Sistemes Operatius II - Pràctica 4

Novembre del 2018

Les dues darreres pràctiques de Sistemes Operatius 2 se centren en l'ús de múltiples fils per incrementar l'eficiència computacional de l'aplicació. Aquesta pràctica se centra en la utilització de les funcions de creació i bloqueig de fils. La darrera pràctica se centrarà en la utilització de variables condicionals.

Índex

1 Introducció

Les tres primeres pràctiques s'han centrat en aprendre a utilitzar diverses estructures per a la manipulació de dades, l'ús de funcions de lectura i escriptura de dades a disc.

En aquesta pràctica ens concentrarem en utilitzar múltiples fils per al processament de les dades dels avions. Per realitzar aquesta pràctica es pot fer servir el codi proporcionat al campus, tot i que no hi ha inconvenient si continueu desenvolupant el vostre codi.

A les següents seccions es detallen les tasques a realitzar. Per a la implementació de la solució es demana utilitzar monitors[1]. Les funcions que es mencionen en aquest document es trobareu descrites a les dues fitxes del campus.

2 Funcionalitat a implementar

A continuació es descriu l'esquema a implementar.

1. En executar el vostre programa només hi haurà un fil. Anomenarem aquest fil el **fil principal**. Aquest fil serà el que imprimirà per pantalla el menú, el que permetrà desar l'arbre a disc, carregar-lo de disc o bé imprimir el nombre de vegades que una paraula apareix a l'arbre.

2. En seleccionar del menú l'opció de creació d'arbre, el fil principal demanarà per teclat el fitxer dels aeroports així com el fitxer que conté les dades dels vols d'avions. A continuació el fil principal crearà l'estructura d'arbre a partir del fitxer d'aeroports. A més, el fil principal, abans de crear els fils secundaris, obrirà el fitxer que conté les dades a processar i en llegirà la capçalera. D'aquesta forma els fils secundaris no s'han d'encarregar d'ignorar la capçalera.

3. Per processar les dades dels vols es faran servir múltiples fils. El fil principal crearà, amb la funció *pthread_create*, un mínim de $F = 2$ **fils secundaris** que accediran de forma concurrent al fitxer de dades per extreure'n la informació que conté (més endavant es detalla aquest procés). Per fer-ho el fil principal ha de passar (per argument) als fils secundaris el fitxer de dades obert així com l'arbre en què s'inseriran les dades extretes.

4. Mentrestant, el fil principal es quedarà esperant, amb la funció *pthread_join*, que els fils secundaris acabin la seva feina.

5. Un cop hagin finalitzat tots els fils secundaris amb la seva feina, el fil principal es despertarà i tornarà a visualitzar el menú.

Suposem que s'han creat els fils secundaris. Cada fil secundari realitzarà un bucle per extreure les dades del fitxer i inserir la informació a l'arbre, de forma similar a la pràctica anterior.

Cada fil llegirà un bloc de N línies **seguides** del fitxer (N podrà tenir valors de 10, 100, o 10.000, per exemple, i pot estar definit amb un *#define* al codi C). Per tal fer-ho caldrà fer servir una clau compartida entre els fils i utilitzar les funcions de bloqueig *pthread_mutex_lock* i *pthread_mutex_unlock* per assegurar que el fil pot llegir aquest bloc. Un cop s'ha llegit el bloc es passarà a extreure la informació de cada línia d'aquest bloc i inserir-la l'arbre. La implementació es pot fer de forma iterativa: línia a línia es va extraient la informació de cada línia del bloc i s'insereix a l'arbre.

[1] A data d'inici d'aquesta pràctica possiblement encara no s'han impartit a teoria. La pràctica es pot iniciar, però, sense aquest coneixement. Veure secció 3.

L'arbre estarà compartit entre tots els fils i aquests hauran d'anar en compte a l'hora d'inserir-hi la informació extreta. Es demana implementar una solució de forma que el bloqueig es realitzi a nivell de node de l'arbre: fils diferents no poden accedir al mateix temps al mateix node a l'arbre, però sí que poden inserir informació en dos nodes diferents. Per tal de fer-ho caldrà que cada node de l'arbre també tingui la seva pròpia clau.

Observeu que es comenta que cada fil llegeixi un bloc de N línies seguides. Quina és la raó d'això? La solució més senzilla seria agafar $N = 1$, oi? És a dir, fer que cada fil llegeixi una línia, n'extregui la informació, la insereixi a l'arbre i torni a començar. Bé, aquesta va ser de fet la primera solució implementada i es va comprovar que no es podia augmentar l'eficiència computacional del codi. Això és per l'estructura del fitxer de dades: les dades estan estructurades de forma que línies seguides tenen sovint el mateix origen. Això fa que fils diferents, en extreure la informació, intentin inserir la informació a mateix node, amb la qual cosa es bloquegen entre sí i no es pugui augmentar l'eficiència del codi. Per tal de poder augmentar l'eficiència del codi s'adoptat la solució que els fils llegeixin les dades en blocs de N línies. A la següent pràctica se n'adoptarà una de diferent.

3 Planificació i implementació

Per planificar-vos la feina, és important entendre bé el funcionament dels fils. Es proposen els següents punts de treball:

1. Llegiu i experimenteu amb la fitxa del campus que parla de la creació de fils (document de programació amb fils, 1a part).

2. Modifiqueu el vostre codi perquè, en seleccionar el menú de creació de l'arbre, el fil principal crei només 1 únic fil secundari que processarà tot el fitxer de dades dels avions. El fil principal esperarà que el fil secundari finalitzi. Observar que encara no cal utilitzar les funcions de bloqueig *pthread_mutex_lock* i *pthread_mutex_unlock*. Només us esteu assegurant que el procediment de creació de l'arbre es realitza correctament si es fa servir un fil que no és el fil principal.

3. Llegiu i experimenteu amb les les funcions de bloqueig que s'expliquen també a la següent fitxa penjada al campus. Tingueu en compte que per a la realització d'aquesta pràctica no cal utilitzar variables condicionals. Les variables condicionals es faran servir a la següent pràctica.

4. Ara podeu crear múltiples fils secundaris (com a mínim $F = 2$) i introduir al codi les funcions de bloqueig *pthread_mutex_lock* i *pthread_mutex_unlock* necessàries per tal d'assegurar exclusió mútua. Tingueu en compte que el bloqueig s'ha d'implementar tant a nivell global (per llegir el bloc de N línies) com a nivell de node (perquè fils diferents puguin inserir-hi la informació extreta). Per inicialitzar les claus dels nodes es recomana fer servir la funció *pthread_mutex_init*.

5. Assegureu-vos que els resultats estadístics que obteniu en crear l'arbre amb múltiples fils són els mateixos que els obtinguts en crear l'arbre amb un sol fil.

4 Entrega

El fitxer que entregueu s'ha d'anomenar `P4_NomCognom1NomCognom2.tar.gz` (o `.zip`, o `.rar`, etc), on `NomCognom1` és el cognom del primer component de la parella i `NomCognom2` és el cognom del segon component de la parella de pràctiques. El fitxer pot estar comprimit amb qualsevol dels formats usuals (`tar.gz`, `zip`, `rar`, etc). Dintre d'aquest fitxer hi haurà d'haver dues carpetes: `src`, que contindrà el codi font, i `doc`, que contindrà la documentació addicional en PDF. Aquí hi ha els detalls:

- La carpeta `src` contindrà el codi font de la pràctica. S'hi han d'incloure tots els fitxers necessaris per compilar i generar l'executable. El codi ha de compilar sota Linux amb la instrucció `make`. Editeu el fitxer *Makefile* en cas que necessiteu afegir fitxers C que s'hagin de compilar.

- El directori `doc` ha de contenir un document (màxim 5 pàgines, en format PDF) explicant la discussió de les proves realitzades i els problemes obtinguts. En aquest document no s'han d'explicar en detall les funcions o variables utilitzades. És particularment interessant que feu una anàlisi del temps d'execució fent servir $F = 2$ (o 4, si teniu 4 processadors) i diferents valor d'N, la mida de bloc. A les fitxes (1a part) s'explica com mesurar avaluar el temps d'execució a les fitxes. Feu totes les proves en un únic ordinador; els resultats obtinguts dependran molt del tipus de disc (magnètic o SSD) així com el tipus de processador i altres característiques de l'ordinador. Es recomana fer les proves pel fitxer de 100.000.000 de línies (disponible al campus), atès que pel fitxer de 7.000.000 de línies el temps d'execució és relativament petit per a un sol fil (és inferior a 10 segons).

  És important que no feu servir màquines virtuals per fer aquestes proves ja que una màquina virtual acostuma a executar-se en un sol processador. A més, també és convenient que proveu el codi compilat amb opcions d'optimització: no feu servir l'opció `-g` en compilar sinó que feu servir l'opció `-O`.

A la qualificació d'aquesta pràctica, el codi tindrà un pes d'un 80% i el document i les proves el 20% restant.

5 Funcions POSIX *thread-safe*: el cas de *strtok*

Una funció que pot ser cridada per múltiples fils a la vegada es diu que és *thread-safe*, és a dir, és segura per a múltiples fils. Les funcions *thread-safe* realitzaran doncs la tasca que s'espera que facin encara que utilitzeu múltiples fils. Per exemple, les funcions estàndard d'entrada/sortida (com *fgets*) així com la funció *malloc* ho són. Però no totes les funcions que ens ofereix el sistema operatiu són *thread-safe*.

Un exemple d'una funció que no és *thread-safe* és la funció *strtok* que permet manipular cadenes. Això vol dir que hem d'anar molt amb compte en utilitzar aquesta funció en el cas d'utilitzar múltiples fils: en particular, aquesta funció utilitza una "variable interna global" que estarà compartida entre els múltiples fils, cosa que pot portar a resultats inesperats en utilitzar múltiples fils. Per evitar problemes, es recomana utilitzar una versió de *strtok* que sigui *thread-safe*. En el cas de *strtok* el manual ens indica que la funció *strtok_r* és *thread-safe*. És molt important doncs que, si esteu utilitzant la funció *strtok*, utilitzeu la funció *thread-safe* per realitzar la implementació mulfifil.

asctime	ecvt	gethostent	getutxline	putc_unlocked
basename	encrypt	getlogin	gmtime	putchar_unlocked
catgets	endgrent	getnetbyaddr	hcreate	putenv
crypt	endpwent	getnetbyname	hdestroy	pututxline
ctime	endutxent	getnetent	hsearch	rand
dbm_clearerr	fcvt	getopt	inet_ntoa	readdir
dbm_close	ftw	getprotobyname	l64a	setenv
dbm_delete	gcvt	getprotobynumber	lgamma	setgrent
dbm_error	getc_unlocked	getprotoent	lgammaf	setkey
dbm_fetch	getchar_unlocked	getpwent	lgammal	setpwent
dbm_firstkey	getdate	getpwnam	localeconv	setutxent
dbm_nextkey	getenv	getpwuid	localtime	strerror
dbm_open	getgrent	getservbyname	lrand48	strtok
dbm_store	getgrgid	getservbyport	mrand48	ttyname
dirname	getgrnam	getservent	nftw	unsetenv
dlerror	gethostbyaddr	getutxent	nl_langinfo	wcstombs
drand48	gethostbyname	getutxid	ptsname	wctomb

Figura 1: Funcions que *POSIX* no garanteix que siguin *thread-safe* (veure secció 5). Aquesta taula s'ha extret de Stevens, W.R. Advanced Programming in the Unix Environment. Addison Wesley, 2005.

Al llistat de la figura 1 es mostren les funcions que no tenen perquè ser *thread-safe*. Aquest llistat inclou només les funcions POSIX, no inclou pas informació d'altres funcions d'altres llibreries (per exemple, la Standard Template Library). Per això, abans d'utilitzar qualsevol funció amb múltiples fils, es recomana consultar el manual per saber si la funció es pot cridar de forma concurrent des de múltiples fils o bé s'ha de cridar des d'una secció crítica.