

10/01/2021

Joel Farré Cortés (78103400T)

Joel Aumedes Serrano (48051307Y)

Escola politècnica superior – universitat de lleida

Pràctica 2

Grau en Enginyeria Informàtica

Sincronització

- Evitar les condicions de carrera

Respecte a la pràctica 1 hem pogut evitar les condicions de carrera de forma més eficient utilitzant *mutexs* i blocs *syncronized.* Degut al disseny de l’algoritme emprat és probable que les estadístiques no donin sempre el mateix ja que es pot donar el cas de que l’accés al processador no sigui sempre el mateix.

- Thread de missatges

Hem utilitzat variables de condició per a implementar el *thread* de missatges de forma eficient. La idea principal es que cada *thread* que vol escriure un missatge adquireix un *mutex* o un bloc *syncronized* per escriure el seu missatge a una cua de missatges. Un cop escrit el missatge i abans d’alliberar el *mutex* el *thread* notifica al *thread* de missatges. El *thread* de missatges ha estat esperant en un *wait()* amb la variable de condició. Un cop despert, comprova si la condició es compleix o no. Aquesta condició és si tenim 100 missatges a la cua. Si no disposem de 100 missatges, el *thread* de missatges se’n torna a esperar. Si en canvi, disposem d’aquests 100, els imprimeix per pantalla tots i els elimina per a deixar espai per a la resta. La implementació és molt semblant entre C i Java.

- Espera del resultat final

Aquí es realitza de forma diferent entre C i Java. A les dues implementacions utilitzem un mètode de sincronització i variables compartides per al resultat però també fem servir la funció *join()* per a assegurar-nos de que el *thread* ha finalitzat correctament. La diferència entre els dos llenguatges es troba en el mètode de sincronització implementat. En C utilitzem una barrera i en Java un semàfor.

En C inicialitzem una barrera per a esperar a dos *threads*. Aquesta barrera es planta en el *thread* que espera el resultat quan el necessita i en el *thread* que el calcula quan aquest resultat ja està disponible per a l’altre *thread*. Un cop sortit de la barrera, el *thread* que ha calculat simplement acaba i l’altre *thread* fa el *join*.

En Java utilitzem un semàfor. Aquest semàfor l’inicialitzem a zero *permits*, llavors quan el *thread* que necessita el resultat, el necessita, intenta adquirir un *permit*, l’altre *thread* alliberarà aquest *permit* quan el resultat ja estigui disponible a la variable compartida.

- Estadístiques

Per a l’emmagatzematge i tractament de les estadístiques en C s’ha utilitzat un *struct* i en Java una *class*. Degut a l’arquitectura del programa, solament contem les estadístiques per a cada *slot* de del *thread* i no per a cada *thread* ja que es creen i s’eliminen amb molta freqüència. Les estadístiques es comptabilitzen de la següent manera:

- Nombre de combinacions vàlides: Es sumarà una unitat cada cop que el *thread* tingui un equip complet.

- Nombre de combinacions no vàlides: Es sumarà una unitat quan s’intenti afegir un jugador en un equip però que no sigui possible ja que o l’equip estigui complet o no es disposi del suficient pressupost. També es sumarà quan s’hagi comprovat tots els jugadors i l’equip no estigui complet.

- Nombre de combinacions avaluades: És la suma de les combinacions vàlides i no vàlides.

- Cost mig de les combinacions vàlides: És la divisió del cost total de les combinacions vàlides entre el nombre de combinacions vàlides.

- Puntuació mitja de les combinacions vàlides: És la divisió de la puntuació total de les combinacions vàlides entre el nombre de combinacions vàlides.

- Millor i pitjor combinacions: Combinació amb tots els jugadors amb millor i pitjor puntuació.

Quan un *thread* modifica les seves pròpies estadístiques, no utilitza cap mètode de sincronització ja que sabem que l’únic *thread* que les modificarà serà ell mateix. Per l’altra banda, per als *threads* globals, es regula l’accés amb l’ús de semàfors.

- Progrés

Per al progrés es té en compte el nombre de combinacions avaluades de cada *thread*. L’arquitectura del programa podria causar que, en el cas de que quan un *thread* arriba a M combinacions s’esperi a que arribin els altres, ens trobéssim en una situació de *deadlock* ja que un *thread* parat no generaria més *threads* i això faria que no arribés a les M combinacions. Aleshores, el que fem es imprimir les estadístiques globals i les parcials cada cop que tots els *threads* hagin completat una etapa, però quan un *thread* ha completat l’etapa, no s’espera a que els altres la completin.

- Anàlisi del rendiment