Programació multifil: paradigma productor-consumidor

Novembre 2022

1 Introducció

La darrera pràctica s'ha centrat en el processament de l'arxiu fent servir múltiples fils. Cada fil secundari llegeix de disc un bloc de N línies seguides del fitxer per extreure'n després la informació i així actualitzar la matriu `num_flights`. En aquesta nova pràctica es realitzarà una implementació multifil fent servir **algorisme productor-consumidor**. Amb aquest paradigma aconseguirem separar, en fils diferents, la tasca de la lectura de dades de disc de l'extracció de la informació de les dades.

La implementació de l'algorisme es realitzarà amb **monitors**. Per implementar l'algorisme del productor-consumidor es farà servir doncs les **variables condicionals dels monitors**. 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. Les funcions que es mencionen en aquest document es trobareu descrites també a les dues fitxes que teniu disponibles al campus virtual.

2 Funcionalitat a implementar

Per descriure la funcionalitat a implementar s'utilitzen els noms de les funcions del codi base.

En començar a executar el vostre programa només hi haurà un fil. Anomenarem aquest fil el **fil principal**. El fil principal llegirà els codis dels aeroports, funció `read_airports`. A continuació es crida a la funció `read_airports_data`, que és la que llegeix les dades del fitxer de vols.

L'objectiu d'aquesta pràctica és fer servir el **paradigma del productor-consumidor** per augmentar l'eficiència del processament de les dades dels vols. En particular, el fil principal serà el productor, l'encarregat de llegir les línies del fitxer en blocs de N línies. Els **fils secundaris**, creats pel fil principal, seran els consumidors, és a dir, els encarregats d'extreure la informació dels blocs de dades que ha llegit el productor i actualitzar la matriu `num_flights`.

A continuació es descriu l'esquema a implementar, el qual inclou una descripció de l'esquema del productor-consumidor a implementar. A la secció de planificació 3 es proposa com procedir per arribar a implementar aquest esquema:

1. El fil principal crida a la funció `read_airports_data`. El fil principal, abans de crear els fils secundaris, haurà de crear el buffer de comunicació entre el fil principal i els fils secundaris (vegeu el tema de "Introducció a la concurrència"). El buffer de comunicació emmagatzemarà els blocs de N línies que el fil principal llegeixi perquè els fils secundaris hi puguin accedir i processar-los. El buffer de comunicació tindrà una mida fixa, és a dir, podrà emmagatzemar B blocs de N línies.

2. A continuació el fil principal, abans d'obrir el fitxer a processar, crearà, amb la funció *pthread_create*, $F = 2$ fils secundaris. En crear els fils secundaris el fil principal passarà als fils secundaris, preferentment per argument (i no per variables globals), totes les variables necessàries que els fils secundaris necessiten per processar les dades del fitxer.

3. El fil principal obrirà el fitxer a processar: en llegirà la capçalera, la qual descartarà, i a continuació llegirà el fitxer de dades en blocs de N línies. Cada bloc de N línies serà "transferit" al buffer de comunicació. Observar que si el buffer de comunicació s'omple amb B blocs, el fil principal s'haurà d'adormir a una variable condicional per esperar que els fils secundaris agafin blocs per processar. Quan un consumidor agafi un bloc del buffer despertarà al productor per avisar que pot introduir un bloc al buffer.

4. Els $F = 2$ fils secundaris seran els encarregats de processar les dades que s'hagin introduït al buffer. Si hi han blocs a processar al buffer de comunicació, el fil en podrà agafar un (que encara no s'hagi processar). Un cop agafat, el fil secundari extraurà les dades del bloc i actualitzarà la matriu `num_flights`. Un cop s'hagi processat el bloc, el fil tornarà a agafar un altre bloc del buffer de comunicació. En cas que no hi hagi cap bloc a processar, el(s) fil(s) consumidor s'adormirà a una variable condicional per esperar que el productor insereixi un nou bloc al buffer de comunicació. Així que el productor introdueixi un nou bloc al buffer, aquest despertarà als fils consumidors per avisar-los que hi ha blocs a processar al buffer.

5. Quan el fil principal hagi llegit tot el fitxer i hagi "transferit" els blocs al buffer de comunicació, tancarà el fitxer i es quedarà esperant, amb la funció *pthread_join*, que els fils secundaris acabin de processar els blocs que hi pugui haver al buffer d'intercomunicació.

6. Un cop hagin finalitzat tots els fils secundaris amb la seva feina, el fil principal es despertarà del *pthread_join*, sortirà de l'aplicació imprimint per pantalla els resultats de l'anàlisi del fitxer.

Observar que s'implementa l'esquema del productor-consumidor. En concret,

1. El productor, el fil primari, és l'únic fil encarregat de llegir el fitxer de dades. Llegirà el fitxer de dades en blocs de N línies i "transferirà" cada bloc al buffer que comparteixen productor i consumidors. El buffer tindrà una mida per poder emmagatzemar B blocs, i es recomana que B sigui igual o superior al nombre F de fils secundaris. Teniu alguna noció de perquè ha de ser així? Podeu fer algun experiment per veure què passa si agafeu, per exemple, $B = 1$ amb $F = 2$ fils?

2. Els consumidors, els fils secundaris, agafaran del buffer els blocs de N línies, i extrauran de cada línia la informació necessària per actualitzar la matriu `num_flights`. Observar que només els fils consumidors accedeixen a aquesta matriu.

Atès l'esquema presentat en aquesta secció es demana respondre a les següents preguntes a l'informe a entregar

3. Els fils secundaris accedeixen a recursos (variables) compartits entre ells. Quines seran les seccions crítiques? Quines parts del codi són les que s'han de protegir? Cal protegir la lectura del fitxer? Cal protegir l'extracció de dades del bloc? Cal protegir l'actualització de la variable `num_flights`? Comenteu la vostra resposta.

3 Planificació

Per planificar-vos la feina, és important entendre bé el funcionament dels fils així com l'esquema del productor-consumidor. Es proposen els següents punts de treball:

1. És recomanable que dissenyeu l'estructura del buffer de comunicació perquè el consumidor pugui hi pugui "transferir" molt ràpidament un bloc de N línies. De la mateixa forma, és important els consumidors puguin agafar un bloc molt ràpidament. Vegeu la secció 4 per a una proposta d'implementació.

2. Llegiu i experimenteu amb la fitxa del campus que parla dels monitors i les variables condicionals (document de programació amb fils, 2a part). Aquesta és la que us permetrà implementar l'esquema del productor-consumidor amb un un productor i un consumidor, amb una mida de buffer $B = 1$. A teoria s'explicarà el cas general.

3. Implementeu l'esquema del productor-consumidor fent servir un productor i un consumidor (és a dir, el fil principal i un únic fil secundari). A més, si voleu, podeu simplificar el buffer de comunicació perquè només s'hi emmagatzemi un únic bloc, és a dir, $B = 1$. Podeu fer servir l'exemple mostrat a la fitxa per implementar aquesta part.

   El consumidor haurà de finalitzar de forma "neta", és a dir, haurà de sortir de la seva funció d'entrada executant el "return" al final de la funció. No està permès fer servir funcions que "matin" els fils per finalitzar-los (hi ha una funció en C que permet fer-ho). Vegeu la següent secció per a una proposta.

4. A continuació modifiqueu el codi per tal de fer servir $F = 2$ fils secundaris i un buffer en què s'hi puguin emmagatzemar B blocs. Haureu d'implementar l'algorisme de comunicació per a múltiples consumidors i productors. Una de les complicacions es troba en el fet de saber quan acaben els consumidors ja que els consumidors han d'acabar de forma "neta". Vegeu la secció 4 per a una proposta d'implementació.

5. Assegureu-vos que els resultats estadístics que obteniu en processar el fitxer amb múltiples fils són els mateixos que els obtinguts executar el codi amb el codi de la pràctica 4 fent servir els fitxers `2007.csv` i `2008.csv`.

4 Implementació

A continuació es descriu una proposta per implementar l'algorisme proposat. Es descriu, de fet, l'algorisme que s'ha implementat per assegurar que les propostes que es descriuen a continuació són factibles. Sou lliures d'implementar un altre algorisme si així ho desitgeu. En tot cas, l'algorisme que es descriu aquí ha sigut dissenyat perquè sigui eficient a l'hora de "transferir" la informació del productor al buffer i del buffer al consumidor. Implementar un algorisme poc eficient serà penalitzat.

Suposem la següent declaració d'una cel·la (que emmagatzema un bloc de dades):

```
typedef struct cell {
  int nelems;
  char **lines;
} cell;
```

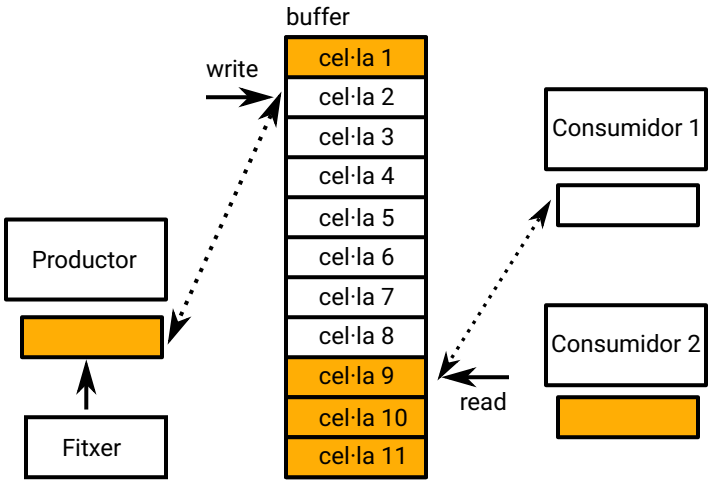

Figura 1: Esquema del productor-consumidor.

Ens recolzarem en l'esquema de la figura 1. Observar que a la figura el buffer de comunicació té capacitat per a $B = 11$ cel·les. A més, tant al productor com als consumidors hi ha també una cel·la. Les cel·les de color ataronjat són cel·les que contenen informació per processar (productor, algunes cel·les del buffer) o que està essent processada en aquell moment (consumidor 2). Les cel·les de color blanc són cel·les que no contenen informació o cel·les en què la informació ja ha sigut processada (consumidor 1 i algunes cel·les del buffer).

Es comenta una forma de procedir per aconseguir una implementació eficient:

1. En reservar memòria la memòria dinàmica, es reserva memòria per a totes les cel·les que hi ha al dibuix. Cada bloc, representat amb la variable `lines` a l'estructura, tindrà una mida per emmagatzemar N línies. La memòria associada aquests blocs no s'alliberarà fins que s'hagin processat tots els blocs del fitxer. Cal evitar, en la mesura del possible, reservar i alliberar memòria mentre es processen els fitxers ja que aquestes funcions són costoses.

2. El productor llegirà el fitxer en blocs d'N línies. La variable `nelems` serveix per indicar quantes línies s'han llegit del fitxer. El valor d'aquesta variable serà N, a excepció del darrer bloc de fitxer, pel qual es podran haver llegit menys línies. El valor d'`nelems` indicarà el nombre de línies que s'han llegit el fitxer.

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 el bloc de N línies del productor a la posició que correspon dintre del buffer. Per fer això caldrà fer servir funcions com *strcpy*, funcions que permeten copiar cadenes de caràcters. El fet de fer-ho això 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 a la posició corresponent al buffer. 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
// Ara intercanviem les dues cel·les
tmp = cell_buffer;
cell_buffer = cell_producer;
cell_producer = tmp;
```

El que estem fent es "transferir" un bloc de dades del productor al consumidor, i agafar-ne un que estigui lliure del buffer. Els consumidors procediran de forma similar: suposem que un consumidor ha acabat de processar les seves dades i vol agafar el següent bloc a processar. La idea és "intercanviar" el bloc del qual disposa el consumidor amb un bloc per processar del buffer. En poques instruccions es pot fer!

Observar que en fer aquest darrer intercanvi (entre el consumidor i el buffer) estem posant una cel·la amb dades ja processades al buffer. No passa res, atès que en algun moment 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 el buffer amb les dades del fitxer, tornarà a fer la "transferència" com s'ha comentat abans.

L'esquema que s'ha presentat fa un moment és el cas particular en què el buffer té mida $B = 1$. Vosaltres haureu d'ampliar la idea perquè el buffer emmagatzemi B cel·les, tal com es veu a la figura 1.

Una altra pregunta que és planteja és: com sabran tots els consumidors que el productor ja ha llegit tots els blocs? Es plantegen dues solucions:

1. La primera solució es basa en el fet que el darrer bloc tindrà, segurament, un nombre d'elements inferior a N. Quan un fil detecti aquest cas sabrà que ell està processant el darrer bloc i haurà de comunicar d'alguna forma a la resta de fils que ja no hi ha més blocs a processar. Aquest esquema només funcionarà si el darrer bloc que insereix el productor al buffer té un nombre d'elements inferior a N,amb la qual cosa aquesta solució ja queda descartada perquè no funcionarà en tots els casos.

2. La segona solució, que és senzilla i curiosa, es basa en procedir de la següent forma: el productor, un cop hagi llegit i "transferit" tots els blocs al buffer, "transferirà" al buffer un total de $F = 2$ cel·les amb un nombre d'elements igual a zero (`nelems=0`). Cada fil secundari, en agafar una cel·la amb un nombre d'elements igual a zero, sabrà que ja no hi ha més blocs a processar i deixarà d'executar el bucle (sense agafar cap cel·la més). I voilà, el fil secundari ja haurà acabat!

5 Entrega

Caldrà entregar fitxer ZIP que inclogui el codi i un informe associat a la feina feta. El nom del fitxer ha d'indicar el número del grup, seguit del nom i cognom dels integrants del grup (e.g. **P5_Grup07_nom1_cognom1_nom2_cognom2.zip**). L'informe a entregar ha d'estar en **format PDF** o equivalent (no s'admeten formats com `odt`, `docx`, ...). El codi font s'inclourà dins de la carpeta `src`. 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`. No us oblideu doncs d'incloure el fitxer *Makefile* corresponent. 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.

- L'informe (**màxim 4 pàgines**, en format PDF, sense incloure la portada) en què es respongui a les preguntes plantejades a la secció 2, així com el experiments de temps d'execució que s'han fet amb els fitxers `2007.csv` o `2008.csv` (no pas el `fitxer_petit.csv`). En fer les proves amb aquests fitxers es recomana no apuntar el primer o segon resultat que es pugui obtenir atès el primer cop que s'executi el codi el fitxer es quedarà, parcialment, a la memòria cau (caché) associada a la màquina. És per això que es recomana repetir múltiples vegades el mateix experiment per veure el temps d'execució associat a un determinat experiment.

  Es proposa fer experiments analitzant el temps d'execució per a l'aplicació sense fils secundaris (el codi de la pràctica 3) així com l'aplicació amb $F = 2$ fils i un valor de $B = 10$. Compareu el temps d'execució amb el que heu obtingut a la pràctica 5. Observeu alguna diferència de rendiment entre les dues solucions? Ha de quedar clar que no necessàriament ha d'augmentar. Els resultats obtinguts dependran molt del tipus de disc (magnètic o SSD) així com el tipus de processador i altres característiques de l'ordinador.

  També es proposa analitzar el temps d'execució per a diferents valors d'N i B. Comenceu per fixar $B = 10$. Proveu a continuació diferents valors d'N(N pot tenir valors de 1, 10, 100, o 10.000, o més). Un cop hagueu obtingut un bon valor d'N, proveu de modificar el valor de B. Pot ser interessant provar valors de $B = 1$, 2, 5 o 10. Assegureu-vos de fer totes les proves al mateix ordinador.

  A l'informe no s'han d'explicar les funcions o variables utilitzades. En tot cas, es pot incloure, si es creu necessari, un esquema de la solució implementada.

Els pesos de la pràctica són: 70% per al codi font i 30% per a l'informe entregat. Respecte el codi, es demana que funcions estiguin comentades, codi modular i net, bon estil de programació (per a les seccions crítiques només es permet que n'hi hagi un punt d'entrada i un punt de sortida), que el programa funcioni correctament, tota la memòria ha de ser alliberada i sense accessos invàlids a memòria. El document ha de tenir una **llargada màxima de 4 pàgines** (sense incloure la portada). El document 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%.