu) `/dev/sda` s hlavním číslem **8** a vedlejším číslem **0**. Hlavní číslo se používá pro adresaci ovladače uvnitř jádra a vedlejší číslo pro rozlišení mezi zařízeními ovládané tímto ovladačem (driverem).

3.2.3 Udev

V mém návrhu zařízení na duplikaci paměťových médií využívám možnosti za běhu systému připojovat zařízení a to konkrétně paměťová média (i pevné disky přes rozhraní SATA). Tuto vlastnost Linuxu od verze 2.6 obstarává **udev**. Od této verze jádra může být adresář `/dev` umístěn v paměti RAM (souborový systém TmpFS) a speciální soubory pro všechna zařízení v tomto adresáři vytvoří program udev.

Systém udev má dvě fáze v první *(coldplug)* inventarizuje stávající zařízení a vytvoří jim speciální soubory v `/dev`. V druhé fázi pak reaguje na *hotplug* události od jádra a poté vytváří nebo ruší soubory v `/dev` podle toho jestli je připojeno nebo odpojeno nové zařízení [14]. Pokud tedy po zapnutí je v systému přítomen jeden disk, který reprezentuje soubor `/dev/sda` a poté je za běhu připojen do systému druhý pevný disk, tak udev vytvoří soubor `/dev/sdb`.

Výhodou systému udev je, že běží v uživatelském prostoru a je proto možné pro něj pomocí konfiguračních souborů vytvářet pravidla (například jak bude pojmenovávat zařízení atd.). Tyto soubory se umísťují do adresáře `/etc/udev/rules.d`.

3.3 Paralelní port

Pavel Tišnovský [22] ve svém článku o paralelním portu píše, že paralelní port je typickým zástupcem paralelního přenosu dat, který je na osobních počítačích (IBM PC) také znám pod zkratkou LPT (Line Print Terminal). Z toho vyplývá, že paralelní port byl původně používán pro komunikaci s tiskárnou (pouze jedním směrem - od PC k tiskárně). Poté byl rozšířen o další takzvané módy, aby bylo možné komunikovat i s jinými periferiemi a oběma směry. V roce 1994 byl standardizován [18] pod IEEE 1284. Tento standard definuje 5 komunikačních módů činnosti:

1. SPP mode - Compatibility Mode (Centronics mode)
2. Nibble Mode

3. Byte Mode
4. EPP Mode (Enhanced Parallel Port)
5. ECP Mode (Extended Capabilities Mode).

Základním módem je SPP mode, v něm je definován jen přenos typu forward, tedy od PC k periferii, a rychlost přenosu se pohybuje jen do 150KB/s. Tento mód je řízen třemi základními osmibitovými registry [18]:

- Datový registr (Data Register) - určený pro zápis vysílaných dat
- Stavový registr (Status Register) - určený pro čtení stavů na lince
- Řídící registr (Control Register) - určený pro řízení periferie, které jsou data posílána

Tyto registry jsou mapovány do I/O oblasti procesoru a mají následující adresy (platí pro LPT1):

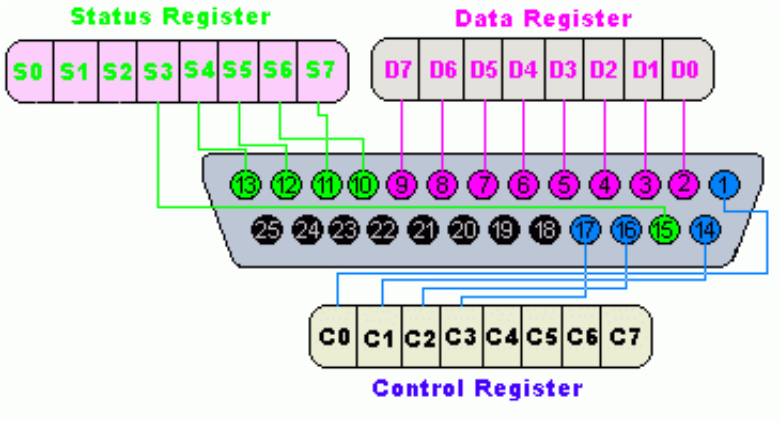
- Datový registr - 0×378
- Stavový registr - 0×379
- Řídící registr - 0×37a

Na paralelní port je možné připojit v podstatě jakékoliv zařízení, i vlastní výroby. Pavel Tišnovský dále uvádí:

> *„Připojení vlastních zařízení na paralelní port je zpravidla velmi jednoduché a i způsob ovládání paralelního portu (většinou se jedná o přímé ovládání jednotlivých vstupních a výstupních pinů) nepředstavuje ani pro začínající programátory žádný větší problém.”*

Standardní paralelní port používá konektor s dvaceti pěti piny uspořádanými do dvou řad, který je označován jako DB25. Význam jednotlivých pinů tohoto konektoru je vyobrazen na obrázku 3.5. Pomocí pinů 2 - 9 (Data register) se posílají vlastní data (paralelně) – počet osm zde samozřejmě není náhodný, odpovídá jednomu bajtu (původně jednomu znaku či řídicímu kódu poslanému na tiskárnu). Dále jsou zde řídicí piny 1,14,16,17 (Control register), pomocí nichž může počítač řídit veškerou komunikaci se zařízením. Piny 10,11,12,13,15 jsou stavové, kterými naopak zařízení (typicky tiskárna) posílá počítači zpět informaci o svém stavu, připravenost pro příjem dalšího bajtu (znaku) atd. Přes poslední skupinu pinů se připojují zemnící vodiče (Ground, GND) – propojovací kabely bývají konstruovány tak, aby každému datovému vodiči odpovídal jeden vodič zemnící.

Na paralelní port není vyvedeno žádné napájecí napětí, připojené zařízení tedy musí být napájeno „externě”, například přímo z počítačového zdroje.



Obrázek 3.5: Konektor DB25 - význam jednotlivých pinů, převzato z [22]

3.3.1 Ovládání paralelního portu

V operačním systému Linux(viz. 3.2.2) je možné ovládat paralelní port z uživatelského prostoru[22] bez nutnosti psát si vlastní ovladač (modul do jádra). Je pouze nutné, aby program, který přistupuje k paralelnímu portu běžel pod uživatelem, který má potřebná práva. Jsou v podstatě dvě možnosti jak přistupovat k paralelnímu portu z uživatelského prostoru. Jeden způsob je využít nízko úrovňových maker obsažených ve standardní knihovně C (GNU C library). Jedná se především o funkce ioperm(), inb(), outb(), iopl(). Druhým způsobem je využití obecného ovladače (modulu jádra) pro paralelní port - PPDEV. Tento modul vytvoří soubor /dev/parportX[26], se kterým je pak možné pracovat pomocí standardních funkcí pro práci se soubory, respektive se souborovými deskriptory (file descriptor).

Já jsem se rozhodl program pro ovládání celého zařízení napsat v jazyce Python (viz. 3.5.1), pro který existuje modul na ovládání paralelního portu. Tento modul pracuje právě nad souborem /dev/parportX a potřebuje tedy, aby byl v jádře načten modul PPDEV (příkaz `modprobe ppdev`).

3.4 Modul LCD2L4P02A

Jako interface jsem zvolil modul LCD2L4P02A (obr.3.6) od firmy MLAB[27], který obsahuje dvouřádkový displej [17] s Hitachi řadičem HD44780 (ten se používá téměř u všech řádkových LCD displejů). Tento modul je primárně určen jako interface pro různé mikroprocesory.

Řadič HD44780 obsahuje paměť RAM (nazývanou DD RAM) pro zapisované znaky o kapacitě 2x40 znaků. Dále obsahuje paměť RAM (nazývanou CG RAM) pro definici 8 uživatelských znaků. LCD displej se ovládá zápisem do řídících registrů (RS=0), do paměti pro zobrazovaný text (DD RAM), nebo do paměti definice znaků (CG RAM). Z pamětí DD RAM a CG RAM je možné rovněž číst, ale tato funkce nebývá potřebná a je možné trvale čtení deaktivovat (tzn. RW trvale uzemnit). Komunikace s řadičem probíhá po paralelní sběrnici o šířce 8 nebo 4 bitů, tato šířka se nastavuje po zapnutí displeje pomocí příslušné řídící instrukce. To, že má řadič na všech pinech platná data, mu „sdělíme" tím, že pošleme kladný pulz na pin **E** (enable pin).

[26] X je pořadové číslo paralelního portu - pro LPT1 je to /dev/parport0
[27] http://www.mlab.cz

Pin	Jméno signálu	Směr přenosu	Registr	Invertovaný bit?
1	nStrobe	Out	Control-0	ano
2	Data0	In/Out	Data-0	ne
3	Data1	In/Out	Data-1	ne
4	Data2	In/Out	Data-2	ne
5	Data3	In/Out	Data-3	ne
6	Data4	In/Out	Data-4	ne
7	Data5	In/Out	Data-5	ne
8	Data6	In/Out	Data-6	ne
9	Data7	In/Out	Data-7	ne
10	nAck	In	Status-6	ne
11	Busy	In	Status-7	ano
12	Paper-Out	In	Status-5	ne
13	Select	In	Status-4	ne
14	Linefeed	Out	Control-1	ano
15	nError	In	Status-3	ne
16	nInitialize	Out	Control-2	ne
17	nSelect-Printer	Out	Control-3	ano
18	Ground	x	x	x
19	Ground	x	x	x
20	Ground	x	x	x
21	Ground	x	x	x
22	Ground	x	x	x
23	Ground	x	x	x
24	Ground	x	x	x

Tabulka 3.1: Signály a registry paralelního portu, převzato z [22]

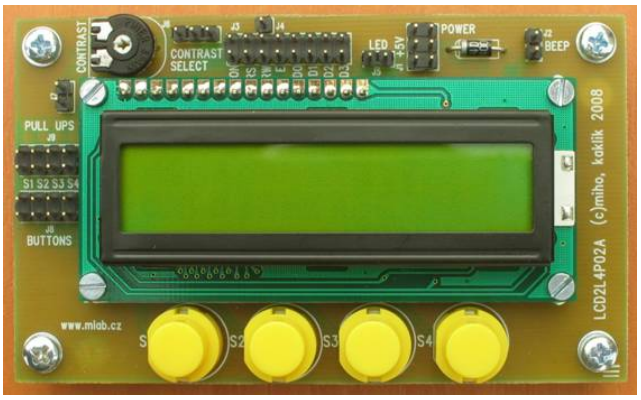
Jak je vidět na obrázku 3.7, jsou v modulu LCD2L4P02A jednotlivé piny LCD displeje vyvedeny na konektor J3, z toho pouze piny s označením DB4 - DB7 jsou datové. To znamená, že se s displejem musí v tomto modulu komunikovat 4 bitově, a tedy po zapnutí nastavit funkci displeje na přenos dat po čtyřech bitech. Kontrast je možné ovládat pomocí trimru P1, nebo pomocí programu podle toho, jak se propojí konektor J6.

Na obrázku 3.8 je schéma zapojení tlačítek. Ty jsou zapojeny na konektor J8 a do země. Při propojení konektoru J9 (pomocí jumperů) je možné na ně přes odpory R6 - R9 připojit napájecí napětí.

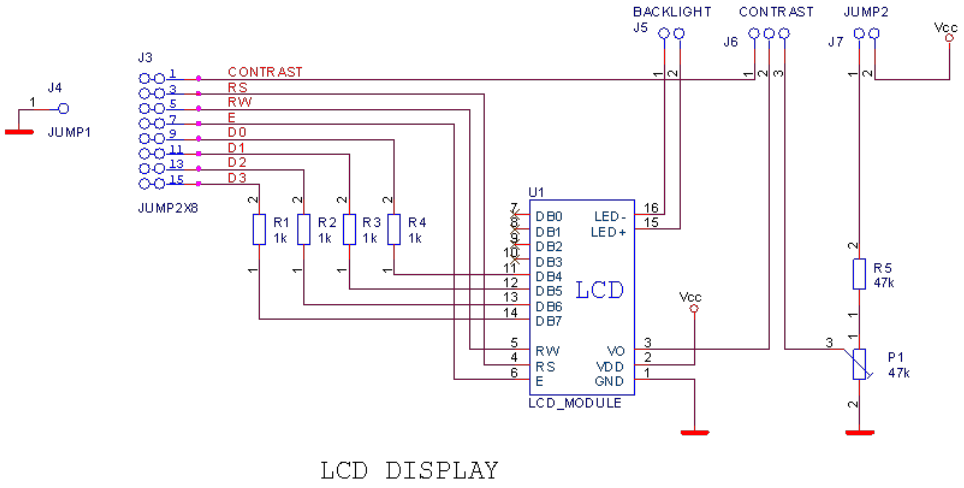
3.4.1 Připojení modulu LCD2L4P02A

Celý modul (tlačítka a LCD displej) je připojen na paralelní port, jak ukazuje obrázek 3.9 a napájen je přímo z počítačového ATX zdroje a to 5V. Piny datového registru paralelního portu (viz. obrázek 3.3) jsou připojeny na konektor J3 (viz. obrázek 3.7) a piny stavového registru paralelního portu jsou připojeny na konektor J8 (viz. obrázek 3.8), tedy k tlačítkům na modulu.

Na všech pinech stavového registru je ve standardním nastavení paralelního portu logická jednička (více než 3,5V), to znamená, že konektor J9 nesmí být propojen. Při stisknutí tlačítka je tak konkrétní pin uzemněn, a tím pádem je na něm v tu chvíli



Obrázek 3.6: Modul LCD2L4P02A, převzato z [17]



Obrázek 3.7: Schéma zapojení LCD displeje, převzato z [17]

logická nula (méně než 0,8V).

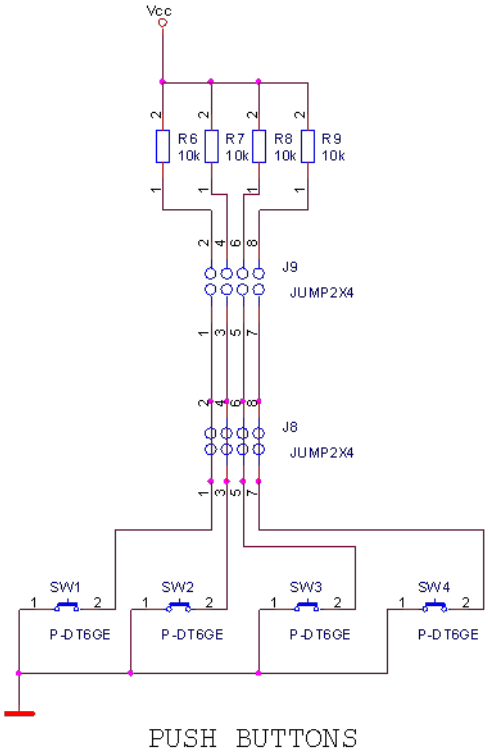
3.5 Řídící program

Jak už jsem psal výše, program, který řídí komunikaci s modulem MLAB (viz. 3.4) a také celý proces pořizování bitové kopie média, jsem napsal v jazyce Python a to ve verzi 2.7. Program nebyl testován ve verzi 3 a vyšší[28]. Tento jazyk jsem zvolil kvůli jeho dobrým vlastnostem (viz. 3.5.1). Řídící program běží pod právy uživatele **root**[29] tak, aby měl plný přístup ke všem zdrojům. Z hlediska „Linuxovské" filozofie (každý program by měl běžet pouze s takovými právy, jaká nezbytně potřebuje) by to mohlo být chápáno jako bezpečnostní riziko, ale vzhledem k tomu, že se jedná jen o jednoúčelové zařízení bez možnosti se jakkoliv do systému přihlásit, si myslím, že možnosti zneužití (například nějaké chyby v kódu) jsou naprosto minimalizované. Program se spouští automaticky po startu systému - pomocí souboru `/etc/rc.local` a pak běží v nekonečné smyčce až do vypnutí celého systému.

Pro lepší přehlednost zdrojového kódu jsem program rozdělil na tři části (tři Py-

[28]Python od verze 3 změnil syntaxi a není tedy od této verze kompatibilní s poslední stabilní verzí 2.7

[29]root - superuživatel - má v systému práva na zápis, nebo čtení do, nebo z každého souboru



Obrázek 3.8: Schéma zapojení tlačítek v modulu LCD2L4P02A, převzato z [17]

thonské moduly):

- menu.py - hlavní program (viz. 3.5.8)
- tools.py - pomocné funkce (viz. 3.5.7)
- mlab.py - ovládá modul MLAB LCD2L4P02A (viz. 3.5.6)

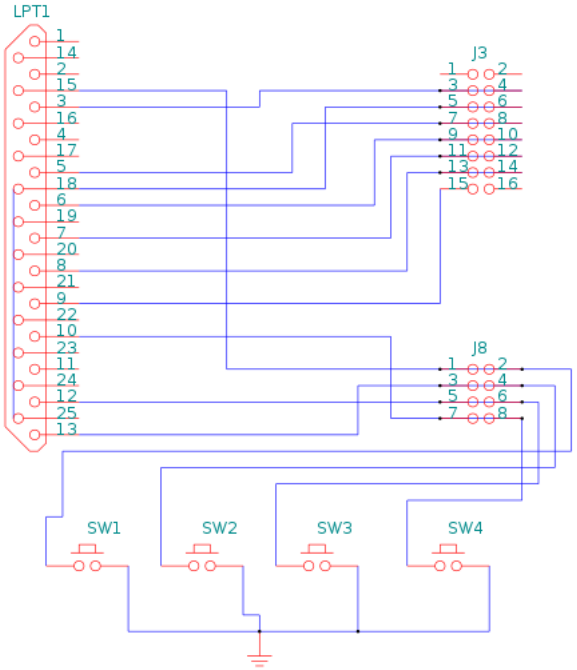
K tomu, aby mohl řídící program správně fungovat, je potřeba, aby v systému byl nainstalován interpret jazyka Python ve verzi nejvýše 2.7. Dále musí být v systému nainstalované podpůrné utility, které program používá. Jedná se o tyto programy:

dcfldd - viz. 2.2. Tento program používám k samotnému pořízení bitové kopie paměťového média a jeho haše.

gdisk - patří do balíku GPT fdisk. Je to program určený pro práci s GUID[30] partition table (GPT - tabulka rozdělení disku), se kterou naopak neumí pracovat starší *fdisk*. Ten umí pracovat pouze s MBR. Já *gdisk* používám pouze pro získání informací o rozdělení disku na jednotlivé partition - `gdisk -l`. Tento příkaz vypíše jaké je rozdělení disku i v případě, že je použit MBR.

sfdisk - program určený pro práci s tabulkou rozdělení disku na jednotlivé partition (stejně jako *fdisk* je omezen pouze na MBR). Je ale také schopen vypsat velikost celého disku v blocích, bez ohledu na použitý formát partition table (například `fdisk -s /dev/sda`), a to je právě jedna z vlastností, které využívám v řídícím programu. Standardní velikost bloku pro většinu programů (kromě *sfdisk* ještě

[30] Globally Unique Identifier



Obrázek 3.9: Připojení modulu LCD2L4P02A k paralelnímu portu

například *ls* a *df*) je 1024 bajtů. Vydělíme-li tedy počet bloků číslem 1024 (2^{10}), získáme velikost disku v megabajtech (MB). Druhou vlastností, kterou využívám, je schopnost vypsat všechna paměťová média, která jsou v systému přítomna.

hwinfo - tento program vypíše informace o veškerém hardwaru, který je přítomný v systému. Používám ho ke zjišťování informací o konkrétním paměťovém médiu - jeho název (např. Corsair_Flash_Disk), výrobce a sériové číslo (`hwinfo -disk /dev/sdb`).

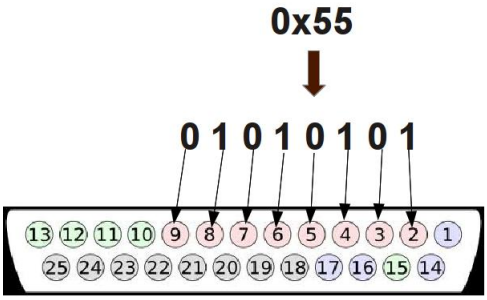
smartmontools - jedná se o dvě utility (*smartcl* a *smartd*) na řízení a monitorování `S.M.A.R.T.`[31] V řídícím programu je použitá jen utilita *smartcl* a to ke zjištění sériového čísla pevného disku, který je připojen přes rozhraní USB - například `smartctl -i /dev/sdc`.

Některé z výše uvedených programů jsou standardně v mnou zvoleném systému Debian GNU/Linux přítomny hned v základní instalaci, ostatní lze jednoduše doinstalovat z internetových repozitářů pomocí správce balíčků.

```
apt-get install dcfldd hwinfo smartmontools
```

Také je nutné mít nainstalovaný potřebný modul Pythonu pro práci s paralelním portem - pyParallel, který lze získat z webu vývojářů tohoto modulu[32], ale protože je také již připraven v repozitářích Debianu může být nainstalován (aktualizován) z něj. Interpret (běhové prostředí) jazyka Python včetně modulu pyParallel nainstalujeme pomocí příkazu:

[31] Self-Monitoring, Analysis and Reporting Technology System
[32] http://pyserial.sourceforge.net/pyparallel.html



Obrázek 3.10: Datový registr paralelního portu a jednotlivé piny.

```
apt-get install python python-parallel
```

3.5.1 Python

Python je plně objektový jazyk, který má čistou a snadno pochopitelnou syntaxi. Python je multiplatformní jazyk tzn., že aplikace v něm napsané fungují na operačních systémech Windows, Linux/Unix, OS/2, Mac, Amiga a dalších, některé verze Pythonu lze spustit i v prostředí .NET (IronPython) a Java virtual machine (Jpython)[20].

Je to interpretovaný programovací jazyk, který je zaměřen na rychlý vývoj aplikací a zároveň se snaží o co největší zachování výkonu. Existuje široká škála modulů a knihoven, které činí tento jazyk použitelný v nepřeberném množství situací s minimálním úsilím programátora. Mně se na Pythonu nejvíce líbí délka (resp. „krátkost") zdrojových kódů, ve srovnání s jinými jazyky.

3.5.2 Paralelního port v jazyce Python

Pro Python existuje modul, který se jmenuje pyParallel. Ten je součástí projektu pySerial. Do jádra musí být zaveden modul **ppdev** (`modprobe ppdev`) a pokud je v jádře modul **lp**, tak se musí z jádra odebrat (`rmmod lp`). S modulem pyParallel se pak v Pythonu pracuje tak, jak ukazuje následující příklad kódu.

```
1 import parallel
2 p = parallel.Parallel()
3 p.setData(0x55)
```

Nejprve tedy naimportujeme modul (řádek číslo 1), pak vytvoříme (inicializujeme) objekt třídy parallel.Parallel() a poté nastavíme datový registr paralelního portu (viz. 3.3) na hexadecimální hodnotu **0x55**, což je v binárním kódu **01010101**. Na paralelním portu pak bude nejméně významný bit (LSB) na pinu „2" a nejvíce významný bit (MSB) na pinu „9" (viz. obrázek 3.10).

Třída parallel.Parallel() kromě metody **setData()**, která nastavuje data v datovém registru paralelního portu, ještě obsahuje tyto metody:

setDataStrobe(level) - nastavuje signál "nStrobe" (tzn. pin 1, viz. tabulka 3.1) na požadovanou logickou úroveň (True - logická „1", False Logická „0")

setAutoFeed(level) - nastavuje signál "Linefeed" (pin 14) na požadovanou logickou úroveň

setInitOut(level) - nastavuje signál "nInitialize" (pin 16) na požadovanou logickou úroveň

getInSelected() - čte stav signálu "Select" (viz. tabulka 3.1), vrací *True* pokud má hodnotu logické „1", tzn. že na pinu 13 je napětí větší než 3.5V

getInPaperOut() - čte stav signálu "Paper Out" (pin 12)

getInAcknowledge() - čte stav signálu "nAck" (pin 10)

getInBusy() - čte stav signálu "Busy" (pin 11)

getInError() - čte stav signálu "nError" (pin 15)

3.5.3 Podprocesy v Pythonu

V řídícím programu jsou využívány externí systémové programy (gdisk, sfdisk, dcfldd, hwinfo, smartctl atd.). V jazyce Python lze spouštět externí programy několika způsoby, z nichž nejvíce podporovaným a doporučovaným je modul **subprocess**, který nahrazuje některé starší moduly a funkce. [21] Tento modul je tedy schopen spustit externí program (podproces) a připojit se k jeho vstupní, výstupní a chybové rouře (pipe). Také je schopen získat návratový kód podprocesu.

Modul subprocess poskytuje funkce pro jednodušší spouštění podprocesů, ale také obsahuje třídu subprocess.**Popen()**, která v sobě má další objekty a je určená ke komplexnějšímu zacházení s podprocesem. S třídou Popen() se pracuje jak ukazuje následující příklad:

```
1 import shlex, subprocess
2 prikaz = "cat /etc/hosts"
3 args = shlex.split(prikaz)
4 print args
5 p = subprocess.Popen(args)
```

První argument, který konstruktor třídy Popen() očekává, je „pythonský" seznam[33]. Proto je v příkladu použitý ještě modul **shlex** a jeho metoda **split()**, která jako argument bere řetězec znaků a vrátí seznam jednotlivých slov v tomto řetězci. Pokud tedy necháme vytisknout proměnnou **args** (`print args`), pak se na standardním výstupu v našem příkladu objeví:

```
["cat", "/etc/hosts"]
```

Tento seznam je pak předán jako první argument konstruktoru třídy Popen() a tím dojde ke spuštění nového procesu (cat), který na standardní výstup vypíše obsah souboru /etc/hosts. Konstruktor třídy Popen() má ještě další argumenty, které jsou přednastaveny na nějakou hodnotu (většinou na **None**[34]). Z těchto argumentů už uvedu pouze ty, které jsem použil ve svém programu. Jedná se o stdout a stderr, které se používají k přesměrování standardních výstupů podprocesu.

```
1 import shlex, subprocess
2 prikaz = "cat /etc/hosts"
3 args = shlex.split(prikaz)
4 print args
5 p = subprocess.Popen(args, stdout=subprocess.PIPE, stderr
    =subprocess.PIPE)
```

[33] seznam (angl. list) je v Pythonu datový typ

[34] None je v Pythonu speciální datový typ, který se používá pro naplnění proměnné, o které nevíme jakého bude typu a jakou bude mít hodnotu.

```
6 vystup = p.communicate(input)
7 print vystup[0]
```

V tomto příkladu jsme „otevřeli" roury (pipes) k programu cat (řádek 5). Dále je zde použita metoda **communicate()**, která interaguje s procesem (posílá data na stdin a čte stdout a stderr), čeká na jeho skončení a pak vrací n-tici[35]**(stdoutdata, stderrdata)**. Pokud tedy vytiskneme na terminál první prvek n-tice vystup (řádek 7) uvidíme na něm obsah souboru /etc/hosts.

Z třídy Popen() ještě v řídícím programu používám metodu **poll()**, která kontroluje jestli podproces běží, nebo jestli už byl ukončen, přičemž ale na rozdíl od metody communicate() nečeká na to až podproces skončí. Poté nastavuje a vrací atribut **Popen.returncode**, který má hodnotu None do té doby, než je přenastaven metodou poll() nebo wait() na hodnotu návratového kódu podprocesu.

3.5.4 Regulární výrazy v jazyce Python

V některých funkcích řídícího programu používám ke zpracování výstupu externích programů regulární výrazy. Proto bych se chtěl krátce zmínit o tom, jak je tato problematika řešena v Pythonu. Regulární výraz je způsob, jakým se na základě vzoru rozpoznávají nebo získávají data. Regulární řetězec může obsahovat metaznaky a speciální posloupnosti. Veškerá funkčnost spojená s regulárními výrazy se v Pythonu nachází v modulu **re**.[19]

Tento modul obsahuje poměrně velké množství funkcí a metod na zpracování regulárních výrazů. Já v řídícím programu používám pouze dvě funkce. Jednou z nich je re.**sub()**, která nahrazuje v řetězci hledaný regulární výraz zadaným textem a vrací změněný řetězec (pokud v něm nalezne hledaný výraz). Funkce očekává tři argumenty:

1. vzorek (pattern) - výraz který se má nahradit
2. náhrada (replacement) - řetězec kterým se nahradí vzorek (první argument)
3. řetězec (string) - řetězec ve kterém se provádí náhrada

Druhou funkcí je re.**search()**. Ta jako první argument očekává vzorek a jako druhý řetězec, který prohledá a pokud v něm najde hledaný vzorek vrátí objekt typu **MatchObject**, jinak vrací **None**.

```
1 import re
2 re.search('^M?M?M?$', 'M')
3 re.search('^M?M?M?$', 'MMMM')
4 re.search('^M?M?M?$', 'MCL')
```

Na ukázce je příklad použití funkce re.search(), přičemž prohledává řetězec '**M**' a hledaný vzorek je '**^M?M?M?$**'.

Tento vzorek má tři části. Znak ^ zajistí vazbu další části výrazu na začátek řetězce. Pokud bychom jej nepoužili, pak by vzorek pasoval nezávisle na tom, kde by se znaky M nacházely. Zápis M? odpovídá nepovinnému výskytu jednoho znaku M. A protože se opakuje třikrát, odpovídá výraz výskytu žádného až tří znaků M za sebou. Znak $ odpovídá konci řetězce. Když to dáme dohromady se znakem ^ na začátku, znamená to, že vzorek musí odpovídat celému řetězci. Znakům M nemůže žádný jiný znak předcházet a ani za nimi nemůže následovat.

[35] n-tice (angl. tuple) je datový typ podobný seznamu (list)

V této ukázce by tedy funkce re.search() na řádku 2 vrátila objekt MatchObject, protože vzorek „pasuje" na prohledávaný řetězec. Na řádcích 3 a 4 by ale funkce vrátila **None**.

3.5.5 Ovládání LCD displeje v Pythonu

Na webu projektu pySerial (pyParallel)[16] je dostupný příklad modulu (lcd.py) Pythonu na ovládání znakového LCD displeje s řadičem HD44780 . Tento modul jsem upravil a použil ve svém řídícím programu, protože je vydaný pod licencí GNU GPL (stejně jako celý Python), která to umožňuje.

Modul obsahuje třídu **EightBitIO()**, která jak název napovídá, je určená pro komunikaci s řadičem HD44780 přes paralelní port osmibitově a dále třídu **FourBitIO()** pro čtyřbitovou komunikaci.

Jelikož se v modulu MLAB (viz. 3.4) musí s řadičem HD44780 komunikovat čtyřbitově, používám proto v řídícím programu pouze třídu FourBitIO(). Tato třída využívá pro komunikaci s řadičem pouze datový registr paralelního portu (viz. 3.3). Konektor J3 modulu MLAB (viz. obrázek 3.7), na který jsou vyvedeny řídící signály a datové vodiče, je tedy připojen k paralelnímu portu tak, jak ukazuje obrázek 3.9.

Při tomto druhu komunikace jsou využity jen 4 datové vodiče D4-D7 řadiče HD44780 a zbytek je uzemněn. Data nebo instrukce jsou vyslány vždy nadvakrát. Nejprve se na D4-D7 pošle horní polovina bajtu, vygeneruje se kladný pulz na E, na sběrnici se pošle spodní polovina bajtu a generuje se druhý pulz na E. Pouze při první instrukci, která inicializuje displej jako 4-bitový se pošle jen horní polovina inicializačního bajtu a jen jeden pulz na pin E. Bajt se pak vyšle znovu, ale již celý a klasickým čtyřbitovým komunikačním formátem, aby se nastavily i spodní 4 bity instrukce „Function set" (v tabulce 3.2 „*Nastavení funkce displeje*")[1].

Pro posílání impulzu na pin E má třída FourBitIO() metodu nazvanou **toggleE()**, která v podstatě pouze přepne pin „5" paralelního portu, na který je připojen pin E řadiče HD44780 z logické 0 do logické 1 a zpět do 0. Dále tato třída obsahuje metodu **instr(cmd)**. Ta jako argument očekává 8 bitové slovo, tedy 1 bajtovou instrukci. Pokud bychom chtěli nastavit funkci displeje na 4 bitovou komunikaci a jako dvouřádkový, použili bychom jako argument této metody hexadecimální hodnotu 0x28, což se v binární podobě rovná 00101000. Třetí metodou třídy FourBitIO() je **putc(char)**. Tato metoda je učená k zápisu jednoho znaku na displej. Je v podstatě totožná s předcházející metodou instr(). Nastavuje ale pin 3, na nějž je vyveden signál RS řadiče HD44780, na logickou „1", čímž sděluje řadiči, že se nejedná o instrukci, ale o data, která má zobrazit na displeji.

3.5.6 Ovládání celého modulu MLAB LCD2L4P02A

Protože na modulu LCD2L4P02A, jsou kromě LCD displeje osazena ještě tlačítka, která jsem chtěl využít pro ovládání jednoduchého uživatelského rozhraní, musel jsem pythonský modul lcd.py (viz. 3.5.5) rozšířit o potřebné funkcionality. Takto rozšířený modul jsem pojmenoval **mlab.py**. Do něj jsem připrogramoval třídu MLAB(), která si při své inicializaci vytváří dva objekty. První objekt - **mlab** je instancí třídy HD44780(), jenž je potomkem třídy FourBitIO(). Tento objekt má na starosti komunikaci s LCD displejem. Metoda **show([])** této třídy očekává jako argument seznam o dvou prvcích typu řetězec (string), přičemž první prvek (řetězec) pomocí metody write() objektu mlab zobrazí na prvním řádku LCD displeje a druhý prvek zobrazí na druhém řádku

Instrukce	Řízení		Datová sběrnice							
	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
Smaž displej a nuluj adresu DD ($2ms$)	0	0	0	0	0	0	0	0	0	1
Nuluj adresu DD na 0 a zruš posunutí ($2ms$)	0	0	0	0	0	0	0	0	1	*
Nastavení pohybu kurzoru ($40\mu s$)	0	0	0	0	0	0	0	1	ID	S
Nastavení módu displeje ($40\mu s$)	0	0	0	0	0	0	1	D	C	B
Okamžitý posuv kurzoru nebo textu ($40\mu s$)	0	0	0	0	0	1	SC	RL	*	*
Nastavení funkce displeje ($40\mu s$)	0	0	0	0	1	DL	N	F	*	*
Nastavení adresy CG ($40\mu s$)	0	0	0	1	A5	A4	A3	A2	A1	A0
Nastavení adresy DD ($40\mu s$)	0	0	1	A6	A5	A4	A3	A2	A1	A0
Čtení adresy a busy bitu	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Zápis dat ($40\mu s$)	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Čtení dat ($40\mu s$)	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Tabulka 3.2: Instrukce řadiče HD44780, převzato z [17]

Označení Bitu	Funkce / status	
I/D	0 = Posun kurzoru vlevo	1 = Posun kurzoru vpravo
S	0 = Neposouvat text	1 = Posouvat celý text
D	0 = Vypnutí displeje	1 = Zapnutí displeje
C	0 = Vypnutí kurzoru	1 = Zapnutí kurzoru
B	0 = Vypnutí blikání kurzoru	1 = Zapnutí blikání kurzoru
S/C	0 = Posun kurzoru	1 = Posun displeje
R/L	0 = Vlevo	1 = Vpravo
DL	0 = 4-bit interface	1 = 8-bit interface
N	0 = jednořádkový displej	1 = dvouřádkový displej
F	0 = 5x7 font (základní)	1 = 5x10 font
BF	0 = Displej připraven	1 = Právě probíhající interní operace

Tabulka 3.3: Speciální bity instrukcí řadiče HD44780, převzato z [17]

LCD displeje. Dále třída MLAB() obsahuje objekt **tlacitko**, jenž uchovává informaci o tom jaké tlačítko bylo naposledy stisknuto a ještě metodu **stisknuto()**, která čeká v nekonečné smyčce, dokud není stisknuto jedno z tlačítek na modulu MLAB (dojde k uzemnění jednoho z pinů stavového registru paralelního portu) a poté přenastaví hodnotu proměnné **tlacitko**. K vyhodnocení toho, že bylo stisknuto tlačítko, se v metodě stisknuto() používají metody třídy Parallel() (viz. 3.5.2).

3.5.7 Pomocný modul tools.py

V řídícím programu jsou využívány různé externí programy, kromě programu dcfldd, který provádí samotnou duplikaci média, ještě další utility na zjišťování informací o duplikovaném médiu (výrobní číslo, velikost, rozdělení na jednotlivé oddíly, atp.). Tyto programy jsou spouštěny pomocí modulu subprocess (viz. 3.5.3). Funkce, které externí programy využívají jsou uloženy v pomocném modulu tools.py.

Funkce **detekce_disku()** spouští příkaz sfdisk s parametrem -s (`sfdisk -s`), výstup tohoto příkazu, pak pomocí regulárních výrazů (viz. 3.5.4) upravuje a vrací seznam paměťových médií (blokových zařízení) aktuálně přítomných v systému. Další funkcí v modulu tools.py je **info_o_disku()**. Ta přebírá jako argument řetězec, který specifikuje cestu k blokovému zařízení, jehož název pak tato funkce vrací v podobě řetězce. Ke zjištění názvu paměťového média tato funkce využívá program `hwinfo` s potřebnými parametry, přičemž výstup tohoto programu je opět upraven pomocí regulárních výrazů. Název duplikovaného média je předán jako argument funkci **vytvor_adresar()**, která vytvoří adresář s tímto názvem v cílovém disku a do tohoto adresáře je pak ukládán obraz a informace o duplikovaném médiu.

Funkce **zapis_info()** zjišťuje informace o duplikovaném médiu (výrobní číslo, velikost, rozdělení na jednotlivé oddíly, atp.) a tyto informace zapisuje do souboru v adresáři, který byl vytvořen v cílovém médiu funkcí vytvor_adresar(). Informace získává postupým spouštěním externích programů `hwinfo`, `gdisk` a `smartctl`. Modul tools.py jestě obsahuje tyto funkce:

- mount() - spouští příkaz `mount`
- umount() - spouští příkaz `umount`
- porovnej_seznamy() - porovná dva seznamy a vrátí rozdílový seznam
- volne_misto() - porovnává velikost volného místa v cílovém médiu s velikostí duplikovaného média

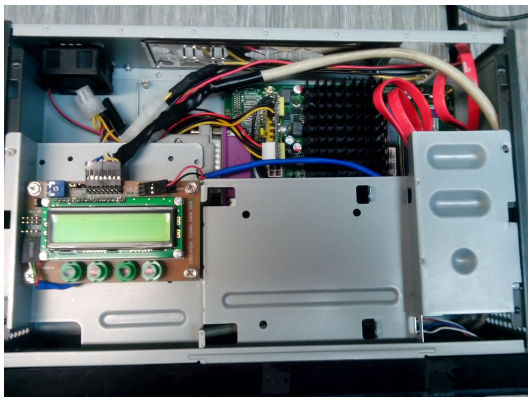
3.5.8 Hlavní modul menu.py

Základem hlavního programu menu.py je nekonečný cyklus while, který běží až do vypnutí celého systému. Při spuštění programu se nejprve spustí dva podprocesy pro načtení a odstranění modulů `ppdev`, respektive `lp`, do, respektive z, jádra. Poté program vytváří instanci třídy **MLAB()** z modulu mlab.py. Dále tento modul obsahuje třídu **Menu()**, která zobrazuje jednoduchou uživatelskou nabídku a uchovává informaci o tom, jaká položka menu byla zvolena a tedy jaký se má vykonat příkaz. Nabídka obsahuje pouze dvě možnosti. Jedna z nich je „Vypnout". Pokud uživatel zvolí a potvrdí tuto možnost, spustí se funkce **vypnout()**, která spustí podproces `shutdown -h 0` a ten vypne celé zařízení.

Druhou možností pro uživatele je „Spustit zálohu disku". Pokud je vybrána a potvrzena tato možnost, spouští se funkce **zaloha()**. Tato funkce komunikuje s uživatelem prostřednictvím LCD displeje na modulu MLAB a směruje ho k úkonů, které musí provést, aby mohl být spuštěn proces bitového kopírování paměťového média. Pokud všechny přípravné kroky proběhnou bezchybně, je spuštěn podproces `dcfldd` s potřebnými parametry.

Tento program se spustí po startu systému pomocí skriptu `/etc/rc.local`, který je spouštěn (jednotlivé příkazy v něm obsažené) po naběhnutí všech systémových služeb (démonů) v multiuživatelském „runlevelu". Všechny programy spuštěné pomocí tohoto souboru běží s právy superuživatele root. Tento soubor tedy obsahuje následující řádek kódu:

```
/home/atom/menu.py
```



Obrázek 4.1: Hotové zařízení na duplikaci paměťových médií bez horního krytu

Obrázek 4.2: Uživatelský interface kopírovacího zařízení

K tomu je potřeba dodat, že všechny moduly Pythonu (menu.py, tools.py a mlab.py) jsou uloženy v domovské složce uživatele atom a soubor menu.py má příznak spustitelného souboru.

4 Hotové zařízení

Všechny komponenty (základní deska, MLAB modul s LCD displejem, zdroj pico PSU, kabeláž) jsem osadil do své starší mini-ITX počítačové skříně (tzv. case) o rozměrech 31x22x7cm, ve které byl vadný integrovaný zdroj. Tento zdroj jsem odstranil a nahradil jsem ho již zmíněným pico PSU. Protože největší komponent - základní deska má rozměry 17x17cm, bylo by možné použít i menší „case", ale pro mě to v tuto chvíli již byla investice navíc a použil jsem ten, který jsem měl k dispozici.

Do mini-ITX skříně jsem vyřízl, respektive vyvrtal, otvory pro LCD displej a tlačítka na modulu MLAB LCD2L4P02A. Tento modul je pak připevněn k plechovému držáku, kde byl původně osazen systémový pevný disk počítače. Operační systém je nainstalován na USB flashdisk, tím pádem není potřeba pevného disku a vznikl tak tento prostor pro osazení MLAB modulu, pomocí kterého se celé zařízení ovládá.

Zařízení se od zapnutí, tedy stisknutí vypínače, dostane do použitelného stavu po 26 sekundách. Jinými slovy tzv. boot systému trvá 26 vteřin, poté se spustí program menu.py (viz. 3.5.8) a na displeji se objeví nápis „Zarizeni je pripraveno" (LCD displej nezobrazuje českou diakritiku - to lze doprogramovat). V této chvíli může uživatel začít zařízení ovládat pomocí tlačítek (viz. obrázek 4.2).

Pokud pak uživatel vybere na displeji možnost „Spustit zalohu disku", začne se

Obrázek 4.3: Hotové zařízení na duplikaci paměťových médií včetně horního krytu

provádět připravený scénář.

Nejprve je uživatel požádán o připojení cílového média, to znamená média, na které se bude ukládat obraz duplikovaného disku. Poté je disk rozpoznán a připojen (příkaz `mount`) do adresáře `/mnt`. Pak je uživatel vyzván k tomu, aby připojil zdrojové médium, tedy to duplikované, které je též rozpoznáno a uživatel poté musí potvrdit, že chce skutečně spustit bitové kopírování. V tomto okamžiku je jediný prostor na to, aby uživatel celý proces korektně zastavil (dojde k odpojení cílového média z adresáře `/mnt`) a to stisknutím tlačítka 2 (X).

Jestliže je ale stisknuto tlačítko 1 (√), pokračuje řídící program tím, že zjistí velikost duplikovaného média (příkaz `sfdisk -s`) a porovná ji s volným místem v adresáři `/mnt` (příkaz `df`). Pokud je tedy v adresáři `/mnt` dost volného prostoru na to, aby se tam vešel soubor o stejné velikosti jako duplikované médium, spustí se následující příkaz.

```
dcfldd␣conv=noerror,notrunc␣bs=4096␣hash=sha1␣hashlog=/mnt/<
    název>/hash.sha1␣hashconv=before␣if=<duplikované␣médium>
    ␣of=/mnt/<název>/obraz.img
```

V opačném případě je sděleno uživateli, že v cílovém médiu není dostatek volného místa, cílové médium je odpojeno (`umount`) a uživatel je instruován k fyzickému odpojení obou paměťových médií.

Spuštěný proces `dcfldd` je řídícím programem kontrolován, jestli stále běží, nebo jestli skončil s chybovým návratovým kódem. Zároveň řídící program průběžně zjišťuje velikost souboru **obraz.img** pomocí příkazu `sfdisk -s`, porovnává ji s velikostí kopírovaného média a na základě toho pak na displeji zobrazuje počet procent zkopírovaných bloků dat.

Jestliže proces `dcfldd` skončí bez chyb a duplikované médium je tak kompletně zkopírováno, tak řídící program pomocí utilit (`gdisk`, `hwinfo`, `smartctl`) zjistí veškeré informace o duplikovaném disku a zapíše je do souboru **info_o_disku.txt** umístěného ve stejné složce jako obraz.img. Po té je odpojen příkazem `umount` cílový disk z adresáře `/mnt` a na displeji se objeví „Odpojte vsechna media". Tím je kompletní duplikace hotova.

5 Závěr

Cílem mé bakalářské práce bylo navrhnout jednoduché elektronické zařízení, které bude provádět duplikaci paměťových médií pro potřeby dalšího forenzního zkoumání a vyšetřování, tak aby tento úkon zvládali běžní uživatelé a aby nedošlo k přepsání nebo smazání zájmových dat na kopírovaném (duplikovaném) médiu.

Takové zařízení se mi podařilo navrhnout a realizovat za poměrně nízkou pořizovací cenu veškerého hardwaru - 3286 Kč (základní deska 1396 Kč, paměť RAM 291 Kč, zdroj 1000 Kč, MLAB modul 599Kč). Software včetně operačního systému jsem použil takový, který je dostupný zdarma (většinou šířený pod licencí GPL). Jako svobodný software je šířen i programovací jazyk Python, v němž je napsán řídící program na ovládání LCD displeje.

Inspirací pro návrh tohoto zařízení mi byl Forensic Duplicator od firmy Digital Intelligence, jehož základní verze je však několikanásobně dražší. Celkové náklady by bylo možné ještě snížit použitím počítače Raspberry Pi, jehož cena je v současnosti $35. O tom proč jsem tento jednodeskový počítač nakonec nepoužil se zmiňuji v kapitole 3.1. Pokud by se někdy objevil podobně levný jednodeskový počítač, který by byl navíc vybaven rozhraním SATA, nebyl by problém moje řešení na něj aplikovat, díky použití multiplatformního operačního systému a stejně multiplatformního programovacího jazyka.

Zařízení je možné dále rozšiřovat o některé funkce. Například by mohla být přidána funkcionalita pro kopírování výsledného obrazu média do síťového úložiště NAS. To by bylo využitelné v případě, že by duplikovaný harddisk měl příliš velkou kapacitu (3TB a více). Také by mohla být v zařízení implementována možnost během probíhající duplikace analyzovat zájmová data na duplikovaném médiu, a to buď pomocí programu bulk_extractor nebo balíkem nástrojů The Sleuthkit (viz. kapitola 2.5 resp. 2).

Literatura

[1] Ovládání znakových LCD s řadičem HD44780 - 1. díl In: *Padatron: Elektrotechický magazín*. [online]. [cit.2013-02-18]. Dostupné z: <http://pandatron.cz/?685&ovladani_znakovych_lcd_s_radicem_hd44780_%96_1._dil>. 3.5.5

[2] Forenzní vědy. In: *Wikipedia: the free encyclopedia*, . [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-12-20]. Dostupné z: <http://cs.wikipedia.org/wiki/Forenzní_vědy>. 1

[3] GUID Partition Table. In: *Wikipedia: the free encyclopedia*, . [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-12-20]. Dostupné z: <http://cs.wikipedia.org/wiki/GUID_Partition_Table>. 11

[4] ALLEN, B. Introducing bulk_extractor. In: *GitHub*, 2013. [online]. [cit. 2013-02-20]. Dostupné z: <https://github.com/simsong/bulk_extractor/wiki/Introducing-bulk_extractor>. 2.5

[5] CARRIER, B. *Defining Digital Forensic Examination and Analysis Tools Using. Abstraction Layers*. [online]. Dostupné z: <www.digital-evidence.org/.../dfrws_define.pdf>. 1

[6] CARRIER, B. Autopsy Forensic Browser. [online], 2003-2012. [cit.2012-09-24]. Dostupné z: <http://www.sleuthkit.org/autopsy>. 2.4, 2.1

[7] CORBET, J. – RUBINI, A. – KROAH-HARTMAN, G. *Linux Device Drivers*. 3rd edition edition, 2005. 3.2.2, 3.4

[8] DIGITAL INTELLIGENCE, I. Computer Forensics. [online], 2013. [cit.2013-02-07]. Dostupné z: <http://www.digitalintelligence.com/>. 3.1

[9] FREE SOFTWARE FOUNDATION. *GNU Operating System*. [online]. [cit.2012-09-25]. Dostupné z: <http://www.gnu.org>. 3.2, 3.2.1

[10] GRUNDY, B. A Practitioner's Guide to Linux as a Computer Forensic Platform. [online]. Dostupné z: <http://www.linuxleo.com/Docs/linuxintro-LEFE-3.78.pdf>. 2, 2.3

[11] HALDERMAN, J. A. et al. LEST WE REMEMBER: COLD BOOT ATTACKS ON ENCRYPTION KEYS. [online]. Dostupné z: <http://citp.princeton.edu/pub/coldboot.pdf>. 1.2

[12] KADLEC, J. Forenzní analýza In: *Root: Informace nejen ze světa Linuxu*. [online], . [cit.2012-12-20]. Dostupné z: <http://www.root.cz/clanky/forenzni-analyza-1/>. (document), 1, 1.1, 1.2

[13] KADLEC, J. Forenzní analýza 2 In: *Root: Informace nejen ze světa Linuxu*. [online], . [cit.2012-12-20]. Dostupné z: <http://www.root.cz/clanky/forenzni-analyza-2/>. 1.2

[14] KASPRZAK, J. Desktop a jádro Linuxu In: *Konference EurOpen.CZ*. [online]. [cit.2012-12-20]. Dostupné z: <http://www.root.cz/knihy/desktop-a-jadro-linuxu/stahnout/927/>. 3.2.3

[15] KUMPOŠT, M. Hašovací funkce. Bakalářská práce, Masarykova universita Fakulta informatiky, 2002. 2.2.1

[16] LIECHTI, C. pySerial v2.6 documentation. [online]. [cit.2013-02-15]. Dostupné z: <http://pyserial.sourceforge.net/index.html>. 3.5.5

[17] MLAB. *LCD2L4P02A*. [online]. [cit.2012-12-10]. Dostupné z: <http://www.mlab.cz/Modules/HumanInterfaces/LCD2L4P02A/DOC/HTML/LCD2L4P02A.cs.html>. 3.4, 3.6, 3.7, 3.8, 3.2, 3.3

[18] OLMR, V. *HW server představuje: Paralelní port - LPT (IEEE 1284)*. [online]. [cit.2012-12-14]. Dostupné z: <http://www.hw.cz/lpt>. 3.3, 3.3

[19] PILGRIM, M. *Ponořme se do Python(u) 3*. Cz.Nic, c2010. ISBN 978-80-904248-2-1. 3.5.4

[20] PYTHON SOFTWARE FOUNDATION. *Python Programming Language; Official Website*. [online], . [cit.2012-09-24]. Dostupné z: <http://www.python.org>. 3.5.1

[21] PYTHON SOFTWARE FOUNDATION. 17.1. subprocess - Subprocess management In: *Python Programming Language; Official Website*. [online], . [cit.2013-02-15]. Dostupné z: <http://docs.python.org/2.7/library/subprocess.html>. 3.5.3

[22] TIŠNOVSKÝ, P. Paralelní port a rozhraní Centronics In: *Root: Informace nejen ze světa Linuxu*. [online]. [cit.2012-12-14]. Dostupné z: <http://www.root.cz/clanky/paralelni-port-a-rozhrani-centronics/>. 3.3, 3.5, 3.3.1, 3.1

[23] WALKER, D. H. *A Comparison of NVMe and AHCI*. [online], 2012. Dostupné z: <http://sata-io.org/documents/NVMeandAHCI__long_.pdf>. 3.1.1

[24] ZATYKO, K. Commentary: Defining Digital Forensics. *Forensic Magazine*. [online], 2007. [cit. 2012-12-26]. Dostupné z: <http://www.forensicmag.com/node/128>. 1

Příloha

Výpis souboru zdrojového kódu programu **menu.py**:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# licence GNU GPL

"""
Created on Tue Dec  4 14:19:30 2012

@author: František Doule
"""
import time
from mlab import *
from tools import *


class Menu:

    def __init__(self):
        self.menu = [
                            ["Vypnout ?",""],
                                                # i=0
                            ["Spustit", "zalohu disku ?"],  #
                                i=1
                        ]


        self.i=0

    def posun_right(self):
        if self.i>=0 and self.i < (len(self.menu)-1):
            self.i+=1
        else:
            self.i = 0
        return self.i

    def posun_left(self):
        if self.i>0 and self.i <= (len(self.menu)-1):
            self.i-=1
        else:
            self.i = (len(self.menu)-1)
        return self.i

    def prikaz(self):
        return self.i

    def polozka(self):
        return self.menu[self.i]

```

```
47
48 def vypnout():
49     global mlab
50     print mlab.show(["Vypinam", "zarizeni ..."])
51     podproces("shutdown -h 0")
52
53
54 def zaloha():
55
56     global mlab
57     partition = "1"
58     int_disk=[]
59     cilovy_disk=[]
60     zdrojovy_disk=[]
61     nazev_c = ""
62     nazev_z = ""
63     size = 0
64     cesta = "/mnt/"
65     args=None
66     dcfldd=None
67
68     print mlab.show(["Odpoj vsechna", "media a potvrd"])
69     while 1:
70         mlab.stisknuto()
71         if mlab.tlacitko == "1":
72             print mlab.show(["probiha", "kontrola..."])
73             time.sleep(2)
74             int_disk = detekce_disku()
75             print int_disk
76             break
77
78     print mlab.show(["Pripoj CILOVY", "disk a potvrd"])
79     while 1:
80         mlab.stisknuto()
81         if mlab.tlacitko == "1":
82             print mlab.show(["detekuji ", "disk..."])
83             time.sleep(2)
84             pom_disky = detekce_disku()
85             cilovy_disk = porovnej_seznamy(int_disk,
                   pom_disky)
86             if cilovy_disk:
87                 nazev_c = info_o_disku(cilovy_disk[0])
88                 print mlab.show(["nalezen disk:", nazev_c
                       ])
89                 time.sleep(1)
90                 print cilovy_disk[0]+partition
91                 print mlab.show([nazev_c,"  Pokracovat ?"
                       ])
92                 mlab.stisknuto()
93                 if mlab.tlacitko == "1":
```

```
94                      print mlab.show(["mount disku:",
                            nazev_c])
95
96                      # !!! nasledujici radek musi byt kvuli
                            mount FUSE !!!
97                      del mlab.mlab #!!!
98                      try:
99                          mount(cilovy_disk[0]+partition)
100                     except:
101                         print mlab.show(["mount disku se",
                                "nezdaril..."])
102                         time.sleep(2)
103                         print mlab.show(["odpoj disk","a
                                naformatuj ho"])
104                         return
105                     print mlab.show(["cilovy disk","
                            pripraven"])
106                     time.sleep(1)
107                     break
108                 if mlab.tlacitko == "2":
109                     print mlab.show(["pro navrat","pouzij
                            sipky"])
110                     return
111
112             else:
113                 print mlab.show(["disk nenalezen","zkusit
                        znovu?"])
114                 mlab.stisknuto()
115                 if mlab.tlacitko == "1":
116                     continue
117                 if mlab.tlacitko == "2":
118                     print mlab.show(["pro navrat","pouzij
                            sipky"])
119                     return
120
121     print mlab.show(["Pripoj ZDROJOVY", "disk a potvrd"])
122
123     while 1:
124         mlab.stisknuto()
125         if mlab.tlacitko == "1":
126             print mlab.show(["detekuji ", "disk..."])
127             time.sleep(2)
128             zdrojovy_disk = porovnej_seznamy(pom_disky,
                    detekce_disku())
129             if zdrojovy_disk:
130                 nazev_z = info_o_disku(zdrojovy_disk[0])
131                 print mlab.show(["nalezen disk:", nazev_z
                        ])
132                 time.sleep(3)
133                 print mlab.show(["vyrobit bitovou:","kopii
                        ....?"])
```

```
134                 mlab.stisknuto()
135                 if mlab.tlacitko == "1":
136                     del mlab.mlab
137                     size= float(podproces("sfdisk -s "+
                            zdrojovy_disk[0]))
138                     if volne_misto("/mnt",size):
139                         cesta += nazev_z
140                         vytvor_adresar(cesta)
141 
142                         tik= 60
143                         if int(size/1024/1024) < 150:
144                             tik= 20
145                         if int(size/1024/1024) < 2:
146                             tik= 3
147 
148                         args = shlex.split("dcfldd conv=
                                noerror,notrunc bs=4096 hash=
                                sha1 hashlog="+cesta+"/hash.
                                sha1 hashconv=before if="+
                                zdrojovy_disk[0]+" of="+cesta+"
                                /obraz.img")
149                         try:
150                             #dcfldd = subprocess.Popen(
                                    args, shell=False,stdout=
                                    subprocess.PIPE, stderr=
                                    subprocess.PIPE)
151                             dcfldd = subprocess.Popen(args
                                    , shell=False,stdout=
                                    subprocess.PIPE)
152                         except:
153                             print mlab.show(["chyba","
                                    pouzij sipky"])
154                             umount("/mnt")
155                             return
156                         while dcfldd.returncode==None:
157 
158                             try:
159                                 hotovo=(float(podproces("
                                        sfdisk -s "+cesta+"/
                                        obraz.img"))/size) *100
160                             except:
161                                 hotovo=float(0)
162                             procent = "hotovo: "+str(int(
                                    hotovo))+"%"
163 
164                             print mlab.show(["zalohuji..",
                                    procent])
165 
166                             dcfldd.poll()
167                             if dcfldd.returncode!=None:
168                                 break
```

```
169                         time.sleep(tik)
170
171                     print dcfldd.returncode
172
173                     if dcfldd.returncode==0 :
174                         print mlab.show(["zalohuji..."
                                ,"hotovo 100%"])
175                         time.sleep(1)
176                         print mlab.show(["Zaloha","
                                dokoncena"])
177                         time.sleep(1)
178                         print mlab.show(["Zapisuji
                                informace","o disku..."])
179                         zapis_info(zdrojovy_disk[0],
                                cesta+ "/info_o_disku.txt")
180
181                         umount("/mnt")
182                         print mlab.show(["Hotovo",""])
183                         time.sleep(1)
184                         print mlab.show(["odpojte
                                vsechna","media..."])
185
186                         break
187                     else:
188                         print mlab.show(["chyba","
                                pouzij sipky"])
189                         with open(cesta+"/errorlog.txt
                                ", mode='a') as soubor:
190                             soubor.write(dcfldd.
                                    communicate())
191                         umount("/mnt")
192                         return
193                 else:
194                     print mlab.show(["v cily neni mis-
                            ","to odpoj media..."])
195                     umount("/mnt")
196                     return
197             if mlab.tlacitko == "2":
198                 print mlab.show(["pro navrat","pouzij
                        sipky"])
199                 umount("/mnt")
200                 return
201         else:
202             print mlab.show(["disk nenalezen","zkusit
                    znovu?"])
203             mlab.stisknuto()
204             if mlab.tlacitko == "1":
205                 continue
206             if mlab.tlacitko == "2":
207                 umount("/mnt")
```

```
208                     print mlab.show(["pro navrat","pouzij
                            sipky"])
209                     return
210
211     return
212
213
214
215
216
217
218
219 if __name__ == '__main__':
220
221
222     podproces("modprobe -r lp")
223     podproces("modprobe ppdev")
224
225     mlab=MLAB()
226     mlab.show(["Zarizeni je", "pripraveno"])
227
228     menu = Menu()
229     kontrola=[]
230
231     while 1 :
232
233         mlab.stisknuto()
234
235         if mlab.tlacitko == "4":
236             menu.posun_right()
237             kontrola = mlab.show(menu.polozka())
238
239
240         if mlab.tlacitko == "3":
241             menu.posun_left()
242             kontrola = mlab.show(menu.polozka())
243
244
245         if mlab.tlacitko == "2":
246             pass
247
248         if mlab.tlacitko == "1" and kontrola == menu.
                polozka():
249             kontrola=mlab.show(["Opravdu ?",""])
250
251             while 1:
252                 mlab.stisknuto()
253                 if mlab.tlacitko == "1":
254                     if menu.prikaz() == 0:
255                         vypnout()
256                         break
```

```
257                         if menu.prikaz() == 1:
258                             zaloha()
259                             break
260 
261 
262                         break
263                     if mlab.tlacitko == "2":
264                         mlab.show(menu.polozka())
265                         break
266                     time.sleep(0.1)
267                 #podproces(menu.prikaz())
268 
269             time.sleep(0.1)
```

Výpis zdrojového kódu souboru **tools.py**:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''
Created on 13.9.2012

@author: František Doule
'''
import subprocess
import shlex
import re
import os
import shutil

def podproces(prikaz):
    args = shlex.split(prikaz)
    proc = subprocess.Popen(args, shell=False,stdout=
       subprocess.PIPE, stderr=subprocess.PIPE)
    vystup = proc.communicate(input)
    return vystup[0]


def detekce_disku():
    args = shlex.split("sfdisk -s")

    try:
        vystup = subprocess.check_output(args)
    except:
        return("nelze spustit \"sfdisk\" - pravdepodobne
           nemate potrebna prava")

    drives = shlex.split(vystup)
    disky=[]
    for i in range(len(drives)):
        if re.search("/dev/*",drives[i]):
            disky.append(re.sub(":$","",drives[i]))

    return disky

def porovnej_seznamy(list1,list2):

    if len(list1) >= len(list2):
        return list(set(list1) - set(list2))
    else:
        return list(set(list2) - set(list1))

def info_o_disku(disk):
    try:
        args = shlex.split("hwinfo --short --disk --only
           "+disk)
        hwinfo=subprocess.check_output(args)
```

```
48          infor=re.sub(r"\n","",hwinfo)
49          info=re.sub(disk,"",infor)
50          inf=re.sub("disk:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣","",info)
51          bez_mezer=re.sub("␣","_",inf)
52          return bez_mezer
53      except:
54          return "Nebylo␣možné␣získat␣informace␣o~disku␣(
                chybí␣utilita␣hwinfo)"
55
56
57  def mount(partition):
58      args=shlex.split("mount␣-w␣"+partition+"␣/mnt" )
59      return subprocess.check_call(args)
60
61  def umount(partition):
62      args=shlex.split("umount␣"+partition)
63      return subprocess.call(args)
64
65
66  def vytvor_adresar(nazev):
67      try:
68          shutil.rmtree(nazev)
69      except:
70          pass
71      try:
72          os.mkdir(nazev)
73          return True
74      except:
75          return False
76
77  def zapis_info(disk,cesta_k_souboru):
78      args_hwinfo = shlex.split("hwinfo␣--disk␣--only␣"+
            disk)
79      hwinfo=subprocess.check_output(args_hwinfo)
80      with open(cesta_k_souboru, mode='a') as soubor:
81          soubor.write(hwinfo + "\n")
82
83      args_fdisk = shlex.split("gdisk␣-l␣"+disk)
84      fdisk=subprocess.check_output(args_fdisk)
85      with open(cesta_k_souboru, mode='a') as soubor:
86          soubor.write("\n" + fdisk)
87
88      args_smart = shlex.split("smartctl␣-i␣"+disk)
89      try:
90          smart=subprocess.check_output(args_smart)
91          with open(cesta_k_souboru, mode='a') as soubor:
92              soubor.write("\nS.M.A.R.T␣info:␣\n" + smart)
93      except:
94          pass
95
96  def volne_misto(directory, size):
```

```
97      df = shlex.split(podproces("df "+ directory))
98      print df
99      volno = float(df[10])/1024/1024
100     velikost = (float(size)/1024/1024)+2
101     if volno >= velikost:
102         return True
103     else:
104         return False
105
106 if __name__ == '__main__':
107
108     #print mount("/dev/sdb1")
109     #print str(int(float(podproces("sfdisk -s /dev/sdd"))
            /1024/1024))
110     volne_misto("/mnt",1937879644)
```

Výpis zdrojového kódu souboru **mlab.py**:

```
1  #!/usr/bin/env python
2  # program (pseudo ovladac) pro modul MLAB LCD2L4P02A
3
4  #prevzato a upraveno z ~:
5  #                          # character LCD example for
       pyparallel
6  #                          #
7  #                          #(C) 2002 Chris Liechti <
       cliechti@gmx.net>
8  # this is distributed under a free software license - GNU
        GPL
9  #
10 # musi bezet pod pravama roota
11 # !!!! pred spustenim :
12 #                  modprobe -r lp
13 #                  modprobe ppdev
14
15 import sys, time
16 sys.path.insert(0, '..')
17 import parallel
18
19
20 LCDON           = 0x01 #0x00000001 Switch on  display
21 LCDOFF          = 0x08 #0x00001000 Switch off display
22 LCDCLEAR        = 0x01 #0x00000001
23 LCDLINE1        = 0x80 #0x10000000
24 LCDLINE2        = 0xc0 #0x11000000
25 LCDMOVE         = 0x07 #0b00000111 posouvat text doleva
26 LCDCURSORON     = 0x0f #0x00001111 turn on cursor
       blinking
27 LCDCURSOROFF    = 0x0c #0x00001100 Disable cursor
       blinking. The cursor is hidden.
28 LCDCGADRSET     = 0x40 #0b01000000
29 LCDDDADRSET     = 0x80 #0b10000000
30 LCD2LINES       = 0x28 #0b00101000 Set display mode to
       two lines.
31 LCD8BITS        = 0x30 #0b00110000 select 8 Bit interface
32 LCD4BITS        = 0x20 #0b00100000 select 4 Bit interface
33 LCD_DATA_OFF    = 0x05 #0x00000101 mask used to clear the
        data lines
34 LCDCLEARDD      = 0x03 #0b00000011
35 LCDROLL         = 0x18  #0b00011000 okamzity posun
       kurzoru
36
37 LCD_RS          = 1<<1
38 LCD_RW          = 1<<2
39 LCD_E           = 1<<3
40 LCD_D4          = 1<<4
41 LCD_D5          = 1<<5
42 LCD_D6          = 1<<6
```

```
43 LCD_D7            = 1<<7
44
45
46 class FourBitIO(object):
47     def __init__(self):
48         self.data = 0
49
50         self.out(0)                    #reset pins
51         time.sleep(0.050)              #wait more than 30ms
52         #send the reset sequece (3 times the same pattern
               )
53         self.out(LCD8BITS)             #set 8 bit interface
54         self.toggleE()                 #toggle LCD_E, the
               enable pin
55         time.sleep(0.005)              #wait a bit
56         self.toggleE()                 #toggle LCD_E, the
               enable pin
57         time.sleep(0.005)              #wait a bit
58         self.toggleE()                 #toggle LCD_E, the
               enable pin
59         time.sleep(0.005)              #wait a bit
60
61         self.out(LCD4BITS)             #now set up the 4 bit
               interface
62         self.toggleE()                 #toggle LCD_E, the
               enable pin
63         time.sleep(0.002)              #wait until instr is
               finished
64         self.instr(LCD2LINES, wait=0.002)     #set 2 lines
               display
65         self.instr(LCDCURSOROFF, wait=0.002) #hide cursor
66         self.instr(LCDCLEAR, wait=0.002)      #clear
               display
67
68     #my connector has the wrong pinorder....
69     #better swap them in software than to solder ;-)
70     def reveseout(self, x):
71         #r = ((x & (1<<0) and 1) << 7) |\
72         #    ((x & (1<<1) and 1) << 6) |\
73         #    ((x & (1<<2) and 1) << 5) |\
74         #    ((x & (1<<3) and 1) << 4) |\
75         #    ((x & (1<<4) and 1) << 3) |\
76         #    ((x & (1<<5) and 1) << 2) |\
77         #    ((x & (1<<6) and 1) << 1) |\
78         #    ((x & (1<<7) and 1) << 0)
79         #print "%02x" % r, "%02x" %x
80         #self.p.setData(r)
81         self.p.setData(x)
82
83     def toggleE(self):
84         """toggle enable pin"""
```

```
85          self.data |= LCD_E;          #toggle LCD_E, the
                enable pin
86          self.reveseout(self.data)
87          self.data &= ~LCD_E;         #back to inactive
                position
88          self.reveseout(self.data)
89
90
91      def out(self, data):
92          """set data to LCD port"""
93          self.data = data
94          self.reveseout(self.data)
95
96      def instr(self, cmd, wait=0.000040):
97          """send instruction byte to LCD"""
98          self.out(cmd & 0xf0)         #output upper nibble
99          self.toggleE()               #toggle LCD_E, the
                enable pin
100         time.sleep(wait)
101         self.out((cmd << 4) & 0xf0) #and then the lower
                nibble
102         self.toggleE()               #toggle LCD_E, the
                enable pin
103         time.sleep(wait)            #wait until instr is
                finished
104
105     def putc(self, c):
106         """send a data byte to the LCD"""
107         c = ord(c)
108         self.out((c & 0xf0) | LCD_RS) #output upper
                nibble
109         self.toggleE()               #toggle LCD_E, the
                enable pin
110         time.sleep(0.002)
111         self.out(((c << 4) & 0xf0) | LCD_RS) #and then
                the lower nibble
112         self.toggleE()               #toggle LCD_E, the
                enable pin
113         time.sleep(0.002)            #wait until instr is
                finished
114
115
116 class HD44780(FourBitIO):
117 #class HD44780(EightBitIO):
118     def __init__(self):
119         self.p = parallel.Parallel()
120         super(HD44780, self).__init__()
121
122     def write(self, str):
123         """write a string to the LCD"""
124         for c in str:
```

```
125             self.putc(c)  #write each character
126
127     def downloadFont(self, fontdata):
128         """Set the memory pointer and download a font"""
129         self.instr(LCDCGADRSET);
130         self.write(fontdata)
131         self.instr(LCDLINE1)        #just in case, set
                cursor to a visible pos
132
133 class MLAB():
134     def __init__(self):
135         self.mlab=HD44780()
136         self.tlacitko=""
137
138     def stisknuto(self):
139
140         while 1:
141             if not self.mlab.p.getInAcknowledge():
142                 self.tlacitko = "4"
143                 break
144
145             if not self.mlab.p.getInPaperOut():
146                 self.tlacitko = "3"
147                 break
148
149             if not self.mlab.p.getInSelected():
150                 self.tlacitko = "2"
151                 break
152
153             if not self.mlab.p.getInError():
154                 self.tlacitko = "1"
155                 break
156
157             time.sleep(0.1)
158
159
160     def show(self,radky=[]):
161         try:
162             del self.mlab
163         except:
164             pass
165         time.sleep(0.2)
166         self.mlab = HD44780()
167         try:
168             self.mlab.write(radky[0])
169         except:
170             self.mlab.write("")
171         self.mlab.instr(LCDLINE2)
172         try:
173             self.mlab.write(radky[1])
174         except:
```

```
175                 self.mlab.write("")
176             #zobrazeno = radky[0] + radky[1]
177             return radky
178 
179 def zobraz (lcd,radky=[]):
180     lcd.write(radky[0])
181     lcd.instr(LCDLINE2)
182     try:
183         lcd.write(radky[1])
184     except:
185         lcd.write("")
186     #zobrazeno = radky[0] + radky[1]
187     return radky
188 
189 if __name__ == '__main__':
190     lcd = HD44780()
191     lcd.write("Zarizeni je")
192     lcd.instr(LCDLINE2)
193     lcd.write("pripraveno")
194 
195     while 1 :
196 
197         if not lcd.p.getInAcknowledge():
198             del lcd
199             time.sleep(0.2)
200             lcd = HD44780()
201             zobraz(lcd,["tlacitko 4",""])
202 
203         if not lcd.p.getInPaperOut():
204             del lcd
205             time.sleep(0.2)
206             lcd = HD44780()
207             zobraz(lcd,["tlacitko 3",""])
208 
209         if not lcd.p.getInSelected():
210             del lcd
211             time.sleep(0.2)
212             lcd = HD44780()
213             zobraz(lcd,["tlacitko 2",""])
214 
215         if not lcd.p.getInError():
216             del lcd
217             time.sleep(0.2)
218             lcd = HD44780()
219             zobraz(lcd,["tlacitko 1",""])
220 
221         time.sleep(0.1)
```