Pràctica 5: Processos i memòria compartida

Sistemes Operatius — iTIC

Sebastià Vila-Marta
Toni Escobet Canal

7 de novembre de 2012

Índex

1 Organització

Aquesta sessió s'organitza com una seqüència d'exercicis que condueixen finalment a una aplicació concurrent formada per diversos processos que, emprant memòria compartida, col·laboren per calcular un producte matricial.

Els objectius inclouen la familiaritzar-se amb aquestes funcionalitats concretes però, sobre tot, la millora de la comprensió del conceptes procés i comunicació per memòria compartida. La pràctica requereix consultes freqüents a `man` i l'ús de la toolchain de `C` sobre GNU/Linux amb la que ja teniu experiència.

1.1 Material necessari

Simplement us cal un computador amb sistema operatiu GNU/Linux i connexió a xarxa. Pel que fa a la documentació de referència, les pàgines de `man` han de ser suficients.

1.2 Lliurament

Cal lliurar els exercicis en un tarfile a través d'Atenea en la data fixada. En aquest cas també caldrà fer una demostració de les aplicacions en la classe de laboratori que ja s'anunciarà.

1.3 Durada

La durada de la pràctica és de 2 sessions de laboratori.

2 Disseny

El producte de dues matrius quadrades de reals A i B és una operació fàcilment paral·lelitzable amb l'objectiu de reduir-ne el temps de càlcul. La idea és la següent, considereu $R = A \times B$, la matriu resultant R es pot entendre subdividida en quatre submatrius de la següent forma:

$$R = \begin{pmatrix} R_{00} & R_{01} \\ R_{11} & R_{12} \end{pmatrix}$$

de manera que R_{ij} és una submatriu de R. Cadascuna d'aquestes submatrius pot ser calculada concurrentment sense dificultats atès que d'una banda no se sobreposen i per tant no hi ha conflictes d'accés i d'altra banda, l'accés a les matrius A i B és únicament de lectura i conseqüentment tampoc ofereix conflictes d'accés.

Per desenvolupar aquesta idea assumiu que tant la matriu A com la B tenen aquesta forma:

$$A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,m-1} & a_{0,m} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,m-1} & a_{m-1,m} & \cdots & a_{m-1,n-1} \\ a_{m,0} & \cdots & a_{m,m-1} & a_{m,m} & \cdots & a_{m,n-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,m-1} & a_{n-1,m} & \cdots & a_{n-1,n-1} \end{pmatrix}$$

Fixeu-vos que la matriu és de dimensió $n \times n$ i els elements estan indexats entre 0 i $n-1$. El valor d'm és la meitat aproximada de n i.e. $m = \lfloor n/2 \rfloor$.

Per a qualsevol matriu A, denotem com $A[a, b; c, d]$ la submatriu d'A definida per la intersecció entre les files que van de la a fins la b i les columnes que van de la c fins la d. Si s'ometen a i b s'entén que significa "totes les files"; de forma similar, si s'ometen c i d s'entén que significa "totes les columnes". Aplicant aquesta notació, observeu les següents equivalències:

$$A[0, m-1; 0, m-1] = \begin{pmatrix} a_{0,0} & \cdots & a_{0,m-1} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,m-1} \end{pmatrix}$$

$$A[0, m-1;] = \begin{pmatrix} a_{0,0} & \cdots & a_{0,m-1} & a_{0,m} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,m-1} & a_{m-1,m} & \cdots & a_{m-1,n-1} \end{pmatrix}$$

$$A[;] = A$$

Usant aquesta notació és senzill definir les submatrius R_{ij} que se citaven al principi:

$$\begin{aligned} R_{00} &= R[0, m-1; 0, m-1] \\ R_{01} &= R[0, m-1; m, n-1] \\ R_{10} &= R[m, n-1; 0, m-1] \\ R_{11} &= R[m, n-1; m, n-1] \end{aligned}$$

En aquest context succeeix que

$$R_{00} = R[0, m-1; 0, m-1] = A[0, m-1;] \times B[; 0, m-1]$$

i de forma similar poden calcular-se la resta de submatrius d'R.

La idea és doncs molt senzilla. Un procés pare crea una zona de memòria compartida en que emmagatzema les matrius A i B i reserva espai per a la matriu resultat R. Acte seguit crea quatre subprocessos que comparteixen la mateixa zona de memòria. Cadascun dels processos fills calcula el producte parcial que dona lloc a una de les submatrius R_{ij} i emmagatzema el resultat a la zona compartida. Finalment el procés pare espera que els fills acabin i escriu la matriu resultat que ha quedat emmagatzemada a la zona compartida.

3 Desenvolupament

El desenvolupament es proposa fer-lo de manera incremental, és construirà un primer programa que s'aproxima lleugerament al que es demana i s'anirà modificant successivament fins arribar al resultat desitjat.

Exercici 3.1 El primer pas consisteix a crear un programa de nom `parent` que quan s'executa crea quatre processos fills i espera a que acabin. Els processos fills solament cal que escriguin el missatge `"Soc el proces 1"` (o 2,3,...) i acabin.

Heu d'assegurar-vos que comproveu tots els possibles errors derivats de crides al sistema i que, en cas de detectar-ne algun, escriviu el corresponent missatge i avorteu. També és important que el procés pare emmagatzemi el pid de tots els fills en una taula de pid's.

Les principals crides implicades en aquest exercicis són fork(), exit() i wait(). Com el codi que desenvolupareu segueix l'estàndard POSIX, cal que poseu exactament al principi del fitxer font, abans que cal altre header, la definició:

```
#define _POSIX_C_SOURCE 200809L
```

Aquesta definició indica que han de ser visibles les crides al sistema d'acord amb la versió del 2008 de l'estàndard POSIX.

Exercici 3.2 Modifiqueu el programa anterior per resoldre la següent situació:

Imagineu que es es creen 3 processos fill i el quart fork falla. Aleshores el programa ha fallat i cal avortar. Succeeix, però, que per aturar el programa cal aturar també els subprocessos que havien arrencat correctament. Per avortar, el procés pare ha de matar els subprocessos fills, esperar a que morin i finalment acabar amb EXIT_FAILURE.

Per que el procés pare pugui matar els subprocessos fills cal que els hi envii un *signal*. És exactament el mateix que feu per matar un procés des de la terminal usant l'ordre kill. En aquest cas, però, cal usar la crida kill() i enviar a cada procés el signal SIGTERM. Quan un procés rep aquest signal, se suïcida i mor. Estudieu la pàgina de man corresponent per veure els paràmetres escaients.

Exercici 3.3 El següent pas consistirà en crear un programa que anomenarem `child` i fer que els subprocessos de `parent` facin un exec de `child`. Més endavant el programa `child` serà el encarregat de calcular un subproducte parcial.

A tal efecte escriviu un programa `child` que comprova que se li passen exactament 3 arguments via argc[], escriu pel canal de sortida els tres arguments i acaba. En cas que no se li passin els tres arguments exactament, el programa falla i acaba amb EXIT_FAILURE.

Modifiqueu el programa `parent` per tal que cada subprocés fill que crea faci un execlp() de `child`. Els paràmetres que es passen en el moment de fer l'exec han de ser, per ordre:

1. El nom del procés, que serà `child0`, `child1` i així successivament.
2. Un string arbitrari que, més endavant, servirà per comunicar al subprocés el nom del fitxer de memòria compartida.
3. Un string que correspon al número de fill: pel subprocés 0, l'string `"0"`, pel 1 l'string `"1"`, etc.

EXERCICI 3.4 Modifiqueu el programa `parent` per tal que comprovi que l'status amb que ha acabat cadascun dels quatre fills és EXIT_SUCCESS. En cas que no sigui així, cal que escrigui un missatge d'error i `parent` acabi amb EXIT_FAILURE.

EXERCICI 3.5 Definiu un `Makefile` per generar els executables del projecte i, a partir d'aquest punt, manteniu-lo convenientment.

EXERCICI 3.6 Implementeu el mòdul `matrix`. Aquest mòdul ha de tenir un header com el següent:

```
#ifndef MATRIX_H
#define MATRIX_H

/* dimensio de la matriu */
#define DIM 8
/* espai que ocupa una matriu */
#define SIZE (DIM*DIM*sizeof(float))

/* el tipus matriu */
typedef float (* const matrix) [DIM];

/* escriu pel canal de sortida la matriu `a' */
void print_matrix(const matrix a);

/* calcula el producte parcial de matrius seguent:
 * r[minf,maxf;minc,maxc] = a[minf,maxf;] x b[;minc,maxc]
 */
void prod_matrix(const matrix a, const matrix b, matrix r,
                 int minf, int maxf, int minc, int maxc);

/* omple `m' amb el valor `v' */
void const_matrix(matrix m, float v);
#endif
```

Aquest mòdul defineix el tipus matriu quadrada de dimensió `DIM` i algunes operacions sobre aquestes matrius. Implementeu `matriu.c`.

És important entendre la definició de `matriu`. Aquest tipus és un apuntador constant a taules de `DIM` reals. Si definiu la variable **float** m[DIM][DIM], el tipus d'm és exactament aquest. Així doncs usarem el tipus `matriu` com un apuntador a matrius quadrades de dimensió `DIM`. Això permet construccions com la següent:

```
void X(matriu m) {
  m[1][3] = 3.14;
}
```

EXERCICI 3.7 En aquest pas caldrà modificar `parent` per tal que crei una regió de memòria compartida abans d'engegar els subprocessos i la esborri una vegada acabat el càlcul. A tal efecte cal usar les crides al sistema shm_open(), ftruncate(), mmap(), close() i shm_unlink(). L'estratègia a seguir és la següent:

1. Creeu un fitxer de memòria compartida usant shm_open(). Doneu-li al fitxer el nom que us sembli.

2. Modifiqueu la seva mida per tal que pugui encabir 3 matrius, A, B i R. La dimensió total en bytes és de $3 \cdot$ `SIZE`. Useu ftruncate() a tal efecte.

3. Mapegeu a memòria aquest fitxer usant mmap(). Assumiu que us retorna addr com l'apuntador a la zona de memòria compartida.

4. Tanqueu el fitxer de memòria compartida, atès que ja no serà necessari en endavant.

5. Per accedir a les tres matrius podeu definir les variables següents:

   ```
   matrix A = addr;
   matrix B = addr + SIZE;
   matrix R = addr + 2*SIZE;
   ```

   Això us permet d'accedir a les matrius que es troben a la zona compartida de forma natural com, per exemple en l'expressió A[2][5]. Mireu d'entendre per què podeu definir les matrius així.

6. Just abans d'acabar el procés pare, esborreu el fitxer de memòria compartida usant shm_unlink(). Noteu que el procés pare pot acabar en molts punts (a totes les crides a exit()). Organitzeu el codi per tal que en qualsevol cas s'esborri el fitxer de memòria compartida.

7. Modifiqueu el necessari per tal que quan un subprocés fa exec() passi com a segon paràmetre el nom del fitxer de memòria compartida. D'aquesta forma els subprocessos podran conèixer quina fitxer han d'obrir per poder accedir a la memòria compartida.

EXERCICI 3.8 Modifiqueu el programa `child` per tal que accedeixi a la zona de memòria compartida que ha creat el procés `parent`.

EXERCICI 3.9 Primera prova de càlcul. Feu que el procés `parent` ompli les matrius A i B amb valors constants. Modifiqueu-lo també per tal que escrigui la matriu resultat R quan tots els subprocessos han acabat. Useu a tal efecte el mòdul `matrix`.

Modifiqueu també el programa `child` per tal que cada subprocés calculi la part d'R que li correspon. A tal efecte heu d'establir una correspondència entre en número de subprocés i la submatriu de la que es fa càrrec.

Proveu l'aplicació. Si tot va bé hauria de calcular el producte sencer concurrentment.

Exercici 3.10 Per finalitzar la pràctica anem a polir la recollida de dades. L'objectiu és que les matrius no siguin fixes i que el resultat no s'escrigui per pantalla sinó que es treballi amb fitxers. A tal efecte caldrà afegir dues funcions més al mòdul `matrix`: una per llegir una matriu d'un fitxer de text i una altra per escriure una matriu en un fitxer de text. Els prototips han de ser:

```
void save_matrix(const char filename[], const matrix m);
void load_matrix(const char filename[], matrix m);
```

El format amb que s'emmagatzemen les matrius en un fitxer ha de ser el mateix que usa `octave`, així podreu interaccionar amb aquest. Proveu el següent programa `octave` per veure'n el format:

```
% Matriu aleatòria de 4x4
M = rand(4,4)

% L'escrivim en un fitxer
dlmwrite("matA.dat",M);

% La llegim de nou
M2 = dlmread("matA.dat")
```

Modifiqueu ara el programa `parent` per tal que, en comptes de usar matrius constants, les llegeixi dels fitxers. Per aconseguir-ho feu el següent:

1. Modifiqueu els paràmetres de main() i considereu que rebrà tres paràmetres via argv[] sempre. Comproveu que és així. Assumiu que el primer paràmetre és el nom del fitxer que conté la matriu A, el segon la B i el tercer és el nom del fitxer on es desarà el resultat. Per arrencar el programa s'haurà d'invocar, per exemple, així:

   ```
   $ ./parent matA.dat matB.dat matR.dat
   ```

2. Feu que el programa inicialitzi les matrius de la zona compartida usant les dades dels fitxers corresponents.

3. Feu que el programa desi la matriu resultant en el fitxer corresponent.