Sistemes Operatius II

Pràctica 4

Vicent Roig / Igor Dzinka

21/12/2014

1. Part 1

A aquesta part se'ns demana paral·lelitzar el processament de base de dades i fer servir múltiples fils.

Amb aquesta implementació hem d'anar en compte, ja que tenim dues zones crítiques que s'han de protegir amb mutex. Una zona es l'accés a la llista de fitxers on no podrem tenir mes d'un fil llegint de la llista alhora. L'altra zona es l 'accés a l'arbre global, ja que copiar les dades a l'arbre implica canviar l'estructura del mateix i en cas de que hi accedeixi mes d'un fil a l'hora podríem tenir problemes de consistència de dades i per tant errors.

Proves realitzades:

A continuació posem els temps d'execucions amb diferent quantitat de fils. Es passa com a paràmetre tota la base de dades de Gutenberg (595 fitxers).

Threads	1	2	4	8	16	32
Time	real 0m48.007s user 0m47.085s sys 0m0.859s	real 0m31.369s user 0m59.179s sys 0m0.987s	real 0m23.358s user 1m10.572s sys 0m1.198s	real **0m20.855s** user 1m7.271s sys 0m1.246s	real 0m20.905s user 1m5.303s sys 0m1.314s	real 0m21.808s user 1m7.577s sys 0m1.285s

Bloqueig de tota la funció de copia vs. Bloqueig de nomes parts d'accés a l'arbre

Es molt **millor bloquejar tota la funció de copia de la estructura local a l'estructura global** que fer bloqueig dins de la funció de copia a les parts on s'accedeix a l'arbre. La raó d'això es que **bloquejant tota la funció**, hem de realitzar el bloqueig **un cop per fitxer** processat.

Al fer el **bloqueig de nomes parts d'accés a l'arbre** fa que el programa trigui molt més. Si analitzem el que s'està fent, ens adonem de que **s'agafa i s'allibera la clau de mutex** cada cop que s'intenta buscar un node o inserir-ne un a l'arbre. Això es fa **milers de cops per cada fitxer** processat i les operacions d'agafar i alliberar la clau son costoses.

Opcional

Hem fet una versió del programa que es reparteix els fitxers entre els fils equitativament ***(anomenats 'CHUNKS').*** Per exemple, si tenim 2 fils, el primer farà la primera meitat del fitxer (de 1 a (fitxers)/2) i el segon la segona meitat.

A continuació les proves realitzades amb aquest plantejament. El codi d'aquesta versió esta al fitxer ***main.c*** de la carpeta ***src0***

Threads	2	4
Time	real 0m33.725s user 0m55.775s sys 0m0.911s	Real 0m23.254s user 1m7.615s sys 0m1.084s

Com podem veure, el fet de paral·lelitzar el processament d'aquesta forma, proporciona resultats pitjors que amb el mètode anterior per la raó que s'ha comentat. Un fil pot acabar abans que l'altre i romandre inactiu, en aquest cas perdem eficiència.

Part 2

Aquesta segona part es centra en la utilització combinada de funcions de bloqueig i variables condicionals.

Se'ns demana implementar el nostre codi de tal manera que es creï un total de N + 1 fils, és a dir, un fil més que a la part 1. N d'aquests fils estaran especialitzats en extreure les paraules dels fitxers, mentre que un únic fil estarà especialitzat únicament en copiar les paraules de l'estructura local a la global.

Paradigma utilitzat

El paradigma de programació multifil que fem servir per implementar aquesta proposta es el de **productors i consumidors**. Tindrem N fils productors (El valor de la N el definim nosaltres) que llegiran els fitxers de text i crearan les HashTables corresponents i les desaran a un buffer circular. El fil consumidor esperarà a que hi hagi dades al buffer i quan hi hagi una dada, es dedicarà a copiar- la a l'estructura global.

Proves realitzades

Aquestes son les característiques de l'ordinador amb el que hem realitzat les nostres proves, *és important tenir en compte que en tot moment em realitzat les proves sobre la mateixa màquina per tal de poder comparar el rendiment obtingut amb cada versió multifil:*

```
Arquitectura:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Orden de bytes:          Little Endian
CPU(s):                  4
On-line CPU(s) list:     0-3
Hilo(s) por núcleo:      2
Núcleo(s) por zócalo:    2
Socket(s):               1
Nodo(s) NUMA:            1
ID del vendedor:         GenuineIntel
Familia de CPU:          6
Modelo:                  58
Stepping:                9
CPU MHz:                 774.000
BogoMIPS:                3591.94
Virtualización:          VT-x
caché L1d:               32K
caché L1i:               32K
caché L2:                256K
caché L3:                3072K
NUMA node0 CPU(s):       0-3
```

Hem remarcat el nombre de nuclis i fils per nucli donat que empíricament esdevé com informació important. Més endavant veurem com **la màxima eficiència multifil obtinguda**, efectivament **succeix quan el nombre de fils a executar coincideix amb el nombre de fils** total**s** en funció de **l'arquitectura de la màquina** que ho executa. En aquest cas **4x2 = 8 fils**

A continuació mostrem els resultats d'execució del programa. Processem tota la base de dades Gutenberg (595 fitxers) amb diferents quantitats de fils productors i sempre amb 1 fil consumidor

NTHREADS	**1**	**2**	**4**	**8**	**16**	**32**
Time	real **0m40.751s** user 0m52.252s sys 0m1.050s	real **0m25.703s** user 1m1.071s sys 0m1.232s	real **0m21.644s** user 1m8.191s sys 0m1.197s	real **0m21.314s** user 1m7.016s sys 0m1.443s	real **0m22.164s** user 1m7.770s sys 0m1.386s	real **0m23.547s** user 1m8.799s sys 0m1.741s

Tal i com hem comentat, als dos casos **obtenim el millor rendiment quan el nombre de threads és el mateix que suporta la pròpia arquitectura de la CPU**.

Per tal d'optimitzar l'execució en funció del processador, hem fet una adaptació per tal d'obtenir el nombre de fils més adequat per processador que ho executi mitjançant una crida a una funció de l'estàndard POSIX.

A mode conclusiu, podem determinar que en aquest cas obtenim un rendiment lleugerament inferior respecte a la part 1, probablement donat al fet que comptar amb un sol consumidor junt a un buffer limitat resulta un punt crític, a més, en aquest cas també cal tenir en compte que el nombre de bloquejos és prou superior.

Mida de buffer

La mida del buffer circular efectivament si pot tenir efecte sobre el comportament global. Per un costat, si comptem amb un buffer massa petit (menys posicions que productors), es pot donar el cas de que productors preparats per executar s'hagin de quedar adormits i no puguin treballar per falta d'espai al buffer.

Per altra banda (si tenim moltes més posicions al buffer que no pas productors cas això te coses positives, com ara el fet de que els productors disposin de més espai per tal d'anar executant i produint noves dades. Encara que cal no oblidar que la limitació prové en que només hi ha un únic consumidor.). En cap cas ens interessarà tenir més posicions al buffer que dades a processar ja que malbaratarem memòria.