Lista Enlazada Paralelo con Pthreads

daryl.butron@ucsp.edu.pe

1. Link al repositorio

https://github.com/dabc312GitHub/tarea678.git

2. Funciones

Se tienen principalmente 3 funciones las cuales crearan problemas en tratar una estructura que almacene elementos:

1. Member: Esta se encargará solo de recorrer la lista hasta encontrar el elemento o sea nulo.
2. Insert: Esta se encargará de crear nodos y agregar el valor correspondiente en el determinado nodo de forma que sea una lista ordenada.
3. Delete: Esta se encargará de eliminar nodos con su respectivo valor.

3. Contexto Paralelo

Para trabajar con paralelismo con pthreads se requiere saber como solucionar el tema de colisiones entre threads al realizar el uso de un recurso compartido en simultáneo. Para esto se tendra que gestionar adecuadamente un recurso compartido como el puntero de tipo nodo $\text{head}_p como variable global, por lo cual se presentarán los s$

3.1. Solución 1:

Se podría colocar un bloqueo por mutex antes de la operación y colocar el desbloqueo despues de la operación. Tratandose así, la lista se bloquea y desbloquea por operación.

Listing 1: Solución 1

```
Pthread_mutex_lock(&list_mutex);
```

```
Member(value);
Pthread_mutex_unlock(&list_mutex);
```

Problema: Esta solución covierte a las operaciones de forma secuencial

3.2. Solución 2:

Se podrían bloquear nodos en lugar de toda la lista, para que así se puede tener mayor libertad de manejo de las operaciones en simultáneo. Para que esta solución funcione se procede a crear un mutex por nodo, es decir que en la misma estructura del nodo se tiene el mutex.

Listing 2: Solución 2 en la operación Member

```
struct list_node_s {
  int data;
  struct list_node_s* next;
  pthread_mutex_t mutex;
};

struct list_node_s** head_p;
struct pthread_mutex_t ** head_p_mutex;

void MemberThreads(int value) {
  struct list_node_s* temp_p;
  pthread_mutex_lock((pthread_mutex_t *) &head_p_mutex);
  temp_p = (struct list_node_s *) head_p;
  while (temp_p != NULL && temp_p->data < value) {
    if (temp_p->next != NULL)
      pthread_mutex_lock(&temp_p->next->mutex);
    if (temp_p == *head_p)
      pthread_mutex_unlock((pthread_mutex_t *) &head_p_mutex);
    pthread_mutex_unlock(&temp_p->mutex);
    temp_p = temp_p->next;
  }
}
/* MemberPthreads */
```

Problema: Al agregar el campo mutex a cada nodo, esto llevara un mayor almacenamiento en memoria que podría ser considerable a la perfomance de la estructura.

3.3. Solución 3: Bloqueo de lectura y escritura

Para el caso de Member se realiza lectura, pero para Insert y Delete se realiza bloqueo por escritura. Para esto se crea bloqueos mas que todo en escritura y en la lectura se dejan pasar los threads ya que no hace modificaciones críticas a la estructura.

Listing 3: Solución 3 bloqueo Read-Write

```
pthread_rwlock_rdlock(&rwlock);
Member(value);
pthread_rwlock_unlock(&rwlock);
. . .
pthread_rwlock_wrlock(&rwlock);
Insert(value);
pthread_rwlock_unlock(&rwlock);
. . .
pthread_rwlock_wrlock(&rwlock);
Delete(value);
pthread_rwlock_unlock(&rwlock);
```

4. Tiempos

Para 8 threads:

1. Read-Write: -4.357145e-312 seconds
2. Mute por lista: 1.484375e+00 seconds
3. Mutex por nodo:

5. Hardware utilizado

1. Intel Core i7
2. 16 GB RAM