Solución eficiente de los lectores/escritores sin hambruna.

Basadas en las slides del profesor Luis Mateu B.

Rodrigo Arenas A., Luis Mateu B. & Lucas Torrealba A.

28 de marzo de 2024

Contenidos

1. Lectores/Escritores por orden de llegada
2. Los cambios de contexto
3. Patrón *request*
4. Otros criterios para el orden de atención

Clase anterior: Lectores/Escritores por orden de llegada

```
int readers = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITALIZER;
pthread_cond_t cond = PTHREAD_COND_INITALIZER;
int display=0, ticket_dist=0;
```

```
void enterRead(){
    lock(&m)
    int my_num = ticket_dist++;
    while(my_num!=display)
        wait(&cond, &m);
    reader++;
    display++;
    broadcast(&cond);
    unlock(&m);
}
```

```
void exitRead(){
    lock(&m)
    readers --;
    if(readers==0)
        broadcast(&cond);
    unlock(&m);
}
```

```
void enterWrite(){
    lock(&m);
    int my_num = ticket_dist++;
    while(my_num!=display || readers > 0)
        wait(&cond, &m);
    unlock(&m);
}
```

```
void exitWrite(){
    lock(&m);
    display++;
    broadcast(&cond);
    unlock(&m);
}
```

Clase anterior: numero explosivo de cambios de contexto

- Considere que hay n lectores en espera.
- Cuando el escritor sale despierta a los n threads.
- Algunos threads no tendrán el siguiente número y se volverán a dormir.
- Cuando por fin se despierta el thread con el siguiente número, vuelve a despertar a todos los lectores.
- En el peor caso se producen $O(n^2)$ cambios de contexto inútiles.
- Solución: patrón *request*.

```
1 typedef struct{
2    int ready;
3    pthread_cond_t w;
4    ...
5 } Request;
```

```
1 Request req = {0,
      ↪ PTHREAD_COND_INITIALIZER
      ↪ };
2 while(!req.ready)
3    pthread_cond_wait(&req.w, &m);
```

Lectores/escritores por orden de llegada con patrón request

```
1  typedef enum { READER, WRITER } Kind;
2  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
3  int readers= 0, writing= 0;
4  Queue q; // = makeQueue();
```

```
1  void enterWrite(){
2      pthread_mutex_lock(&m);
3      if (readers==0 && !writing)
4          writing = 1;
5      else
6          enqueue(WRITER);
7      pthread_mutex_unlock(&m);
8  }
9
10 void exitWrite(){
11     pthread_mutex_lock(&m);
12     writing = 0;
13     wakeup();
14     pthread_mutex_unlock(&m);
15
16 }
```

```
1  void enterRead(){
2      pthread_mutex_lock(&m);
3      if (!writing && emptyQueue(q))
4          readers++;
5      else
6          enqueue(READER);
7      pthread_mutex_unlock(&m);
8  }
9
10 void exitRead(){
11     pthread_mutex_lock(&m);
12     readers--;
13     if (readers==0)
14         wakeup();
15     pthread_mutex_unlock(&m);
16 }
```

Lectores/escritores por orden de llegada con patrón request (2da parte)

```
typedef struct {
    int ready;
    Kind kind;
    pthread_cond_t w;
} Request;

void enqueue(Kind kind){
    // ¡'Patrón request!
    Request req= { 0, kind,
      ↪ PTHREAD_COND_INITIALIZER
      ↪  };
    put(q, &req);
    while (!req.ready)
       pthread_cond_wait(&req.w, &m);
    }
```

```
void wakeup() {
    Req *pr= peek(q);
    if (pr==NULL)
       return;
    if (pr->kind==WRITER) {
       writing= 1; // entra un escritor
       pr->ready= 1;
       pthread_cond_signal(&pr->w);
       get(q); // Sale de la cola
    }
    else { // ¡' pr->kind==READER !
       do { // entran lectores consecutivos
          readers++;
          pr->ready= 1;
          pthread_cond_signal(&pr->w);
          get(q); // Sale de la cola
          pr= peek(q);
          // Si es escritor no sale de la cola
       } while ( pr!=NULL && pr->kind==
      ↪ READER );
    }
}
```

Comparación

- Solución que saca número: $O(n^2)$ cambios de contexto inútiles en el peor caso
- Solución con patrón *request*: ningún cambio de contexto inútil

Criterios de entrada para los lectores/escritores

- Orden indefinido o con prioridades: pueden conducir a *hambruna*.
- Por orden de llegada: intuitivo, pero *reducen paralelismo*.
- Por orden de llegada, pero cuando le toca a un lector, entran todos los lectores en espera. Si llega un nuevo lector, ingresa si y solo si no hay un escritor dentro o en espera. *No reduce tanto el paralelismo.*
- Entradas alternadas (propuesto).

Lectores/escritores: orden de llegada, entran todos los lectores en espera (I)

```
typedef enum { READER, WRITER } Kind;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
int readers= 0, writing= 0;
Queue wq, rq; // = makeQueue();x2
```

```
void enterWrite(){
    pthread_mutex_lock(&m);
    if (readers==0 && !writing)
        writing = 1;
    else
        enqueue(WRITER);
    pthread_mutex_unlock(&m);
}

void exitWrite(){
    pthread_mutex_lock(&m);
    writing = 0;
    wakeup();
    pthread_mutex_unlock(&m);

}
```

```
void enterRead(){
    pthread_mutex_lock(&m);
    if (!writing && emptyQueue(wq))
        readers++;
    else
        enqueue(READER);
    pthread_mutex_unlock(&m);
}

void exitRead(){
    pthread_mutex_lock(&m);
    readers--;
    if (readers==0)
        wakeup();
    pthread_mutex_unlock(&m);
}
```

Lectores/escritores: orden de llegada, entran todos los lectores en espera (II)

```
typedef struct {
    int ready; //Kind kind;
    pthread_cond_t w;
} Request
Request dummy;
void enqueue(Kind kind){
    // Patron request!
    Request req= { 0,
      ↪ PTHREAD_COND_INITIALIZER
      ↪  };
    if (kind==WRITER)
        put(wq, &req);
    else{
        // Para recordar el turno de los
      ↪ lectores
        if (emptyQueue(rq))
            put(wq, &dummy);
        put(rq, &req);
    }
    while (!req.ready)
        pthread_cond_wait(&req.w, &m);
}
```

```
void wakeup() {
    Req *pr= get(wq);
    if (pr==NULL)
        return;
    if (pr!=&dummy) {
        // es el turno de un escritor
        writing = 1 ;
        pr->ready= 1;
        pthread_cond_signal(&pr->w);
    }
    else { // es el turno de todos los
         // lectores
        while (!emptyQueue(rq)){
            pr= get(rq);
            readers++;
            pr->ready= 1;
            pthread_cond_signal(&pr->w);
        }
    }
}
```

Ejercicio propuesto

- Resuelva el problema de los lectores/escritores con entradas alternadas: ..., entra un escritor, entran todos los electores en espera, luego un escritor, luego todos los lectores en espera, luego un escritor, etc.
- Si llega un lector, ingresa si y solo si no hay un escritor dentro o en espera.
- Si llega un escritor ingresa si y solo si no hay nadie dentro.
- Fue pregunta de control el semestre primavera de 2018: https://users.dcc.uchile.cl/~lmateu/CC4302/controles/c1-182.pdf
- El enunciado ayuda bastante a resolver el problema.