KIV/PGS

První semestrální práce

Dokumentace

Překlad

javac –d bin src/.java*

jar cfm main.jar Manifest.txt –C bin .

Spuštění

java –jar main.jar <nazev_vsupniho_souboru.txt>

např.

java –jar main.jar lesmiserables–input.txt

Popis implementace

Implementace obsahuje 5 hlavních tříd, jeden vstupní bod v **Main** třídě a jedna pomocná třída **Tool,** která obsahuje pomocné funkcionality pro čtení a zápis souborů. Všechny výkonné třídy (**Boss, Underboss, Master, Foreman a Slave**) implementují rozhraní **Callable**, které na rozdíl od rozhraní **Runnable** umožňuje vracet návratovou hodnotu. Tou je Mapa<slovo, četnost>. Tím je zajištěno předání informací nadřazeným vrstvám, které tyto vlákna vytvářejí.

V implementaci je použito mnoho regulérních výrazů. Používají se pro vyhledání klíčových slov (kapitol, knih, svazků…) a pro práci s jedním slovem textu, slova spojená pomlčkou jsou brána za různá slova, nikoliv jedno.

Konfigurace

Ve třídě Main se nachází parametry, kterými lze ovlivňovat chod programu. Jsou to počty instancí jednotlivých tříd. Nejsou zadávány absolutní hodnotou vláken, nýbrž odpovídají na otázku, kolik instancí dané třídy bude mít k dispozici nadřazená třída. Tudíž UNDERBOSS_NUMBER= 2, znamená, že každý z N Bossů bude mít k dispozici 2 Underbosse.

Kritické sekce

Jedná se o místo v programu, do kterého má přístup více vláken a které v něm vykonávají nějaké změny. Pokud by tyto změny vykonávaly současně, mohlo by dojít

k neočekávanému chování. Proto je potřeba tyto sekce ošetřit tak, aby do nich mohlo vždy jen jedno vlákno současně.

Zápis do souborů

V programu se jedná především o místa zápisů do souborů, ať už se jedná o /state.txt či /book.txt a podobné. Toto je ošetřeno příslušným zámkem nad každým ze souborů pomocí následující metody:

```
public static void appendStateMessage(String filePath, String message) {
  Object fileLock = getLockForFile(filePath);
  synchronized (fileLock) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath,
true))) {
      writer.append(message);
    } catch (IOException e) {
      System.err.println("Chyba při zápisu do " + filePath);
      e.printStackTrace();
    }
  }
}
```

Všechny zápisy do daného souboru tak probíhají v rámci synchronizovaného bloku, který zaručuje, že v jednom okamžiku může do souboru zapisovat pouze jedno vlákno.

Zápis do souboru /lesmiserables.txt není kritickou sekcí, jelikož tento zápis vykoná vždy Boss, který dokončí svou činnost jako poslední.

Jednotlivé instance podtříd (Underboss, ..., Slave), nepracují se vstupním souborem, nýbrž je jim předán text ve formě String atributu.

Sdílené datové struktury

Sloučení výsledků z různých vláken do jedné centrální mapy může být zdrojem nepředpokládaného chování, pokud není přístup správně synchronizován.

Pro sdílení výsledků je použita třída **ConcurrentHashMap**, která zajišťuje bezpečné operace při slučování a aktualizaci výsledků. Díky tomu není třeba používat explicitní synchronizační bloky při slučování výsledků, což zjednodušuje implementaci.

Přístup ke sdíleným frontám a rozdělování práce

V implementaci třídy Boss je využita sdílená fronta pro svazky. To by mohl být problém, kdybych do fronty přistoupilo víc vláken současně. Proto je použita **BlockingQueue**, která je již implementována jako bezpečná struktura.

Běh programu

Idea: Čím větší počet vláken vytvoříme, tím rychlejší program bude, protože se bude provádět víc operací současně. Existuje však hraniční bod, kdy těchto vláken bude již příliš mnoho a většina z nich nebude mít reálné využití. Bude docházet k častým blokacím na zámcích a běh programu se bude zpomalovat.

Trvání [s]	Boss [n]	Underboss [n]	Master [n]	Foreman [n]	Slave [n]
1.045	1	1	1	1	1
0.714	2	4	8	16	32
0.921	5	5	5	5	50
0.743	2	4	8	16	160
0.760	5	10	10	20	100
1.159	2000	2000	2000	2000	2000

Z dat můžeme vyčíst, že vstupní idea se v zásadě potvrdila.

Měření bylo uskutečněno s vypnutým vypisováním diagnostických informací na obrazovku, které by běh programu výrazně zpomalilo.

Výstup

Výstupem programu je požadovaná adresářová struktura a jednotlivé soubory popisující běh programu. Pokud taková adresářová struktura ještě neexistuje, je vytvořena. Pokud existuje, je smazána a vytvořena znovu.