Universitat de Barcelona

Facultat de Matemàtiques i Informàtica

Informe Pràctica 5: Programació multifil: paradigma productor-consumidor

Sistemes Operatius 2

Junjie Li i Manuel Liu Wang

Desembre 2022

1 Introducció

En aquesta pràctica hem d'implementar una estructura proveïdor consumidor/consumidors, on el proveïdor proveeix les dades dels fitxers i el consumidor/consumidors agafen les dades per actualitzar les dades.

2 Funcionalitat a implementar

Pregunta 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?

Si B és més petit que F o B = 1 i F = 2 fils, el model de productor i consumidor que utilitzem no té cap significat, perquè només hi ha un buffer, i només un fil pot accedir a aquest buffer alhora, és a dir, quan el productor escriu el fitxer al buffer, el consumidor ha d'esperar quan acabi, i quan el consumidor llegeix el fitxer al buffer, mentre el productor ha d'esperar. El resultat és que això fa que el programa sembli un programa lineal d'un sol fil i es perden els beneficis del model productor-consumidor.

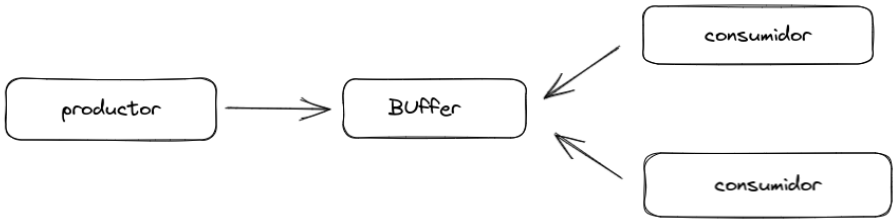

Figure 1: Buffer Block = 1, Fils = 2

Pregunta 2: 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.

Les seccions crítiques són la part on el productor ha d'accedir al buffer i quan ha d'introduir les dades del fitxer i la part on el consumidor ha d'accedir al buffer per extreure les dades que ha introduït el productor. Aquestes dues parts s'han de protegir per evitar que s'accedeixi simultàniament al buffer. Pel que fa la variable num_flights, l'hem de protegir per evitar que dos o més consumidors accedeixin a la mateixa vegada.

3 Planificació

La part més important de la nostra planificació són els 2 structs, **Node** i **Queue**, on el node guarda la informació del fitxer i el nombre de línies i el Queue és el buffer circular. D'altra banda, hem definit diferents mètodes per extreure dades i introduir-les al buffer. A l'utilitzar una Queue, tenim els mètodes add i get que per intercanviar els nodoes de la queue, l'intercanvi de nodes fa servir els punters interns de cada node.

```c
typedef struct Node;
Node *CreateNode();
void DestroyNode(Node *node);

typedef struct Queue;
Queue *CreateQueue();
void DestroyQueue(Queue *queue);
int isFull(Queue *queue);
int isEmpty(Queue *queue
int addElement(Queue *queue, Node *info)
Node *getElement(Queue *queue);
```

Figure 2: Queue i mètode

```c
typedef struct Node {
  char **lines;
  int nelems;
} Node;

typedef struct Queue {
  Node **buffer;
  int front;
  int rear;
  int size;
  int capacity;
  int finish;
  pthread_mutex_t lock;
} Queue;
```

Figure 3: Estructura de buffer

```c
pthread_mutex_lock(&mutex);

while (isFull(queue)){
    pthread_cond_wait(&prod, &mutex);
}


Node *temp = CreateNode();

while (temp->nelems < N_BLOCK && !feof(fp)){
    fgets(temp->lines[temp->nelems], MAXCHAR, fp);
    temp->nelems++;
}

addElement(queue, temp);

pthread_cond_signal(&cons);
pthread_mutex_unlock(&mutex);

if (feof(fp)){
    queue->finish = 1;
    pthread_cond_signal(&cons);
    pthread_mutex_unlock(&mutex);
    return ((void *)0);
}
```

Figure 4: Codi productor

Aquesta és la zona crítica del productor, on llegeix la dada del fitxer i la introdueix al buffer. Si el buffer està ple, hem d'esperar que el consumidor agafi la dada del buffer, en cas contrari introduirà la dada al buffer i cridar al consumidor a què llegeixi les dades. Quan acaba de llegir el fitxer, surt i avisa al consumidor que ha acabat de llegir el fitxer.

```c
if (queue->finish && isEmpty(queue)){
    return ((void *)0);
}

pthread_mutex_lock(&mutex);


while (isEmpty(queue)){
    if (queue->finish && isEmpty(queue)){
        pthread_mutex_unlock(&mutex);
        return ((void *)0);
    }

    pthread_cond_wait(&cons, &mutex);
}

Node *temp = getElement(queue);

for (int i = 0; i < temp->nelems; i++){

invalid = extract_fields_airport(origin, destination, temp->lines[i]);

if (!invalid){
    index_origin = get_index_airport(origin, airports);
    index_destination = get_index_airport(destination, airports);

    if ((index_origin >= 0) && (index_destination >= 0)){
          num_flights[index_origin][index_destination]++;
        }
    }
}

pthread_cond_signal(&prod);
pthread_mutex_unlock(&mutex);

if (queue->finish && isEmpty(queue)){
    pthread_mutex_unlock(&mutex);
    break;
}
```

Figure 5: Codi productor

Aquesta és la zona crítica del consumidor, quan el buffer no està ple, el consumidor llegeix les dades, en cas contrari avisa al productor que introdueixi les dades en cas que encara hi hagi dades per llegir en el fitxer. Quan el productor li avisi que ja ha acabat de llegir totes les dades del fitxer, el consumidor també sortirà.

4 Implementació

En aquesta secció expliquem la implementació de comprovació dels resultats i l'explicació de les diferents proves de rendiment realitzades amb diferents mides del buffer i nombre de línies de cada cel·la de buffer.

Vam fer diferents proves amb diferents fitxers per trobar els mateixos resultats que la pràctica anterior per poder comprovar si el que tenim és correcte o no, en la següent imatge es pot veure que hem aconseguit els mateixos resultats.

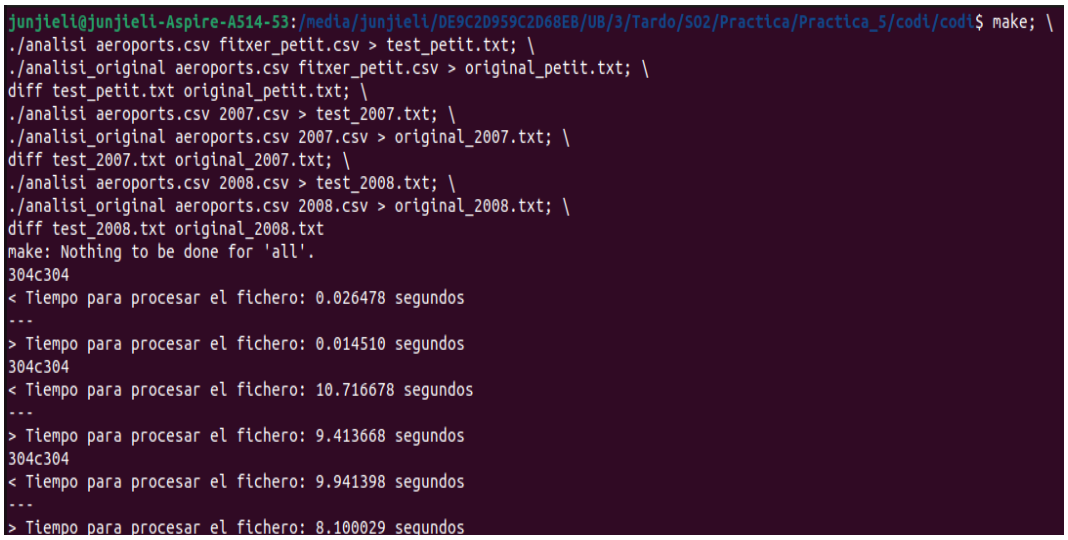

Figure 6: Buffer Block = 1, Fils = 2

En aquest test vam utilitzar una mida de buffer, B = 10, i 2 fils, F = 2. Vam fer aquest test amb diferents nombres de línies:

N_BLOCK	10	100	1000	10000
petit.csv	0.021727	0.021727	0.022788	0.048703
2007.csv	13.716811	11.324812	10.917184	10.462914
2008.csv	12.196950	10.757839	10.770648	10.316503

Table 1: Experiments amb diferents nombre N Block

Com podem veure en els resultats, com més gran sigui les línies del bloc, té un rendiment més alt, ja que amb més línies, pots guardar més informació en cada bloc i es té menys accessos a nous blocs.

En aquest test mirem quan el bloc té 1000 línies, el nombre de fils es manté igual que en l'anterior test però amb diferent mida de buffer.

B_BLOCK	1	2	5	10
petit.csv	0.021568	0.018542	0.020890	0.021883
2007.csv	10.684869	10.891360	10.715122	10.704903
2008.csv	10.411578	10.478517	9.773771	9.824954

Table 2: Experiments amb diferents nombre B Block

Amb experiment anterior podem veure que els resultats no varien molt, els temps de rendiment són similars, perquè el temps computacional es reflecteix en les línies de bloc, ja que s'accedeixen més o menys vegades a un nou bloc. És a dir, com que el consumidor no para d'escriure, l'única cosa que pot endarrerir el seu treball és quan s'ha de canviar a un nou bloc per continuar escrivint.

5 Conclusió

En aquesta pràctica vam implementar una estructura amb un proveïdor i uns consumidors, els resultats són els mateixos que les pràctiques anteriors. I vam fer diferents experiments de rendiments amb diferents mides de buffer, nombre de línies.