CIS 415 Operating Systems

Assignment 3 Report

Submitted to:

Prof. Allen Malony

Author:

Xuehai Zhou

Report

Introduction

This program simulates a publisher/subscriber model. It will take in and parse a command file, which contains commands to create topic, add publisher or subscriber files, query topics, publishers and subscriber, and to start the program. In the publisher and subscriber files we added in contains commands to enqueue and get entries of photo URL and photo caption in different topic queues. We have a topic queue array where each queue is a struct has a queue ID, a filename, the length of the queue, an integer number head and tail to implement a circular ring queue, a struct buffer to store topic entries, and a mutex lock. And there is an entry number, a time stamp, a publisher ID, a string of photo URL, and a string of photo caption in each topic entry. The program will create two set of threads for publisher and subscriber and a clean thread to dequeue entries is too old, which stayed in the queue longer than a certain time. The main job of this project is to enqueue, get entry and dequeue to or from the topic queue synchronously. Finally, the program will write the photo caption and photo URL in a table with a topic name to a html file.

Background

In terms of complete synchronization, we have to know the usage of the mutex and functions list follow:

1. Thread

- *int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);*
- *int pthread_join(pthread_t thread, void **retval);*

2. Mutex and scheduling

- *int pthread_mutexattr_init(pthread_mutexattr_t *attr);*
- *int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);*
- *int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);*
- *int pthread_mutexattr_setrobust(const pthread_mutexattr_t *attr, int robustness);*
- *int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);*
- *int pthread_mutex_destroy(pthread_mutex_t *mutex);*
- *int sched_yield(void);*

3. Queue

- *int gettimeofday(struct timeval *tv, struct timezone *tz);*

Implementation

1. topicStore: topicStore.c topicStore.h

We have two structs to indicate topic entries and topic queues.

```c
typedef struct topicEntry {
    int entryNum;
    struct timeval timeStamp;
    long int pubID;
    char photoURL[URLSIZE]; // URL to photo
    char photoCaption[CAPSIZE]; // photo caption
} topicEntry;
```

```c
typedef struct topicQueue {
    int qid;
    char name[NAMESIZE];
    int entryCtr;
    int length;
    topicEntry buffer[MAXENTRIES];
    int head;
    int tail;
    pthread_mutex_t mylock;
} topicQueue;
```

And a topic store struct to store a topic queue array and the number of topics.

- *Circular ring queue: head/tail points to the next position that an entry was enqueued/dequeued*
  - *Enqueue: copy the new entry values to the queue buffer in tail position then increment the tail, and set the return value to be 1 when the enqueue is success. When the tail is in the last position in the ring, which is when tail is equal to the length of ring, then we set the tail to 0, which is the first position in the ring. When the tail minus the head equals -1, that means the ring is full, then enqueue cannot be processed, enqueue function will return 0.*
  - *Dequeue: copy the head position entry to a new entry, then we increment the head. All other behaviors are similar to enqueue. When head is equal to tail, the ring is empty, return 0.*
  - *Get entry: each entry has an entry number. First, we want to make sure the queue isn't empty, or return a 0. (case 1) If we can find an entry with entry number equals to the number of last entry, it means we can find the last entry in the topic queue. Then we want to load the entry with the entry number of the last entry to the new entry. (case 2) Else we want to find if there are some entry number is greater than the last entry number. If so, we load that entry to the new entry and return 1, else we return a 0. (case 3-ii and 3-i)*
- *Mutex*

  As far as I know the implementations of the mutex are vary. The way I did was having a mutex in each topic queue. Then we lock or unlock in the front and back of enqueue, dequeue and get entry functions.
  We can also have a mutex array that has a length of MAXQUEUE. In each topics, when we want to enqueue, dequeue or get entry, we will lock the corresponding mutex.

  - *Initialization: we can either use PTHREAD_MUTEX_INITIALIZER or call pthread_mutex_init(pthread_mutex_t* *mp,*const pthread_mutexattr_t* *mattr*);*

2. quacker: quacker.c quacker.h

- *File parsing*

  We have a global topic store initialized in the very front, where stores a topic queue array and the length of the array. In main function we parse the file taken in. When we encounter a command to create topic, then we initialize a queue to the position of the number of queues in the topic store. When we see query topic, publisher, subscriber commands, we print the information of all topics, publishers and subscribers. If the command is add publisher or subscriber, we want to pass in the file name and save it when the program starts. When the command is delta, then we save the delta number into a integer buffer to save it for the later use in the clean thread. There was one difficulty that took me pretty long to solve is a character called carriage reture. When we want to tokenize a line, we may sometimes see carriage return, which is similar to the newline character, hanging around, so we also want to remove '\r' when tokenize.

- *Thread create*
  - *Useage: We will create thread by calling pthread_create(). There are four parameters of pthread_create. The first is a thread of pthread_t type, the second is an attribute of pthread_attr_t type, the third is a void*

pointer type function, and the fourth is a void pointer type parameter that will be passed into the third parameter function.

- *Void *Publisher/subscriber: since these two functions only take one parameter in a certain type of void pointer, we have to come out a way to solve if there are multiple parameters that we may want to use. Therefore, the way out is to create a struct I named it thread pool:*

```
typedef struct threadPool {
    char filename[FILENAME_MAX];
    int thread_idx;
    pthread_t thread;
    int isFree;
} threadPool;
```

isFree is the indicator whether this thread is available or not. Thread index tells us the thread ID. The thread is the first parameter we want to pass in of pthread_create(). The filename is the char array to save the filename read from argv[2].

I start the thread pool to be an array with a length of NUMPROXIES. Then initialize all isFree to be 1 that indicates all threads are available. The isFree is being set to 0 when pthread_create() call the publisher/subscriber functions. And in the bottom of those two functions, we set isFree back to 1. Before we want to create a thread, we always want to check whether the thread in pool is available by checking isFree.

Then parse file by using getline and strtok in a standard way. When we read put/get in, I did enqueue or get entry each for 30 times, and sleep 100 milliseconds after each failures. After 30 times enqueue or get entry, if they haven't succeeded, then we report an error. To avoid the program run into a infinite loop, I didn't use while loop here.

- *Join: join all thread when all threads created. Let the main thread wait all threads terminates.*

- *Mutex destroy*

Destroy all mutex in in topic struct by looping through the topics and calling pthread_mutex_destroy().

3. HTML

When we want to view a webpage's source, the tricky is go to that page in chrome and right click the page and choose the view page source. Then the HTML source code will pop up. Thus, we can call fputs() to add the HTML code to a html file line by line.

Performance Results and Discussion

All parts run properly excepting some possible memory leak by conditional jump of uninitialized blocks existing.

Conclusion

This project is a good training of threads coding. Thanks for the guidance from Jared Hall, Grayson Guan, and Joseph Goh.