Sistemes Operatius II - Pràctica 4

Novembre del 2014

La darrera pràctica de sistemes operatius 2 se centra en l'ús de múltiples fils per incrementar l'eficiència computacional de l'aplicació. Aquesta pràctica té dues parts: la primera se centra en la utilització de les funcions de creació i bloqueig de fils mentre que la segona se centra en la utilització de variables condicionals. La data d'entrega d'aquesta pràctica és el diumenge **21 de desembre del 2014**.

Índex

1 Introducció

Les tres primeres pràctiques s'han centrat en aprendre a utilitzar diverses estructures per a la manipulació de dades, així com en l'ús de canonades per la comunicació interprocés. En aquesta darrera pràctica ens concentrem en utilitzar múltiples fils per a la creació de l'arbre. El punt de partida d'aquesta pràctica és el codi de la pràctica 3.

Recordem a grans trets el procediment per construir un arbre:

1. S'obre un fitxer de configuració. Aquest fitxer de configuració conté, a la primera línia, el nombre de fitxers a processar. La resta de les línies contenen els noms dels fitxers a processar.
2. Cada fitxer es processa analitzant les paraules i copiant-les a una estructura local.
3. Un cop finalitzat el processament de les paraules d'un fitxer es copien les paraules de l'estructura local a l'estructura global.

Aquest algorisme s'ha implementat fent servir un sol fil d'execució. L'objectiu d'aquesta pràctica se centra en modificar el codi perquè l'arbre sigui creat amb un mínim de dos fils. Si la implementació es realitza de forma correcta es podrà crear l'arbre amb tants fils com es vulguin. A les següents seccions es detallen les tasques a realitzar.

2 Part 1

La primera part utilitza funcions de creació de fils i de bloqueig. Es recomana finalitzar aquesta part abans del **dijous 4 de novembre**. Aquesta primera part s'entregarà juntament amb la segona part en finalitzar la pràctica, el **21 de desembre**.

2.1 Implementació

Per a la implementació de la solució es demana utilitzar monitors, veure fitxa 3. A continuació es descriu l'esquema a implementar (cada grup de pràctiques tindrà variacions sobre aquesta proposta):

1. En executar el vostre programa, només hi haurà un fil. Anomenarem a 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 analitzar les paraules de l'arbre.
2. En seleccionar del menú l'opció de creació d'arbre, el fil principal demanarà per teclat el fitxer de configuració. El fil principal obrirà aquest fitxer i llegirà tot el fitxer de configuració: la primera línia, que conté el nombre de fitxers a processar, així com tota la resta de línies, que contenen els noms dels fitxers a processar. Una forma senzilla d'implementar-ho és fer servir un vector per emmagatzemnar els fitxers a processar.

3. Un cop carregat a memòria la llista de fitxers a processar s'han de crear els fils – com a mínim $N = 2$ fils – amb la funció `pthread_create`. Haureu de passar a cada fil la informació necessària perquè pugui accedir als fitxers a processar i copiar les dades a l'arbre. Vegeu la fitxa 4 per veure com passar informació als fils.

4. Un cop s'han creat els fils el fil principal es quedarà esperant que els fils creats finalitzin la creació de l'arbre. Ho pot fer amb la funció `pthread_join`.

5. Cada fil creat començarà a executar. Cada fil haurà de realitzar el següent procediment iteratiu fins que tots els fitxers s'hagin processat:

   (a) Agafar el primer element no processat del vector de fitxers a processar. A l'hora d'agafar el primer element no processat heu d'anar amb compte que cap altre fil també agafi el mateix fitxer. Per això utilitzeu les funcions de bloqueig `pthread_mutex_lock` i `pthread_mutex_unlock`. No és bona idea repartir prèviament els fitxers entre fils – per exemple, un fil fa els fitxers senars i l'altre els parells – ja que es pot donar el cas que un fil acabi molt abans que un altre de processar la seva part de les dades degut a la mida de les dades a processar. A la secció d'entrega es proposa comparar el temps de còmput entre la solució proposada aquí i, una solució alternativa, en què es reparteix prèviament entre els fils (és a dir, abans de crear-los) els fitxers a processar.

   (b) El fil extraurà les paraules del fitxer i les inserirà en una estructura local. Heu de tenir en compte que cada fil tindrà la seva pròpia estructura local i, per tant, no hi haurà interferència entre fils (encara que utilitzin les mateixes funcions). Assegureu-vos que no hi ha cap variable global compartida entre els fils a l'hora d'inserir les paraules a l'estructura local.

   (c) Un cop creada l'estructura local el fil haurà de copiar les dades de l'estructura local a la global. Tingueu en compte que heu d'evitar deixar l'arbre en un estat inconsistent degut a la manipulació per múltiples fils. Per tal d'assegurar això s'han d'utilitzar les funcions de bloqueig `pthread_mutex_lock` i `pthread_mutex_unlock`. Hi ha diverses formes d'utilitzar aquestes funcions per aconseguir consistència de l'arbre: la primera solució es basa en utilitzar aquestes dues funcions per protegir (tota) la funció que copia l'estructura local a l'arbre. La segona solució es basa en utilitzar les dues funcions per protegir realment el que cal protegir: les funcions de manipulació d'arbre (inserció d'un element a l'arbre, cercar un element de l'arbre, etc.).

   (d) Un cop copiada l'estructura local a l'arbre cal alliberar l'estructura local i agafar el següent element no processat del vector de fitxers a processar, el pas (a). En cas que no hi hagi cap més fitxer a processar el fil acabarà (i sortirà doncs de la seva funció d'entrada).

6. Un cop hagin finalitzat tots els fils que processaven els fitxers de text, el fil principal es despertarà (estava esperant a `pthread_join`) i tornarà a visualitzar el menú.

2.2 Planificació

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

1. Modifiqueu el vostre codi perquè en una una única funció que creï l'arbre (és a dir, el procediment descrit al punt 5 de la secció 2.1). Encara no cal que es creïn els fils sinó només esteu preparant el codi per fer-lo multil. Tampoc cal que feu servir, de moment, funcions de bloqueig.

   Es proposa doncs que modifiqueu l'aplicació perquè en obrir el fitxer de configuració es llegeixi tot el fitxer i s'emmagatzemi en un vector els noms dels fitxers a processar. Escriviu una altra funció, separada de l'anterior, que permeti a) Agafar el primer element no processat del vector , b) Extreure les paraules del fitxer i inserir-les en una estructura local, c) Copiar l'estructura local a la global, d) Alliberar estructura local i tornar al primer pas. Comproveu que el codi funciona correctament abans de passar al següent punt. Tingueu en compte que la funció *strtok* (en cas que l'utilitzeu a la vostra implementació) no és *thread-safe*, és a dir, no funcionarà com s'espera en el cas d'utilitzar múltiples fils. Per a mes informació mireu la secció 5.

2. Llegiu i experimenteu amb la fitxa 4 al campus que parla de la creació de fils.

3. Modifiqueu el codi perquè, en seleccionar el menú de creació de l'arbre, el fil principal crei només 1 sol fil que realitzarà la creació de l'arbre (executant la funció esmentada al punt 1 d'aquesta llista). 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.

4. Llegiu i experimenteu amb les les funcions de bloqueig que s'expliquen a la fitxa 5 penjada al campus. Tingueu en compte que per a la realització de la part 1 no cal utilitzar variables condicionals. Les variables condicionals s'utilitzaran a la segona part de la pràctica.

5. Ara podeu crear múltiples fils (com a mínim $N = 2$) i introduir al codi les funcions de bloqueig `pthread_mutex_lock` i `pthread_mutex_unlock` necessàries per tal d'assegurar exclusió mútua entre els múltiples fils. Un cop creats, el fil principal ha esperar que aquests finalitzin.

3 Part 2

Aquesta segona part se centra en la utilització combinada de funcions de bloqueig i variables condicionals. Tingueu en compte que la data d'entrega final és el **21 de desembre del 2014**. Per fer aquesta part cal tenir finalitzada la part 1 de la pràctica.

3.1 Implementació

A continuació es descriu l'esquema a implementar (cada grup de pràctiques tindrà variacions sobre aquesta proposta):

1. En seleccionar del menú l'opció de creació d'arbre el fil principal demanarà per teclat el fitxer de configuració i el llegirà emmagatzemant els noms de fitxer en un vector, igual que abans. El fil principal crearà ara 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. A continuació es donen els detalls.

2. Tal com s'ha comentat al punt anterior, N dels fils creats estaran especialitzats en a) agafar un fitxer del vector fitxers a processar, b) extreure les paraules i inserir-les en una estructura local, i c) inserir (el punter a) a l'estructura local en un *buffer* circular de mida M compartit entre tots els fils. Anomenarem a aquest fils "fils processadors". Per defecte, $M = N$, és a dir, que la mida del *buffer* circular és igual al nombre de fils que existeixen. Un cop el fil ha inserit (el punter a) l'estructura local al *buffer* circular, el fil pot passar a processar el següent fitxer sense necessitat d'esperar que s'hagin copiat les dades a l'estructura global. Cal assegurar, però, que a la nova iteració es faci servir una nova estructura local per emmagatzemar les paraules extretes. En cas que el *buffer* cicular sigui ple, el fil processador, en voler inserir (el punter) a l'estructura local al *buffer*, s'haurà d'esperar amb un `pthread_cond_wait`. Qui el despertarà? Resposta: el fil especialitzat en copiar l'estructura local a la global (vegeu següent punt).

3. Tal com es comenta al punt 1 hi haurà un fil especialitzat en copiar les dades de les estructures locals del *buffer* circular a l'estructura global. Anomenarem a aquest fil "fil copiador". En cas que no hi hagi cap element al *buffer* circular el fil copiador es quedarà dormint amb un `pthread_cond_wait`. El fil copiador es despertarà per un fil processador, amb un `pthread_cond_signal`, en el moment en què aquest insereixi una estructura local al *buffer*. De la mateixa forma, quan fil el fil copiador agafi un element del *buffer* circular per copiar-lo a l'estructura global podrà despertar, amb un `pthread_cond_broadcast`, als fils processadors que estiguin esperant per inserir elements.

3.2 Planificació

Aquí hi ha unes recomanacions per planificar-vos la feina:

1. Per fer aquesta part és important entendre bé el funcionament de les funcions `pthread_cond_wait` i `pthread_cond_broadcast`. Llegiu i experimenteu doncs amb la segona part de la fitxa 5 penjada al campus que parla de les variables condicionals.

2. Quin és el paradigma que s'ha d'utilitzar per implementar la proposta de la secció 3.1? Una barrera? Un esquema de productors i consumidors? Lectors i escriptors, potser? Abans de continuar programant heu de tenir clar quin paradigma heu d'utilitzar. Un cop tingueu clar quin és el paradigma a implementar utilitzeu l'esquema corresponent de les transparències de teoria.

3. ~~Tingueu en compte que el fil copiador ha de posar-se a dormir a l'inici (ja que no hi haurà cap element al buffer circular). Sou capaços de fer un programa que asseguri que aquest fil està dormint abans que qualsevol dels fils processadors comenci a inserir les estructures locals al buffer cicular? Tingueu en compte que els fitxers de text a processar podrien ser molt i molt petits (amb una paraula a processar, per exemple) i per tant no és adequat tenir l'esperança que els fils processadors trigaran una mica en començar a inserir elements al buffer — el temps necessari perquè el fil copiador pugui posar-se a dormir. No, es tracta d'assegurar al 100% que el fil copiador estarà dormint! Per què és important això? Podeu trobar algun cas en què el programa es quedi "penjat" (i.e. tots els $N + 1$ fils dormint) esperant que algun altre fil els desperti? Per ajudar-vos penseu en el cas $N = M = 1$.~~

   ~~Una forma d'aconseguir solucionar el problema anterior (és a dir, que el fil copiador s'adormi abans que els fils processadors puguin posar un element al buffer cicular) és que el fil principal creï el fil copiador abans que la resta i, que el fil principal s'esperi a un `pthread_cond_wait` que el fil copiador s'adormi. Just abans de posar-se a dormir el fil copiador senyalitzarà al fil principal, amb un `pthread_cond_signal`, que pot continuar l'execució. Quin és l'algorisme que heu de fer servir per implementar això? Una barrera!~~

4. Recordeu que el fil principal s'haurà d'esperar a un `pthread_cond_join` perquè els fils finalitzin d'executar.

4 Entrega

El fitxer que entregueu s'ha d'anomenar `P4_Cognom1Cognom2.tar.gz` (o `.zip`, o `.rar`, etc), on `Cognom1` és el cognom del primer component de la parella i `Cognom2` 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 tres carpetes: `src`, que contindrà el codi font, `proves`, que contindrà resultats d'execució i `doc`, que contindrà la documentació addicional en PDF. Aquí hi ha els detalls per cada directori:

- La carpeta `src` contindrà el codi font de les dues parts per separat (per exemple, en subdirectoris `part1` i `part2`). 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. Es necessari comentar com a mínim les funcions que hi ha al codi.
- La carpeta `proves` ha d'incloure qualsevol informació que considereu rellevant. Aquesta informació haurà d'estar comentada a la memòria.
- El directori `doc` ha de contenir un document (dues o tres pàgines, en format PDF) explicant el funcionament de l'aplicació, 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, però, que comenteu sobre els següents aspectes de la part 1
  - Una comparativa sobre el temps d'execució fent servir 1 sol fil o 2 (o més). A la fitxa 4 teniu exemples que mostren com es pot mesurar el temps d'execució d'una funció.
  - Quina solució d'exclusió mútua és més eficient? És millor protegir tota la funció que copia l'estructura local a l'arbre o és millor protegir les funcions que permeten manipular els arbres?
  - Part opcional: recordeu que a la secció 2.2, punt 5(a), es comenta que "No és bona idea repartir prèviament els fitxers entre fils – per exemple, un fil fa els fitxers senars i l'altre els parells – ja que es pot donar el cas que un fil acabi molt abans que un altre de processar la seva part de les dades degut a la mida de les dades a processar." Es proposa doncs que comproveu aquesta afirmació. En particular, es proposa que compareu la vostra solució (en què cada fil agafa un nou element del vector cada cop que ha acabat amb l'anterior) amb la solució en què els fils es reparteixen prèviament els fitxers de configuració entre ells (per exemple, per $N = 2$, un fil processa els fitxers d'índexs senars i l'altre processa els fitxers d'índexs parells).

  Pel que fa la segona part és interessant que comenteu sobre els següents punts
  - Una explicació breu del paradigma utilitzat per implementar la comunicació entre els fills que processen els fitxers i el fil copiador.
  - Hi ha alguna influència del valor de M (la mida del buffer circular) en el temps computacional total?

La data límit d'entrega d'aquesta pràctica és el **21 de desembre del 2014**. El codi tindrà un pes d'un 80% (40% part 1, 40% part 2) i el document i les proves el 20% restant.

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.

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 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 (veure fitxa 2) així com la funció *malloc* ho són. Però no totes les funcions que ens ofereix el sistema operatiu són *thread-safe*. Un exemple 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. O bé podem 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.

El següent llistat mostra 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.