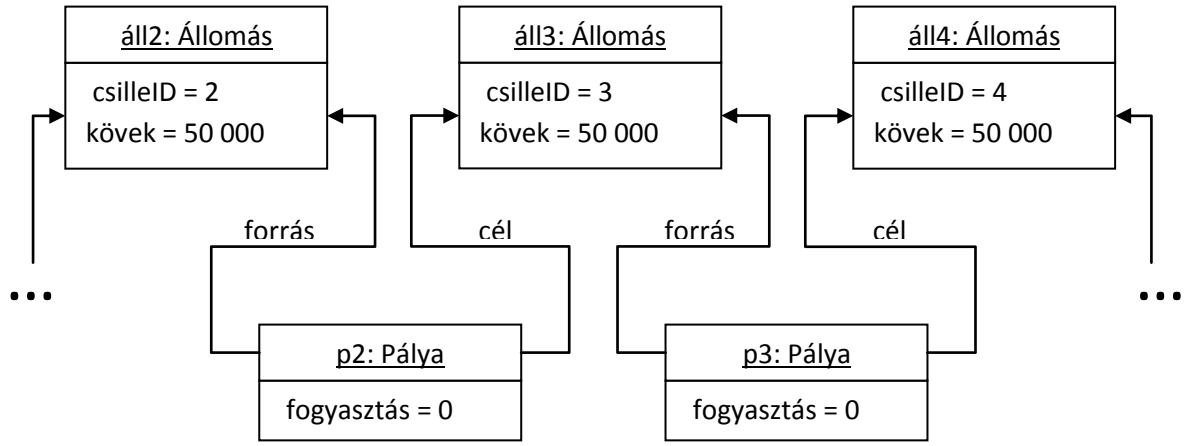
Csillerendszer – párhuzamos programozás gyakorlófeladat

Bevezető

A feladat egy kőbánya csillerendszerének modellezése. A bánya területén N állomása van a csillerendszernek, mindegyiken tárolunk bizonyos mennyiségű követ. Kezdetben az összes állomáson egyformán K kő van. Az állomásokat egy kötélpálya-lánc köti össze, minden két szomszédos állomás közötti pályaszakaszon külön motor mozgatja a csilléket, a többi szakasztól függetlenül. A csillékre egyedi azonosítószám van felfestve; ugyanannyi csille van, mint állomás (nyugalmi állapotban minden állomáson pontosan egy csille parkol). Ha egy pályaszakaszon valamilyen irányban követ kell szállítani, akkor – a kötélpálya kiegyensúlyozása miatt – a célállomásról is át kell vinni az üres csillét a forrásállomásra; tehát mindig egyszerre cserél helyet két szomszédos állomás csilléje, és utána minden állomáson újra egy csille lesz. A szimuláció során minden pályaszakasz összesen M csillecserét végez.



Szemléltető objektumdiagram

1. Készítsünk egy Állomás osztályt, amely (védett mezőben) nyilvántartja az állomáson lévő kövek mennyiségét, és az állomáson parkoló csille azonosítóját! Publikus accessorok (get/set) olvassák és írják a csille azonosítót, és olvassák a kőmennyiséget; szintén publikus metódussal megnövelhető vagy lecsökkenthető a kövek mennyisége. Szükség van továbbá a Pályaszakasz osztályra, amely ismer egy forrás és egy cél Állomást, van egy saját randomgenerátora, és méri a motor eddigi összfogyasztását. A publikus futtató metódusa M alkalommal végrehajtja a csillecserét szimuláló ciklusmagot:
    a. sorsol egy véletlen bitet (felfelé vagy lefelé kell követ szállítani?), melynek kimenetelétől függően vagy csökkenti a forrás állomás köveinek mennyiségét és növeli a célállomásét, vagy fordítva;
    b. kicseréli a két állomás csilleazonosítóját;
    c. kiszámolja a forrás állomás kőmennyiségének négyzetgyökét (ez jelképezi a csillecsere energiaigényét), és hozzáadja a motor fogyasztásmérőjéhez.
2. Példányosítsunk N állomást, közéjük sorban N-1 pályaszakaszt (elsőről másodikra, másodikról harmadikra, stb.). Legyen az *i*-edik állomáson kezdetben az *i* azonosítójú csille, és K kő. Futtassuk le egymás után összes pályaszakaszt! A megfelelő rendszerhívásokkal mérjük meg a

futási időt, és szükség esetén növeljük vagy csökkentsük M értékét, hogy 1-10 másodperc közé essen. Ajánlott paraméterek: N = 5, M = 5 000 000, K = 10M (igazítsuk mindig M-hez).

3. Utána módosítsuk a programot, hogy külön-külön szálakon futtassa le a pályaszakaszok szimulációját, majd várja be az összes indított szál befejezését. Hasonlítsuk össze a két változat futási idejét! Ha többmagos processzor esetén is a szekvenciális változat a gyorsabb, akkor a szálkezelés overheadje elnyomja a szimuláció számításigényét. Ebben az esetben bonyolítsuk kicsit el a ciklusmagot számításigényes lépésekkel, pl. a négyzetgyök négyzetgyöke legyen a fogyasztás, vagy 1 helyett véletlenszerűen 1 és 4 közötti számú követ vigyen egy csille; majd ismételjük meg a soros és párhuzamos változat összemérését.
4. A futás végén ellenőrzésként írassuk ki az egyes állomásokon parkoló csille számát, valamint adjuk össze és írjuk ki az összes tárolt kő mennyiségét. Ugyanaz lesz az eredmény párhuzamos és szekvenciális futás esetén? Magyarázzuk meg az eltéréseket!
5. Alakítsuk át az Állomás osztályt, hogy a kőmennyiség növelése / csökkentése atomikus művelet legyen. Használjuk a szinkronizációs könyvtár által biztosított atomi integer-műveleteket. Hogyan és miért változik a többszálú futási idő és az ellenőrzés eredménye?
6. Atomi műveletek *helyett* használjunk zárakat (mutexet): a Pályaszakasz osztály a ciklusmag legelején zárolja a forrás- majd a célállomást, a ciklusmag végén pedig engedje el a zárakat. Hogyan és miért változik a többszálú futási idő és az ellenőrzés eredménye?
7. Zárjuk körré a pályákat, vagyis vegyünk fel egy N-edik pályaszakaszt is, amelynek az utolsó állomás a forrása és az első a célja. Többszálú, zárolásos futás esetén mit tapasztalunk? Mi a magyarázat?

Javasolt technológiák

Használjunk többmagú processzorral vagy több processzorral felszerelt számítógépet a mérésekhez.

Java vagy .NET platformon az osztálykönyvtár biztosít szálkezelő és szinkronizációs lehetőségeket. C++ nyelv esetén érdemes kipróbálni az *OpenMP,* vagy *Amino Concurrent Building Blocks* könyvtárat. A többszálú programok fejlesztése során sok a hibalehetőség; használjunk kényelmes debugger programot (pl. Java esetén *Eclipse IDE*), amellyel töréspontnál vagy hosszadalmas futás esetén kézzel meg lehet állítani a programot, ellenőrizni az objektumok és változók értékeit, valamint a különböző szálak állapotát és lefoglalt zárait.

Akkor igazán szép a megoldás, ha közös a forráskód az egyes részfeladatok között, és a futás módja egyszerű kapcsolóval vagy konstanssal jelölhető ki.

Kontakt

Bergmann Gábor, bergmann@mit.bme.hu