CS636 Assignment 2 : Concurrent Debugger (Conc-GDB)

Rohan Bavishi
13196

Features Implemented (Conc-GDB)

1) The standard "***break***", "***continue***" and "***next***" commands are supported. They take an additional argument which specifies the particular thread for which the operation needs to be performed. This arg can be a "*" to signify that the operation needs to be performed for all threads

2) The "***info***" and "***getfields***" commands display the current status of all threads, and the current context of the thread (limited to class variables, no local variables). This is useful for identifying which thread is which, and for tracking changes in variables

3) The "***replay***" command plays back the commands stored in a file. This is useful for reproducing a bug as the commands in essence describe a concrete schedule. All the user commands are automatically logged in a debug file, which can then be replayed.

4) Other standard commands can be listed by using the "***help***" command in Conc-GDB.

Installation and Usage

JRE 1.7 is required to be installed. The soot-2.5.0 jar is provided. The Makefile needs to be updated according to the path of the installation directory for JRE 1.7

The tool can be invoked using the master shell script "conc-gdb.sh" as follows -
./conc-gdb.sh <java-file-path>

The tools takes a java file as an input; class files are not supported as of now.

Implementation Details

1) The Soot framework is used to instrument the input class files (obtained after compilation). Specific calls to static methods are inserted before every distinct line (the lines correspond to actual line numbers in the original java file). Examples include a call to check whether a breakpoint has been inserted at this place.

2) A class named "Controls" with static methods and static variables acts as a global state, and is used for interaction between the master Conc-GDB thread and the input program.

3) A separate object is associated with every thread in the original program. The master Conc-GDB thread interacts with each individual thread using “wait” and “notify” on these objects.

4) A special pause is added to the main function of the input program so as to prevent it from running before the Conc-GDB can issue commands to it.

5) Each thread also registers its thread object with the Controls class, so the Conc-GDB thread can extract relevant information about these threads.

Limitations

1) Debuggers are supposed to work with object/class files. So even if the user doesn’t have access to source-code, he/she can reason about the program. Our tool does not have support for a direct input of class files currently. This is primarily done to simplify the design (for example, eliminate the manual class dependency checks)

2) The implementation is very inefficient as threads perform a check for a breakpoint after every line. So large programs cannot be handled in reasonable time.

3) The “break”, “next” and “continue” commands only take a single thread ID as an argument. The support for a collection of thread IDs has not been implemented yet.

4) The implementation relies on the internal thread IDs used by the Thread library in Java. Although they are guaranteed to be unique for every thread that is alive, they may be reused once a thread exits. This can create problems as the mapping between our custom ID and these IDs would break. Fortunately, the IDs are not reused when we have a reasonably small number of threads.

Tests (replay files are provided in the folder)

1) ***Deadlock*** - A simple program to demonstrate a deadlock when two threads sync using two different objects without any partial ordering.

2) ***WriteOrderViolation*** - A simple program to demonstrate a write-order-violation (as described in Shan Lu’s paper) which performs an early write to a variable that is missed by one thread, which causes it to go into an infinite loop.

3) ***AtomicityViolation*** - A simple program to demonstrate an atomicity violation (as described in Shan Lu’s paper).

4) ***Dining Philosophers*** - The example discussed in class, wherein we can get into a deadlock state when the philosophers take forks without any partial ordering.

5) ***HashTable*** - The Unsynchronized hash-table from Assignment-1 is used to reproduce a Null-Pointer Exception.