Generikus Pipeline modellezése

Ghadri Najib

2016-04-16

Feladat

Tervezzen objektummodellt csővezeték (pipeline) feldolgozási architektúra modellezésére! A csővezeték-rendszerű feldolgozásban a feldolgozó egységek egymás után vannak fűzve úgy, hogy a bemenetük a sorban megelőző elem kimenetére van kapcsolva. A legegyszerűbb esetben minden feldolgozó egység bemenetén van egy tároló, feldolgozás pedig két fázisban zajlik: Az első fázisban minden egység kiveszi a bemeneti tárolóból az ott levő adatot és feldolgozza, majd a második fázisban az eredményt továbbadja a következő elem bemenetére. Ezután a feldolgozási ciklus kezdődik elölről. Megj: A feldolgozó egységek elvileg párhuzamosan működnek, de a tároló lehetőséget ad az egymás utáni végrehajtásra is. Olyan modellt tervezzen, hogy tetszőleges hosszúságú csővezetéket ki tudjunk alakítani, amin bármilyen (GENERIKUS) adat végighaladhat! Az objektumok műveletvégző eleme legyen szintén definiálható! Mutassa be a működést összefésülő rendezés (merge sort) csővezetékkel történő megvalósításával! Az összefűző rendezés lényege az, hogy önmagukban rendezett adathalmazokból egyre nagyobb rendezett halmazokat képezünk. Csővezetékkel ez pl. úgy valósítható meg, hogy a feldolgozó egységek bemenetére olyan halmazpárokat adunk, mely halmazok önmagukban rendezettek, a kimeneten pedig ezek összefésült halmaza jelenik meg. Pl:
5,7 — 1,8 – 1,2,3,8 – 1,2,3,4,5,6,7
4,6 — 2,3 – 4,5,6,7
(A halmazpárokat egymás alatt tüntettük fel, a – pedig a feldolgozó egységet jelöli.) Válasszon megfelelő adatstruktúrát a rendezett halmazpárok tárolására! Valósítsa meg a feldolgozó elemeket! Értelemszerűen 2 bemeneti adatból kapunk 1 kimeneti adatot, így a feladat elején ismertetett általános modellt ki kell egészítenie úgy, hogy a feldolgozás ne mindig induljon el.

Implementáció

ShellTask

Bázis osztály az összes shell-ben futó taszkhoz

virtual void step-task

A későbbi metódus ebből lesz meghívva.

Task

ShellTaskból származik. Minden task osztálynak van egy inputja és egy outputja, és egy metódusa, amit öröklés során definiálunk. Absztrakt osztály.

adatok

MyStream input MyStream output Ezek a pipestreamek

Task()

default

bool isRunning()

Megmondja hogy fut-e.

virtual void eval(MyStream input,MyStream output)

Itt lesz örökléssel megvalósítva a kész függvény.

void set-input(MyStream input-stream)

Beállítja a bemenetet.

void set-output(MyStream output-stream)

Beállítja a kimenetet.

void step-task(void)

Meghívja a később definiált evalt.

PipeLine

Létrehoz egy task-ok ból álló csövet. Segítségével kaszkádosíthatunk bármilyen metódust, ami a taskban van benne. Konstruktor paraméterek: A köztes taszkok, és az input output leírók a pipe elején és végén.

adatok

std::vector Taskp tasks-p : A taskok tárolója std::vector MyStreamp pipe-streams : A streamek tárolója

PipeLine(Taskp new-task-p)

Első task-kal init.

void add-task(Taskp new-task-p)

Hozzáad a task-p sorhoz egy taskot, és beállítja a közös be/kimeneteket az előző task-kal.

void step-task()

Sorban vérehajtja a taskokat a saját step-task-jukkal.

Merger

Merge rendezést szimuláló osztály: két rendezett vektort összefésül és egy rendezett vektort ad. Ha ezt a taskot kaszkádosítjuk (pipeban) önmagával, akkor exponenciális mennyiségû vektort tudunk összefésülni. A futását szakaszokra bontottuk, így szimulálva a program egymás utáni bemeneteit. Szálakat használva nincs erre szükség.

adatok:

std::vector int vec1 - első vektor std::vector int vec2 - második std::vector int target-v - cél enum Stage current-stage - jelenlegi szakasz

UserInput

Bemenetkezelő osztály. Belső std::istream adattal inicializáljuk.

UserOutput

Kimenetkezelő osztály. Belső std::ostream adattal inicializáljuk.

MyStream

Saját stream osztály. Szükség volt rá mert az std stream-ek túl bonyolultak voltak. Egyszerû deque string alapú read/write metódusok álnak rendelkezésre. Ez hibamentes üzenetváltást enged meg.

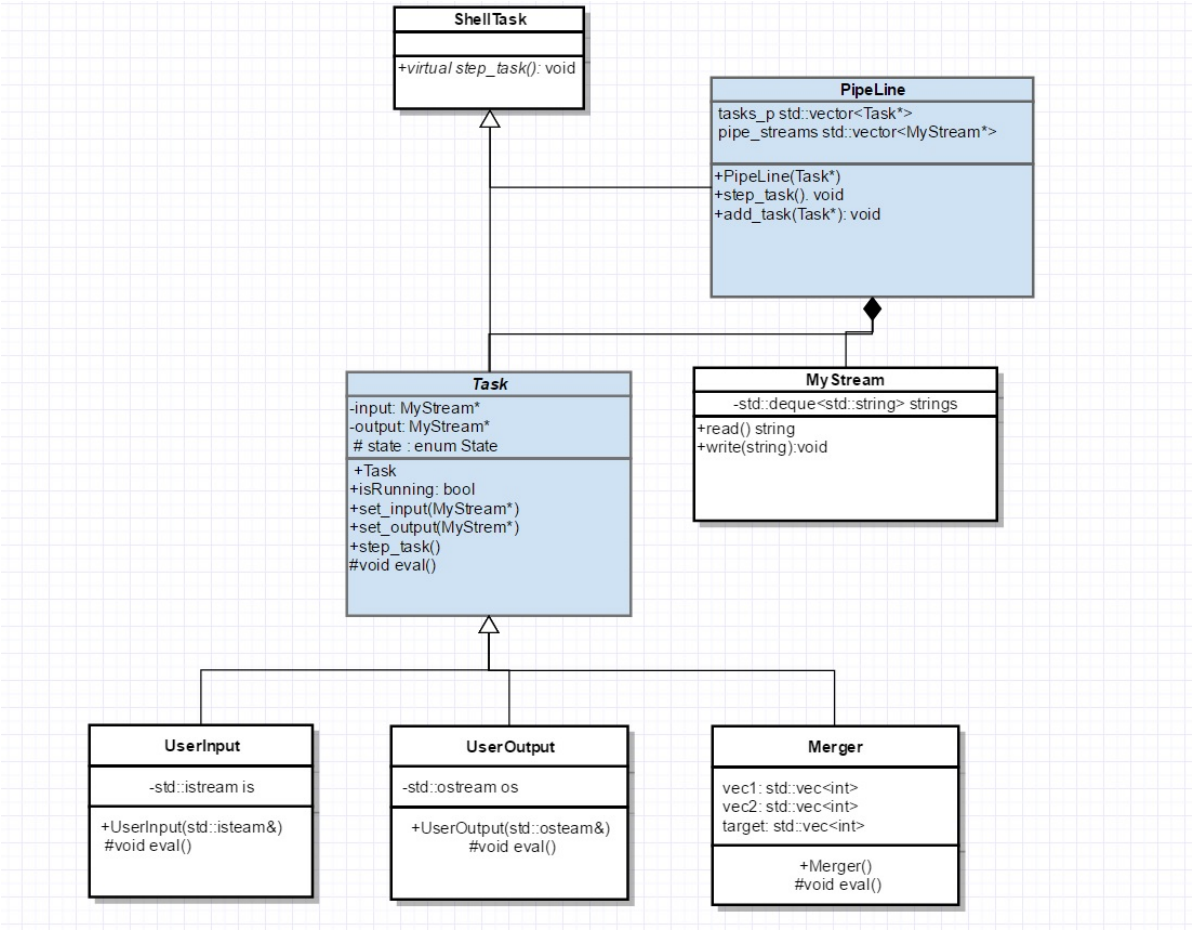

Figure 1: Caption

0.0.1 adatok:

std::deque¡std::string¿ strings : string sor.

const std::string read()

void write(std::string line)

A string soron pop és push műveleteket hajtanak végre.

Megjegyzések

Fontos tudni hogy egy operációs rendszerben a task-ok (folyamatok/programok) futása egyes pillanatokban megelőzheti egymást, az ütemezőtől függően, így

valódi párhuzamos vagy soros modellezést nem tudunk megvalósítani.
Továbbá az itt modellezett utasítások nem felelnek meg programoknak, hiszen egy függvény bemenete paraméter függő, míg egy program bármilyen szöveges paraméter értelmezésére képes (esetleg hibát kiadva). Emiatt a probléma miatt minden függvénynek egy előfeldolgozót kellene megvalósítani ami értelmezi a szöveges paramétereket. Ennek a módszernek az előnye az hogy nem kell bajlódni a generikus bemenettel, elég egy string-et átadni, és a kimenetet csak egy string-gé kell átalakítani. Természetesen a végső megvalósítás során emellett dönthettünk.

Ghadri Najib