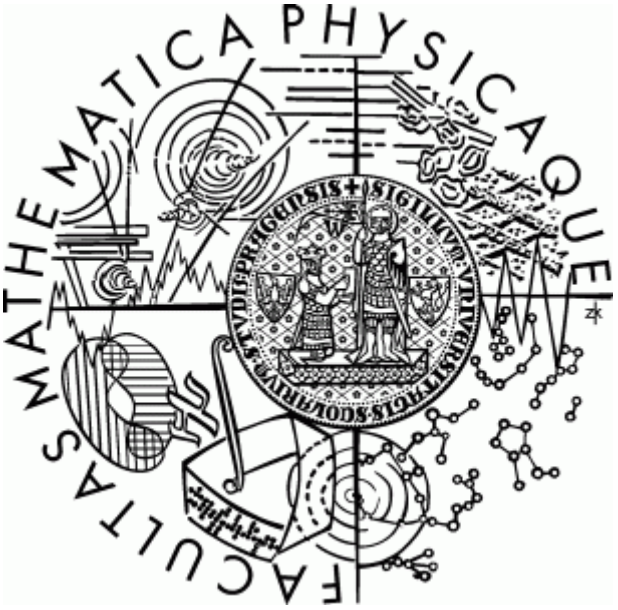
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martina Tomisová

Distribuovaný systém pro ověřování vlastností přirozených čísel

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Jiří Mírovský

Studijní program: Správa počítačových systémů

2007

Ráda bych poděkovala vedoucímu mého projektu panu Mgr. Jiřímu Mírovskému, za ochotu a připomínky, které mi pomohly napsat tuto práci, i jeho trpělivost. Dále děkuji všem vyučujícím i přátelům, kteří mi pomohli drobnými radami, když jsem si nebyla jistá. Rovněž bych ráda vyjádřila vděčnost všem, kteří se zasloužili o bezplatný software, který jsem použila a také těm, kteří se podobnou problematikou zabývali dříve a svými texty mi usnadnili získávání informací.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 11. června 2007 Martina Tomisová

Obsah

1 Zadání práce....6
2 Motivace....7
3 Co je Cuckoo....8
4 Obsah práce....9
5 Úvodní poznámky....10
5.1 Název projektu – Cuckoo....10
5.2 Jazyky....10
5.3 Požadavky na provoz....10
6 Programátorská dokumentace ke klientské části mimo modulární....11
6.1 Struktura projektu....11
6.2 Posloupnost akcí při běhu programu....11
6.2.1 Start....11
6.2.2 Výpočet....13
6.3 Manipulace s čísly....14
6.3.1 Formát čísla, jak ho vidí server, základní část klienta a modul....14
6.3.2 Co se s číslem děje během procesu ověření....14
6.3.3 Formát zápisu čísla v souborech....14
7 Programátorská dokumentace k modulární části klienta....16
7.1 Obecně o tvorbě modulu....16
7.2 Ukázková tvorba modulu – velmi jednoduchý příklad....16
7.3 Ukázková tvorba modulu – složitější příklad....19
8 Programátorská dokumentace k modulu pro ověřování prvočíselnosti....20
8.1 Postup ověření prvočíselnosti....20
8.2 Zásady využívání funkcí aritmetické knihovny....20
8.3 Aritmetická knihovna pro dlouhá čísla....23
8.3.1 Formát čísla....23
8.3.2 Algoritmy použité v knihovně....23
8.4 Algoritmy pro testování čísel....25
8.4.1 Rabinův-Millerův algoritmus....26
8.4.2 Algoritmus jistého ověřování prvočíselnosti....26
9 Rozhraní mezi klientem a serverem....27
10 Dokumentace k serveru....29
10.1 Z čeho se server skládá....29
10.2 Administrátorské skripty....29
10.2.1 config.php....30
10.2.2 admin.php a admin_module.php....32
10.3 Struktura databáze....34
10.3.1 Tabulka uživatelů....34
10.3.2 Tabulka zadání....35
10.3.3 Tabulka řešení....36
10.4 Stránka pro stahování....36
10.5 index.php....36
10.5.1 Registrace klienta....37
10.5.2 Připomenutí hesla....37

10.5.3 Příjem výsledků....37
10.5.4 Zadání práce....37
10.6 Jak nainstalovat a nakonfigurovat server....38
11 Uživatelská dokumentace....39
11.1 Požadavky na systém....39
11.2 Instalace a spouštění....39
11.3 Ukončení programu....41
11.3.1 Konfigurační soubor....41
11.4 Řešení problémů....42
11.4.1 Píše: "I'm already running. You don't need to run me twice." (Ukázka 6.1 na straně 12)....42
11.4.2 Zapomněl jsem heslo....42
11.4.3 Píše: "There are mistakes in configuration file." (Ukázka 6.2 na straně 12)....42
11.5 Přechod mezi projekty....43
11.6 Odinstalace....43
12 Experimenty....44
12.1 První experiment....44
12.2 Druhý experiment....46
13 Závěr....48
13.1 Shrnutí....48
13.2 Návrhy na vylepšení....48
Literatura....50
Obsah CD....51
Použité programy....54
Příloha A – Původní podoba Makefile_module....55
Příloha B – Původní podoba Makefile....56
Příloha C – Nový Makefile pro modul o prvočíselnosti....57

Název práce: Distribuovaný systém pro ověřování vlastností přirozených čísel
Autor: Martina Tomisová
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Jiří Mírovský
E-mail vedoucího: mirovsky@ufal.ms.mff.cuni.cz

Abstrakt: Výsledkem práce je systém pro distribuované ověřování vlastností přirozených čísel. Systém má dvě části – klientskou a serverovou. Komunikace mezi nimi probíhá přes HTTP protokol. Server distribuuje zadání (přirozená čísla) a ukládá výsledky (vlastnost čísel) od klientů. Klienti provádí výpočty. Zadáním jednoho výpočtu je jedno přirozené číslo, výsledkem rovněž. Distribucí výpočtů je možné ověřit danou vlastnost pro více přirozených čísel. Konkrétní postup pro výpočet lze klientu poskytnout jako modul. Součástí práce jsou dva příklady modulů. První je jednoduchý a slouží jako vzor pro jejich vytváření. Druhý modul určuje, zda je číslo prvočíslo. Je v něm zahrnuta vlastní knihovna pro aritmetiku dlouhých čísel. Server tedy může distribuovat (potenciálně velká) čísla, klient ověří, zda dané číslo je prvočíslo.

Klíčová slova: distribuce výpočtů, prvočísla, dlouhá čísla, aritmetika, Rabinův-Millerův algoritmus

Tittle: Distributed System for Verification of Properties of Natural Numbers
Author: Martina Tomisová
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Jiří Mírovský
Supervisor's e-mail address: mirovsky@ufal.ms.mff.cuni.cz

Abstract: The result of my work is a system for distributed verification of properties of natural numbers. It has two parts – server and client. These communicate via HTTP protocol. The clients perform the computation, the server distributes the work (numbers) and gather results (properties of the given numbers). The input of one computation should be one natural number, as well as the result (output). The distribution can be used for verification of a given property for several natural numbers. Particular jobs can be added to the client as plugins. Two examples of plugins are a part of the work. The first one is very simple and shows how to create plugins. The second example searches for prime numbers (and has it's own arithmetics library for long numbers) - the server can distribute (possibly big) numbers, the client will verify whether a given number is a prime number.

Keywords: distributed computing, prime numbers, long numbers, arithmetics, Rabin-Miller's algorithm

1 Zadání práce

Zde cituji, jak přesně byla má práce zadána:

Vytvoření distribuovaného systému pro ověřování vlastností přirozených čísel. Server bude komunikovat s klienty pomocí HTTP protokolu. Výpočet bude prováděn klienty, server bude distribuovat práci (čísla) a sbírat výsledky (vlastnosti daných čísel). Zadáním jednoho výpočtu bude vždy jedno přirozené číslo, výsledkem rovněž. Distribuce výpočtu bude využita pro ověření dané vlastnosti pro více přirozených čísel. Konkrétní výpočet (úlohu) bude do systému možno modulárně vložit.

Součástí práce budou dva příklady modulů. První modul bude velmi jednoduchý a bude sloužit jako vzor pro vytváření modulů. Druhý modul bude vyhledávat prvočísla - server bude distribuovat (potenciálně velká) čísla, klient bude ověřovat, zda dané číslo je prvočíslo.

2 Motivace

Úlohy, které ještě před pár lety řešila velká vědecká výpočetní střediska, jsou dnes schopny spočítat naše domácí počítače. Výkonnost počítačů se stále zvyšuje, stejně tak naše nároky na ně. Chceme stále počítat složitější a větší úlohy. Ne každý, kdo by rád provedl nějakou rozsáhlou kalkulaci, má k dispozici dostatek procesorového času. Jedna z možností, jak i přes tento handicap uskutečnit své výpočty, je sehnat dostatek "slabších" počítačů a mezi ně práci rozdělit.

Distribuci úkolů lze zhruba rozdělit do dvou kategorií. První skupinu charakterizuje jeden dlouhý výpočet, který je paralelizován a každou část počítá jiný stroj. Takovéto části nemusí být vždy stejné. Druhou skupinu, kterou jsem se zabývala, tvoří úkoly, které lze rozdělit na mnoho částí, které se ale počítají každá stejným způsobem.

Distribucí řeší potřebu složitých výpočtů mnoho projektů z různých oborů, jako příklady uveďme matematiku, kryptografii, biologii, medicinu, fyziku, astrofyziku, meteorologii a dokonce i umění.

Jedním z nejznámějších projektů je *Great Internet Mersenne Prime Search (GIMPS)* [1], který se snaží najít prvočísla dlouhá 10 miliónů cifer. Ověřují se čísla ve tvaru 2^n-1. Výhodou takového přístupu je malý přenos dat mezi serverem a klientem a relativně málo využitá databáze – stačí ukládat exponent. Některá prvočísla ovšem tohoto tvaru nejsou a tedy není možné je najít pomocí tohoto projektu. Jejich klientský software podporuje většina běžně používaných operačních systémů.

Mezi další známé projekty patří *Distributed.net* týkající se kryptografie [2]. Populárním projektem je SETI podporovaný Kalifornskou univerzitou, který se snaží odhalit inteligentní mimozemské civilizace [3]. Dalším známým projektem, tentokrát ze světa biomedicíny, je Folding@home [5], zabývá se zkoumáním bílkovin (např. v souvislosti a Alzheimerovou a Parkinsonovou chorobou nebo rakovinou) a podporuje ho stanfordská univerzita.

Některé projekty používají svůj vlastní software, jiné využívají již napsané obecné projekty. Jedním z nejpoužívanějších je BOINC [4], na kterém je postaven např. výše zmiňovaný projekt SETI [3].

3 Co je Cuckoo

Cuckoo je systém pro distribuci a ověřování vlastností přirozených čísel. Skládá se ze dvou částí – klientské a serverové.

Server je jeden pro každý projekt a spravuje čísla určená k ověřování. Rozesílá je klientům a přijímá od nich výsledky.

Klientský program běží na mnoha různých stanicích. Uživatel si nainstaluje klienta, nastaví ho a dále se o něho již nemusí příliš starat. Program si sám říká serveru (bez nutnosti komunikace s uživatelem) o zadání práce, prověřuje čísla a odesílá výsledky.

Součástí mé práce je serverová i klientská část pro dva druhy projektů. První z nich je velmi jednoduchý, jeho cílem je určit, zda je číslo sudé, nebo liché.

Rozsáhlejším z nich je ověřování prvočíselnosti zadaných čísel. Modul je uzpůsoben i pro testování velmi dlouhých čísel, pracuje ve 256kové soustavě a má vlastní aritmetiku.

4 Obsah práce

Kapitoly 1 – 5 jsou určeny k úvodnímu seznámení s projektem, aby čtenář získal hrubou představu, k čemu je projekt určen a jak funguje.

Kapitola 6 je věnována podrobnému popisu základní části klienta, týká se především způsobu práce se zadáním a výsledky. Rovněž popisuje posloupnost akcí, kterou tato část klienta provádí (jak probíhá start programu a podobně).

V kapitole 7 se dočtete o modulární části klienta. Z čeho se skládá a jak pracuje. Důležitou částí této kapitoly je návod, jak si napsat vlastní modul včetně dvou příkladů.

Kapitola 8 se podrobně věnuje příkladu zmíněnému již v minulé kapitole a to ověřování prvočíselnosti. Rozebírá především použitou aritmetickou knihovnu a algoritmy pro samotné ověřování.

Devátou kapitolou postupně přecházíme k serverové části. Tato kapitola je věnována rozhraní mezi serverem a klientem – jaký tvar má jejich komunikace.

Kapitolka 10 nám osvětluje serverovou část projektu. Popisuje jeho fungování, strukturu databázových tabulek pro uchovávání zadaných čísel, výsledků i uživatelů. Závěr této části tvoří návod, jak a co upravit, pokud chceme server používat pro nově vytvořený modul.

Jedenáctá kapitola je věnována uživatelům. Dočteme se zde, jak si nainstalovat klienta i jak ho odinstalovat. Jsou zde popsána řešení častých problémů a podrobně krok po kroku vysvětleno, jak program ovládat.

Dvanáctou kapitolu, tvoří poznatky a úvahy z experimentálního spouštění projektu ve školní laboratoři.

Závěr tvoří kapitola 13 obsahující návrhy na vylepšení a shrnutí.

V posledních listech nalezneme seznam použité literatury, programů, obsah CD a přílohy.

5 Úvodní poznámky

5.1 Název projektu – Cuckoo

Cuckoo je anglické slovo, které v češtině vyjadřuje kukačku. Některé druhy kukaček jsou proslulé svým hnízdním parazitismem. Nebudují vlastní hnízda, místo toho nechávají svá vejce v cizích hnízdech. Podobně se chová můj projekt – jako kukačka roznáší svá vejce, server distribuuje zadání výpočtů, sám je neprovádí.

5.2 Jazyky

Klient je napsán pouze v jazyce C. Server je přizpůsoben pro provoz na webovém serveru, využívá tedy PHP a MySQL.

5.3 Požadavky na provoz

Klient je k dispozici v podobě balíčku zdrojových kódů. Aby ho bylo možné zkompilovat a spustit, je tedy nutné mít nástroje pro rozbalení balíčku typu *tar.gz*, nějaký unixový systém a kompilátor *gcc*. Klient byl testován na systémech Fedora 6, SuSE 10.0, Slackware 10.1, Gentoo a FreeBSD. Program nevyžaduje žádné speciální knihovny ani vlastnosti systému, a proto předpokládám, že bude zkompilovatelný a provozuschopný na mnoha dalších unixových systémech.

Pro provoz serveru je třeba webový server s PHP5 a MySQL databází. Testován byl na serveru Apache 2.0 s verzí MySQL 5.0.38-Debian_1 s protokolem verze 10 a PHP verze 5.2.2.

6 Programátorská dokumentace ke klientské části mimo modulární

6.1 Struktura projektu

Klient má dvě základní části. Pokud se podíváme do zdrojových kódů, první část nalezneme v adresáři *base*. Jak název napovídá, jedná se o základní část. Obsahuje veškeré funkce, které jsou nezávislé na aktuálním typu zadání výpočtů. Tato část je vždy stejná, ať se jedná o libovolný druh výpočtu. Patří sem například funkce pro komunikaci se serverem a správu čísel (udržují pořádek v tom, jaká čísla máme zadaná, jaká vyřešená a s jakým výsledkem).

Další část tvoří moduly. Ve zdrojových kódech je nalezneme v ostatních adresářích (co adresář, to jiný modul). Modul má v zásadě pouze jednu povinnou funkci, kterou musí obsahovat – funkci pro výpočet, které základní část předá zadání a funkce jí vrátí výsledek. Ostatních souborů a funkcí může modul obsahovat nebo využívat libovolně mnoho. Modul je základní částí programu načten (jako sdílená knihovna) krátce po startu. Funkci určenou pro provádění výpočtu pak využívá tu, která je obsažena v načteném modulu.

V každém adresáři *(base* i adresářích obsahujících moduly*)* jsou soubory *definitions.h* a *functions.h*. V prvním jmenovaném jsou uvedeny všechny konstanty vztahující se k dané části kódu. Druhý soubor obsahuje hlavičky všech použitých funkcí vztahujících se k modulu nebo základní části. U každé takto uvedené hlavičky je stručný komentář o funkci a jméno souboru, ve kterém je funkce definována.

6.2 Posloupnost akcí při běhu programu

V této podkapitole je stručně popsán běh programu – kdy, co provádí a podle čeho se rozhoduje. Detaily některých operací, které jsou zde uvedeny jsou blíže popsány v následujících podkapitolách.

6.2.1 Start

Ihned po spuštění program využívá pouze funkce z jeho základní části.

6.2.1.1 Jiné instance

První věc, co klient udělá po startu, je, že se podívá, zda neběží jiná jeho instance. Pokud ano, vypíše hlášení na standardní chybový výstup a ukončí se (viz. Ukázka 6.1). Pokud běží sám, vytvoří speciální soubor a zapíše do něho svůj PID. Tedy každý z tohoto souboru může poznat, že Cuckoo klient běží a s jakým PIDem. Podle existence tohoto souboru se orientuje i klient samotný při zjišťování, zda neběží jiná jeho kopie. Pokud tento soubor nalezne a PID v něm zapsaný je v seznamu aktuálně běžících procesů, usoudí, že není sám.

Jméno a umístění tohoto souboru je uložené v definičním souboru v definici *LOCK_FILE*.

Ukázka 6.1: Dvojité spuštění

```
[bug@localhost client]$ ./cuckoo
I'm allready running. You don't need to run me twice.
[bug@localhost client]$
```

6.2.1.2 Konfigurační soubor a registrace

Další informace, co program zajímá, je, zda je na daném počítači spuštěn poprvé, tedy zda má v konfiguračním souboru uložené potřebné informace. Cesta ke konfiguračnímu souboru je uložena v definici *CONF_FILE*.

Pokud soubor nenalezne, dotáže se uživatele na potřebné informace: přihlašovací jméno, heslo, jméno modulu, který bude používat pro výpočty, adresu serveru a počet čísel, který tvoří jednu dávku (komunikace se serverem probíhá většinou právě po jedné dávce, výjimečně po více dávkách najednou). Poslední tři jmenované hodnoty může uživatel zadat, nebo jen stisknout *Enter* a použity budou implicitní hodnoty, které jsou definovány v konstantách *DEF_MOD*, *DEF_WEB* a *DEF_TASKS*. Klient poté použije získané informace k registraci uživatele. Pokud se mu to podaří, uloží uživatelem zadané hodnoty do konfiguračního souboru a pokračuje dále. Pokud ne, vypíše chybové hlášení, hodnoty neukládá a ukončí se. Ukázka úspěšné registrace a následného započetí výpočtů je umístěna v kapitole uživatelské dokumentace jako Ukázka 11.1 na straně 40.

Jestliže program konfigurační soubor nalezne, pokusí se z něho přečíst potřebná data. Pokud soubor není poškozen (je v očekávaném formátu) a informace načte bez problémů, pokračuje dál. Jestliže soubor není ve formátu, který očekával, vypíše hlášení a ukončí se. V hlášení informuje uživatele, jak tento soubor vidí – uživatel tedy může snadno odhalit chybu a opravit ji.

Ukázka 6.2 ukazuje, jak se program zachová, pokud mu v konfiguračním

Ukázka 6.2: Chování při poškozeném konfiguračním souboru

```
[bug@localhost client]$ ./cuckoo
Content of your config file:
Login:
Password: *****
Name of module: ./primes.so
Server name: cuckoo.my-place.us
Number of tasks at single batch: 50
There are mistakes in configuration file.
```

souboru poškodíme řádek, kde je definováno uživatelské jméno. Z výstupu programu je zřejmé, že nebyl schopen načíst uživatelské jméno (je prázdné). Zbytek dat načetl, zdá se, v pořádku.

Pokud se podařilo konfiguraci zdárně načíst a zároveň je prázdné heslo, klient to vyhodnotí jako žádost o zaslání zapomenutého hesla na email. Zašle tedy serveru příslušný požadavek, informuje o tom uživatele a ukončí se (viz Ukázka 6.3). Uživateli pak přijde email s heslem, kde je požádán, aby si heslo do konfiguračního souboru zkopíroval.

Ukázka 6.3: Zaslání hesla na email

```
[bug@localhost client]$ ./cuckoo
Password successfully sent to your email.
[bug@localhost client]$
```

6.2.1.3 Odeslání výsledků a získání nové práce

Když má klient načteny všechny potřebné údaje, začne se konečně zabývat výpočty. Nejdříve se podívá, zda nemá uložené výsledky z minulých výpočtů, které ještě neodeslal. Jestliže ano, pokusí se je odeslat. Pokud se mu to povede, smaže je. Pokud ne, nechá je uložené a pokračuje dál. Pokusí se je odeslat, až spočítá celou další dávku (nebo dopočítá zbytek dávky od minulého spuštění). Jméno souboru s výsledky je definováno v konstantě *RES_FILE*.

Pak se klient podívá, jestli má nějaká zadaná čísla k výpočtu. Zadání je ukládáno do souboru, jehož jméno je definováno v konstantě *TASK_FILE*. Jestliže tento soubor existuje, lze otevřít pro čtení, a jestliže má nenulovou velikost, klient usoudí, že zadání má, a přejde k výpočtu samotnému.

Pokud však tento soubor neexistuje, nebo má nulovou velikost, je zřejmé, že žádné zadání není k dispozici. Klient se proto připojí k serveru a zažádá si o zadání. Požádá o tolik čísel, kolik má nastavenou velikost dávky. Příchozí čísla si uloží do souboru se zadáním a přejde k výpočtu.

6.2.2 Výpočet

V této fázi běhu programu je konečně využit modul. Klient se ho pokusí načíst. Pokud se povede, vyhledá v něm funkci, kterou musí modul povinně obsahovat.

Tato funkce musí mít následující hlavičku:

```
int count_out_main(char* num_string)
```

Funkce by měla vracet číslo typu *int*, které říká, zda dané číslo má či nemá danou vlastnost, případně do jaké míry ji má nebo do jaké třídy spadá. Toto vrácené číslo je pak beze změn přímo posláno jako výsledek serveru. Funkce má jeden parametr typu *char**, který bude vyplněn zadáním od serveru. Číslo je tedy funkci předáváno ve formě textového řetězce a modul si ho sám musí

převést do podoby, jakou požaduje.

Základní část klienta vezme číslo ze souboru obsahující zadání, zavolá na něho výše zmíněnou funkci *count_out_main* a získaný výsledek si ukládá. Po uložení výsledku smaže právě spočtené zadání ze souboru. Pak vezme další číslo a celou proceduru opakuje.

Po vypočtení celého zadání (právě jedna dávka, nebo zbytek dávky od minulého spouštění) se pokusí výsledky odeslat serveru. Chová se při tom stejně, jako na začátku běhu celého programu. Pokud se mu to podaří, výsledky smaže ze souboru. Pokud ne, nechá je tam a další výsledky připojuje za ně. Po pokusu o odeslání si řekne o nové zadání. Potom ho znovu počítá stejným způsobem.

6.3 Manipulace s čísly

6.3.1 Formát čísla, jak ho vidí server, základní část klienta a modul

Zatím je jedno zadání bráno jako číslo. Mezi serverem a klientem a také mezi základní a modulární částí klienta se však číslo předává jako řetězec znaků. Je tomu tak ze dvou důvodů:

1. Modulu se tak snadno předá číslo v libovolné soustavě nebo tvaru. Základní část klienta formát čísla nemusí zkoumat a nemusí do něho zasahovat, tedy nepotřebuje znát všechny použité soustavy a tvary zadávaných čísel. Modul si zadaný řetězec sám zpracuje na číslo.

2. Je snadné projekt rozšířit. Pokud bychom chtěli v zadání místo čísel používat zcela libovolné řetězce, stačilo by lehce upravit server a modulární část klienta. Základní část klienta by zůstala nezměněná.

6.3.2 Co se s číslem děje během procesu ověření

Nejdříve je číslo vyzvednuto ze serveru (v dávce s ostatními čísly) a uloženo do souboru se zadáním. Pak je každé postupně přečteno a vloženo do funkce *count_out_main*. Výsledek je spolu s číslem uložen do souboru výsledků a číslo je smazáno ze zadání. Následně (opět v dávce) je odesláno na server. Pokud se odeslání povede, je číslo smazáno i z výsledků. Ne vždy se ale manipuluje s celým skutečným číslem, jak je popsáno v následující podkapitole.

6.3.3 Formát zápisu čísla v souborech

Z důvodů urychlení a zjednodušení práce je občas místo čísla samotného používáno jeho ID. Každé číslo ze zadání má své unikátní ID, které je klientovi zasláno spolu s číslem.

Do souboru zadání je číslo uloženo spolu se svým ID, a to ve formátu

```
ID číslo
```

Příklad řádky ze souboru zadání je Ukázka 6.4. Vidíme zde číslo *12 4 55 82 12 45 232 22 0 1 23*, které má ID *233*.[1]

Ukázka 6.4: Číslo uložené v souboru zadání

```
233 12 4 55 82 12 45 232 22 0 1 23
```

Ve výsledcích je číslo uloženo už jen podle svého ID, skutečné zadání tu již není. To proto, že serveru se posílá zpět také pouze ID a výsledek (není důvod mu posílat znovu zpět dlouhé číslo). Ve výsledkovém souboru jsou čísla ve formátu

```
ID RESULT
```

Exemplární řádek nám nabízí Ukázka 6.5, číslo s ID *233* má výsledek testování *0*.

Ukázka 6.5: Číslo uložené v souboru řešení

```
233 0
```

1 *Toto číslo je zřejmě v jiné než desítkové soustavě. Jeho skutečný význam ale nemůžeme dovodit bez znalosti modulu, který ho zpracuje – záleží jen na něm, jakým způsobem číslo přečte. Tento konkrétní příklad je z projektu využívající modul pro ověřování dlouhých čísel, zda nejsou prvočísla. Je ve 256kové soustavě, kde jsou číslice odděleny mezerou.*

7 Programátorská dokumentace k modulární části klienta

7.1 Obecně o tvorbě modulu

Klient využívá tuto část pro samotný výpočet. Modul je tvořen sdílenou knihovnou, která povinně obsahuje funkci *count_out_main* s pevně danou hlavičkou:

```
int count_out_main(char* num_string)
```

Takovýto modul si může napsat každý, kdo má zájem o využití mého projektu. Přizpůsobení projektu pro konkrétní druh výpočtů vyžaduje dvě akce:

1. upravit server
2. napsat modul

Úpravě serveru se věnuje jedna z dalších kapitol. V zásadě jde pouze o úpravy výpisu výsledků a generování zadání do databáze.

Psaní modulu má jen několik omezení. Je třeba ho zkompilovat jako sdílenou knihovnu, která obsahuje výše zmíněnou funkci. Není nutné, aby se skládal z jednoho souboru. Je však třeba specifikovat, ve kterém souboru je uvedená funkce a tento soubor musí být uveden jako jméno modulu.

Funkce *count_out_main* musí být napsána tak, že přijme zadání ve formátu textového řetězce a vrátí číselný výsledek typu *int*.

Pro svůj modul je nutné upravit *Makefile_module* a *Makefile*. První ze zmíněných souborů má na starosti vytvoření modulu, ale nebývá určen k přímému použití. V balíčku bývají soubory *Makefile*, *Makefile_base* a *Makefile_module*. A právě *Makefile* slouží ke kompilaci celého projektu a využívá oba zbylé soubory.

Nic jiného tvorbu modulu nelimituje. Uveďme si nyní ukázkový postup krok po kroku, jak napsat vlastní modul a přidat ho do projektu.

7.2 Ukázková tvorba modulu – velmi jednoduchý příklad

Zde je příklad, jak si vyrobit vlastní modul. Jako jednoduchý příklad jsem vybrala modul, který rozhodne, zda je číslo sudé, nebo liché.

1. Stáhněte si balíček (*plain package*) obsahující základní část klienta z webu:

   http://cuckoo.my-place.us/downloads.php

   Tento balíček je umístěn i na přiloženém CD.

2. Rozbalte si ho

tar xfvz ./cuckoo_plain.tar.gz

3. Vedle adresáře *base* vytvořte adresář s názvem vašeho modulu. V našem příkladu se bude jmenovat *oddeven*.

mkdir oddeven

4. Do tohoto adresáře umístěte veškeré soubory týkající se modulu. Umístění do toho adresáře samozřejmě není povinné, ale vede k přehlednosti.

Povinně zde musí být soubor obsahující funkci *count_out_main*. Dále je doporučené umístit sem soubory *definitions.h* a *functions.h*. Do prvního z nich umístíme všechna makra a do druhého hlavičky všech funkcí.

Náš první příklad je tak jednoduchý, že nám stačí vytvořit zde pouze jediný soubor. Nazveme ho *count_out.c* a jeho obsah nám přibližuje Ukázka 7.1.

Tvar hlavičky funkce *count_out_main* je povinný. Funkce by měla být napsána tak, aby byla schopna přijmout zadané číslo jako posloupnost znaků *(char*)* a vrátit výsledek typu *int*.

5. Dále je nutné upravit soubor *Makefile_module*. Standardně je v balíčku ve stavu, který vidíme v příloze A. V balíčku jsou soubory

Ukázka 7.1: Obsah souboru count_out.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

/************************************************/
/*               count_out_main                 */
/************************************************/
/* zjišťuje, zda je dané číslo sudé, nebo liché  */
int count_out_main(char* num_string)
{
  char c[1];
  strncpy(c,num_string+strlen(num_string)-1,1);
  int x=atoi(c);
  return (x%2);
}
```

Makefile, *Makefile_base* a *Makefile_module*. První z nich slouží ke kompilaci celého projektu a využívá oba následující soubory. *Makefile_base* je soubor pro utváření základní části programu, do něho nezasahujeme. Upravovat budeme zatím pouze *Makefile_module*. Popišme si ho a zároveň si popišme změny, které v něm provedeme.

LIB_NAME je proměnná, ve které je uloženo jméno budoucího modulu. Nastavme ji tedy na oddeven.so. MOD_NAME nám říká, jak se jmenuje adresář, ve kterém máme uloženy zdrojové kódy k modulu. Zde napišme jméno adresáře, který jsme si vytvořili: oddeven.

INCL_LIB je seznam souborů (včetně cesty), které jsou zahrnuté ve zdrojových souborech modulu, jako například definitions.h a funtions.h. V našem příkladu je nastavím na prázdné.

LIB_FILES_C obsahuje seznam jmen zdrojových souborů modulu. *LIB_FILES_O* pak seznam jmen neslinkovaných souborů *.o*. Zachováme standardní nastavení, protože náš soubor se jmenuje stejně.

CC obsahuje cestu ke *gcc*, *CFLAGS* příznaky, se kterými se bude kompilace provádět. *RM* obsahuje cestu k programu *rm*.

Následují instrukce pro *make*, který cíl jak vybudovat. Do nich není třeba zasahovat, pokud nemáme nestandardní požadavky nebo příliš složitý modul. V takovýchto případech je nutné si vytvořit zcela vlastní *Makefile_module*. Z důvodů požadavků *Makefile* je velmi vhodné vždy ponechat funkční cíle *all* i *clean*.

6. Aby šel celý projekt (základní i modulární část) vytvořit jedním příkazem *make*, musíme ještě upravit soubor Makefile. Jeho standardní podoba je Příloze B. Zde je třeba správně nastavit proměnnou *MODL* na jméno souboru, který jsme editovali o odstavec výše. Dále by proměnná *MODL_NAME* měla obsahovat jméno vzniklého modulu. Zbylé proměnné není v tomto případě nutné měnit. Po těchto úpravách by měla kompilace proběhnout bez problémů pouhým spuštěním

```
make all
```

7. Než je celý projekt zabalen a distribuován uživatelům, je vhodné projít a upravit definiční soubor základní části klienta a upravit v něm konstanty, které se pak zobrazují uživateli jako implicitní odpověď na dotazy při instalaci. Konkrétně to jsou jméno serveru, velikost jedné dávky a jméno použitého modulu.

8. Pro plnou funkčnost projektu bývá nutné upravit i serverovou část. Této problematice se věnuji v dalších kapitolách.

7.3 Ukázková tvorba modulu – složitější příklad

Když je v návodu příliš jednoduchý příklad, může být sice snadno pochopitelný a stručně zapsaný, ovšem občas je nedostatečný. Proto ještě doplním minulou podkapitolu o složitější příklad – modul pro ověřování prvočíselnosti dlouhých čísel.

Body 1. a 2. zůstávají stejné. V bodě 3 uděláme drobnou změnu, pojmenujeme adresář např. *module_prime*.

Bod 4 se změní více. Do adresáře dáme soubor *count_out.c* a pomocný soubor *aritm.c*, obsahující funkce pro aritmetiku dlouhých čísel. Také přiložíme soubory *definitions.h* a *functions.h*, protože tento modul obsahuje relativně hodně definic a funkcí. Všechny použité soubory je možno nalézt na přiloženém CD, s ohledem na jejich délku je netisknu ani do přílohy.

V bodech 5 a 6 jsem probírala soubory typu *Makefile*. V tomto složitějším případě bude pozměněný *Makefile_module* mírně složitější – musíme navíc přidat soubor *aritm.c*. Výpis tohoto *Makefile_module* uvádím v příloze C. V souboru *Makefile* opět změníme pouze jméno souboru s modulem a proměnnou MODL v případě, že jsem *Makefile_module* pojmenovali jinak, než je standardní hodnota této proměnné.

Body 7 a 8 zůstávají platné beze změny.

8 Programátorská dokumentace k modulu pro ověřování prvočíselnosti

V mé práci jsou přiloženy moduly dva: pro ověření sudosti/lichosti čísla a prvočíselnosti. První z modulů je triviální a jeho dokumentace je obsažena v návodu pro utváření modulů, jako jednoduchý příklad. Dále se budu věnovat již pouze dokumentaci složitějšího z modulů.

Tento modul používá svoji vlastní aritmetickou knihovnu. Veškerý kód této knihovny je umístěn v souboru *aritm.c*.

8.1 Postup ověření prvočíselnosti

Modul si nejprve převede číslo z tvaru textového řetězce do tvaru dlouhého čísla (funkce *ln_create* a *ln_set_value*).

Pak je číslo prověřeno rychlým algoritmeme - Rabinovým-Millerovým (Ukázka 8.6 na straně 25, funkce *cnt_rm*). Tento algoritmus neumí stoprocentně zjistit, jestli je číslo prvočíslo. Pokud zjistí, že je číslo složené, je to pravda zcela jistě. Pokud se mu ale nepodaří nalézt důkaz o složenosti čísla, ještě to neznamená, že číslo je prvočíslo. A proto, než prohlásíme číslo za prvočíslo, ho musíme prověřit ještě jiným způsobem.

Tedy pokud se ukáže Rabinovým-Millerovým algoritmem, že číslo je složené, je vrácen výsledek *0*, tedy „složené číslo“. Pokud Rabinův-Millerův algoritmus žádného svědka složenosti čísla nenajde, je číslo prověřováno dále, tentokrát algoritmem, který ho ověří sice pomalu, ale s jistotou (Ukázka 8.7 na straně 26, funkce *cnt_real*). Podle tohoto výsledku vrací i celá hlavní funkce, buď *0* (složené), nebo *1* (prvočíslo).

8.2 Zásady využívání funkcí aritmetické knihovny

Knihovna obsahuje mnoho užitečných funkcí, jejichž kompletní seznam (hlavičky funkcí) je uveden v definičním souboru modulu, spolu s popisem, jak funkci správně použít. Zde uvádím pouze seznam hlavních funkcí s jejich stručným popisem. Po přečtení komentáře v kódu lze bez problémů používat i ty pomocné.

Tabulka 8.1.: Hlavní funkce

Účel funkce	Hlavička funkce	Stručný popis
Porovnání čísla s nulou	`int ln_is_zero (PLONG_NUMBER ln);`	Pokud je číslo nulové, vrátí 1, jinak 0.

Účel funkce	Hlavička funkce	Stručný popis
Vynulování čísla	`int ln_zeroize (PLONG_NUMBER ln, int size);`	Vynuluje číslo. Počet nulových číslic bude *size*. Vrací 0, pokud se akce povede.
Nastavení hodnoty	`int ln_set_value (PLONG_NUMBER ln, char *s,int s_len);`	Nastaví číslu hodnotu z řetězce s o délce s_len. Vrací 0, pokud se povede.
Uvolnění čísla	`void ln_free_number (PLONG_NUMBER ln);`	Uvolní paměť čísla.
Vytvoření čísla	`PLONG_NUMBER ln_create_number(void);`	Vytvoří číslo, které bude mít hodnotu nula. Vrátí NULL, pokud se nepovede, jinak ukazatel na číslo.
Porovnání čísel	`int ln_cmp_numbers (PLONG_NUMBER a, PLONG_NUMBER b);`	Vrátí 1, pokud $a>b$, nebo -1 pro $a<b$, nebo 0 pro a rovné b.
Přiřazení	`int ln_copy_number (PLONG_NUMBER a, PLONG_NUMBER b);`	Zkopíruje číslo a do b. Vrátí 0, pokud se povedlo.
Násobení	`int ln_mul (PLONG_NUMBER a, PLONG_NUMBER b, PLONG_NUMBER c);`	$c = a * b$ Vrací 0 při úspěchu.
Odčítání	`int ln_sub (PLONG_NUMBER a, PLONG_NUMBER b, PLONG_NUMBER c);`	$c = a - b$ Vrací 0 při úspěchu.
Sčítání	`int ln_add (PLONG_NUMBER a, PLONG_NUMBER b, PLONG_NUMBER c);`	$c = a + b$ Vrací 0 při úspěchu.

Účel funkce	Hlavička funkce	Stručný popis
Odmocnina	`int ln_sqrt (PLONG_NUMBER a, PLONG_NUMBER v);`	Do *v* uloží dolní celou část odmocniny z *a*. Vrací 0 při úspěchu.
Dělení	`int ln_div (PLONG_NUMBER a, PLONG_NUMBER b, PLONG_NUMBER v, PLONG_NUMBER pz);`	Do *v* uloží *a* div *b* a zbytek uloží do *pz*. Vrací 0 při úspěchu.

Než však můžeme nějakou funkci použít, musíme mít k dispozici číslo ve formátu, který knihovna používá. Jediná správná cesta, jak ho získat, je vytvořit si nejdříve prázdné číslo, pak mu případně nastavit hodnotu a pak až na něj můžeme volat další funkce. Po použití je nutné paměť zabranou číslem uvolnit. Příklad vytvoření a uvolnění takového čísla je v Ukázce 8.2.

Ukázka 8.2.: Příklad vytvoření a uvolnění dlouhého čísla

```
// vytvoříme číslo
PLONG_NUMBER num=ln_create_number();
// nastavíme mu hodnotu
ln_set_value(num,num_string,strlen(num_string));
...
// uvolníme číslo
ln_free_number(num);
```

Při nastavování hodnoty číslu je třeba, aby řetězec *num_string* byl ve správném tvaru. Takový tvar je obyčejný řetězec, kde jsou zapsány číslice v 256kové soustavě a oddělené mezerami. Tedy např. pro číslo 1 000 000 by byl takový tvar *„64 66 15“* (viz. Ukázka 8.3 a Ukázka 8.4 na straně 23).

8.3 Aritmetická knihovna pro dlouhá čísla

8.3.1 Formát čísla

Každé dlouhé číslo je uloženo ve 256kové soustavě[2] jako řetězec znaků (místo čísla ASCII znaku je zde číslice tohoto čísla). Přesný formát ukazuje Ukázka 8.3.

Ukázka 8.3.: Uložení dlouhého čísla

b^0	b^1	b^2	...	...	b^{n-1}	b^n

Znak *b* znamená základ soustavy, který je implicitně 256. Toto číslo lze nastavit v definičním souboru modulu konstantou *BASE*. Tedy např. číslo 1 000 000 bychom zapsali jako

Ukázka 8.4.: milion

64	66	15

protože $64*256^0+66*256^1+15*256^2=1\,000\,000$.

Číslo se ovšem neskládá pouze z číslic. Toto pole je zabaleno do struktury, která pak jako celek tvoří schránku pro jedno číslo. Tato struktura se nazývá *LONG_NUMBER* a je definovaná v definičním souboru modulu. Její definice je rovněž uvedena v Ukázce 8.5 na straně 24.

Struktura obsahuje dvě pomocné proměnné: *len* a *allocated*. Proměnná *len* udává, kolik číslic číslo má, a*llocated* nám říká, na kolik číslic máme pole alokováno. Další dva prvky struktury tvoří výše zmíněné pole obsahující číslice (*num*) a znaménková proměnná *sgn* – pokud je nulová, je číslo kladné, pokud je rovna *1*, je číslo záporné.[3]

8.3.2 Algoritmy použité v knihovně

Knihovna má mnoho funkcí a většina z nich není těžkých na pochopení. V kódu je mnoho komentářů, které funkci vysvětlují. Jsou zde ovšem i funkce, které jsou složitější, a ty vysvětlím zde.

---

2 *Nejmenší typ v jazyce C je „char“ se svou velikostí 1 byte, tedy 8 bitů. Do 8 bitů můžeme napsat číslo (bez znaménka) o maximální velikosti 255. Menší soustavu jsem nezvolila, protože bych plýtvala místem – bylo by zbytečné obsadit 8 bitů číslem, jehož maximální hodnota by byla menší nežli 255, některé bity by zůstaly nevyužité. Větší číslo by mělo smysl volit pouze v případě, že bychom místo typu „char“ využili jiný větší typ, například „int“ nebo „long int“. V takovém případě bychom zvolili velikost odpovídající počtu bitů, které daný typ tvoří.*

3 *Znaménko je sice u čísla zaznamenáno, ale v modulu pro výpočet prvočísel není potřebné, tedy je ignorováno.*

Ukázka 8.5.: Struktura LONG_NUMBER

```
typedef struct _LONG_NUMBER
{
  int len;          // počet číslic
  int allocated;    // na kolik číslic máme alokováno
  BOOL sgn;         // znaménko: 0-kladné, 1-záporné
  PLONG_DIGIT num; // ukazatel na pole s číslem
} LONG_NUMBER, *PLONG_NUMBER;
```

Sčítání a odčítání je implementováno algoritmem „tužka a papír“, jak ho učí děti ve škole.

Násobení zajišťuje Karatsubův algoritmus. Zde je jeho popis, který jsem našla ve wikipedii [6]. Přeložená citace:

Mějme dvě čísla x a y o n cifrách v soustavě o základu B, který nám umožňuje pohodlnou reprezentaci hodnot, jako např. 2 pro dnešní počítače.

Můžeme pak vybrat takové číslo m, menší než n, abychom mohli napsat:

$x = x_1 B^m + x_2$

$y = y_1 B^m + y_2$

kde x_2 a y_2 jsou menší než B^m. Je snadno vidět, že se jedná o jednoznačnou reprezentaci. Nyní máme

$xy = (x_1 B^m + x_2)(y_1 B^m + y_2) = x_1 y_1 B^{2m} + (x_1 y_2 + x_2 y_1) B^m + x_2 y_2$

Standardní metoda je násobit čtyři „podčísla“ zvlášť a nakonec je posunout a sečíst. Takovýto algoritmus funguje v čase $O(n^2)$. Nicméně Karatsuba vyzkoumal, že můžeme spočítat $x*y$ pouze na tři násobení.

Nechť $X = x_1 y_1$

Nechť $Y = x_2 y_2$

Nechť $Z = (x_1 + x_2)(y_1 + y_2) - X - Y$

Tedy víme:

$Z = (x_1 y_1 + x_1 y_2 + x_2 y_1 + x_2 y_2) - x_1 y_1 - x_2 y_2 = x_1 y_2 + x_2 y_1$

Proto $x*y = X B^{2m} + Y + Z B^m$. Snížili jsme počet násobení ze čtyř na tři. Na spočtení těchto tří členů vzniklých z m-ciferných čísel můžeme opět

rekurzivně použít Karatsubův algoritmus. Posuny, sčítání a odčítání zaberou množství času lineárně závislé na délce čísla, což je téměř zanedbatelný čas.

Karatsubovo násobení funguje nejlépe, když jsou oba činitelé stejné délky. I když bychom si teoreticky mohli vybrat m libovolně, nejlepšího času dosáhneme, když nastavíme m tak, že obě výsledná čísla budou zhruba stejně dlouhá – tedy je vhodné vybrat m jako polovinu n.

Ukázka 8.6.: Rabinův-Millerův algoritmus

```
# n je testované číslo
m  = n - 1
mm = n - 2
najdi takové d a maximální s, že:
  s>=2 && m=2^s * d
for i=1..t
  a=rand<2,mm>
  y=a^d mod n
  if(y==1 || y==m) continue
  j=1
  while(j<=s-1 && y!=m)
    y=y^2 mod n
    if(y==1) SLOŽENÉ
    j++
  if(y!=n-1) SLOŽENÉ
MOŽNÁ PRVOČÍSLO
```

Odmocňování a dělení jsou implementována tak, že výsledek je nejdříve odhadnut a následně je odhad zpřesňován, dokud nedojdou ke skutečnému výsledku. Pro účely zpřesňování výsledku se využívá metoda půlení intervalů, tedy potřebujeme mít implementováno dělení dvěma. Toto dělení je uděláno zvlášť metodou „tužka a papír“.

8.4 Algoritmy pro testování čísel

Tyto funkce již nespadají do aritmetické knihovny, ale jsou implementovány v souboru *count_out.c*.

Ukázka 8.7.: Ověřování prvočíselnosti

```
r = dolní celá část z odmocniny z n
a = r
b = r
q = n - r^2
if( !q ) return 0
while( b > 1 )
  while( q >= b )
    a++
    q- = b
  if( !q ) return 0
  b--
  q += a
return 1
```

8.4.1 Rabinův-Millerův algoritmus

Algoritmus jsem převzala z wikipedie [8]. Jeho symbolický zápis je v Ukázce 8.6 na straně 25.

Pro tento algoritmus potřebujeme umět spočítat $a^d \ mod \ n$. Na to jsem použila algoritmus, který jsem nalezla na stránkách univerzity George Masona a který je symbolicky zapsán v Ukázce 8.8 na straně 26 [7].

Ukázka 8.8.: Výpočet $^{základ\ mocnina}$ % modulo

```
výsledek = 1
while mocnina > 0
  if mocnina & 1 == 1
    výsledek = (výsledek * základ) % modulo
  základ = základ^2 % modulo
  mocnina >>= 1;
```

8.4.2 Algoritmus jistého ověřování prvočíselnosti

Tento algoritmus jsem nikde nenašla, poradil mi ho spolužák Martin Molnár. Algoritmus ověřuje, zda je číslo prvočíslo, tentokrát již s jistým výsledkem. Jeho zápis je v Ukázce 8.7 na straně 26. Algoritmus je pomalý, ale přesný. Číslo je testováno v cyklu, jako iterační proměnná je použita dolní celá část z odmocniny čísla. Tato proměnná se postupně zmenšuje o jedničku. V každém cyklu proběhne test. Mimo odmocňování algoritmus vůbec nepracuje s násobením a dělením, pouze sčítá a odčítá.

9 Rozhraní mezi klientem a serverem

Všechny funkce, které klientovi umožňují komunikaci se serverem, jsou umístěny v základní části klienta v souboru *client_communication.c*. Jejich implementace je zcela přímočará a komentovaná v kódu. V následujícím textu je věnuji tvaru zpráv, které si mezi sebou klient a server vyměňují.

Na serveru tuto část (komunikace s klientem) zajišťuje soubor *index.php*.

Server běží na webovém serveru, tedy komunikace probíhá pomocí HTTP protokolu. Klient vždy zašle přesně zformulovaný dotaz na server a dostane odpověď.

Zde je přehled možných dotazů, jejich význam a možné odpovědi serveru.

Dotaz:

```
index.php?reg=1&login=jmeno&email=mail@server.com&passwor
d=password
```

Význam: Registrace uživatele

Klient žádá o registraci nového uživatele, který si přeje mít uživatelské jméno „*jmeno*“, jehož email je „*mail@server.com*“ a jeho heslo bude nastaveno na „*password*“.

Možné odpovědi:

- ERR1 takové uživatelské jméno již existuje
- ERR2 takový email již existuje
- ERR3 již existuje takový email i uživatelské jméno
- OK registrace proběhla v pořádku

Dotaz:

```
index.php?for=1&login=jmeno
```

Význam: Žádost o připomenutí hesla

Klient žádá server, aby zaslal email se zapomenutým heslem na email, který má uložený pro uživatele s uživatelským jménem *„jmeno“*.

Možné odpovědi:

- SEND heslo bylo zasláno na email
- ERR klient s takovým uživatelským jménem neexistuje, nebo jiná chyba

Dotaz:

```
index.php?res=1&login=jmeno&password=heslo&count=2&id0=10
&res0=1&ln0=12&id1=11&res1=3&ln1=16
```

Význam: Zaslání výsledků

Klient posílá serveru výsledky svých výpočtů. Autentizuje se uživatelským jménem *„jmeno“* a heslem *„heslo“*. Zasílá 2 výsledky *(count)*. Výsledek je pro první číslo: ID čísla je 10, výsledek výpočtu je 1 a výpočet trval 12 vteřin. Výsledky pro druhé číslo: ID čísla je 11, výsledek je 3 a výpočet trval 16 vteřin.

Takto je možné připojit větší počet výsledků. Je však důležité, uvést jejich přesný počet do proměnné *count*.

Možné odpovědi:

- AUTH_FAILED nezdařila se autentizace nebo jiná chyba
- OK výsledky přijaty

Dotaz:

```
index.php?get=1&login=jmeno&password=heslo&count=10
```

Význam: Žádost o zadání práce

Klient žádá o zadání práce. Autentizuje se při tom uživatelským jménem *„jmeno“* a heslem *„heslo“*. Žádá dávku práce o velkosti maximálně 10 čísel.

Možné odpovědi:

- AUTH_FAILED nezdařila se autentizace nebo jiná chyba
- Dvojice čísel ve formátu

  x_1 y_1

  x_2 y_2

  : :

  END

  kde x_i je ID čísla a y_i je číslo samotné.

10 Dokumentace k serveru

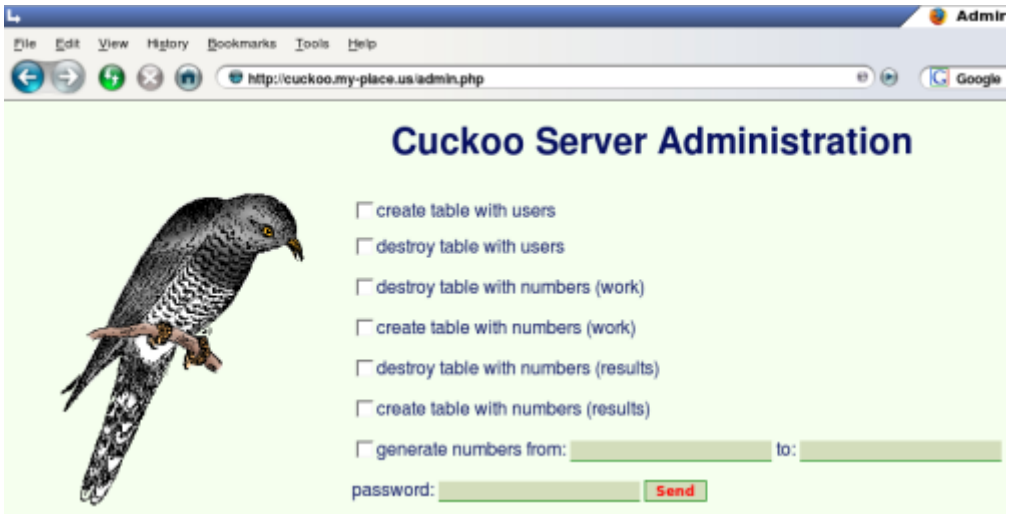
10.1 Z čeho se server skládá

Server lze rozdělit do několika částí:

1. Administrátorské skripty – nabízejí možnost manipulace s tabulkami databáze, generování čísel na prozkoumání, výpisy výsledků
2. MySQL databáze uchovávající data
3. Stránka pro uživatele a zájemce o projekt – obsahuje odkazy na balíčky ke stažení (stránka s adresou *downloads.php*)
4. Stránka pro komunikaci s klienty je popsána na konci minulé kapitoly

10.2 Administrátorské skripty

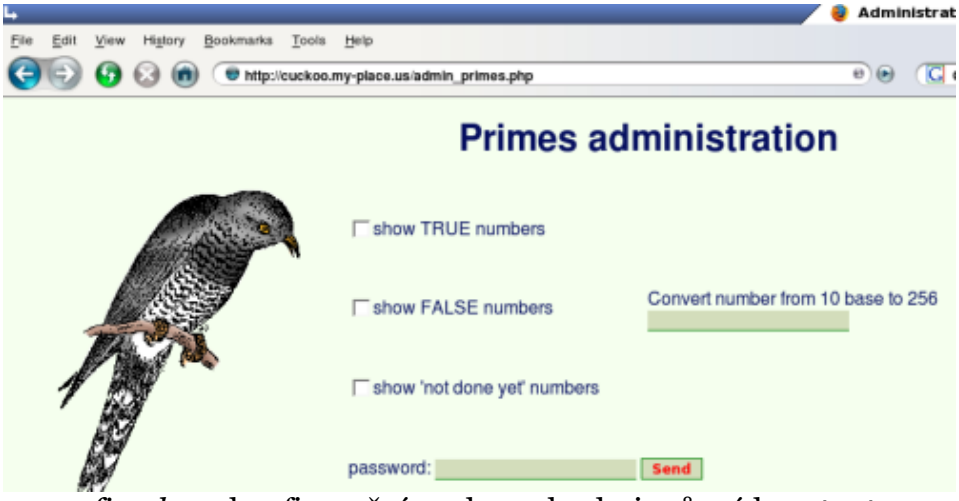
Tato část serveru se skládá z následujících souborů (jejich podrobnější popis následuje níže):



Ukázka 10.1: admin.php

- *admin.php* (Ukázka 10.1), *admin_module.php* (Ukázka 10.2) – hlavní stránky pro administraci, obsahuje formuláře pro základní obsluhu databáze

Ukázka 10.2: admin_primes.php



- *config.php* – konfigurační soubor, obsahuje různé konstanty
- *functions.php, functions_module.php* – soubory obsahující pomocné funkce, které využívá *admin.php, admin_module.php* i *index.php*

Soubory s příponou *_module.php* bývá nutné modifikovat podle aktuálně používaného modulu. Povinně musí obsahovat pouze funkce *convert* a *add_numbers* popsané v podkapitole o souboru *admin.php*.

10.2.1 config.php

Soubor obsahující nastavitelné konstanty. V Ukázce 10.3 vidíme jejich přehled a vysvětlení.

Ukázka 10.3: Nastavitelné konstanty souboru config.php

Jméno konstanty	Popis
`PASS`	MD5 hash hesla, které musí administrátor zadat při používání formulářů na stránce *admin.php*

Jméno konstanty	Popis
NDIF_USERS	Pokud je tato hodnota nenulová, pak bude akceptován více jak jeden výsledek od jednoho uživatele pro jedno zadané číslo. Jedním uživatelem je myšlen uživatel, který má stejné uživatelské jméno nebo IP adresu. Nenulová hodnota je v běžném provozu nebezpečná, usnadňuje padělání výsledků. Pokud je hodnota nastavena na nulu, musí být každý přijatý výsledek od jiného uživatele, výsledky neodpovídající tomuto pravidlu jsou zahazovány. Aby byl pak výsledek celkově uznán za platný a definitivní, je třeba, aby ho dosáhlo všech *NT_COUNT* různých uživatelů.
MAXLEN	Maximální délka uživatelského jména, emailu nebo hesla.
MYSQL_HOSTNAME	Adresa MySQL serveru.
DB_name, DB_login, DB_password	Jméno MySQL databáze, přihlašovací jméno a heslo k ní.
PR_admin	Email na administrátora projektu.
PT_table_name, PT_login, PT_email, PT_password	Jméno tabulky v databázi, která uchovává údaje o uživatelích. Jména sloupců v této tabulce, která obsahují uživatelské jméno, email a heslo uživatele.
TZ_table_name	Jméno tabulky, která obsahuje zadávaná čísla.
TZ_id,TZ_num, TZ_result, TZ_time, TZ_many	Názvy sloupců v tabulce se zadáním pro ID čísla, číslo obsahující výsledek, čas posledního zadání nějakému uživateli, kolikrát bylo uživatelům zadáno celkem.
TR_table_name, TR_id, TR_checking, TR_login, TR_ip, TR_long	Jméno tabulky, co obsahuje výsledky a jména sloupců v ní obsažených: ID čísla, výsledek, uživatel, který výsledek odeslal a jeho IP adresa, jak dlouho (obvykle ve vteřinách) výpočet trval.
NT_count	Kolik uživatelů musí ověřit číslo se stejným výsledkem, aby byl výsledek uznán za platný.

Jméno konstanty	Popis
`NT_dtime_unit, NT_dtime_val`	První konstanta specifikuje jednotku a druhá množství. Společně udávají, po jak dlouhé době může být číslo znovu někomu přiděleno. Pokud bychom totiž přidělovali čísla stále od počátku, dokud bychom neměli potvrzen výsledek, jedno číslo by bylo ověřováno zbytečně mnoha uživateli. Možné hodnoty jednotky jsou: *FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER,* nebo *YEAR*.
`NT_sql_limit`	Kolik generovaných čísel se má přidat do tabulky maximálně najednou.

10.2.2 admin.php a admin_module.php

Soubor *admin.php* je obecný pro téměř všechny druhy projektů. Obsahuje formulář, jehož součástí je kolonka na heslo, tedy nikdo, kromě administrátora, by neměl mít k funkcím tohoto souboru přístup. Popišme si jeho funkce:

- **Zničení tabulky uživatelů, vytvoření tabulky uživatelů**

Zcela přímočaře odstraní nebo založí tabulku uživatelů. Vzhled tabulek je popsán v podkapitole o tabulkách databáze.

- **Zničení nebo vytření tabulek pro uchovávání zadání nebo řešení**

Opět zcela přímočaře řešeno, popis tabulek v podkapitole o tabulkách databáze.

Ukázka 10.4.:

```
   function
convert($x)
   {
     return $x;
   }
```

Ukázka 10.5.: funkce convert pro převod soustav

```
function convert($num)
{
  $result=array();
  $i=0;
  do
  {
    $r=div_num($num,256);
    $vysledek=$r[0];
    $zbytek=$r[1];
    $num=$vysledek;
    $result[$i]=$zbytek;
    $i++;
  }while($vysledek);
  $res=array_reverse($result);
  $r=implode(" ",$res);
  return $r;
}
```

- **Generování čísel zadání**

Vygeneruje čísla na prozkoumání podle zadaného intervalu. Interval je zleva uzavřený. Protože se může stát, že ještě před generováním těchto čísel chceme vstup (hranice intervalu) upravit, prochází vstup nejprve funkcí *convert*, V této funkci je například pro modul prvočísel uskutečněn převod mezí intervalu do 256kové soustavy, v jednoduchém modulu o sudosti je tato funkce ponechána prázdná a vrací, co do ní bylo zadáno. Tato funkce by měla být vždy umístěna do souboru *functions_module.php*. Je povinná, tedy pokud není žádná konverze potřeba, měla by být prázdná a vracet, co do ní přichází. V Ukázce 10.5 na straně 33 vidíme její definici pro modul zkoumající prvočíselnost, tedy převádí čísla do 256kové soustavy. Z Ukázky 10.4 na straně 33 vidíme, jak napsat tuto funkci, pokud si žádnou konverzi nepřejeme.

Generování čísel pak zařizuje funkce *add_numbers*, která je opět povinná a měla by být umístěna taktéž do souboru *functions_module.php*. Její návratová hodnota není důležitá. Jako argumenty dostává funkce čísla *od, do* a handle na otevřenou databázi. Měla by do ní vložit čísla z intervalu <*od, do*). Pro jednoduché moduly lze využít dvě předepsané funkce, umístěné v *functions.php*. Tyto

funkce se nazývají *add_numbers_short* a *add_numbers_long*. První vloží čísla jednoduše v desítkové soustavě, druhá očekává meze intervalu již ve 256kové soustavě a uloží do databáze čísla ve 256kové soustavě. Ukázka 10.6 nám ukazuje, jak je vyřešena funkce *add_numbers* ve *functions_oddeven.php* (pro jednoduché vložení čísel využijeme již předepsanou funkci *add_numbers_short*).

Ukázka 10.6.:

```
function add_numbers($from,$to,$mysql)
{
  add_numbers_short($from,$to,$mysql);
}
```

Soubor *admin_module.php* je značně závislý na modulu, pro který má sloužit, je tedy na administrátorovi projektu, aby si ho upravil dle své potřeby. Soubor není povinný, v projektu se nemusí vůbec vyskytovat. Může obsahovat pomocné formuláře – například *admin_primes.php* obsahuje formulář pro převod čísla z desítkové do 256kové soustavy, který se může občas hodit. Také může obsahovat funkce pro základní výpis výsledků (toto dělá funkce *show* v souboru *functions_primes.php* i *functions_oddeven.php*).

10.3 Struktura databáze

Názvy tabulek i jejich sloupců lze nastavit v souboru *config.php*. Více v podkapitole o tomto souboru.

Celá databáze se skládá ze tří tabulek: tabulka uživatelů, tabulka zadání a tabulka řešení. Popišme si je blíže.

10.3.1 Tabulka uživatelů

10.3.1.1 Struktura

Ukázka 10.7: Struktura tabulky uživatelů

Jméno	Proměnná obsahující jméno sloupce	Typ
Uživatelské jméno	PT_login	varchar(64)
Email	PT_email	varchar(64)
Heslo	PT_password	Varchar(64)

Klíčem v této tabulce je uživatelské jméno a email.

10.3.1.2 Příklad tabulky

Ukázka 10.8: Příklad tabulky uživatelů

login	email	password
Anicka	anna@seznam.cz	Honzicek
Honza	Honzik@email.cz	MujPesBobik
Pepa	Pepa@google.com	Mojeheslo

10.3.2 Tabulka zadání

10.3.2.1 Struktura

Ukázka 10.9.: Struktura tabulky zadání

Jméno	Proměnná obsahující jméno sloupce	Typ
ID	TZ_id	int(11)
Číslo	TZ_num	varchar(1024)
Výsledek	TZ_result	tinyint(4)
Poslední	TZ_time	timestamp
Kolikrát	TZ_many	int(11)

Klíčem je ID čísla.

10.3.2.2 Příklad tabulky

Ukázka 10.10: Příklad tabulky zadání

id	num	result	last_time	many
1	15 66 64	0	2007-05-05 09:26:05	5
2	15 66 65	0	2007-05-05 09:26:05	5
3	15 66 66	0	2007-05-05 09:26:05	5
4	15 66 67	1	2007-05-05 09:26:05	5
5	15 66 68	0	2007-05-05 09:26:05	5
6	15 66 69	0	2007-05-05 09:26:05	5
7	15 66 70	0	2007-05-05 09:26:05	5

10.3.3 Tabulka řešení

10.3.3.1 Struktura

Ukázka 10.11.: Struktura tabulky řešení

Jméno	Proměnná obsahující jméno sloupce	Typ
ID	TR_id	int(11)
Výsledek	TR_checking	tinyint(4)
Uživatelské jméno	TR_login	varchar(64)
IP adresa	TR_ip	varchar(15)
Doba	TR_long	bigint(20)

Tato tabulka nemá žádný klíč. Z použití přesto plyne, že by se zde neměly vyskytnout řádky, které mají stejné ID a uživatelské jméno nebo ID a IP adresu.

10.3.3.2 Příklad tabulky

Ukázka 10.12: Příklad tabulky řešení

id	chck	login	ip	how_long
1	0	Anicka	195.113.21.148	0
2	0	Anicka	195.113.21.148	0
3	0	Honzik	195.113.21.149	0
4	1	Honzik	195.113.21.149	1
4	1	Jack	195.12.1.111	1
6	0	Anicka	195.113.21.148	0
5	0	Anicka	195.113.21.148	0
8	0	Honzik	195.113.21.149	0

10.4 Stránka pro stahování

Stránka *download.php* je určena uživatelům, kteří si chtějí stáhnout balíček se zdrojovými kódy nebo toto povídání.

10.5 index.php

Jak jsem zmínila výše, tento soubor slouží výhradně pro komunikaci s klientským programem. Tvary požadavků jsou vypsány výše. Zde se budu věnovat postupům, jak jsou požadavky plněny.

10.5.1 Registrace klienta

Nejprve se zkontroluje, zda dané uživatelské jméno nebo email ještě neexistují, a pokud jsou volné, uživatel je přidán do databáze.

10.5.2 Připomenutí hesla

Zkontrolujeme, zda uživatele máme v databázi, a pokud ano, získáme z databáze jeho email a na ten odešleme zprávu obsahující zapomenuté heslo.

10.5.3 Příjem výsledků

Nejprve autentizujeme uživatele pomocí jeho uživatelského jména a hesla, a pokud toto proběhne v pořádku, přijmeme výsledky. Postupně rozebíráme řetěz výsledků, co nám klient zaslal, a po jednom je ukládáme do tabulky řešení.

Číslo uložíme, pokud ještě není definitivně otestováno. Pokud již je, výsledek zahodíme. Také kontrolujeme, zda pro stejné číslo neposílá výsledek někdo, kdo ho již jednou ověřoval (má stejné uživatelské jméno nebo IP adresu).

Pokud se právě zaslaný výsledek liší od výsledků předchozích, nastává problém – někde nastala chyba, nebo někdo úmyslně bojkotuje náš projekt. Server v této situaci jen těžko pozná, kdo má výsledek skutečně správně. Proto smaže všechny dosavadní výsledky tohoto čísla, včetně toho naposledy zaslaného (odlišného).

10.5.4 Zadání práce

Opět nejdříve prověříme totožnost klienta. Poté z databáze vybereme jím požadovaný počet čísel a odešleme mu je.

Systém vybírání čísel není jednoduchý. Pokud bychom brali čísla vzestupně podle ID a jen ta, která ještě nejsou definitivně vyřešená, všichni klienti by dostávali první číslo, dokud by se nevyřešilo. To by vedlo k tomu, že by ho počítalo mnohonásobně více klientů, než je třeba. V podstatě by to téměř znehodnotilo celý proces paralelizace.

Každé číslo má v zadání u sebe uloženou ještě další přídavnou informaci o tom, kdy bylo naposledy zadáno k vyšetření. Čísla se tedy zadávají podle tohoto pravidla: vyber čísla, která ještě nejsou definitivně vyřešena. Z těchto čísel vyber ta, která byla zadána méně-krát, nežli je nutný počet nezávislých ověření pro definitivní ověření. Tento výběr sjednoť s výběrem čísel, která nejsou definitivně ověřena, ale již byla zadána dostatečnému počtu klientů a naposledy byla zadána před více než nějakou zadanou dobou (viz. *config.php*).

10.6 Jak nainstalovat a nakonfigurovat server

Stáhneme si balíček se soubory pro server, rozbalíme ho a soubory na server nakopírujeme. Poté upravíme soubor *config.php* podle našich požadavků. Následně upravíme soubor *functions_module.php* – zde je nutné nastavit funkce *add_number* a *convert*. Více se o nich dočtete v podkapitole o souboru *admin.php*. Volitelně je možné si vytvořit soubor *admin_module.php*. Nyní již zbývá pouze vytvořit příslušné tabulky v databázi a vygenerovat zadání práce. To lze pomocí formuláře na stránce *admin.php*, kde zaškrtneme políčka pro vytvoření všech tří tabulek a také políčko pro generování čísel, u něhož vyplníme i meze intervalu zadání. Zadáme heslo a klikneme na „Send“.

Výsledky si lze prohlížet pomocí funkce *show*, která je napsána pro oba přiložené moduly v souboru *functions_module.php*. Až na výjimky mají dnešní MySQL databáze k dispozici takzvaný *PhpMyAdmin*, kterým lze databázi prohlížet velmi pohodlně a bez omezení. Proto mi přišlo zbytečné psát jiný rozsáhlý systém na prohlížení výsledků a tedy doporučuji výsledky probírat pomocí tohoto modulu.

11 Uživatelská dokumentace

Následující stránky jsou určeny uživatelům. Obsahují návod, jak nainstalovat klienta, provedou uživatele prvním spuštěním, seznámí ho s tipy pro běžné použití. Nalézají se zde a návody pro řešení běžných problémů a situací, které mohou nastat.

11.1 Požadavky na systém

- nástroj pro rozbalení balíčku typu *tar.gz* (např. program *tar*)
- unixový systém (testováno na systémech Fedora 6, SuSE 10.0, Slackware 10.1, Gentoo a FreeBSD)
- kompilátor *gcc*

11.2 Instalace a spouštění

Stáhněte si balíček a uložte ho do svého */home* adresáře.

1. Rozbalte balíček

```
tar xfvz ./cuckoo.tar.gz
```

2. Přejděte do adresáře se zdrojovými kódy

```
cd ./cuckoo/source
```

3. Zkompilujte

```
make all
```

4. Přesuňte binární soubor a modul do *~/cuckoo*

```
make install
```

5. Smažte nepotřebné soubory

```
make clean
```

6. Jděte do adresáře *~/cuckoo* a spusťte program

```
cd ~/cuckoo
```

Při prvním spuštění bude mít program mnoho dotazů. Příklad spuštění vidíte v Ukázce 11.1 na straně 40. Je třeba, abyste zadali uživatelské jméno, které chcete používat, kontaktní email pro případ ztráty hesla a heslo. Dále se program bude ptát na jméno modulu, který má použít. Pokud nevíte, stiskněte pouze *Enter* a program si vybere modul, který má implicitně nastaven. Další dotaz se týká serveru, který má být použit. Pokud nevíte, opět stiskněte *Enter*. Poslední otázka, kterou Cuckoo položí, se týká velikosti dávky, což je počet zadaných čísel, která bude program řešit najednou bez komunikace se serverem. Doporučuji nechat přednastavenou velikost, protože lze předpokládat, že tvůrce projektu tuto otázku již vyřešil a

doporučené číslo nastavil právě jako implicitní hodnotu.

Ukázka 11.1: První spuštění Cuckoo

```
[bug@localhost client]$ ./cuckoo

There is no config file. That means you are a new user who is not
registered. If I am wrong and you are registered, please copy
the example congif file to cuckoo.conf and edit your login and
password there. Press Ctrl+C to stop registration of a new user.
If you don't remember your password, set it to the blank one by
editing your configuration file to PASSWORD=

If you are new user, please give me your login and password now,
I will register you.

Login: my_login
Email: my_email@server.cz
Password:
Used module: [./primes.so] ./other_module.so
Server (without any http://): [cuckoo.my-place.us]
Number of tasks in one batch: [50] 100
Registered on the web successfully!
Username and password stored succesfully at the file cuckoo.conf.
New tasks successfully brought.
Checking numbers...
Module selected: ./other_module.so
New tasks successfully brought.
...
```

Po úspěšném zodpovězení všech těchto otázek začne Cuckoo pracovat a již mu nemusíte věnovat pozornost. Během výpočtů občas vypisuje informační hlášení. Pokud nechcete, aby vás tím obtěžoval, ukončete ho pomocí *Ctrl+C* nebo mu jinak pošlete signál *SIGINT*.

Potom ho znovu spusťte, tentokrát na pozadí s přesměrovaným výstupem, jak ukazuje Ukázka 11.2. Tento řádek můžete i jednoduše umístit do startovacích skriptů.

Ukázka 11.2.: Spuštění Cuckoo na pozadí a přesměrováním výstupu

```
./cuckoo &> /dev/null &
```

Při dalším spouštění se již nebude na nic ptát (pokud předchozí registrace

proběhla bez chyb) – všechny potřebné údaje má již uložené ve svém konfiguračním souboru *cuckoo.conf*, který je umístěn ve stejném adresáři jako Cuckoo.

Na druhou stranu není na škodu, když Cuckoo různá hlášení vypisuje. Jste tak informováni o tom, zda pracuje, případně proč nepracuje.

Nemusíte se bát, že vám bude Cuckoo příliš zatěžovat procesor a že tím bude zpomalovat ostatní aplikace. Ihned po spuštění si Cuckoo samo sníží prioritu na nejnižší možnou.

11.3 Ukončení programu

Pokud program běží na pozadí a vy ho chcete ukončit, použijte příkaz z Ukázky 11.3.

Ukázka 11.3.: Ukončení programu

```
pkill cuckoo
```

Pokud Cuckoo spustíte a ono se samo ihned ukončí bez vypsání chyby, je to v pořádku. Znamená to, že server momentálně nemá žádnou práci, kterou by mohl program počítat. Zkuste program spustit znovu za nějakou dobu. Může to být pár vteřin, ale i několik dní, záleží na správci serveru, jak server nastavil a jak ho obsluhuje. Pokud chcete tyto informace vědět, napište mu email.

11.3.1 Konfigurační soubor

Standardní název konfiguračního souboru je *cuckoo.conf* a bývá umístěn ve stejném adresáři jako Cuckoo. V každém balíčku Cuckoo je přibalen soubor *cuckoo.conf.example* jako ukázkový konfigurační soubor. Většinou se ale needituje ručně – Cuckoo si při prvním spuštění sám vytvoří vhodný konfigurační soubor.

Každý řádek, který začíná (tedy hned první znak na řádku je) „#“, je brán jako komentář. Stejně tak jsou ignorované řádky prázdné nebo obsahující pouze mezery a tabulátory.

Hodnota se zadává ve tvaru

Ukázka 11.4.: Syntax konfiguračního souboru

```
PROMĚNNÁ=hodnota
```

Každá proměnná musí být na samostatném řádku. Nepoužívejte uvozovky, ani escape sekvence. Všechny znaky za „=“ jsou brány přesně tak, jak jsou napsány, až do konce řádku (včetně mezer).

Ukázka 11.5 na straně 42 ukazuje, jak může soubor vypadat. V tomto výpisu jsou použity všechny proměnné, které umí cuckoo zpracovat.

Ukázka 11.5.:

```
# uživatelské jméno
LOGIN=your_login
# heslo
PASSWORD=your_password
# použitý modul
MODULE=/path/to/the/module
# adresa serveru (bez http://)
WEB=www.server.address
# velikost dávky
TASKS=50
```

11.4 Řešení problémů

11.4.1 Píše: "I'm already running. You don't need to run me twice." (Ukázka 6.1 na straně 12)

Toto hlášení Cuckoo vypisuje, pokud si myslí, že běží dvakrát. Pokud jste si zcela jisti, že dvakrát neběží, smažte soubor */tmp/cuckoo.pid*.

11.4.2 Zapomněl jsem heslo

Pokud jste pouze zapomněli heslo (a z nějakých příčin ho nemáte uložené v konfiguračním souboru) a pamatujete si své uživatelské jméno, bude vám posláno na email, který jste zadali při registraci. Opravte konfigurační soubor tak, že místo hesla neobsahuje nic - za znakem „=“ nechte pouze prázdný řádek (nesmí obsahovat ani neviditelné znaky jako jsou mezery a podobně). Pak spusťte Cuckoo. Mělo by vypsat to, co vidíte v Ukázce 6.3 na straně 13. Server vám zaslal email s heslem, to zkopírujte do konfiguračního souboru. Tím by měla být situace vyřešena.

Pokud si nepamatujete ani uživatelské jméno, bohužel není jiné pomoci, než se znovu zaregistrovat – nemáme žádnou možnost, jak vás identifikovat. Při registraci musíte zvolit nové uživatelské jméno a zadat nový email.

11.4.3 Píše: "There are mistakes in configuration file." (Ukázka 6.2 na straně 12)

Zdá se, že je poškozený konfigurační soubor. Cuckoo vám vypisuje, jak soubor vidí. Z Ukázky 6.2 je vidět, že se mu nepodařilo přečíst uživatelské jméno. To by vám mělo poradit, kde je chyba.

Jsou dvě možnosti řešení. Můžete soubor opravit, jeho správný formát je uveden např. v Ukázce 11.5 na straně 42 nebo v souboru

cuckoo.conf.example. Pokud si nevíte rady, soubor smažte, spusťte Cuckoo a znovu se zaregistrujte. Budete ale muset uvést nové uživatelské jméno i email.

11.5 Přechod mezi projekty

Cuckoo můžete používat pro různé projekty. Jakmile ho máte jednou zkompilované, stačí k němu přidat požadovaný modul. Modul si stáhněte a zkompiluje podle návodu. Poté ukončete Cuckoo, jestli stále běží. Nakonec je třeba smazat soubory *tasks.dat, results* a *cuckoo.conf* a znovu spustit Cuckoo. Zadejte registrační údaje a cestu k novému modulu.

11.6 Odinstalace

Pokud jste nainstalovali Cuckoo do *~/cuckoo*, stačí smazat celý tento adresář.

```
rm -rf ~/cuckoo
```

12 Experimenty

Projekt jsem uvedla do provozu ve školní laboratoři. Provedla jsem dva pokusy, oba s modulem pro ověřování prvočíselnosti.

12.1 První experiment

První pokus jsem provedla s dvěma tisíci čísly v intervalu od 10 000 do 12 000. Klienta jsem nainstalovala na pěti počítačích. Číslo bylo vyhodnoceno jako vyšetřené, pokud se na výsledku shodli tři nezávislí klienti.

Nejdříve jsem nastavila velikost dávky na všech klientech na 50, což se ukázalo jako příliš malé číslo. Klient neustále čekal na server, který nestíhal. Jako vhodná se ukázala velikost dávky alespoň 100. S tímto nastavením jsem experiment úspěšně dokončila.

Test trval pouze několik minut a jeho výsledky byly správné. Na přiloženém CD jsou uloženy kompletní soubory obsahující výpisy databázové tabulky zadání, tabulky řešení a výpis pomocí administrátorské stránky pro oba dva testy. Také je přiložen soubor, obsahující seznam prvočísel ve zkoumaných intervalech v jejich desítkové podobě a 256kové. Tento seznam a seznam mých výsledků se shodují. Seznam těchto prvočísel jsem stáhla ze stránek Johna Moyera [9].

Zde jsem umístila pouze několik ukázek z těchto příloh, protože kompletní výpisy jsou příliš rozsáhlé. V tabulce 12.1 je část tabulky výsledků. Výsledky jsou zde seřazeny v pořadí, v jakém přicházely. Vidíme, že klient s uživatelským jménem 005 právě prozkoumal čísla s ID okolo 220, klient s uživatelským jménem 003 testoval čísla s ID okolo 80 a klient 002 čísla s ID kolem 1990. Zbylí dva klienti tou dobou zřejmě svá čísla právě zkoumali a se serverem nekomunikovali.

Jako prvočísla byla z tohoto výběru vyhodnocena pouze čísla i ID 80 a 224. Doba ověřování přesáhla vteřinu pouze jednou a to právě při ověřování čísla s ID 224.

Z dvojic ID – uživatel můžeme také vypozorovat, v jakém stavu se celý projekt zřejmě nacházel. Klient 002 počítá čísla s vysokým ID, z konce seznamu zadání. A klienti 003 a 005 zkoumali čísla s nižším ID, z počátku seznamu zadání. Z podmínek experimentu (všechny počítače stejné, klient na všech spuštěn v podobnou dobu) lze nahlédnout, že pokud klient 005 počítal *x*-té ověření čísel, pak klienti 002 a 003 počítali *(x+1)*-té ověření, protože za běžných okolností jsou čísla klientům přidělována postupně.

Tabulka 12.1: tabulka výsledků

ID čísla	Výsledek	Uživatelské jméno	IP adresa uživatele	Doba výpočtu
221	0	005	195.113.21.132	0

ID čísla	Výsledek	Uživatelské jméno	IP adresa uživatele	Doba výpočtu
1990	0	002	195.113.21.149	0
79	0	003	195.113.21.150	0
80	1	003	195.113.21.150	0
81	0	003	195.113.21.150	0
222	0	005	195.113.21.132	0
1991	0	002	195.113.21.149	0
223	0	005	195.113.21.132	0
1992	0	002	195.113.21.149	0
224	1	005	195.113.21.132	1
1993	0	002	195.113.21.149	0
82	0	003	195.113.21.150	0
83	0	003	195.113.21.150	0
84	0	003	195.113.21.150	0
225	0	005	195.113.21.132	0
1994	0	002	195.113.21.149	0
226	0	005	195.113.21.132	0
1995	0	002	195.113.21.149	0

Podívejme se také na tabulku zadání, jak vypadá její část po experimentu - Tabulka 12.2. V této vybrané ukázce se nachází pouze jedno prvočíslo a to číslo s ID 400 a 256kovou hodnotou 40 159, což je v desítkové soustavě 10 399. Čísla na začátku tabulky byla zadána právě třikrát, což je minimální počet zadání, aby mohlo být číslo ověřeno. Poslední tři čísla byla zadána pětkrát. Jsou dvě možnosti, jak k tomu mohlo dojít. Buď byla zadána příliš mnoha klientům (výsledky od ostatních tu nevidíme, protože v době, kdy přišly již bylo číslo označeno za definitivně vyřešené) nebo byla zadána znovu klientům, kteří již toto číslo řešili. Druhá možnost je předmětem diskuze v kapitole o dalším možném vylepšování práce.

První možnost ovšem znamená, že jsem server nenastavila příliš dobře – intervaly mezi přidělením jednoho čísla by měly být větší. Z tohoto experimentu je vidět, že správná konfigurace serveru i klienta je důležitá. Pokud zvolíme intervaly mezi jednotlivými zadáními čísla krátké, počítá ho zbytečně mnoho klientů. Jestliže je ovšem zvolíme dlouhé, mohou klienti zbytečně otálet, když by mohli počítat. Stejně důležitá je i velikost dávky. Ukazuje se, že vhodná velikost dávky je zhruba taková, aby její ověření zabralo klientovi alespoň několik desítek minut – aby server nebyl zbytečně nepřetěžován častou komunikací.

Tabulka 12.2: tabulka zadání

ID čísla	Číslo	Výsledek	Čas posledního zadání	Počet zadání
394	40 153	0	2007-05-05 09:12:19	3
395	40 154	0	2007-05-05 09:12:19	3
396	40 155	0	2007-05-05 09:12:19	3
397	40 156	0	2007-05-05 09:12:19	3
398	40 157	0	2007-05-05 09:12:19	3
399	40 158	0	2007-05-05 09:12:19	3
400	40 159	1	2007-05-05 09:12:19	3
401	40 160	0	2007-05-05 09:14:03	5
402	40 161	0	2007-05-05 09:14:03	5
403	40 162	0	2007-05-05 09:14:03	5

Experiment ukázal, že z dvou tisíc zadaných čísel je 209 prvočísel.

12.2 Druhý experiment

Druhý pokus jsem prováděla s tisícem čísel v intervalu od 1 000 000 do 1 001 000. Klienta jsem nainstalovala opět na pěti počítačích a pro vyhodnocení čísla stačily tři nezávislé výsledky.

Přílohy na CD k tomuto pokusu jsou stejného typu jako u předchozího pokusu a i tento experiment proběhl úspěšně – všechna čísla byla ověřena správně a opět za několik málo minut.

V tabulce 12.3 je část tabulky výsledků. Výsledky jsou zde seřazeny v pořadí, v jakém přicházely. Vidíme, že se serverem v tuto chvíli komunikoval klient s uživatelským jménem 001 a chvíli po něm navázal komunikaci i klient 004. Výpočty netrvaly dlouho, vteřinu přesáhl pouze jeden výpočet a to čísla s ID 722.

Tabulka 12.3: tabulka výsledků

ID čísla	Výsledek	Uživatelské jméno	IP adresa uživatele	Doba výpočtu
719	0	001	195.113.21.148	0
720	0	001	195.113.21.148	0
721	0	001	195.113.21.148	0
722	1	001	195.113.21.148	1
723	0	001	195.113.21.148	0
724	1	001	195.113.21.148	0

ID čísla	Výsledek	Uživatelské jméno	IP adresa uživatele	Doba výpočtu
725	0	001	195.113.21.148	0
726	0	001	195.113.21.148	0
727	0	001	195.113.21.148	0
601	0	004	195.113.21.134	0
728	0	001	195.113.21.148	0
602	0	004	195.113.21.134	0
729	0	001	195.113.21.148	0
603	0	004	195.113.21.134	0
730	0	001	195.113.21.148	0
731	0	001	195.113.21.148	0
604	0	004	195.113.21.134	0
732	0	001	195.113.21.148	0

Podívejme se také na tabulku zadání, jak vypadá její část po experimentu - Tabulka 12.4. Z vybraných čísel není ani jedno prvočíslo a všechna byla zadána čtyřikrát, ve stejné dávce.

Tabulka 12.4: tabulka zadání

ID čísla	Číslo	Výsledek	Čas posledního zadání	Počet zadání
959	15 69 254	0	2007-05-05 09:26:31	4
960	15 69 255	0	2007-05-05 09:26:31	4
961	15 70 0	0	2007-05-05 09:26:31	4
962	15 70 1	0	2007-05-05 09:26:31	4
963	15 70 2	0	2007-05-05 09:26:31	4
964	15 70 3	0	2007-05-05 09:26:31	4
965	15 70 4	0	2007-05-05 09:26:31	4
966	15 70 5	0	2007-05-05 09:26:31	4
967	15 70 6	0	2007-05-05 09:26:31	4
968	15 70 7	0	2007-05-05 09:26:31	4

Z tisíce zkoumaných čísel je 75 prvočísel.

13 Závěr

13.1 Shrnutí

Výsledkem mé práce jsou balíčky se zdrojovými kódy. Tyto balíčky lze rozdělit do tří kategorií: projekt pro ověření prvočíselnosti, projekt pro ověření sudosti čísla a obecný nekompletní balík určený pro přidání nového modulu. Každá z těchto tří částí pak obsahuje serverovou a klientskou část. Dokumentaci k projektu tvoří tento text a komentáře ve zdrojových kódech.

Projekt je vhodný hlavně pro nasazení v komunitních sítích. V takovýchto sítích nemívá příliš mnoho uživatelů zájem společný projekt poškodit, naopak uživatelé bývají pro věc nadšeni a jsou ochotni hostit klienta na svém počítači. Přesto projekt skýtá částečnou ochranu proti pirátům – každý výsledek má u sebe informaci, od kterého je uživatele a z jaké IP adresy se uživatel připojil, uživatelé se musí autentizovat hesly. Také lze nastavit počet uživatelů, kteří se musí shodnout na výsledku pro jedno číslo, aby byl výsledek uznán za platný. Tato ochrana ovšem není dokonalá, projekt lze relativně snadno napadnout. Více zmiňuji v následující podkapitole.

Při zvažování vhodné celkové velikosti zadání je nutné uvažovat kapacitu databáze a celkovou výkonnost serveru i předpokládanou výkonnost klientů. Aby nebyl server přetížen nebo aby nedošlo místo na disku. Výkonnost klientů je také dobré zvážit – v případě, že by klient počítal jedno zadání déle, než několik hodin, není pravděpodobné, že ho dopočítá, protože do té doby bude vypnut. Tento problém je zmíněn i v další podkapitole.

Zadáním jednoho výpočtu je jedno přirozené číslo, výsledkem je číslo typu *int*. Ovšem není těžké projekt ještě dále zobecnit, jak je popsáno v následující podkapitole.

13.2 Návrhy na vylepšení

Projekt nyní umí počítat s přirozenými čísly - jako zadání i řešení jsou přirozená čísla (řešení je omezeno na velikost typu *int*). Přitom reálně již teď lze jako zadání dát libovolný řetězec. Pro zcela obecné využití by tedy stačilo upravit projekt tak, aby uměl přijmout jako libovolný řetězec i řešení.

To, že už nyní můžeme jako zadání poslat libovolný řetězec, nám umožňuje do tohoto řetězce zakódovat více čísel. Klient pak může jako úlohu počítat např. které ze zadaných čísel má nějakou vlastnost nejsilnější a podobně.

Dalším bodem je formát uložených hesel. Administrátorovo heslo na webovém serveru je již uložené pouze v podobě svého hashe, ovšem klienti stále posílají svá hesla a mají je uložená v čisté podobě. Bylo by dobré, aby se i v jejich případě zacházelo pouze s hashy.

Z výsledků simulace provozu vyplynulo, že čísla stále ještě nejsou

přidělována nejlepším možným způsobem. Některá byla přidělena i vícekrát, než je počet klientů. Z toho plyne, že čísla byla přidělována i klientům, kteří je již jednou prověřili a tedy jejich další výsledky byly ignorovány. Takovýto přístup vede k plýtvání prostředky.

Rovněž by bylo vhodné zvážit, zda výsledky, co nám přišly „navíc“, tedy výsledek přišel v době, kdy už je číslo prohlášeno za definitivně ověřené, také neshromažďovat.

V provozu se může stát, že nějaký záškodník bude server „zasypávat“ chybnými výsledky. V takovém případě by se mohl hodit seznam ignorovaných IP adres, z nichž by server nikdy neukládal žádné výsledky. Nyní by se tento problém dal řešit pomocí firewallu, což má své nevýhody. Záškodník by se pak nedostal k žádnému z webů na serveru, což ještě není takový problém, jako to, že se útočník dozví, že byl odhalen. Tato informace může způsobit, že se dotyčný pokusí projekt likvidovat jinou cestou. Pokud si bude myslet, že jeho výsledky jsou přijímány a přitom budou zahazovány, dává to administrátorům projektu určitý náskok.

Pokud uživatel pouští klienta pouze na několik hodin a jedno zadání je příliš složité na to, aby ho v té době stihl dopočítat, jeho práce přichází vniveč. V takovém případě by se hodilo, kdyby si klient uměl průběžně ukládat výsledky výpočtů, tedy aby uměl výpočet přerušit a po restartu v něm pokračovat.

Literatura

[1] Great Internet Mersenne Prime Search (GIMPS), http://www.mersenne.org

[2] Distributed.net, http://distributed.net/projects.php

[3] Search for Extraterrestrial Intelligence (SETI), http://www.seti.org/

[4] Berkeley Open Infrastructure for Network Computing (BOINC), http://boinc.berkeley.edu/

[5] Folding@home, http://folding.stanford.edu/

[6] Wikipedia - Karatsubovo násobení, http://en.wikipedia.org/wiki/Karatsuba_algorithm

[7] Dong Wan Han: Generating Strong Prime Numbers Using Probabilistic Tests for Primality, http://ece.gmu.edu/crypto/student_projects/projects99/han/han_slides/sld006.htm

[8] Wikipedia - Rabin-Miller, http://en.wikipedia.org/wiki/Miller-Rabin_primality_test

[9] John Moyer: Some Prime Numbers, http://www.answers.com/topic/miller-rabin-primality-test

Obsah CD

Jméno souboru nebo adresáře	Popis
Dokumentace.pdf	Tento soubor

cuckoo_plain
• base
  • client_comunication.c
  • definitions.h
  • functions.h
  • main.c
  • number_manipulation.c
• cuckoo.conf.example
• Makefile
• Makefile_base
• Makefile_module
	Adresář obsahující klientský balíček Cuckoo bez modulu

cuckoo_oddeven
• base
  • client_comunication.c
  • definitions.h
  • functions.h
  • main.c
  • number_manipulation.c
• oddeven
  • count_out.c
• cuckoo.conf.example
• Makefile
• Makefile_base
• Makefile_oddeven
	Adresář obsahující klietský balíček Cuckoo s modulem pro ověření sudosti čísla

Jméno souboru nebo adresáře	Popis

cuckoo_primes
• base
  • client_comunication.c
  • definitions.h
  • functions.h
  • main.c
  • number_manipulation.c
• module_prime
  • aritm.c
  • count_out.c
  • definitions.h
  • functions.h
• cuckoo.conf.example
• Makefile
• Makefile_base
• Makefile_prime
	Adresář obsahující klietský balíček Cuckoo s modulem pro ověření prvočíselnosti čísla

cuckoo_server_plain
• down
  • cuckoo_oddeven.tar.gz
  • cuckoo_plain.tar.gz
  • cuckoo_primes.tar.gz
• admin.php
• config.php
• cuckoo.gif
• downloads.php
• functions_module.php
• functions.php
• index.php
• style.css
	Adresář obsahující serverový balíček Cuckoo bez modulu

cuckoo_server_oddeven
• down
  • cuckoo_oddeven.tar.gz
  • cuckoo_plain.tar.gz
  • cuckoo_primes.tar.gz
• admin_oddeven.php
• admin.php
• config.php
• cuckoo.gif
• downloads.php
• functions_oddeven.php
• functions.php
• index.php
• style.css
	Adresář obsahující serverový balíček Cuckoo pro ověření sudosti čísla

Jméno souboru nebo adresáře	Popis

cuckoo_server_primes
• down
  • cuckoo_oddeven.tar.gz
  • cuckoo_plain.tar.gz
  • cuckoo_primes.tar.gz
• admin.php
• admin_primes.php
• config.php
• cuckoo.gif
• downloads.php
• functions_primes.php
• functions.php
• index.php
• style.css
	Adresář obsahující serverový balíček Cuckoo pro ověření prvočíselnosti

test1
• admin_primes_files
  • cuckoo.gif
  • style.css
• admin_primes.html
• tabulka_vysledku.pdf
• vysledky_spravne.pdf
• tabulka_zadani.pdf
	Adresář obsahující výsledky prvního testu

test2
• admin_primes_files
  • cuckoo.gif
  • style.css
• admin_primes.html
• tabulka_vysledku.pdf
• vysledky_spravne.pdf
• tabulka_zadani.pdf
	Adresář obsahující výsledky druhého testu

binary • cuckoo • primes.so • oddeven.so	Zkompilovaný program cuckoo a dva moduly (sdílené knihovny).

Použité programy

vim	-	http://www.vim.org/
gcc	-	http://gcc.gnu.org/
gdb	-	http://sourceware.org/gdb/
OpenOffice	-	http://www.openoffice.org/
KDevelop	-	http://www.kdevelop.org/
Firefox	-	http://www.firefox2.com/en/
XChat	-	http://www.xchat.org/
Gnome	-	http://www.gnome.org/
WindowMaker	-	http://www.windowmaker.info/

Příloha A – Původní podoba Makefile_module

```
# module name and directory where sources are placed
LIB_NAME = module_name.so
MOD_NAME = module_name

# definitions files
INCL_LIB = $(MOD_NAME)/definitions.h
$(MOD_NAME)/functions.h

# source files
LIB_FILES_C = ./$(MOD_NAME)/count_out.c
LIB_FILES_O = ./$(MOD_NAME)/count_out.o

# Compiler, Linker Defines
CC      = /usr/bin/gcc
CFLAGS  = -std=gnu99 -Wall
RM      = /bin/rm -f

all: lib

lib: $(LIB_FILES_O) $(INCL_LIB)
     gcc -shared -Wl,-soname,$(LIB_NAME).1 -o $(LIB_NAME)
$(LIB_FILES_O) -lc

($LIB_FILES_O): $(LIB_FILES_C)
     gcc -fPIC -c $(LIB_FILES_C)

clean:
     RM -f *.o $(EXE) *.so ./$(MOD_NAME)/*.o
```

Příloha B – Původní podoba Makefile

```
MAKE = /usr/bin/make
BASE = Makefile_base
MODL = Makefile_module

MODL_NAME = module_name.so
EXE = cuckoo

all:
     $(MAKE) -f $(BASE) all; $(MAKE) -f $(MODL) all

clean:
     $(MAKE) -f $(BASE) clean; $(MAKE) -f $(MODL) clean

install:
     cp $(MODL_NAME) $(EXE) ~/cuckoo
```

Příloha C – Nový Makefile pro modul o prvočíselnosti

```
# module name and directory where sources are placed
LIB_NAME = primes.so
MOD_NAME = module_prime

# definicni soubory
INCL_LIB   = ./$(MOD_NAME)/functions.h \
./$(MOD_NAME)/definitions.h

# soubory, ze kterych budou knihovny
LIB_FILES_C = ./$(MOD_NAME)/aritm.c \
./$(MOD_NAME)/count_out.c
LIB_FILES_O = ./$(MOD_NAME)/aritm.o \
./$(MOD_NAME)/count_out.o

# Compiler, Linker Defines
CC      = /usr/bin/gcc
CFLAGS  = -std=gnu99 -Wall
RM      = /bin/rm -f

all: lib

lib: $(LIB_FILES_O) $(INCL_LIB)
     gcc -shared -Wl,-soname,libprime.so.1 -o \
     $(LIB_NAME) $(LIB_FILES_O) -lc

($LIB_FILES_O): $(LIB_FILES_C)
     gcc -fPIC -c $(LIB_FILES_C)

clean:
     RM -f *.o $(EXE) *.so ./$(MOD_NAME)/*.o
```