CS490: Concurrent Program Testing

Problem statement

To find a conccurent bug, both buggy input and buggy schedule are required. However in a normal execution, we can specify input but we cannot specify the schedule. Thus it is hard to find a conccurrent bug with a normal execution and we need a tool to explore schedules.

In this project, you'll implement a simplified version of CHESS to explore schedules systematically.

Step 1 (7pt)

Step 1: Implement deterministic scheduler

In this step, you'll implement a deterministic scheduler. That allows only one thread to execute at one time and repeated executions with the same input produce the same sequence of instructions. Your deterministic scheduler will be tested with 2-thread concurrent programs.

Algorithm

Enter a thread	`LOCK(GL)`

Leave a thread	`UNLOCK(GL)`

- Lock the global lock before entering a new thread. This should be done for the main thread too. One way to lock the global lock in the main thread is that lock at the first instance of pthread_create. You can reuse pthread library for the global lock.

pthread_mutex_lock	`if (L is held by other threads)`
	`  UNLOCK(GL);`
	`  /* select next available thread to execute */`
	`  /* wait till L is not held by any other threads */`
	`  LOCK(GL);`
	`  LOCK(L);`
	`else`
	`  LOCK(L);`

- The algorithm uses the existing LOCK functionality to suspend the current thread until the lock L is available. However with the existing pthread library we can't know if a thread is waiting for a lock or is ready to execute. Therefore we cannot simply use the pthread lock and unlock primitives to realize the polling lock state and LOCK(L) in the semantics of pthread_mutex_lock, we have to re-implement LOCK semenatics. With your LOCK implementation, you should be able to know 2 things: whether this lock is held by other thread or not, and whether this thread is waiting for a lock or not. In order to map the pthread_mutex_t object and your mutex object, you can use hash table.

pthread_join	`UNLOCK(GL);`
	`/* select next available thread to execute */`
	`/* wait till joinee terminates */`
	`LOCK(GL);`

- Similar to the LOCK, you should maintain status(whether a thread is running or terminated) for each thread. You can use hash table to map the pthread_t object and your own thread object.

sched_yield	`UNLOCK(GL);`
	`/* select next available thread to execute */`
	`LOCK(GL);`

- If there's no other available thread, the calling thread should be executed. If there is another available thread, the other thread should be executed.

You will implement these primitives by using the provided template. The implementation will be compiled to a dynamic library in place of the standard pthread library. When it is linked and executed with a concurrent program, deterministic execution can be achieved. Note that you should not need to change anything in the test programs.

Template

You can use the following template for the basic part of the tool. The template has a basic implementation to wrap pthread functions. Your code should be compiled in the unix or linux machine.

chess.tar.gz

Build

Use the makefile included in the package. Type 'make' in the directory with makefile and source code and chess.so will be built.

Run

Use run.sh in the package. Type './run.sh ./sample' in the directory with chess.so and run.sh will execute program sample in the current directory with the tool.

Sample test program: sample1.c

This program will print out numbers with two threads and the order of numbers will change as schedules. The deterministic scheduler should produce same input at every execution.

You can build sample1.c by typing 'gcc -o sample1 -lpthread -lrt sample1.c' on the shell.

Step 2 (8pt)

Step 2: Implement CHESS scheduler

1. In the first execution, record all synchronization points where you can switch to another thread. The synchronization points are after a thread is created, before mutex is locked, and after mutex is released.
2. In the (n+1)-th execution, switch to another thread at the n-th sycnhronization point.

Sample test program: sample2.c

This program has an atomicity violation bug and it will crash the program in a certain schedule. Use your tool to find the bug in this program.

You can build sample2.c by typing 'gcc -o sample2 -lpthread -lrt sample2.c' on the shell.

Extra credit (2pt)

Find a concurrent bug in pbzip2.

The pbzip2 is a parallel compression/decompression program. An old version of pbzip2 has a concurrent bug. Download the following buggy pbzip2 and input and find a concurrent bug with your tool. You'll need to implement **pthread_cond_wait (wait)** and **pthread_cond_broadcast (notify)** to find a bug.

pbzip2.tar.gz, input

Submission

- By 10/11, 11:59pm, you should finish deterministic scheduler. Your code should implement global lock (2 points), thread status (2 points), lock semantic (2 points), and scheduler (1 point). Submit all of your code. Your code will be tested with several simple 2-thread concurrent programs.
- By 10/18, 11:59pm, you should finish CHESS scheduler. Your code should be able to records and try all possible synchronization points. Submit all of your code and readme about how to run your tool. Your code will be tested with several programs that will express a bug in a certain schedule.

Where to get help

Dohyeong Kim(kim1051@purdue.edu) is managing this project. His office hour is Wednesday 10:30am -- 11:30am at LWSN 3151 #4. We recommend you to use the Piazza for sharing your questions and get answers faster. Every day our instructors check the Piazza several times to process students' questions.

Appendix

You can find a document about pthread from internet including pthread tutorial from Lawrence Livermore National Laboratory