Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Osztott rendszerek specifikációja és implementációja

IP-08bORSIG

Dokumentáció az 2. beadandóhoz

Abonyi-Tóth Ádám
DKC31P

2017.november 30.

1. Kitűzött feladat

Feladatunk annak eldöntése, hogy egy adott halmaznak létezik e olyan részhalmaza, melyben található elemek összege pontosan megegyezik egy előre megadott számmal!
A szükséges adatokat a program 3 parancssori paraméteren keresztül kapja.
Az első, egy egész értékű adat, mely a feladatban definiált számot jelzi, ezt kell valamilyen módon elérni a halmazelemek összegével.
A második, egy fájl neve - ez tartalmazza a kiinduló halmaz elemeit (inputfájl). A felépítése az alábbi: A fájl első sorában egy nemnegatív egész szám (N) áll - a kiinduló halmazunk elemszáma tehát N. A következő sorban összesen N db egész számot olvashatunk (pozitív és negatív egyaránt), melyek a halmaz elemeit jelölik (sorrendiséget nem kötünk meg köztük).
Egy megfelelő bemeneti fájl (például data.txt) ekkor:

```
6
3 34 4 17 5 2
```

A harmadik paraméter, annak a fájlnak a neve, melybe a feladat megoldása során kapott választ kell írni.
A kimeneti fájlban található információ az alábbi legyen:

```
   létezik N összegű részhalmaz:
      'May the subset be with You!'
   nem létezik ilyen részhalmaz:
'I find your lack of subset disturbing!'
```

2. Felhasználói dokumentáció

2.1. Rendszer-követelmények, telepítés

A program futtatásához szükséges a PVM rendszer megléte. Telepítéséhez a két bináris állomány, a *master* és a *child* megfelelő helyen való elhelyezése szükséges.

2.2. A program használata

A program futtatásához előbb lépjünk be PVM-be, és inicializáljuk tetszés szerint a bladeeket. Az indítás a PVM-en belül a következő paranccsal történik:
`spawn -> master sum input.txt output.txt`,
ahol `sum` a kérdéses összeg, `input.txt` a bemeneti fájl neve, `output.txt` pedig a kimeneti fájl neve a felhasználó könyvtárához képest.
Figyeljünk az inputfájlban található adatok helyességére és megfelelő tagolására, mivel az alkalmazás külön ellenőrzést nem végez erre vonatkozóan. Futás után a paraméterben megadott fájl tartalmazza a kapott eredményt.

3. Fejlesztői dokumentáció

3.1. Megoldási mód

A kódot logikailag három egységre osztjuk: a probléma reprezentációja, a főprogram, és a megoldó program. A főprogram feladata a be- és kimenet kezelése, a gyermeké a feladat elosztott megoldása, a harmadik pedig a probléma (és alproblémák) konzisztenciáját biztosítja. A végeredményt a főfolyamat írja ki a kimeneti állományba.

3.2. Implementáció

A probléma reprezentációja egész számok egy sorozata, és a kívánt összeg. Ehhez a kód a `problem.hpp` fájlban található. Az implementációs osztály tartalmazza a domain-hez tartozó műveleteket is: alproblémák elkészítése, a teljes probléma küldése és fogadása PVM-mel, illetve a probléma állapotának felmérése (megoldott, megoldhatatlan, nem eldönthető).
A `master.cpp` fájlban van a főprogram megvalósítása. A `main` függvény létrehozza a kezdeti feladatot az összeggel és a bemeneti fájlban lévő számokkal, majd a PVM segítségével meghív egy gyermek folyamatot a feladat megoldására. Amint a gyermek processz végzett, a program kiírja az eredményt a harmadik paraméterben kapott kimeneti fájlba.
A gyermek folyamat először fogadja a szülőtől a megoldandó problémát. Ha a probléma állapota ismert (megoldott, vagy megoldhatatlan), akkor visszaadja ezt az értéket, ha nem akkor a gyermek *Divide and Conquer* módszert alkalmazva próbálja megszerezni az eredmény. Ehhez a problémát felosztja két részproblémára, majd megpróbál létrehozni egy-egy újabb gyermek folyamatot ezek kiértékelésére. Ha nem sikerül új folyamatot indítani, a program áttér egy naiv rekurzív, de szekvenciális megoldásra. Ha az első folyamat eredményéből látszik, hogy a feladat megoldható, akkor a gyermek nem várja be a második folyamat eredményét.
Mind a két gyermek bevárása után a részfeladat eredménye mindenképp ismert lesz. Ilyenkor a futó gyermekfolyamat ezt az eredményt elküldi az eredményt a szülőnek, és terminál.

3.3. Fordítás menete

A fordítás a tantárgy honlapjáról letöltött `Makefile.aimk` fájl és az `aimk` eszköz segítségével történik. A makefile tartalmából kiderül, hogy a kódban használhatjuk a C++11-es szabványos elemeket. A modulokat úgy jelöljük ki fordításra, hogy hozzáfűzzük a makefile első sorához a nevüket, kiterjesztés nélkül.

3.4. Tesztelés

A program tesztelése két részből állt, az egyik a megoldás helyességét vizsgálta, a másik a kód sebességét.

A helyesség vizsgálatához az alapeseteket teszteltük, illetve a beadandó leírásánál mellékelt példabemenetek le lettek futtatva egy-egy helyes és helytelen összeggel.

A sebesség tesztelésére lefuttattam a legnagyobb elemszámú bemenetet az atlaszon különböző mennyiségű PVM-m blade hozzáadásával. Az eredmények alább láthatóak.

Bladek száma	0	1	2	4
Teszt1	0.061	0.061	0.031	0.053
Teszt2	0.061	0.036	0.033	0.279
Teszt3	0.072	0.063	0.357	0.139

Amint láthatjuk, a blade-ek száma ebben az esetben nem pozitívan, hanem inkább negatívan befolyásolja a program futásidejét. Ez a jelenség két dolognak tulajdonítható. Az egyik az, hogy egy-egy gyermekfolyamat által végzett számítás elenyésző, a folyamatok létrehozása és az üzenetek küldése több időt vesz igénybe. A másik az, olyan elemszámokra is párhuzamosan történik a kiértékelés, amikre a szekvenciális kiértékelés gyorsabb lenne. Ez a két tényező együtt fokozottan rontja az algoritmus teljesítményét. Megoldás lehetne egy adott elemszám alatt átváltani szekvenciális számításra, illetve egy-egy gyermekfolyamatnak komplexebb alfeladatot adni.