Pràctica 4

SISTEMES OPERATIUS II

DAVID DE LA PAZ / ORIOL RABASSEDA

ÍNDEX

Funcionament del programa

En aquesta quarta pràctica tornem a treballar sobre les pràctiques anteriors, centrant-nos en la utilització de fils per tal de processar les dades d'aeroports de forma més ràpida. El main és el mateix menú de la pràctica 3, tot i que ara hem modificat la funció 1 de creació de l'arbre (***database.c***) per tal de crear l'arbre binari utilitzant fils. D'aquesta manera, tindrem un fil secundari per inicialitzar l'arbre d'aeroports i un nombre variable de fils addicionals per processar la informació del fitxer de vols i inserir-la a l'arbre. Haurem d'anar amb compte per tal de bloquejar els fils a l'hora de llegir informació del fitxer principal i per inserir dades als nodes. Anem a veure amb més detall el funcionament de la pràctica:

Per tal d'evitar repetir explicar el funcionament de la pràctica anterior, només ens centrarem en explicar les noves instruccions relacionades amb l'ús dels fils. A l'hora de crear un arbre, comencem per cridar a la funció ***airport_tree(*fp)*** per tal d'inicialitzar-lo amb el fitxer d'aeroports. La única diferencia amb l'altra pràctica és que hem modificat l'estructura dels nodes per afegir un *mutex* individual per cadascun per tal de poder-los bloquejar al modificar els seus vols. Abans d'inserir un node a l'arbre, l'inicialitzem a NULL amb la funció ***pthread_mutex_init(&(n_data→mutex))*** (com es pot veure, el mutex per dada node està a la struct que defineix el node)***.***

Un cop tenim l'arbre, el següent pas és crear una llista dels threads que utilitzarem per omplir l'arbre amb els diferents vols. També inicialitzem una estructura *thread_data* per poder enviar informació als threads, la qual conté el nom del fitxer de vols i l'arbre en qüestió. A continuació utilitzem un *for* per crear tots els threads a utilitzar amb la funció ***pthread_create(&ntid[i], NULL, thread_ini, (void *)t_data)***, la cual va creant els diferents threads de la llista (amb atributs per defecte) utilitzant la funció *thread_ini*, que veurem a continuació amb més detall. També enviem el struct *t_data* per tal de poder enviar informació a cada thread. Un cop creats, utilitzem un altre *for* per cridar per a cada fil la funció ***pthread_join(ntid[i], NULL)***, la qual espera a que aquests acabin la seva execució.

Què fa exactament la funció ***thread_ini(*args)***? Cada fil utilitzarà un bloc de mida donada per anar llegint grups de N línies del fitxer de vols per posteriorment introduir les dades a l'arbre. Els arguments de la funció són el struct que conté el arbre i el nom del fitxer de vols. Mentre encara quedin línies per llegir, fem malloc per un crear un bloc de mida «BLOCK_SIZE» (variable). A continuació fem un ***lock*** de f_mutex, el qual hem declarat a dalt de tot de manera estàtica amb *PTHREAD_MUTEX_INITIALIZER*. Això ens permet bloquejar el file de vols per lectura, de manera que un bloc pugui llegir N línies seguides. Utilitzant un bucle *while* anem llegint les N línies amb fgets, fem el seu malloc corresponent i les anem guardant al bloc amb un ***memcpy***. Com que l'últim bloc pot ser de mida menor a la resta, si passa això activem un flag 'eof' per saber que ja hem acabat. Un cop omplert el bloc, desbloquegem f_mutex fent ***unlock*** i introduïm a l'arbre la informació del bloc. D'aquesta manera, cada fil va introduint informació a l'arbre en blocs de N línies fins acabar de llegir el fitxer (Quan acabem d'utilitzar un bloc l'alliberem amb el seu *free* corresponent). El flag de fi de fitxer, també para l'execució de lectura de la resta de fils perquè està declarat de forma global dins de la zona del mutex de file (cada fil executa lectura més escriptura de forma iterativa).

Finalment, a l'hora d'inserir la informació dels vols a l'arbre també hem de fer ***lock*** per a cada node per tal d'evitar que 2 o més fils escriguin al mateix node alhora. D'aquesta manera, quan la funció *flight_list()* troba el node a modificar, si aquest existeix el programa **bloqueja** el mutex propi del node (n_data->mutex) per poder introduïr les dades corresponents a un vol. Al finalitzar, es fa ***unlock*** del mutex pertanyent al node per tal de que altres fils puguin introduir també informació sobre aquest node. D'aquesta manera aconseguim evitar qualsevol tipus de conflicte relacionat amb els fils. Un cop aquest acabin d'introduir totes les dades corresponents, el fil principal despertarà i el programa tornarà al menú principal (s'espera a l'execució dels fils amb ***pthread_join(ntid[i], NULL)***).

Proves realitzades

Hem realizat la creació de l'arbre amb diferents nombres de fils i diferents mides de blocs, concretament per 2, 4, 8, 16, 32, 64 i 128 fils i per 10, 100, 1.000, 10.000, 100.000 i 1.000.000 línies per blocs. Hem fet les pràctiques sobre el fitxer complert de 2008 amb 7 milions de línies (l'ordinador del que disposàvem a casa no té suficient espai per provar-ho amb el fitxer gran). Els resultats són els següents:

```
Introdueix fitxer que conte llistat d'aeroports: data/aeroports.csv
Introdueix fitxer de dades: data/2008.csv
Block size: 10  Num threads: 2   Time: 7.351
Block size: 10  Num threads: 4   Time: 7.306
Block size: 10  Num threads: 8   Time: 5.970
Block size: 10  Num threads: 16  Time: 5.880
Block size: 10  Num threads: 32  Time: 5.851
Block size: 10  Num threads: 64  Time: 6.170
Block size: 10  Num threads: 128         Time: 5.310
Block size: 100 Num threads: 2   Time: 6.263
Block size: 100 Num threads: 4   Time: 6.098
Block size: 100 Num threads: 8   Time: 5.602
Block size: 100 Num threads: 16  Time: 6.542
Block size: 100 Num threads: 32  Time: 6.640
Block size: 100 Num threads: 64  Time: 6.569
Block size: 100 Num threads: 128         Time: 6.546
Block size: 1000        Num threads: 2   Time: 7.524
Block size: 1000        Num threads: 4   Time: 7.521
Block size: 1000        Num threads: 8   Time: 7.308
Block size: 1000        Num threads: 16  Time: 7.138
Block size: 1000        Num threads: 32  Time: 6.998
Block size: 1000        Num threads: 64  Time: 6.924
Block size: 1000        Num threads: 128         Time: 6.858
Block size: 10000       Num threads: 2   Time: 7.529
Block size: 10000       Num threads: 4   Time: 7.506
Block size: 10000       Num threads: 8   Time: 7.178
Block size: 10000       Num threads: 16  Time: 7.035
Block size: 10000       Num threads: 32  Time: 6.973
Block size: 10000       Num threads: 64  Time: 7.068
Block size: 10000       Num threads: 128         Time: 7.080
Block size: 100000      Num threads: 2   Time: 7.310
Block size: 100000      Num threads: 4   Time: 7.138
Block size: 100000      Num threads: 8   Time: 6.964
Block size: 100000      Num threads: 16  Time: 7.019
Block size: 100000      Num threads: 32  Time: 6.878
Block size: 100000      Num threads: 64  Time: 6.993
Block size: 100000      Num threads: 128         Time: 6.966
Block size: 1000000     Num threads: 2   Time: 7.430
Block size: 1000000     Num threads: 4   Time: 7.686
Block size: 1000000     Num threads: 8   Time: 7.397
Block size: 1000000     Num threads: 16  Time: 6.875
Block size: 1000000     Num threads: 32  Time: 6.924
Block size: 1000000     Num threads: 64  Time: 7.148
Block size: 1000000     Num threads: 128         Time: 7.589
```

D'aquests resultats, extraiem diverses conclusions:

- A partir dels 8 fils la diferència és mínima en tots els casos, això es del al fet que l'ordinador no en té més.
- Quants més fils pràctics millor, és a dir, s'han d'usar els 8 fils disponibles no menys
- Quant més petita la mida del bloc millor.

Per tant, el millor calibrat és usar el nombre de fils de l'ordinador (posar-ne com a mínim el nombre disponible), i fer la mida del bloc prou petita per no haver de bloquejar massa temps consecutiu la lectura de fitxer alhora que aprofitar la llesca de temps en el que el tens disponible.

Addicionalment, els fitxers estan compilats amb les comandes d'execució optimitzada.