Sistemes Operatius II - Pràctica V

Gabriel Lluís i Pau Soler

3 de Gener de 2020

1 Introducció

En aquesta pràctica crearem l'arbre amb tècniques oncurrents però a nivell de fil, i farem la sincronització amb monitors. En concret, els fils hauran de fer:

1. El fil principal llegirà el diccionari de paraules i crearà l'estructura de l'arbre, aleshores, crearà n fils secundaris que s'encarregaran de processar els fitxers.
2. Cada fil secundari tindrà una còpia local de l'arbre del fil principal i llegirà el fitxer que li pertoqui tot actualitzant el seu propi arbre. Després, actualitzarà l'arbre del fil principal.

En aquesta pràctica entreguem dues implementacions, una que només assigna als nodes de l'arbre dels fils secundaris les dades dels nodes de l'arbre del fil principal, i una altra que per cada fil crea un arbre local nou.

2 Un únic fil secundari

Cada fil té les següents funcionalitats:

- **Fil Principal:** No té mètode propi (està dins de *crear_arbre_fil()*). La seva funció és indexar les paraules del dicionari en un arbre que després serà copiat pel fil, crear un fil secundari i, un cop aquest hagi acabat la seva funció, tornar a juntar-lo amb el fil principal amb un join.
- **Fil Secundari:** La seva funció és recórrer els fitxers de la base de dades actulitzant els valors dels nodes del seu arbre propi segons correspongui, i després actualitza l'arbre del fil principal.

En aquest cas, com es tracta d'una implementació amb un únic fil secundari, encara no és necessarí l'ús de monitors, com sí ho serà en l'apartat següent.

3 Múltiples fils

En aquest cas, el **fil principal** fa el mateix que en l'apartat anterior, la única diferència que hi trobem és que, donat que farem servir múltiples fils, aquest haurà

de crear n fils secundaris enlloc de només un i, per tnt, també haurà de fer el join quan pertoqui n vegades, un cop tots els fils secundaris hagin acabat les seves tasques.

Ara és el moment en què es torna necessari diferenciar entre les dues versions del codi que entreguem:

3.1 Versió 1

Cada fil crea un arbre propi i copia els punters que apunten a la root i al nombre d'elements de l'arbre del fil principal, permetent així treballar amb un arbre igual que el que tenim al fil principal. Aleshores augmenta un comptador global que fem sevir per saber quin fitxer ha de llegir cada fil i per saber si ja s'han llegit tots els fitxers. Aquest augment del comptador el fem dins d'un monitor (*mutex_write*) per controlar que dos fils no hi accedeixin alhora.

Un cop fet això, fa el *search_words* del fitxer corresponent i, altre cop dins d'un monitor (*mutex_join*), actualitza l'arbre del fil principal.

3.2 Versió 2

Cada fil crea un arbre propi i crida al mètode *copiar_nodes_arbre*, que crea un arbre nou node per node copiant l'arbre del fil principal, que hem passat per paràmetre dins d'un struct. Tant el comptador com l'accés al fitxers funcionen igual que en la versió anterior, ara bé, on sí que hem fet canvis és en el mètode que actualitza l'arbre, on hem posat el monitor que abans tancava tot el mètode perquè únicament bloqueji el node quan actualitzem el seu *num_times*.

En aquesta segona versió hem usat també un altre monitor (*mutex_malloc*) en tots els mallocs i els frees que impliquen operacions amb fils secundaris, és a dir, en el propi mètode dels fils secundaris i en el mètode *copiar_nodes_arbre*.

4 Qüestions a respondre

En quant a rendiment, hem pogut observar que amb múltiples fils millora respecte al rendiment amb un sol fil, ja que podem llegir diversos fitxers alhora (tot i que només actualitza l'arbre un fil secundari a la vegada) així com també millora respecte el rendiment respecte a la pràctica anterior, en la qual usàvem processos.

Respecte a la segona pregunta que se'ns planteja, no hem provat a comparar les dues propostes, ho hem implementat amb una secció crítica que només permet a un fil a la vegada actualitzar l'arbre, ja que vam estar comentant aquest mateix tema amb el nostre professor de pràctiques i vam arribar a la conclusió que no valia la pena, en aquest cas, bloquejar a nivell de node, com sí que vam fer en la pràctica anterior, la pràctica 4.