**Word Counter**

**Funkcionalita**

* Načíst soubor podle parametru programu.
* Zjistit počet opakování daného slova v souboru.
* Paralelizovat běh pro více souborů.
* Detekovat přepínače v příkazové řádce.
* V případě nesouladu na vstupu upozornit uživatele a ukončit program.
* Uložit všechna slova do jednoho souboru

**Návrh**

**Přepínače**

V případě špatného vstupu, program vyhodí výjimku a vyšle uživateli pokyn pro zadání přepínače - - help, který uživateli vypíše obecný příklad vstupu.

**[\*Paralelizace\*** **\*rozpoznání velikosti písmen\*** **\*Smazání interpunkce\*** **\*soubor\* \*další soubor\* … \*n-ty soubor\***]

**Paralelizace**

-st : „Single Thread“

-mt : „Multi Thread“

**Rozpoznání velikosti písmen**

-c: „capital“

**Smazání interpunkce**

-i: „interpunction“

**Spuštění**

Zkompilovaný soubor je třeba umístit do podsložky /compilated. Pro zadání zdrojových souborů je třeba soubor umístit do složky /resources následně pak vložit **jméno souboru** jako vstupní argument programu – **nenastavovat relativní cestu k souboru od zkompilovaného programu.**

**Výstup pak bude ve složce /resources/counted.**

**Implementace**

Program je koncipován do 3 částí – main, parallel, sequence. Main slouží k úvodnímu zpracování vstupu od uživatele a dle tohoto vstupu od uživatele deleguje požadavky do částí parallel a sequence.

**Parallel** – program běžící při požadavku na vice vláken.

**Sequence** – program běžící při požadavku na proces bez vláken navíc.

Třídy parallel a sequence dědí od společného předka counter. Přišlo mi vhodné dát těmto třídám společné rozhraní a zbytečně neporušovat „DRY“.

Třída counter poskytuje třídám parallel a sequence pro reprezentaci dat knihovní strukturu std::unordered\_map. Z počátku implementace mě napadla možnost reprezentovat každé slovo instancí nějaké mé vlastní struktury. Usoudil jsem ale, že by vytváření těchto instancí bylo dost nákladné, nakonec bych je stejně musel uchovávat v jiné struktuře (např. std::vector). Samozřejmě jsem tímto řešením eliminoval prostor pro chybu při tvorbě vlastní struktury.

Finální implementace je mimo jiné rozdílná od prvotní také v užívání přepínačů. Uživatel nyní nemusí zadávat na vstup všechny druhy přepínačů. Minimální požadavek na vstup je tedy alespoň jeden textový soubor. Defaultně se program volá jako varianta se sekvenčním přístupem.

**Problémy**

Samotnou implementaci nedoprovázely výrazné problémy. Avšak při testování programu na různá vstupní data jsem zjistil, že rozdíl mezi během single a multi thread je opravdu malý. Nakonec se mi stalo, že single thread varianta „předběhla“ multi thread variantu. S tím samozřejmě nastala panika a pokusy o otestování. Dle statistik uvedených dále v dokumentu jsem usoudil, že problém nastane, když je v souboru vysokých počet různých slov (možná se pletu a budu rád za vysvětlení příčiny) – problém je dále zanalyzován v dokumentu.

Na doporučení jsem užil knihovní strukturu std::unordered\_map, která naprosto markantně ovlivnila sekvenční přístup. Problém však nastal, při letmé kontrole výstupních souborů – všiml jsem si, že při užívání std::unordered\_map **jsou některá slova ignorována – na každý běh aplikace se totiž výsledný počet slov liší.** Usoudil jsem tak dle porovnání s užitou strukturou std::map, která byla v původní implementaci. Zároveň mi také std::map poskytovala automaticky seřazený výpis slov – což přispívá k větší náročnosti programu. Ve finální práci je užita unordered\_map (problém analyzován dále v dokumentu).

**Testovací data**

Test1.txt, Test2.txt, Test3.txt – Každý soubor vychází z cca 350 slov, která jsou několikrát nakopírovaná za sebe.

Lotr1.txt, Lotr2.txt, Lotr3.txt, Hobit.txt – Testovací data obsahující vysoký počet různorodých slov.

**Statistiky**

Testováno na: Intel Core i5-2500k 3,30GHz, 8GB DDR3 RAM, HDD - WD1002FAEX

Pro co nejlepší zanalyzování situace jsem přidal měření času pro zpracování jednotlivých souborů, ukládání dat a samozřejmě celého programu. Veškeré časy jsou uvedeny v milisekundách [ms]. Zároveň pro porovnání uvádím data, která jsem naměřil s prvnotní implementací.

1. Sekvenční zpracování souborů test1.txt test2.txt test3.txt.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test# | **Sequence** | test1.txt | test2.txt | test3.txt | SaveFile |  | **OLD** |
| **1** | 170 | 56 | 45 | 58 | 1 |  | 648 |
| **2** | 182 | 72 | 44 | 57 | 1 |  | 845 |
| **3** | 181 | 65 | 44 | 58 | 1 |  | 862 |
| **4** | 181 | 64 | 50 | 58 | 1 |  | 738 |
| **5** | 167 | 55 | 44 | 59 | 0 |  | 713 |
| **6** | 166 | 56 | 45 | 57 | 0 |  | 568 |
| **7** | 190 | 56 | 49 | 76 | 0 |  | 677 |
| **8** | 167 | 58 | 44 | 56 | 0 |  | 731 |
| **9** | 178 | 65 | 45 | 56 | 0 |  | 597 |
| **10** | 184 | 68 | 46 | 61 | 0 |  | 743 |
| **Avg.** | **176,6** | 61,5 | 45,6 | 59,6 | 0,4 |  | **712,2** |

1. Paralelní zpracování souborů test1.txt test2.txt test3.txt. **Nyní je paralelní zpracování rychlejší než sekvenční.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test# | **Parallel** | test1.txt | test2.txt | test3.txt | SaveFile |  | **OLD** |
| **1** | 105 | 73 | 71 | 74 | 0 |  | 219 |
| **2** | 92 | 66 | 80 | 83 | 1 |  | 212 |
| **3** | 97 | 66 | 81 | 81 | 1 |  | 195 |
| **4** | 100 | 80 | 85 | 90 | 1 |  | 297 |
| **5** | 105 | 73 | 84 | 92 | 1 |  | 198 |
| **6** | 121 | 106 | 108 | 112 | 1 |  | 225 |
| **7** | 104 | 76 | 84 | 95 | 1 |  | 196 |
| **8** | 133 | 71 | 88 | 68 | 1 |  | 217 |
| **9** | 110 | 82 | 93 | 99 | 1 |  | 210 |
| **10** | 110 | 88 | 92 | 101 | 1 |  | 226 |
| **Avg.** | **107,7** | 78,1 | 86,6 | 89,5 | 0,9 |  | **219,5** |

1. Sekvenční zpracování souborů lotr1.txt lotr2.txt lotr3.txt. Markantní zlepšení programu po použití std::unordered\_map je ze statistik evidentní.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test# | **Sequence** | lotr1.txt | lotr2.txt | lotr3.txt | SaveFile |  | ***OLD*** |
| **1** | 172 | 59 | 44 | 38 | 23 |  | 648 |
| **2** | 195 | 63 | 52 | 44 | 26 |  | 845 |
| **3** | 193 | 79 | 44 | 39 | 22 |  | 862 |
| **4** | 204 | 92 | 41 | 38 | 23 |  | 738 |
| **5** | 172 | 57 | 44 | 40 | 23 |  | 713 |
| **6** | 192 | 75 | 46 | 40 | 23 |  | 568 |
| **7** | 195 | 86 | 37 | 42 | 22 |  | 677 |
| **8** | 191 | 63 | 46 | 48 | 28 |  | 731 |
| **9** | 177 | 64 | 42 | 40 | 22 |  | 597 |
| **10** | 176 | 55 | 43 | 38 | 22 |  | 743 |
| **Avg.** | **186,7** | 69,3 | 43,9 | 40,7 | 23,4 |  | **712,2** |

1. Paralelní zpracování souborů lotr1.txt lotr2.txt lotr3.txt. Časové zlepšení při paralelním zpracování nastalo, ovšem proces **je pomalejší než při sekvenčním přístupu.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test# | **Parallel** | lotr1.txt | lotr2.txt | lotr3.txt | SaveFile |  | **OLD** |
| **1** | 567 | 497 | 52 | 527 | 29 |  | 958 |
| **2** | 575 | 509 | 525 | 535 | 33 |  | 836 |
| **3** | 592 | 523 | 535 | 543 | 36 |  | 866 |
| **4** | 537 | 468 | 473 | 483 | 41 |  | 812 |
| **5** | 633 | 560 | 581 | 597 | 28 |  | 909 |
| **6** | 587 | 504 | 523 | 530 | 45 |  | 901 |
| **7** | 588 | 456 | 464 | 474 | 102 |  | 830 |
| **8** | 524 | 472 | 483 | 491 | 26 |  | 822 |
| **9** | 770 | 702 | 713 | 726 | 28 |  | 747 |
| **10** | 775 | 685 | 724 | 727 | 31 |  | 816 |
| **Avg.** | **614,8** | 537,6 | 507,3 | 563,3 | 39,9 |  | **849,7** |

1. Paralelní zpracování souborů lotr1.txt lotr2.txt lotr3.txt s přepínači -i -c. **Oproti běhu bez přepínačů došlo k cca dvojnásobnému zrychlení.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **-i -c switch** | | | | | | | |
| Test# | **Parallel** | lotr1.txt | lotr2.txt | lotr3.txt | SaveFile |  | **OLD** |
| **1** | 333 | 234 | 237 | 246 | 31 |  | 367 |
| **2** | 181 | 80 | 75 | 80 | 15 |  | 533 |
| **3** | 316 | 228 | 231 | 253 | 16 |  | 347 |
| **4** | 354 | 230 | 232 | 241 | 89 |  | 344 |
| **5** | 315 | 241 | 254 | 248 | 19 |  | 542 |
| **6** | 288 | 229 | 233 | 241 | 38 |  | 336 |
| **7** | 316 | 241 | 259 | 273 | 19 |  | 372 |
| **8** | 345 | 255 | 260 | 268 | 41 |  | 356 |
| **9** | 325 | 261 | 263 | 286 | 20 |  | 315 |
| **10** | 315 | 272 | 273 | 286 | 19 |  | 374 |
| **Avg.** | **308,8** | 227,1 | 231,7 | 242,2 | 30,7 |  | **388,6** |

1. Pro porovnání jsem veškeré zdrojové soubory **přesunul na rychlejší SSD disk (Samsung 850 EVO)**. Paralelní zpracování je o něco rychlejší. Sekvenční se zdá být o něco pomalejší, tyto hodnoty nejsou však převratné.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| lotr1.txt lotr2.txt lotr3.txt | | |  | lotr1.txt lotr2.txt lotr3.txt | | |
| **Parallel** | | |  | **Sequence** | | |
|  | without switch | -i -c |  |  | without switch | -i -c |
| 1 | 483 | 289 |  | 1 | 171 | 188 |
| 2 | 480 | 157 |  | 2 | 225 | 249 |
| 3 | 528 | 338 |  | 3 | 212 | 244 |
| 4 | 531 | 172 |  | 4 | 198 | 297 |
| 5 | 646 | 282 |  | 5 | 207 | 202 |
| 6 | 585 | 157 |  | 6 | 209 | 229 |
| 7 | 534 | 345 |  | 7 | 224 | 168 |
| 8 | 649 | 163 |  | 8 | 189 | 166 |
| 9 | 632 | 337 |  | 9 | 201 | 242 |
| 10 | 497 | 152 |  | 10 | 193 | 167 |
| **Avg.** | **556,5** | 239,2 |  | **Avg.** | **202,9** | 215,2 |

1. Při použití struktury std::unordered\_map jsem zaznamenal, že na každé spuštění se stejnými vstupními daty dochází ke změně počtu slov ve výsledném souboru (cca 200-500). Po zpětné změně na std::map tento problém nenastává.

**Závěr**

Problém, který nastal se mi bohužel nepodařilo vyřešit. Paralelní aplikace běží na vstupních datech, která obsahují obrovské množství různorodých slov, a běží pomaleji než aplikace sekvenční.

Jelikož jsem do metod readfile a saveWords ve třídě counter umístil měření času, mohu vyloučit možnost pomalého zápisu na disk – viz. statistiky. Problém pravděpodobně nastává při čtení dat a vkládání do datové struktury std::unordered\_map – tedy v metodě readFile. Nerad bych se pouštěl do vymýšlení přehnaných teorií, kde by mohl být problém… Možná problém paralelního přístupu na disk? Načítání velkého množství dat do mezipaměti? – to vše paralelně pak zpomaluje celý chod…? Skutečně nevím a nedokážu usoudit.

Zajímavé bylo však zjištění, jak markantně ovlivnila změna datové struktury sekvenční chod aplikace. Dle všeho však za cenu neseřazeného výstupu a již zmíněné chyby, kdy struktura unordered\_map některá slova nepojme.