Programació productor-consumidor

Novembre 2021

Índex

1 Introducció

L'objectiu d'aquesta pràctica se centra en utilitzar el paradigma del productor-consumidor de la programació concurrent per realitzar la implementació de la funció *getMoviesFromFile* que es troba al fitxer `CSVReader.c`. Aquesta funció és la que extreu del fitxer CSV la informació de les pel·lícules que els usuaris han valorat. Aquesta funció realitza, al seu interior, diverses operacions que són costoses i que poden ser implementades de forma concurrent mitjançant el paradigma del productor-consumidor.

La raó per la qual s'ha escollit aquesta funció és el fet que el *callgrind* (l'eina que vau fer servir a la pràctica 3) indica que la funció *getMoviesFromFile* ocupa aproximadament un 27% del temps CPU del total l'aplicació. N'hi ha una altra que ocupa més; en particular la funció *fillRecommendationMatrix,* aproximadament un 60%. Aquesta darrera funció pot ser implementada de forma concurrent utilitzant el paral·lelisme per descomposició en les dades que és la tècnica que s'ha fet servir a la pràctica 4. És per això que s'ha escollit *getMoviesFromFile* per aquesta pràctica atès que necessita fer servir un altre paradigma per poder-la imlementar.

A la figura 1 es pot veure l'esquema gràfic – generat amb el *callgrind* — de les funcions que crida *getMoviesFromFile*. Com es pot veure, les funcions que més temps de CPU necessiten són *fgets* (5.31%), *getRating* (15.71%) i *lineIsMovie* (4.08%). Dintre de la funció *getRating* les dues funcions que més temps necessiten són *atoi* (9.84%) i *strtok* (5.01%).

Ha de quedar clar, però, que l'objectiu d'aquesta pràctica no és implementar la funció *getMoviesFromFile* de forma més eficient que l'original amb el paradigma del productor-consumidor. L'objectiu principal és implementar aquesta funció fent servir l'esquema de productor-consumidor. El fet que s'aconsegueixi reduir el temps d'execució no es prioritari.

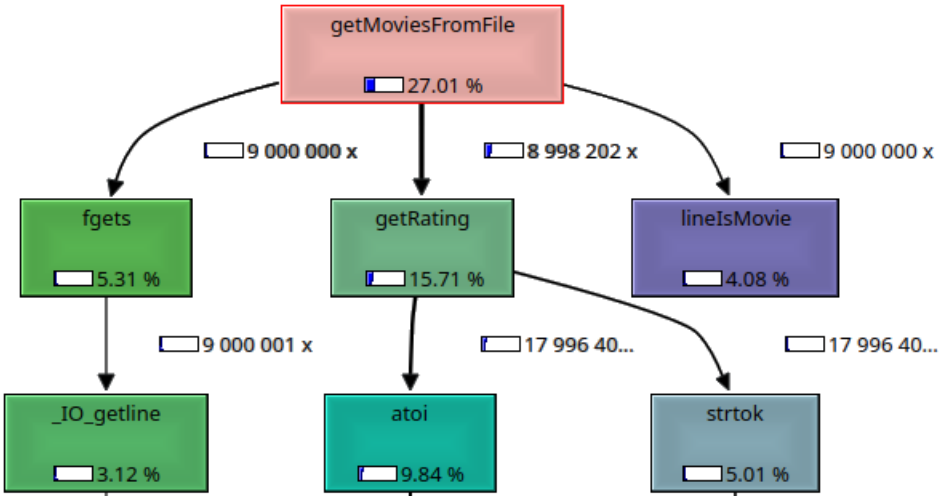

Figura 1: Eficiència de CPU de les funcions que crida *getMoviesFromFile*. Només es mostren aquelles funcions que ocupen un mínim d'un 1% de CPU (valor per defecte en fer la gràfica).

2 Algorisme a implementar

L'objectiu de la pràctica es realitzar una implementació concurrent de la funció *getMoviesFromFile* mitjançant el paradigma del productor-consumidor, veure figura 2. La comunicació entre el productor i el consumidor es realitzarà amb un búffer que té B cel·les. Cadascuna de les cel·les del dibuix – tant al búffer, al productor com al consumidor – és un vector de parells d'elements [*movieId*, *line*] de mida N. En aquesta pràctica es consideren dos casos que tindran una puntuació diferent: per a $B = 1$ es podrà obtenir una qualificació de 10, mentre que la implementació del cas general (qualsevol valor de $B \geq 1$) permet obtenir una qualificació de 12.

Si analitzeu l'estructura del fitxer de dades CSV veureu que és un fitxer de text en què un identificador de pel·lícula (*movieId*) és un sencer seguit del símbol ":" (e.g. "1:", "1234:"). El fitxer CSV comença amb un identificador de pel·lícula (en particular, "1:", amb la qual cosa *movieId* té el valor 1). Després de l'identificador de pel·lícula les línies que segueixen contenen els usuaris i les valoracions associades a aquella pel·lícula. En arribar un nou identificador de pel·lícula, se sap que ja s'han llegit totes les valoracions d'aquella pel·lícula i s'extreu el nou identificador de pel·lícula. Després hi haurà la informació dels usuaris amb les valoracions associades a l'identificador de pel·lícula llegit, i així successivament. Se suposa desconegut el nombre de pel·lícules que hi ha al fitxer CSV.

Atesa la forma amb què es llegeix el fitxer al codi original es proposa realitzar la implementació del productor-consumidor fent servir un únic productor i un únic consumidor. El fet que només es pugui fer servir un productor i un consumidor ve donat per l'estructura del fitxer. Hem d'assegurar d'alguna forma que les dades es processaran de forma seqüencial pel consumidor.

- Al productor es realitzarà la lectura de dades del fitxer (*fgets*) i es comprovarà si la línia es un identificador de pel·lícula (*lineIsMovie)*. Si és un identificador de pel·lícula, pot extreure l'identificador de pel·lícula (*getMovieId*). Les línies que es llegiran a continuació són les valoracions associades a la pel·lícula *movieId*. A partir d'aquesta informació el productor podrà construir les cel·les. Cada cel·la emmagatzema N parells d'elements [*movieId*, *line*], veure

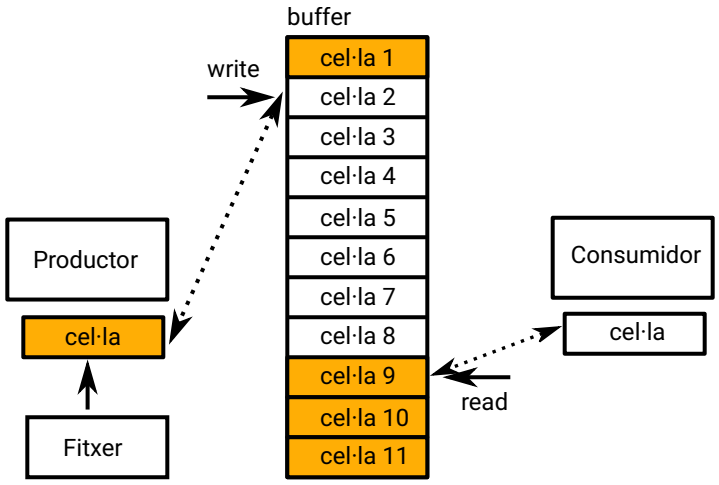

Figura 2: Esquema del productor-consumidor a implementar. Hi ha un únic productor i un únic consumidor. El búffer té B cel·les. Cadascuna de les cel·les emmagatzema N parells d'elements [*movieId*, *line*]. Veure text per a més detalls.

figura 2, on *line* és una línia del fitxer CSV que conté la informació d'un usuari i la valoració corresponent associada a *movieId*. Quan el productor hagi llegit tots els elements per omplir una cel·la, la col·loca al búffer compartit (sempre que hi hagi espai lliure disponible).

- Al consumidor es processaran cadascuna de les cel·les que el productor ha col·locat al búffer. Donada una cel·la es processaran els parells [*movieId*, *line*]: s'extraurà de *line* la informació de l'usuari i valoració (*getRating*) sabent que té associada la pel·lícula amb identificador *movieId*. Quan el valor de *movieId* canviï de valor, caldrà afegir les valoracions realitzades tal com es fa al codi original (*addRatingsToMovie* i *addMovieToCurrentMovies*).

La mida de la cel·la, N, és un valor constant que no depèn del nombre de valoracions que els usuaris hagin fet d'una determinada pel·lícula. Amb això s'aconsegueix que el consumidor pugui extreure informació d'una cel·la mentre el productor llegeix informació del disc. Observar doncs que una cel·la pot contenir parells d'elements de pel·lícules diferents.

Mitjançant aquest algorisme es pot repartir el processament de les funcions més costoses a nivell de CPU entre dues CPUs. S'insisteix, però, que l'objectiu d'aquesta pràctica no és implementar la funció *getMoviesFromFile* de forma més eficient que l'original amb el paradigma del productor-consumidor.

3 Implementació de l'algorisme

Per a la implementació d'aquesta pràctica s'hauran de fer servir monitors: s'utilitzaran claus de bloqueig (mutex) així com variables condicionals. Les claus de mutex i les variables condicionals poden ser variables globals al codi. Les seccions crítiques hauran de tenir un únic punt d'entrada i un únic punt de sortida.

Es demana que totes les dades es passin com a argument als fils secundaris que es creïn. A la solució implementada pel professorat el productor és el fil principal i el consumidor un fil secundari.

Observar que el consumidor obtindrà el resultat del processament. En cas que el consumidor sigui un fil secundari, es demana que el resultat sigui "retornat" sense fer servir variables globals. Sou lliures d'implementar l'estructura de fils que desitgeu.

Com s'ha comentat a la secció 2, a l'hora de qualificar la pràctica es consideren dos casos:

- Cas $B = 1$. En aquest cas el búffer emmagatzema només una cel·la. Disposeu d'un l'exemple de la implementació del productor-consumidor amb un búffer de mida $B = 1$ a "Programació amb fils (2a part)". L'exemple mostrat és per a transferir un sencer i s'haurà d'adaptar per al cas d'aquesta pràctica. La implementació d'aquest cas permet obtenir una qualificació de 10.

- Cas general $B \geq 1$. Es tracta d'implementar el cas general mostrat a la figura 2. A nivell de concurrència una de les avantatges que pot aportar implementar un búffer de mida B és el fet que el productor pot anar col·locant les cel·les al búffer mentre que el consumidor les agafa i les processa. Per al cas de $B = 1$ el productor ha d'esperar que el consumidor agafi la cel·la del búffer per poder-hi col·locar una cel·la. La implementació d'aquest cas permet obtenir una qualificació de 12.

A la secció 3.2 es comenta una forma per transferir de forma eficient una cel·la del productor al buffer compartit així com que el consumidor agafi una cel·la d'aquest perquè pugui ser processada. La transferència de dades entre el productor-consumidor i el búffer ha se ser eficient per assegurar que no es facin operacions innecessàries.

En acabar la implementació de l'algorisme, assegureu-vos que funciona correctament comprovant que l'execució dels exemples del fitxer `Readme.txt` donen el mateix resultat que el codi original. Assegureu-vos també que s'ha alliberat tota la memòria dinàmica que s'hagi fet servir. Vegeu la secció 3.3.

3.1 Codi thread-safe

El codi multifil a implementar no és *thread-safe*, és a dir, no totes les funcions permeten un accés concurrent. La funció que no és *thread-safe* és *strtok*, una funció de la llibreria d'usuari que es fa servir per extreure subcadenes d'una cadena original. Aquesta funció es fa servir a les funcions *getMovieId* i *getRating* que són utilitzades al productor i consumidor respectivament. El fet que dos fils diferents facin crides a *strtok* pot fer que l'aplicació peti sense motiu aparent. La raó és el fet que *strtok* no està preparada per poder ser cridada des de diferents fils al mateix temps.

La solució a aquest problema la proporciona el manual de la funció *strtok*. En executar "man strtok" veureu que la funció *strtok_r* ho és. Amb l'enunciat d'aquesta pràctica disposeu de les funcions *getMovieId* i *getRating* reimplementades amb la funció *strtok_r* amb la qual cosa assegurem que no hi haurà problemes de concurrència. Es demana doncs utilitzar aquestes funcions per realitzar la implementació. Si ho desitgeu, també podeu fer una implementació alternativa a la proposada.

3.2 Implementació eficient de la transferència de cel·les

Un aspecte important a considerar en implementar el codi és com realitzar la transferència de la cel·la del productor al búffer i del búffer al consumidor. Aquesta transferència ha de ser el més ràpid possible evitant operacions innecessàries. La pregunta que ens podem fer ara és com fer la transferència de forma eficient.

1. Una forma de procedir per fer la transferència de la informació és "copiar" la informació entre les cel·les. En particular, pel cas del productor, es tracta de copiar els N elements de la cel·la del productor a la cel·la del búffer. Per fer això caldrà fer servir funcions com *strcpy*, funcions que permeten copiar cadenes de caràcters. El fet de fer-ho portarà a una solució que serà força ineficient. Podeu imaginar per què serà així?

2. Una altra forma de procedir és "jugar" amb els punters en C. Pel cas d'altres llenguatges com Java o Python es podrà procedir de forma equivalent. L'objectiu aquí és "intercanviar" la cel·la que té el productor amb la cel·la que hi ha al búffer. A l'exemple que es mostra a continuació s'intercanvien les dues cel·les:

   ```
   cell *cell_producer, *cell_buffer, *tmp;
   // Suposem s'ha reservat memoria per a cell_producer i cell_buffer
   // Suposem que ara es llegeixen dades i s'omple el cell_producer
   tmp = cell_buffer;
   cell_buffer = cell_producer;
   cell_producer = tmp;
   ```

   El que estem fent es intercanviar la cel·la del productor amb la del búffer (suposant que la cel·la del búffer no conté dades o bé aquestes ja han sigut processades pel consumidor). El consumidor procedirà de forma similar: suposem que el consumidor ha acabat de processar les seves dades i vol agafar la següent cel·la a processar. La idea és intercanviar la cel·la de la qual disposa el consumidor amb la cel·la que hi ha al búffer (suposant que aquesta conté dades que han sigut col·locades al búffer pel productor). En poques instruccions es pot fer!

   Observar que en fer aquest darrer intercanvi (entre el consumidor i el búffer) estem posant una cel·la amb dades ja processades al búffer. No passa res, atès que el productor agafarà la cel·la amb dades ja processades i la sobreescriurà amb les dades que llegeixi del fitxer. Un cop el productor hagi sobreescrit la cel·la amb les dades del fitxer, tornarà a fer l'intercanvi com es mostra a l'exemple. No fa falta esborrar doncs la cel·la si conté dades processades.

3.3 Comprovació del codi amb valgrind

És important comprovar el bon funcionament del codi. Per tal de fer-ho cal, d'una banda, que els exemples del fitxer `Readme.txt` donin el mateix resultat que el codi original. També és important comprovar amb el `valgrind` que el codi funciona correctament. És convenient esmentar que `valgrind` té una arquitectura interna de màquina virtual amb un únic processador. L'eina `valgrind`, per tant, no simula un sistema multiprocessador. Aquesta eina pot ser útil, però, per detectar (alguns) accessos invàlids a memòria així com la memòria no alliberada.

Per comprovar el codi implementat de forma ràpida i no haver d'esperar gaire que l'execució acabi es proposa a) Comentar a la funció *main* la crida a la funció *fillRecommendationMatrix*, b) Comentar les crides a *fwrite* tal com es fa a la pràctica 3. Amb aquestes dues modificacions el temps d'execució de l'aplicació es redueix força i es pot analitzar si hi ha algun problema a la funció *getMoviesFromFile*. Serà útil, per exemple, per veure si s'allibera tota la memòria dinàmica reservada a la funció *getMoviesFromFile* així com altres problemes d'accés invàlids que `valgrind` detecti. S'insisteix, però, que *valgrind* no servirà per detectar tots els problemes que hi pugui haver (atès que simula un únic processador).

4 Entrega de la pràctica

Es demana entregar un fitxer ZIP que inclogui el codi font així com un informe. Atès que el directori `data` pesa molt es demana no incloure aquest directori en entregar la pràctica. El nom del fitxer ha de tenir l'estructura **P5_XX_nom1_cognom1_nom2_cognom2.zip** on XX és l'identificador de grup de parella. En avaluar el codi font tindrà un pes d'un 70% i l'informe un 30%.

Informe L'informe a entregar ha d'estar en format `PDF` o equivalent (no s'admeten formats com `odt`, `docx`, ...). L'informe pot tenir una llargada de fins a 4 (per a l'opció $B = 1$) o bé 5 pàgines (per a l'opció $B \geq 1$). L'informe s'avaluarà amb els següents pesos: proves realitzades i comentaris associats, un 60%; escriptura sense faltes d'ortografia i/o expressió, un 20%; paginació del document feta de forma neta i uniforme, un 20%.

A l'informe es demana indicar clarament quina de les dues opcions s'ha implementat, $B = 1$ o $B \geq 1$. A més, es demana incloure a) L'esquema de fils utilitzat per implementar la proposta de productor-consumidor, b) Els temps d'execució associats a la crida a la funció *getMoviesFromFile* per a diferents valors d'N així com diferents valors de B (només en cas que s'implementi el cas general $B \geq 1$). Per tal de mesurar els temps d'execució no serà útil utilitzar la comanda `time` tal com s'ha fet a les pràctiques anteriors atès que l'objectiu és mesurar només el temps de la funció esmentada. Es demana utilitzar les funcions comentades a la secció 2.5 (Computar el temps d'execució d'un programa) del document "Programació amb fils (1a part)", c) Els resultats de les proves realitzades amb `valgrind` amb el codi finalitzat. No és necessari comentar els problemes que s'han trobat a mesura que s'ha desenvolupat l'aplicació.

Un informe està estructurat generalment en una introducció, treball realitzat i conclusions. A la part d'introducció es descriu breument el problema solucionat (i.e. resum del que proposa en aquesta pràctica). A la part de del treball realitzat es responen les preguntes realitzades al la pràctica i es descriuen les proves que s'han realitzat. Per respondre les preguntes podeu seguir un fil conductor que us permeti descriure el treball realitzat. També podeu respondre de forma explícita a cadascuna de les preguntes (en aquest cas indiqueu clarament a quina pregunta esteu responent en cada moment). Sigueu breus i clars en els comentaris i experiments realitzats, no cal que us esteneu en el text. Finalment, a la part de conclusions es descriuen unes conclusions tècniques de les proves realitzades. Per acabar, es poden incloure conclusions personals.

Inclogueu, preferentment en format text, les comandes que heu executat i algun comentari breu descrivint el resultat si ho creieu necessari. En cas que preferiu incloure captures de pantalla en comptes d'incloure el resultat en format text, assegureu-vos que el text de la captura es pot llegir bé (és a dir, que tingui una mida similar a la resta del text del document) i que totes les captures siguin uniformes (és a dir, que totes les captures tinguin la mateixa mida de text).

Codi Respecte el codi, es demana que funcions estiguin comentades, codi modular i net, bon estil de programació, que el programa funcioni correctament (i.e. s'executaran els exemples mostrats al `Readme.txt`), tota la memòria ha de ser alliberada i sense accessos invàlids a memòria (i.e. s'executarà el `valgrind` tal com s'indica a la secció 3.3). Les seccions crítiques hauran de tenir un únic punt d'entrada i un únic punt de sortida. Es demana que totes les dades, llevat de les claus de mutex i variables condicionals, es passin com a argument als fils secundaris que es creïn. La transferència d'una cel·la del productor al búffer o del búffer al consumidor s'haurà d'implementar de forma eficient (i.e. evitant copiar dades de forma innecessària).