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.
ACM SIGPLAN 2006 Conf. on Programming Language Design and Implementation (PLDI 2006), Ottawa, Canada, June 2006.

APACHE FTP SERVER CONNECTION HANDLING

```
public void run() {
2  ...
3 dereference m_request, m_writer, m_reader
4 and m_controlSocket to initialize the connection
5 ...
READ NEXT REQUEST


PERFORM REQUESTED ACTION (USING
REQUEST HANDLER'S FIELDS)

}
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() {
CLOSE CONNECTION
  //or not
CLEAR FIELDS (SET FIELDS
TO NULL)
5       return;
6    m_isConnectionClosed = true;
7  }
8  ...
```

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

These fields are shared

Shared Fields

Connection thread — MANIFESTATION — Close Thread

```
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 }
```

Detects that connection is alive

```
public void close() {
2 // check whether already closed
   //or not
3 synchronized(this) {
4   if(m_isConnectionClosed)
      return;
    isConnectionClosed = true;
  }
8 ...
9   m_request = null;
10 m_writer = null;
11 m_reader = null;
12 m_controlSocket = null;
   ...
```

Null Pointer Exception

```
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
  //or not
3 synchronized(this) {
4   if(m_isConnectionClosed)
5     return;
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!

Solution 2

```
public void run() {
2 ...
3 synchronized(this) {
4  init();
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 }
```

```
synchronized(this) {
  if(m_isConnectionClosed)
    break;
  else
    cLine = m_reader.readLine();
}
```

```
synchronized(this) {
  if(m_isConnectionClosed)
    break;
  else{
    m_request.parse(cLine);
    if(!hasPermission()) {
     m_writer.send(530);
     continue;
    }
service(m_request, m_writer);
}
```

Solution 1

```
while(!m_isConnectionClosed) {
  synchronized(this) {
    if(m_isConnectionClosed)
      break;
    else
      cLine = m_reader.readLine();
  }
  if(commandLine == null) {
     break;
  }
  cLine = cLine.trim();
  if(commandLine.equals("")) {
      continue;
  }
  m_request.parse(cLine);
  if(!hasPermission()) {
       m_writer.send(530);
       continue;
  }
  service(m_request, m_writer); }
}
```

Solution 2

```
synchronized(this) {
  if(m_isConnectionClosed)
    break;
  else
    cLine = m_reader.readLine();
}
```

```
synchronized(this) {
  if(m_isConnectionClosed)
    break;
  else{
    m_request.parse(cLine);
    if(!hasPermission()) {
     m_writer.send(530);
     continue;
    }
  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]
  A Novel Test Coverage Metric for Concurrently-Accessed Software Components.