A Classification of Concurrency Bugs in Java Benchmarks by Developer Intent

M. Erkan Keremoglu, Serdar Tasiran, Tayfun Elmas

Center for Advanced Design Technologies @ Koc University
http://designtech.ku.edu.tr

Koç University, Istanbul, Turkey
Introduction and Objective

- Correctness Criteria For Concurrent Software:

  - **Race freedom:**
    Reads and writes totally ordered by synchronization operations.

  - **Atomicity:**
    No interference between a code block and other threads.

  - **Refinement:**
    Concurrent execution consistent with its sequential specification.

- Should pick correctness criterion that best expresses designer’s intent

- Wrong choice of criterion ➔ Only bug symptom treated
  ➔ Wrong fix: Bug remains, performance gets worse, ...

- We illustrate our point using Apache FTP Server concurrency bug from

  Mayur Naik, Alex Aiken, and John Whaley.  
  *Effective Static Race Detection for Java.*  
public void run() {
  ... 
  dereference m_request, m_writer, m_reader 
  and m_controlSocket to initialize the connection 
  ... 
  READ NEXT REQUEST 
}

PERFORM REQUESTED ACTION (USING REQUEST HANDLER’S FIELDS)

m_commandLine.equals("") { 
  continue; 
}

m_request.parse(commandLine);
if(!hasPermission()) {
  m_writer.send(530, "permission", null);
  continue;
}
// execute command 
service(m_request, m_writer);

public void close() {
  CLOSE CONNECTION 
  // or not
  CLEAR FIELDS (SET FIELDS TO NULL) 
  return;
  m_isConnectionClosed = true;
  ... 
  m_request = null;
  m_writer = null;
  m_reader = null;
  m_controlSocket = null;
  ... 
  These fields are shared
  Shared Fields
}
public void run() {
  ... 
  dereference m_request, m_writer, m_reader
  and m_controlSocket to initialize the connection
  ... 
  while(!m_isConnectionClosed){
    String commandLine = m_reader.readLine();
    if(commandLine == null) {
      break;
    }
    commandLine = commandLine.trim();
    if(commandLine.equals("")) {
      continue;
    }
    m_request.parse(commandLine);
    if(!hasPermission()) {
      m_writer.send(530, "permission", null);
      continue;
    }
    // execute command
    service(m_request, m_writer);
  }
  m_request = null;
  m_writer = null;
  m_reader = null;
  m_controlSocket = null;
}

class Thread
  // check whether already closed
  // or not
  synchronized(this) {
    if(m_isConnectionClosed)
      return;
    m_isConnectionClosed = true;
  }
  ... 
  m_request = null;
  m_writer = null;
  m_reader = null;
  m_controlSocket = null;
...
public void run() {

  2 ... 
  3 dereference m_request, m_writer, m_reader 
  4 and m_controlSocket to initialize the connection 
  5 ... 

  6 while(!m_isConnectionClosed){ 
    7 String commandLine = m_reader.readLine(); 
    8 if(commandLine == null) { 
      9   break; 
     10 } 
    11 commandLine = commandLine.trim(); 
    12 if(commandLine.equals("""")) { 
      13   continue; 
     14 } 
    15   m_request.parse(commandLine); 
    16   if(!hasPermission()) { 
      17   m_writer.send(530, "permission", null); 
      18   continue; 
     19 } 
    20 // execute command 
    21   service(m_request, m_writer); 
  22 }

public void close() {

  2 // check whether already closed 
  3 or not 
  4 synchronized(this) { 
    5    if(m_isConnectionClosed) 
    6       m_isConnectionClosed = true; 
    7 } 
  8 ... 

  9   m_request = null; 
 10   m_writer = null; 
 11   m_reader = null; 
 12   m_controlSocket = null; 
 ... 

- Make m_reader, m_request, ... volatile
- All races dissappear, but bug remains!
- Null Pointer Exception still there!
public void run() {

    while(!m_isConnectionClosed) {

        String commandLine = m_reader.readLine();
        if(commandLine == null) {
            break;
        }
        commandLine = commandLine.trim();
        if(commandLine.equals("")) {
            continue;
        }
        m_request.parse(commandLine);
        if(!hasPermission()) {
            m_writer.send(530, "permission", null);
            continue;
        }
        // execute command
        service(m_request, m_writer);
    }
}
To fix concurrency bug, select criterion that best reflects developer’s intent.
Ongoing Work

- Building higher-level concurrency bug benchmarks from practically used software.

- **VYRD+**
  - Race + Atomicity + Refinement Checker

  *T. Elmas, S. Tasiran, and S. Qadeer [PLDI’05]*
  *Vyrd: Verifying Concurrent Programs by Runtime Refinement-Violation Detection.*

- **Location Pairs Coverage Metric**
  - Investigating correspondence to concurrency errors.
  - Building coverage measurement tool.

  *S.Tasiran, T. Elmas, G. Bolukbasi, M. E. Keremoglu [FATES’05]*