International Journal of Computer Science and Informatics

Volume 1 | Issue 4 Article 6

April 2012

Extending Racer with Thread is Alive Mechanism

GC Dinesh
Department of CSE, JNTUACE, Anantapur, A.P., India, gc.dinesh143@gmail.com

B. Eswara Reddy Dr.
Department of CSE , JNTUACE, Anantapur, A.P, eswarcsejntua@gmail.com

A. P. Sivakumar
Department of CSE, JNTUACE, Anantapur, A.P., India, sivakumar.ap@gmail.com

Extending Racer with Thread is Alive Mechanism

GC Dinesh, B. Eswara Reddy & A. P. Sivakumar

Department of CSE, JNTUACE, Anantapur, A.P., India
E-mail : gc.dinesh143@gmail.com, eswarcsejntua@gmail.com, sivakumar.ap@gmail.com

Abstract **-** We know that detecting concurrent programming errors such as live locks, data races, starvation and deadlocks is mainly headed problem in day to today programming. Several algorithms have been developed in order to find the concurrency related bugs, among such type of algorithms the efficient one is RACER algorithm. Previously in racer algorithm RACER only uses events acquiring and releasing locks, as well as calls to Thread.start. So here in this work, we extend racer algorithm by making calls to thread is alive method. We use aspect oriented programming AspectJ. The purpose of extension racer algorithm is, in order to increase the performance of an algorithm. We applied this extension of racer algorithm to two small programs. The experiments state that extension of racer increases the efficiency of racer algorithm.

Keywords **-** *Data races, Pointcuts, Aspect oriented programming, AspectJ.*

I. INTRODUCTION

While developing software systems we come across so many errors in the program. Especially in concurrent programming one of the major issue is coordinating access to resources .Accessing to shared resources is mainly concerned in concurrent programming. While sharing resources we use locking discipline such as acquiring and releasing of locks. Here we come across errors in locking discipline. Data races are the one of the concurrent programming errors. Errors due to data races in multithread program often exhibit non deterministic symptoms and are difficult to find. These data races occur when multiple threads access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access take place. These data races add main sources of failure to get the expected output. So many specialized analyses have taken aid programmers with these tasks. Dynamic approaches as well as static approaches have been considered for analysis over on the programs. The low level byte code instrumentation library is used by the most of the dynamic approaches. These low level byte code instrumentation libraries are difficult to use. Programming with this low level byte code instrumentation is tedious and time consuming. So in order to declare the instrumentation at a high level abstraction aspect oriented programming is the convenient tool. Aspect oriented programming entails breaking down program logic in to distinct parts called concerns .The popular aspect oriented programming language is AspectJ. AspectJ is the compatible language extension to java. In AspectJ programmers can define aspects. Aspects are a new class-like language element that has been added to Java by AspectJ. Aspects are how developers encapsulate concerns that cut across classes, the natural unit of modularity in Java. AspectJ contains join points where these join points are well-defined points in the execution of a program, these join points are described by the pointcut declaration. Pointcuts picks out these join points. In AspectJ we have advice where Advice is code that executes at each join point picked out by a pointcuts. There are three kinds of advice one is before advice, around advice and after advice. As their names suggest, before advice runs before the join point executes; around advice executes before and after the join point; and after advice executes after the join point.

II. RACER ALGORITHM

Racer algorithm has been implemented for finding data races at runtime .Firstly this RACER algorithm was presented at ISSTA in 2008. But due to large number of data races were noticed because of false warnings reported by RACER. These false warnings occur due to ignoring calls to thread.start(). So the next version of the RACER algorithm is implemented making calls to thread.start(). Here in this work AspectJ language extension with three new pointcuts has been proposed. These newly proposed pointcuts are lock (), unlock () and maybeshared().

The basic principle containing in RACER algorithm is LOCKSETS. In the lock sets a set of candidate lock L(f) is maintain for each field ‘f’. A field is qualified by

its owner. L(f) contains the lock objects at each point of program execution. All the threads could agree the lock objects when accessing the field 'f'. We maintain a lock set L(c.f) for a static field 'f' of a class 'c'. We maintain a lockset L(o,f) for an instance field 'f' of an object 'o'.

In implementation of RACER we contain two aspects one is LOCKING another is RACER. In order to keep the track of locksets we use aspect LOCKING. To maintain a mapping from fields to a state and to update the state we use the RACER aspect as well as to keep track of invocation of thread.start() we use RACER aspect.

III. STATE MACHINE

State machine of RACER algorithm contains the five states they are virgin, exclusive, exclusive modified, shared, shared modified. Fig 1 shows the State Machine Diagram.

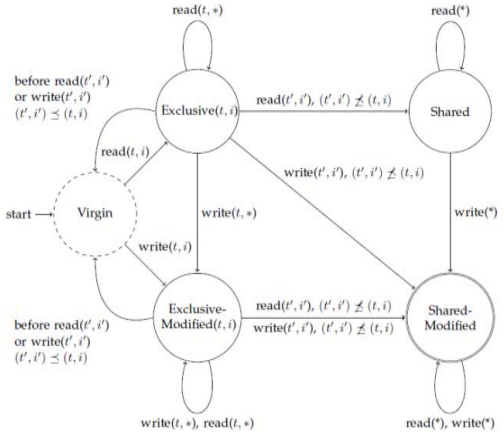

Fig. 1: State Machine Diagram

The starting stage is virgin state. Whenever thread reads value from its main thread then thread state changes to the Exclusive. If the thread writes the value from the main thread then the state changes to the Exclusive modified. If the thread reads the value which is already read by another thread then it changes to state Shared .In the same way if it write from the state from exclusive or Exclusive modified then the state changes to the Shared modified. Whenever it is in state of shared modified then it reports the data race where the lockset becomes empty here.

In the state machine of RACER algorithm we have access periods in order to define in which period the thread't' access the field 'f'. Access period contains ($t.i_t$) where it shows the time of accessing period .In order to keep track of which access periods are visible to other access periods. We maintain a visibility relation. The visibility relation is the one where it shows the smallest relation among access periods in order to keep track of spawned threads when they are spawned.

The RACER algorithm is applied to K9Rover executive application. It finds 12 data races without false warnings.

IV. EXTENSION TO RACER

In this proposed system we extended the racer algorithm by capturing calls to "thread is.alive()". As we know previously in racer algorithm we capture a calls to the thread.start () in order to avoid reporting to the false warnings. Here we use AspectJ language for capturing calls to "thread is. alive()" method. Actually "thread is.alive()" method is used to determine the thread is running or not i.e. alive method returns true if the thread is still running are else it returns false.

Due to improper synchronization in between these threads there will be the occurrence of data races in a way that "thread is.alive()" method returns false instead of true causes the main thread may interrupt instead of other spawned thread. So in order to overcome this we write an aspect by capturing calls to thread is.alive() method. This aspect keeps track of which thread is interrupted and also we write the advice in order to extract the field name as well as for declaring class and the source location from the special constant this Joinpointstaticpart. We write the advice before the execution and after the execution of thread is.alive () method.

Finally in this work we capture the calls to "thread is.alive ()" method and maintain the track of which thread is interrupted. We use AspectJ language for capture the call to methods.

V. EXPERIMENTAL RESULTS

We applied the racer extension to the banking application. The application contains eight threads and each thread makes a random transaction from one account to another. By applying racer extension to bank application one race and one false positive is reported and actual races are zero. The results are shown in Table I.

TABLE I : Experimental Results

Reported races	1
Actual races	0
False positives	1

VI. CONCLUSION AND FUTURE WORK

The conclusion of this work is we use aspect oriented programming language i.e. AspectJ for capturing calls to "thread is.alive()" method .By AspectJ

language we write the aspect for capturing calls to "thread is.alive()" method for tracking of threads interruptions. If there are any wrong interruptions because of improper synchronization then the aspect will report the date race. So here we write an advice to know that where the data race occurs. The future work for this work is we can extend racer algorithm by making calls to object. wait() or notify () methods.

REFERENCES

[1] J. Harrow, "Runtime checking of multithread applications with visual threads," in SPIN Model Checking and Software Verification, ser. Lecture Notes in Computer Science, K. Havelund, J. Penix, and W. Visser, Eds., vol. 1885. Springer, 2000, pp. 331–342.

[2] R. O'Callahan and J.-D. Choi, "Hybrid dynamic data race detection." in ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03). ACM press, 2003, pp. 167–178.

[3] C. von Praun and T. R. Gross, "Object race detection," in Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems,Languages, and Applications (OOPSLA). ACM press, 2001, pp. 70–82.

[4] C. Artho, K. Havelund, and A. Biere, "High-level data races, "Software Testing, Verification and Reliability, vol. 13, no. 4, pp. 207–227, 2003.

[5] Stolz and E. Bodden, "Temporal assertions using AspectJ," Electronic Notes in Theoretical Computer Science (ENTCS), vol. 144,no. 4, pp. 109–124, 2006.

[6] C. Allan, P. Avgustinov, A. S. Christensen, L. J. Hendren, S. Kuzins, O. Lhot ´ak, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble, "Adding trace matching with free variables to AspectJ," in Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), R. Johnson

[7] P.Avgustinov, A.S.Christensen ,L. Hendren, S. Kuzins, J. Lhot ´ak, O. Lhot ´ak, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble, "abc: An extensible AspectJ compiler," in International Conference on Aspect-Oriented Software Development (AOSD). ACM Press, 2005, pp. 87–98.

[8] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: a dynamic data race detector for multithreaded programs," ACM Transactions on Computer Systems, vol. 15, no. 4, pp. 391–411, 1997.

[9] E. Bodden and K. Havelund, "Racer: Effective race detection using AspectJ," in International Symposium on Software Testing and Analysis (ISSTA), Seattle, WA. New York, NY, USA: ACM, Jul. 2008, pp. 155–165.

[10] "The AspectJ home page." [Online]. Available: http://eclipse.org/aspectj/

◈◈◈