Università degli Studi di Modena e Reggio Emilia

RTES-Queue (QoS Lifespan)

Cauchi Gaetano Orazio (matr. n. 185672)

Anno Accademico 2022/2023

Struttura dei messaggi

```
// Definizione della struttura dei messaggi
nella coda

template <typename T> struct element {

    T message;

    chrono::steady_clock::time_point time;

};
```

- `message:` contiene il messaggio inserito dal thread;
- `time:` contiene il tempo trascorso da quando il messaggio è stato inserito.

Variabili utilizzate

```
private:
        struct element<T> *queue;
        size_t dim;
        size_t head, tail;
        size_t count;
        int expiration_time;
        sem_t mutex;
        sem_t push_sem, pop_sem;
        int blocked_push, blocked_pop;
```

- `queue:` vettore che contiene gli elementi della coda;
- `dim:` dimensione della coda (immesso dall'utente);
- `head, tail:` indici utilizzati per scorrere gli elementi della coda;
- `count:` numero di elementi all'interno della coda;
- `expiration_time:` contiene il tempo di scadenza di un messaggio (immesso dall'utente);
- `mutex:` semaforo di mutua esclusione per garantire la consistenza della coda;
- `push_sem, push_pop:` semafori per gestire l'inserimento o l'estrazione di elementi in coda;
- `blocked_push, blocked_pop:` contatori indicanti il numero di lettori/scrittori bloccati in attesa di inserire/estrarre elementi dalla coda.

Costruttore

```
// Costruttore
        LifespanQueue(size_t dim, int expiration_time) {
            if(dim < 1)
                throw logic_error("Invalid dimension.");

            if(expiration_time <= 0)
                throw logic_error("Invalid expiration time.");

            this->dim = dim;
            this->expiration_time = expiration_time;
            this->queue = (struct element<T> *)malloc(this->dim * sizeof(struct element<T>));
            this->head = this->tail = 0;
            this->count = 0;

            sem_init(&this->mutex, 0, 1);
            sem_init(&this->push_sem, 0, 0);
            sem_init(&this->pop_sem, 0, 0);
            this->blocked_push = this->blocked_pop = 0;
        }
```

Metodo expired

```
// Funzione che controlla se vi sono messaggi scaduti nella coda e ritorna se ne sono stati rimossi

        bool expired(void) {

            int previous_tail = this->tail;

            for(int index = this->tail; this->count > 0; index = (index + 1) % this->dim) {

                if((chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now() -
                    this->queue[index].time).count()) < this->expiration_time)

                    break;

                else {

                    this->tail = (this->tail + 1) % this->dim;

                    this->count--;

                }

            }

            return (this->tail == previous_tail) ? false : true;

        }
```

Se il tempo trascorso in coda (`now() - time`) da parte del primo elemento (`tail`) è minore del tempo di scadenza (`expiration_time`), vuol dire che non ci sono messaggi scaduti.

In caso contrario avanzo di un elemento, in modo da escludere il messaggio scaduto dall'essere estratto o dall'occupare inutilmente la coda.

Se è stato rimosso almeno un elemento dalla coda, ritorno `true`; altrimenti `false`.

Metodi push e pop (non bloccanti)

```
// Inserisce un nuovo elemento nella coda (non bloccante)

        void push(T message) {

            sem_wait(&this->mutex);

            if(full() && !expired()) {
                  cout << "Queue is full." << endl;
                  sem_post(&this->mutex);
                  return;
            }

             this->queue[this->head].message = message;
             this->queue[this->head].time = chrono::steady_clock::now();
             this->head = (this->head + 1) % this->dim;
             this->count++;


            for(int i = 0; i < this->count && this->blocked_pop; i++) {
                  this->blocked_pop--;
                  sem_post(&this->pop_sem);
            }

            sem_post(&this->mutex);
      }
```

```
// Preleva un elemento dalla coda (non bloccante)

        T pop(void) {

            sem_wait(&this->mutex);

            expired();

            if(empty()) {
                cout << "Queue is empty." << endl;
                sem_post(&this->mutex);
                return NULL;
            }

            T message = this->queue[this->tail].message;

            this->tail = (this->tail + 1) % this->dim;
            this->count--;

            for(int i = 0; i < (this->dim - this->count) && this->blocked_push; i++)
            {
                this->blocked_push--;
                sem_post(&this->push_sem);
            }

            sem_post(&this->mutex);

            return message;
        }
```

Metodi push e pop (bloccanti)

```
// Inserisce un nuovo elemento nella coda (bloccante)

        void bPush(T message) {
            sem_wait(&this->mutex);

            while(full() && !expired()) {
                this->blocked_push++;
                sem_post(&this->mutex);
                sem_wait(&this->push_sem);
                sem_wait(&this->mutex);
            }

            this->queue[this->head].message = message;
            this->queue[this->head].time = chrono::steady_clock::now();
            this->head = (this->head + 1) % this->dim;
            this->count++;

            for(int i = 0; i < this->count && this->blocked_pop; i++) {
                this->blocked_pop--;
                sem_post(&this->pop_sem);
            }

            sem_post(&this->mutex);
        }
```

```
// Preleva un elemento dalla coda (bloccante)

        T bPop(void) {
            sem_wait(&this->mutex);

            T message;

            while(1) {
                expired();

                if(!empty()) {
                    message = this->queue[this->tail].message;
                    this->tail = (this->tail + 1) % this->dim;
                    this->count--;
                    break;
                }
                else {
                    this->blocked_pop++;
                    sem_post(&this->mutex);
                    sem_wait(&this->pop_sem);
                    sem_wait(&this->mutex);
                }
            }

            for(int i = 0; i < (this->dim - this->count) && this->blocked_push; i++) {
                this->blocked_push--;
                sem_post(&this->push_sem);
            }

            sem_post(&this->mutex);

            return message;
        }
```

Metodo printQueue

```
// Stampa gli elementi attuali che compongono la coda

        void printQueue(void) {

            sem_wait(&this->mutex);
            int index = this->tail;

            if(empty())
                cout << "Impossible to print the queue. Queue is empty." << endl << endl;
            else
                for(int i = 0; i < this->count; i++) {
                    cout << "Elem. n. " << (i + 1) << ":" << endl;
                    cout << "Message: " << this->queue[index].message << " | ";
                    cout << "Elapsed time: " << chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now() - this->queue[index].time).count() << endl;
                    cout << endl;
                    index = (index + 1) % this->dim;
                }

            for(int j = 0; j < 10; j++) cout << "-";
            cout << endl;

            sem_post(&this->mutex);
        }
```

Test.cpp

(main)

```
int main() {
    // Numero di writers stabilito dall'utente
    char n_writers[20];

    // Controllo dell'input
    while(1) {
        int i;
        printf("Enter the writers number: ");
        fgets(n_writers, 20, stdin);

        for(i = 0; i < (strlen(n_writers) - 1); i++)
            if(!isdigit(n_writers[i])) {
                printf("Attention: incorrect number entered.\n");
                break;
            }

        if(i == 0)
            continue;

        if(i == strlen(n_writers) - 1)
             if(atoi(n_writers) < 1)
                printf("Minimum writers count: 1.\n");
             else
                break;
    }

    // Numero di readers stabilito dall'utente
    char n_readers[20];

    // Controllo dell'input
    while(1) {
        int i;
        printf("Enter the readers number: ");
        fgets(n_readers, 20, stdin);

        for(i = 0; i < (strlen(n_readers) - 1); i++)
            if(!isdigit(n_readers[i])) {
                printf("Attention: incorrect number entered.\n");
                break;
            }

        if(i == 0)
            continue;

        if(i == strlen(n_readers) - 1)
            if(atoi(n_readers) < 1)
                printf("Minimum readers count: 1.\n");
            else
                break;
    }
```

```
// Dimensione coda stabilita dall'utente
    char queue_size[20];

    // Controllo dell'input
    while(1) {
        ...
    }

    // Tempo di scadenza dei messaggi stabilito dall'utente
    char expiration_time[20];

    // Controllo dell'input
    while(1) {
        ...
    }

    // Istanziamento della classe con input utente
    lq = new LifespanQueue<TYPE>(atoi(queue_size), atoi(expiration_time));

    pthread_t *p;
    srand(555);

    p = (pthread_t *)malloc((atoi(n_writers) + atoi(n_readers)) * sizeof(pthread_t));

    // Istanziamento dei writers
    for(int i = 0; i < atoi(n_writers); i++) {
        pthread_create(&p[i], NULL, WRITER, NULL);
        usleep(100);
    }

    // Istanziamento dei readers
    for(int i = atoi(n_writers); i < (atoi(n_writers) + atoi(n_readers)); i++) {
        pthread_create(&p[i], NULL, READER, NULL);
        usleep(100);
    }

    // Attesa della fine dei thread
    for(int i = 0; i < atoi(n_writers); i++) {
        pthread_join(p[i], NULL);
    }

    free(p);

    return 0;
}
```

Definita come variabile globale:
`LifespanQueue<TYPE> *lq;`

Test.cpp

WRITERS

```
// Thread che inseriscono messaggi nella coda

void *WRITER(void *args)
{
    for(int i = 0; i < 10; i++) {
        usleep((rand() % 100000) + 100000);
        TYPE mex = rand() % 10;

        cout << "Trying to push the message: " << mex << endl;
        lq->push(mex);

        lq->printQueue();

        usleep(500000 - (rand() % 10000));
    }

    return NULL;
}
```

READERS

```
// Thread che prelevano messaggi dalla coda

void *READER(void *args)
{
   for (int i = 0; i < 10; i++) {
        usleep(500000 - (rand() % 10000));

        TYPE ret = lq->pop();
        cout << "Message taken: " << ret << endl;

        lq->printQueue();

        usleep((rand() % 100000) + 300000);
   }

    return NULL;
}
```

Tipo di dato del messaggio da definire a priori:

```
#define TYPE int
```