Sistemes Operatius II - Pràctica 4

Novembre del 2017

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 següent i 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, així com en l'ús de canonades per la comunicació interprocés.

En aquesta pràctica ens concentrarem en utilitzar múltiples fils per al processament dels fitxers PDF. 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.

2 Funcionalitat a implementar

Per a la implementació de la solució es demana utilitzar monitors[1]. 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 que conté el llistat de fitxers PDF a processar. El fil principal crearà, amb la funció `pthread_create`, un mínim de $F = 2$ **fils secundaris** els quals hauran de repartir-se entre sí els fitxers PDF a processar i crearan l'arbre. Mentrestant, el fil principal es quedarà esperant, amb la funció `pthread_join`, que els fils secundaris acabin la seva feina.

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

Es proposa a continuació una forma de procedir. Podeu introduir-hi variacions si ho creieu convenient.

El fil principal, abans de crear els fils secundaris, obrirà el fitxer que conté els fitxers PDF a processar i en llegirà la primera línia, que conté el nombre de fitxers PDF a processar. En aquest moment podeu optar per dues solucions: 1) el fil principal llegeix el llistat de tots els fitxers a processar, els emmagatzema en un vector i "passa" el vector als fils secundaris, 2) el fil principal només "passa" als fils secundaris el punter al fitxer obert, deixant que els fils secundaris hagin de llegir el fitxer a processar directament de fitxer. En previsió de la pràctica 5, es recomana implementar l'opció 1.

Suposem que s'han creat els fils secundaris. Cada fil secundari realitzarà un bucle amb les següents tasques:

1. El fil secundari obté un nom de fitxer PDF a processar (sigui del vector o directament del fitxer, segons s'hagi implementat 1 o 2). Heu d'anar amb compte que cap altre fil agafi el mateix fitxer. Per això utilitzeu les funcions de bloqueig `pthread_mutex_lock` i `pthread_mutex_unlock`.

[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.

2. El fil secundari farà extraurà les paraules del PDF i les inserirà en un **arbre local**. Cada fil tindrà un arbre local propi que es farà servir per inserir-hi les paraules que aquell fil extregui del PDF. Cada fil secundari només podrà accedir al seu arbre local i no podrà accedir als arbres locals dels altres fils secundaris. El fet de servir un arbre local fa que no hi hagi interferència entre el fils mentre processen el fitxer PDF. No caldrà doncs aplicar cap mena de bloqueig entre els fils mentre processin el fitxer PDF.

3. El fil secundari, en acabar de processar el seu fitxer PDF, copiarà les dades de l'arbre local a l'**arbre global**, l'arbre que conté la informació obtinguda de tots els fils secundaris. Només hi ha un arbre global i està compartit entre tots els fils secundaris. Atès que es poden produir problemes si múltiples fils accedeixen al mateix temps (per escriptura) a l'arbre, cal d'utilitzar les funcions de bloqueig `pthread_mutex_lock` i `pthread_mutex_unlock` en el moment de copiar la informació de l'arbre local a l'arbre global.

4. Un cop el fil secundari ha copiat les dades de l'arbre local a l'arbre global, caldrà alliberar l'arbre local i tornar al pas 1. El fil secundari finalitzarà en cas que no hi hagi cap més PDF a processar.

Tingueu en compte que haureu de "passar" als fils secundaris la informació necessària perquè puguin accedir al fitxers a processar i copiar les dades a l'arbre. Penseu doncs quina és la informació que voleu passar a cada fil secundari: el punter al fitxer de vols que heu obert? l'arbre? altra informació? Llegiu la secció 3 per a més informació al respecte.

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 4 al campus que parla de la creació de fils.

2. Modifiqueu el 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 que conté els fitxers PDF a analitzar. El fil principal esperarà que el fil secundari finalitzi. El fil secundari farà servir un arbre local a l'hora d'extreure les paraules del PDF i copiarà les dades de l'arbre local al global cada cop que finalitzi de processar un PDF. 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. Per copiar les dades de l'arbre local a l'arbre global utilitzeu com a referència la funció que heu fet servir per emmagatzemar dades

3. 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 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.

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. Es necessari comentar com a mínim les funcions que hi ha al codi.

- El directori `doc` ha de contenir un document (màxim 5 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 feu una comparativa sobre el temps d'execució fent servir sol fil o $F = 2$ (o més). 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).

  A la fitxa 4 teniu exemples que mostren com es pot mesurar el temps d'execució d'una funció. Una forma senzilla de mesurar el temps d'excució és aquesta: primer, genereu un fitxer de text `opcions.txt` que tingui aquest contingut

  ```
  1
  fitxers_pdf.txt
  5
  ```

  Aleshores executeu el vostre codi amb

  ```
  time ./main < opcions.txt
  ```

  Amb aquesta instrucció executeu la vostra aplicació fent que l'entrada estàndard sigui el fitxer `opcions.txt`.

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 (veure fitxa 2) així com la funció *malloc* ho són. Però no totes les funcions que ens ofereix el sistema

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.

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: 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.

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.