An Effective Dynamic Analysis for Detecting Generalized Deadlocks


Pallavi Joshi* Mayur Naik† Koushik Sen* David Gay†

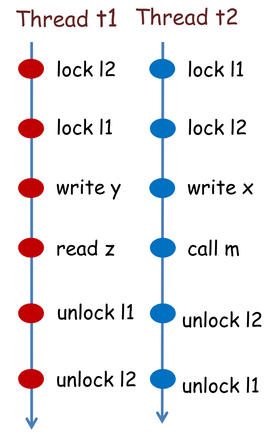
* UC Berkeley † Intel Research Berkeley

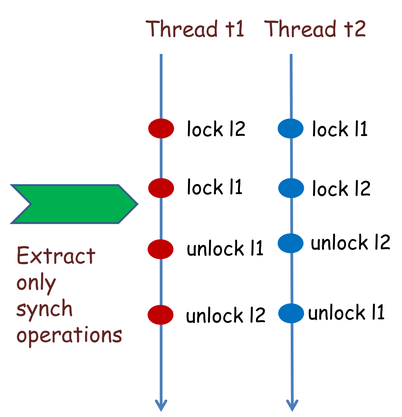

Motivation

- Deadlocks are a common problem in today's multithreaded software
  - 6,500/198,000 of bug reports in Sun's bug database are deadlocks
- Two categories of deadlocks
  - Resource deadlocks
  - Communication deadlocks
- Most of previous work has exclusively focused on resource deadlocks
- In this work, we address all kinds of deadlocks

Our Approach (CHECKMATE)

- Observe a multithreaded program execution
- Retain only the synchronization operations observed during execution
  - Throw away all other operations like memory update operations and method calls
- Create a program from the retained operations (trace program)
- Model check trace program
  - Trace program is usually much faster to model check than the original program





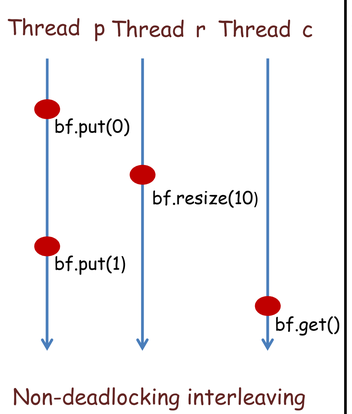

```
public class TraceProgram{
    static Object l1 = new Object();
    static Object l2 = new Object();

    static Thread main = new Thread(){
        public void run(){
            t1.start(); t2.start();
        }
    };
    static Thread t1 = new Thread(){
        public void run(){
            synchronized(l2){
                synchronized(l1) {
                }
            }
    };
    static Thread t2 = new Thread(){
        public void run(){
            synchronized(l1){
                synchronized(l2) {
                }
            }
    };
    public static void main(String[] args){
        main.start();
    }
}
```

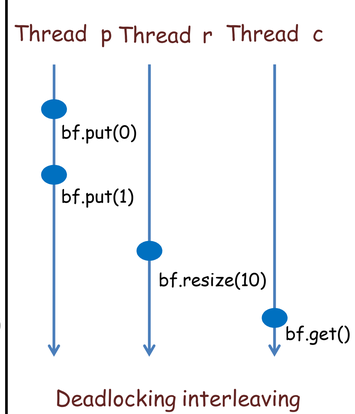
Example

```
public class MyBuffer{
    private List buf = new ArrayList();
    private int cursize = 0, maxsize;
    public MyBuffer (int max) {
        maxsize = max;
    }
    public synchronized void put(Object e){
        while(isFull()) wait();
        buf.add(e); cursize++;
        notify();
    }
    public Object get(){
        Object e;
        synchronized(this){
            while(isEmpty()) wait();
            e = buf.remove(0);
        }
        synchronized(this){
            if(isFull()){
                cursize--; notify();
            } else {
                cursize--;
            }
        }
        return e;
    }
```

```
public synchronized void resize(int
    max){
        maxsize = max;
}
public synchronized boolean isFull(){
        return (cursize >= maxsize);
}
public synchronized boolean isEmpty(){
        return (cursize == 0);
}
public static void main(String[] args){
    final MyBuffer bf = new  MyBuffer (1);
    Thread p = (new Thread(){
        public void run(){
            bf.put(0); bf.put(1);
        }}).start();
    Thread r = (new Thread(){
        public void run(){
            bf.resize(10);
        }}).start();
    Thread c = (new Thread(){
        public void run(){
            bf.get();
        } }).start();
  }
}
```



Non-deadlocking interleaving



Deadlocking interleaving

```
public class TraceProgram{
    static Object bf = new Object();
    static boolean isFull;
    static Thread main = new Thread(){
        public void run(){
            isFull = false;
            p.start(); r.start(); c.start();
        }
    };
    static Thread p = new Thread(){
        public void run(){
            //enter bf.put(0)
            synchronized(bf){
                if(isFull){
                    synchronized(bf) { bf.wait();}
                }
                isFull = true;
                bf.notify();
            }
            //leave bf.put(0)
            //enter bf.put(1)
            synchronized(bf){
                if(isFull){
                    synchronized(bf) { bf.wait();}
                }
                bf.notify();
            }
            //leave bf.put(1)
        }
    };
```

```
    static Thread r = new Thread(){
        public void run(){
            //enter bf.resize(10)
            synchronized(bf){
                isFull = false;
            }
            //leave bf.resize(10)
        }
    };
    static Thread c = new Thread(){
        public void run(){
            //enter bf.get()
            synchronized(bf){
                if(isFull){
                    synchronized(bf){
                        bf.notify();
                    }
                }
            }
            //leave bf.get()
        }
    };
    public static void main(String[] args){
        main.start();
    }
}
```

Implementation

- Implemented our technique in a prototype tool for Java called CHECKMATE
- Experimented with a number of Java libraries and applications
  - log4j, pool, felix, lucene, jgroups, jruby, …

Results

Program Name	# condition annotations	Original Program LOC	Trace Program LOC	Original Program Runtime	Time to generate trace program	JPF time (original program)	JPF time (trace program)	# error traces	Potential Deadlocks (#C/#R)	Confirmed Deadlocks (#C/#R)	Known Deadlocks (#C/#R)
groovy	1	45,796	59	0.118s	1s	> 15m	0.4s	5	1/0	1/0	1/0
log4j	2	48,023	238	0.116s	1.2s	-	9s	198	2/0	1/0	1/0
pool (I)	4	48,024	164	0.116s	1.2s	> 15m	2s	32	1/0	1/0	1/0
pool (II)	4	48,024	215	0.123s	1.5s	> 15m	2s	27	1/0	1/0	1/0
felix	4	73,512	125	0.173s	2.6s	-	-	-	-	1/0	1/0
lucene (I)	9	68,311	507	0.230s	3.1s	> 15m	0.8s	0	0/0	0/0	1/0
lucene (II)	9	81,071	4,193	0.296s	3.4s	> 15m	13s	10	1/0	0/0	1/0
jgroups (v1)	12	92,934	152	0.228s	4.1s	-	3s	43	2/0	1/0	1/0
jigsaw	17	122,806	7,046	-	-	-	> 15m	2415	1/7	1/5	0/2
jruby	16	136,479	9,813	1.1s	13.4s	-	34s	262	1/0	1/0	1/0
jgroups (v2)	15	160,644	3,185	9.89s	20s	-	> 15m	166	1/0	0/0	1/0
java logging	0	43,795	251	0.177s	2s	> 15m	57s	1202	0/2	0/1	0/1
dbcp	0	90,821	522	0.74s	5.9s	-	13s	310	0/2	0/2	0/2
swing	0	264,528	3,995	0.96s	18.6s	-	> 15m	721	2/1	0/1	0/1

#C is no. of communication deadlocks
#R is no. of resource deadlocks