Osztott rendszerek specifikációja és implementációja
2. beadandó - Dokumentáció

Név: Soós Bálint
Neptun kód: HDX9MU
Elérhetőség: soosbalint95@gmail.com

Gyakorlatvezető: Horpácsi Dániel
Csoport: 5.
2016. november 26.

Háttértörténet

A félév közepén járunk, így a felvett diákoknak is számot kell adniuk az eddigi teljesítményükről. Az évfolyam zárthelyin azonban szigorú szabályok vannak, így az áttekinthetőség miatt a tanulmányi rendszerben elfoglalt helyük alapján fogják ültetni a diákokat az írásbeli során, ennek köszönhetően mindenki hamarabb megtalálja a saját helyét. A ZH-t író hallgatók NEPTUN-kódjait egy szöveges fájlban kapták meg a vizsgáztatók, ám ebben még jelentkezési sorrendben voltak megtalálhatóak az adatok, rendezetlenül. Annak érdekében, hogy gyorsan meghatározható legyen, hogy kinek hol a helye, egy, az épületben tartózkodó proginfes hallgató segítségét kérték a tanárok. A teremben lévő számítógépen, melyen a programot meg kellett írni, a rendezési algoritmusokat tartalmazó lib. sérült volt. Mivel már nem maradt idő másik gépet keresni, a vizsgáztatókon úrrá lett a pánik, ám az ELTEs hallgató hidegvérrel válaszolt:

"Ne aggódjanak! Egy pillanat alatt megoldjuk a problémát, hiszen tanultam 'Algoritmusok és Adatszerkezetek'-ből a MergeSort-ot!" - így a vizsga sikeresen, a szabályoknak megfelelően tudott lezajlani.

A programozó hallgatónak tehát az alábbi problémát kellett megoldania:

A bemenet input.txt első sorában egy N pozitív egész olvasható, ennyi diáknak kell biztosítani termet, míg a következő N sorban a hallgatók NEPTUN-kódjai, azaz N string követi egymást, ez mutatja a jelentkezési sorrendet.

Egy lehetséges bemeneti fájl:

```
7
OSAVH1
T0NDJB
4S1UPL
AXKAW4
22TQP7
NM8VPS
PJVNEU
```

Feladat

A megoldás során TILOS a beépített rendezéseket használni (pl. std::sort()), a feladat egy párhuzamosított MergeSort (összefésüléses rendezés) implementálása! A fő folyamat feladata az adatok beolvasása az input.txt fájlból, majd az output.txt kimeneti fájl létrehozása, benne a rendezett adathalmazzal. Rendezéshez kötődő számítást ne végezzen! (Összefésülést sem!)

A megvalósításkor a rekurzívan definiált algoritmusból induljatok ki! A rekurzió két ága mindenképpen külön szálon fusson, ám annak meghatározása, hogy mely ponton áll meg a rekurzív hívás, és történik helyben rendezés (buborékrendezés), a hallgató feladata, külön megkötés erre vonatkozóan nincs. Kérünk titeket, hogy ne használjatok beépített összefuttatási algoritmust, hanem kézzel implementáljátok az erre vonatkozó feladatot is! (összefésülés/összefuttatás programozási tétele.) A dokumentáció során ezt a választást mérésekkel alátámasztva indokoljátok a fejlesztői fejezetben!

Felhasználói dokumentáció

1. ***Környezet***

   A program fordítástól függően `.out` kiterjesztés esetén Linux/OSX oprendszereken, `.exe` kiterjesztés esetén Windows oprendszereken használható. Telepítésre nincs szükség, elegendő a futtatható állományt elhelyezni a számítógépen.

2. ***Használat***

   A program elindításához nincs szükség parancssori paraméterekre, így parancssoron kívül is lehet futtatni. A programmal egy mappaszinten kell elhelyezni az `input.txt` bemeneti fájlt. A program eredményét ugyanezen a szinten az `output.txt` kimeneti fájlba írja.

   Egy lehetséges bemenetet tartalmaz a mellékelt `input.txt` fájl, illetve a `tests` mappában további példák találhatóak. Saját bemeneti fájlok esetén fontos, hogy a feladatban megadott szempontok alapján írjuk az adatokat a fájlba, mivel ezek helyességét a programban nem ellenőrizzük.

Fejlesztői dokumentáció

1. ***Megoldás módja***

   A programot logikailag két részre, főfolyamatra és a rendező algoritmusokra bonthatjuk. A főfolyamat végzi a bemeneti adatok beolvasását, amiket egy `vector<string>` adatszerkezetben tárol. Erre hajtja végre a `MergeSort` rendező algoritmust, majd az eredményt szintén a főfolyamat írja a kimeneti fájlba.

2. ***Implementáció***

   A MergeSort implementálásához a rekurzív összefésülő rendezés algoritmusát használjuk fel, azonban hatékonysági szempontból fejleszthetünk rajta azáltal, hogy egy adott elemszám alatt buborékrendezünk (BubbleSort). A limitszám meghatározásához méréseket kell végeznünk, amelyeket az 5. pontban összegeztük. A rekurzió

két ágát külön szálakon végezzük, ehhez elég csak 1 új szálat nyitni, amelyiken tetszőlegesen az egyik ágat számoljuk, a másik maradhat a főszálon. A teljes implementáció egyetlen forrásfájlba szervezve, a `main.cpp` fájlban található.

3. ***Fordítás***

   A program forráskódja a `main.cpp` fájlban található. Fordításához egy C++11 szabványt támogató fordítóprogram szükséges. A fejlesztéshez a `g++` fordítót használtam: `g++ -std=c++11 main.cpp`

4. ***Tesztelés***

   Teszteléshez a `tests` mappában található bemeneti fájlokat használtam, amelyek mindegyikére az elvárt kimenetet állítja elő a program.

5. ***Mérések***

   A méréseket egy 10 000 soros bemeneti fájl végeztük. A grafikonról leolvasható, hogy a futási időt 64-es limitszámmal volt a legkisebb, azaz 64 elemre még érdemes buborékrendezni, felette azonban hatékonyabb az összefésülő rendezés.