PB161 Programování v jazyce C++
Přednáška 12
Vývoj software

Jiří Weiser

13. prosince 2016

Úvod

Kdo jsem

- vývojář v Seznamu
- bývalý cvičící C/C++ předmětů na FI
- aktuálně pouhý opravující

Úvod

Co vám dneska povím?

- jak funguje vývoj ve větším rozměru
- jaká úskalí na vás mohou čekat při vstupu do firmy
- pohovorové otázky
- náčrt achitektury vyhledávače
- případně i něco jiného...

Anketa

Kdo z pracuje při škole?

Anketa

Kdo z pracuje při škole?

- dobrá věc (doporučuji)
- když už ne ve firmě, tak aspoň v laboratoři na FI (doporučuji ještě víc)
- *nezanedbávejte školu – k titulu se přihlíží*

Programování v malém

- snaha o co nejrychlejší vyhotovení
- žádná (nebo malá) potřeba verzování
  - do první ztráty zdrojových textů
- žádný požadavek na udržitelnost
  - nikdo jiný nebude kód udržovat/rozvíjet
- není třeba plánování
- testy jenom nejnutnější
- typicky domácí úkoly
  - v podstatě se jedná o skripty

Programování ve větším

- snaha o co nejrychlejší vyhotovení
  - jak u současného projektu, tak u **budoucích**
  - úpravy kódu s ohledem na předpokládané změny v budoucnu
- silná potřeba verzování
  - striktní zachování historie změn
  - nutnost verzování binárek (balíčky)
- testování
  - unit, funkční, integrační, zátěžové, bezpečnostní, přívětivosti
- metodika vývoje
  - SCRUM (a další agilní metodiky)
  - vodopád
- velké změny se dějí pomalu

Programování ve větším – stabilita kódu

- budete pracovat převážně s cizím kódem
- budete psát kód, který budou číst další lidé
- mohou to být cizinci
- mohou mít různou úroveň znalostí používaných technologií

Coding style guides

- omezení nástrojů jazyka
- vhodné pojmenování
- komentáře
- **čitelnost má přednost před výkonem**

Programování ve větším – verzování

- využívání robustnějších verzovacích nástrojů
- více komponent systému
  - knihovny, programy
  - zajištění propojení správných verzí
- více programátorů modifikuje stejné části
  - repozitáře, větve
- vytváření patchů/commitů
  - menší kousky
  - popisující komentáře změn

Programování ve větším – robustnost a výkon

Malý web

- jeden server pro CGI
- jeden server pro databázi

Větší web

- vyšší návštěvnost
- více dat
  - různé způsoby využití
- zvládnou to dva servery?

Programování ve větším – robustnost a výkon

Malý web

- jeden server pro CGI
- jeden server pro databázi

Větší web

- vyšší návštěvnost
- více dat
  - různé způsoby využití
- zvládnou to dva servery?

NE

Programování ve větším – robustnost a výkon

- duplikace jedné aplikace na více serverů
- duplikace databází
  - master-slave replikace
- serverům předřadit load balancer
- logování prováděných akcí v komponentách
  - *(jak vůbec debugovat?)*
- mít mechanizmy na restartování pokažených komponent
  - monitoring

Bude to stačit?

Programování ve větším – robustnost a výkon

- duplikace jedné aplikace na více serverů
- duplikace databází
  - master-slave replikace
- serverům předřadit load balancer
- logování prováděných akcí v komponentách
  - *(jak vůbec debugovat?)*
- mít mechanizmy na restartování pokažených komponent
  - monitoring

Bude to stačit?

Databáze nebude stačit vydávat obsah

Programování ve větším – robustnost a výkon

- některá data je potřeba předpočítávat
- je třeba držet ta samá data v různých datových úložištích
  - relační databáze
  - vyhledávací engine
  - Cassandra
  - HBase
  - a další...
- nutnost udržovat konzistentní data
  - migrační/exportní skripty
  - samostatné servery

Komunikace

- s databázemi
  - použije se poskytnuté API
- vlastní komponenty
  - mezi servery
  - v rámci jednoho serveru

Nápady, jak realizovat komunikaci?

(v prostředí primárně C++)

Komunikace mezi servery

- Json
  - velká podpora napříč jazyky
  - lidsky čitelné
  - volná struktura
  - serializace/deserializace

Jaké to má výhody? Má to nevýhody?

Komunikace mezi servery

- Json
  - velká podpora napříč jazyky
  - lidsky čitelné
  - volná struktura
  - serializace/deserializace

Jaké to má výhody? Má to nevýhody?

- volná struktura jde proti udržení rozhraní
- drahá serializace/deserializace
- "ukecaný formát"
  - nekomprimovaná data

Komunikace mezi servery

- pevný předpis dat
- generované rozhraní pro rozličné jazyky
- binární komprimovaný formát

- Protobuffers
  - technologie prověřená léty
  - vyvíjí a používá Google

- Cap'n Proto
  - novější
  - serializace/deserializace "zadarmo"

- volba závisí na použití
  - velké projekty vyžadují stabilitu
  - menší mohou být dravější

Komunikace v rámci serveru

- možnost použít json/proto/...
  - datově bezpečné
  - zbytečně pomalé
- volání přímo metod, používání tříd
  - riziko porušení binární kompatibility

Jaké změny mohou rozbít binární kompatibilitu?

Komunikace v rámci serveru

Nebezpečné změny

- přidání nebo odebrání virtuální metody
- změna atributů
  - pořadí, typ
- použití různých překladačů
  - různých verzí standardní knihovny
- změna hlavičky funkce
  - přidání parametru s defaultní hodnotou také

Komunikace v rámci serveru

Jak zabránit porušení binární kompatibility?

Komunikace v rámci serveru

Jak zabránit porušení binární kompatibility?

- vydat knihovnu s jiným názvem
  - používané v Seznamu
- používání `inline namespace` pro aktuální verzi
  - použito v rámci libc++
- používat čisté C struktury a velikostí určovat verzi
  - použito ve Win32 API
- překompilovat všechno, co závisí na změněné knihovně
  - potřeba continuous integration
  - potenciálně velmi drahé

Hurá do práce

Co vás čeká, když budete chtít pracovat

Pohovor – na co se připravit

- budou po vás chtít napsat kus kódu
  - analogie se zápočtovým testem
  - bude to pravděpodobně na papír/tabuli
  - budete muset komentovat vaše řešení
    - výhody, nevýhody, korektnost
    - jak byste to otestovali

Pohovor – na co se připravit

- budou po vás chtít napsat kus kódu
  - analogie se zápočtovým testem
  - bude to pravděpodobně na papír/tabuli
  - budete muset komentovat vaše řešení
    - výhody, nevýhody, korektnost
    - jak byste to otestovali
- budou vás zkoušet ze všeobecných znalostí
  - směs zkoušky C++/Java/Python, operačních systémů, sítí, databází, paralelizmu, SWE
  - důraz na praktické věci
    - doporučuji vyzkoušet si práci

Pohovor – na co se připravit

- budou po vás chtít napsat kus kódu
  - analogie se zápočtovým testem
  - bude to pravděpodobně na papír/tabuli
  - budete muset komentovat vaše řešení
    - výhody, nevýhody, korektnost
    - jak byste to otestovali
- budou vás zkoušet ze všeobecných znalostí
  - směs zkoušky C++/Java/Python, operačních systémů, sítí, databází, paralelizmu, SWE
  - důraz na praktické věci
    - doporučuji vyzkoušet si práci
- budou vás zkoušet z algoritmů
  - rozsahem předmětu Návrh algoritmů II

Pohovor – jak se připravit

- čerstvě po škole máte obstojné znalosti teorie
- obvykle chybí praxe
- spojte učení na zkoušky alespoň trochu s praxí
  - naprogramujte si něco síťového
  - pohrajte si s databázemi
  - vyzkoušejte si paralelní vykonávání
  - použijte přímo rozhraní OS ve svém programu
- není třeba být odborníkem ve všech jazycích
  - je ale dobré mít jeden jazyk jako "mateřský"
- obvykle se není třeba bát dotazů na technologie
  - firmy si zaměstnance zaškolí samy

Příklady pohovorových otázek

- Napište funkci, která z binárního stromu vytvoří dvojtě spojový seznam.
- Napište funkci, která vytvoří všechny permutace ze zadaného seznamu seznamů.
- Napište funkci, která ve dvou seřazených seznamech nalezne N-tý prvek.
- Napište funkci, která rozhodne, zda se v grafu nachází *sink vertex*.
  - Vrchol, do kterého vedou hrany ze všech ostatních vrcholů, a ze kterého nevede žádná hrana

U každého příkladu si určete časovou i paměťovou složitost a (rámcově) si dokažte korektnost.

Cca 50 % uchazečů práce v Seznamu končí na pohovoru na neznalosti Pythagorovy věty.

Příchod do firmy

- zpočátku vás uvedou do organizace práce v týmu
  - objasnění metodiky
  - vysvětlení základních procesů
    - jak publikovat změny
    - jak dělat code-review
    - kdy a kam chodit na porady
- postupné seznamování kódem
  - začíná se od jednoduchých komponent
  - postupně se vám rozkrývá celý produkt
    - *(tato fáze může trvat roky)*
- podle šikovnosti dostane programátor důvěru pracovat bez přímého dozoru

Příchod do firmy – co by vás nemělo překvapit

- škaredý kód – dlouhé funkce, gigantické třídy
  - původně byly malé a hezké
  - postupným přidáváním funkcionality začaly bobtnat
- požadavky z vedení mají přednost před mazlením kódu
  - důležité jsou prachy
  - manažera nezajímá krása kódu, ale funkčnost
- chodit pravidelně do práce může způsobovat depresi
  - dá se na to zvyknout
- dovolené je málo
- budete muset mluvit s lidmi
  - kolegové v týmu
  - prezentace
  - různé meetingy

Jak může vypadat vyhledávač

Potřeba

- relevantních výsledků
- stabilních výsledků
- velkého výkonu

Jak může vypadat vyhledávač

Potřeba

- relevantních výsledků
- stabilních výsledků
- velkého výkonu

Důležité části vyhledávače

- robot
- indexer
- finder
- agregátor
- query procesor

Každá část se skládá z velké části dalších knihoven, mezi částmi je spousta drobných prográmků, které se starají o komunikaci, vyrovnávání zátěže a mezipaměti.

Jak může vypadat vyhledávač

Robot

- cyklicky prochází web a stahuje informace
  - obecně zdrojů
- hodnotí kvalitu stránek
- slouží k plnění databází
  - zpracovává různé části stránek
- lze paralelizovat
  - obrovské farmy
  - úzkým prvkem je síťové připojení a výstupní fronta
    - zápisy do databází lze provádět až se zpožděním

Jak může vypadat vyhledávač

Indexer

- vyčítá data z databází
- tvoří dokumenty
  - základní jednotka při hledání
  - může kompletovat z různých záznamů v databázi
- vytáhne z dokumentů slova, která zaindexuje
- dokumenty uloží do speciálního formátu
  - přístup pouze pro čtení
  - optimalizován na hledání slov v dokumentech
  - *hledací soubory*
- lze paralelizovat
  - ne tak masivně jako robot
  - úzkým prvkem jsou databáze – čtení

Jak může vypadat vyhledávač

Query procesor

- převádí dotaz do vhodné formy
- hodnotí závažnost slov
- značkuje slova
  - z definičních souborů
  - přiřazuje slovům význam
    - Brno – město
  - používají se učící algoritmy
- opravuje překlepy
- skloňuje, časuje
- doplňuje synonyma
- výsledkem je množina slov ohodnocená dle důležitosti

Jak může vypadat vyhledávač

Finder

- na základě množiny slov hledá v *hledacích souborech* vhodné dokumenty
- hodnotí dokumenty
  - nastavuje jim relevanci
  - může se lišit dle různých kritérií
- lze paralelizovat
  - vhodné rozdělení *hledacích souborů*

Agregátor

- řadí se do stromů
  - uzly stromu jsou další *agregátory*
  - listy stromu jsou *findery*
- jako vstup dostane výsledek z *query procesoru*
- stará se o zavolání *finderů* a sesbírání výsledku
- vrácené dokumenty řadí podle relevance

Závěrem

Děkuji za pozornost