

# Taming Race Conditions in Multithreaded Programs (with Sun Studio Data Race Detection Tool)

Yuan Lin  
Systems Group  
Sun Microsystems Inc.

# One of History's Worst Software Bugs

1985-1987 -- Therac-25 medical accelerator

- “A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities.”
- “Because of a subtle bug called a “race condition,” a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position.”
- “At least five patients die; others are seriously injured.”

wired.com, <http://www.wired.com/news/technology/bugs/0,2924,69355,00.html>

# What is a Race Condition?

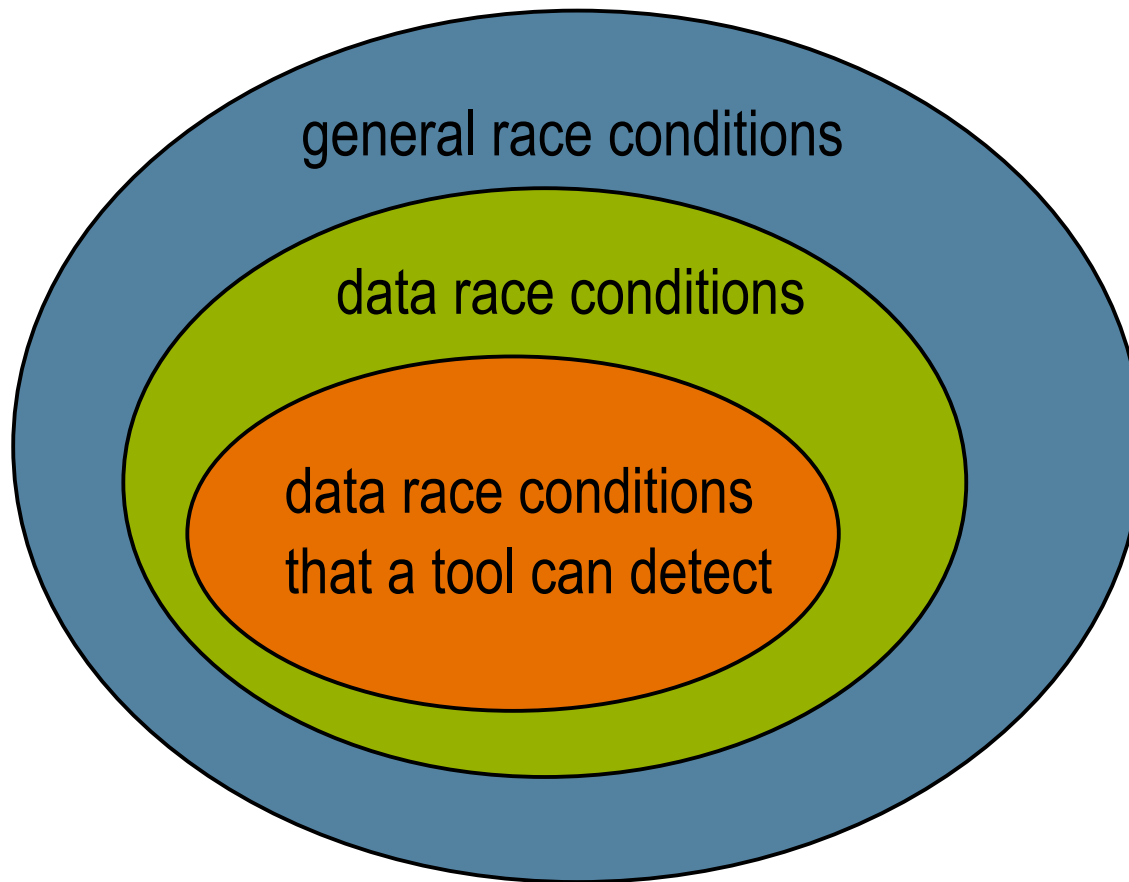
- A race condition happens when two concurrent actions performed on a shared resource are not properly synchronized.
- For example, a race condition may happen between
  - > two threads in one process that try to modify the same variable,
  - > two processes that try to modify the same file, or
  - > two shell scripts that try to copy over the same file.if the modifications are not coordinated correctly.

# Race Conditions

- Many programmers write parallel applications that have race conditions during the development cycle, either consciously or unconsciously.
  - > For example, 4 of 6 submissions to OpenMP Programming Contest (SC 2005) have bugs caused by race conditions. All four were disqualified.
- Race conditions may produce unpredictable program states.
- Race conditions are hard to detect with conventional debugging methods and tools.

A Special Tool is Needed.

# Different Kinds of Race Conditions



# Data Race Condition in MT Programs

- A data race condition in a multi-threaded program happens when two different threads
  - > access the same memory location
    - > at least of the accesses is a modification
  - > concurrently
  - > without holding any common exclusive locks

# Example:

A group of threads work together to find the prime numbers between 2 and N. Each thread finds the prime numbers within an assigned range [thr\_lower, thr\_upper], and stores primes found in a shared array set[] and updates the shared total count of prime numbers.

```
int total = 0;
int set[N];
void prime_work(int thr_lower, int thr_upper)
{
    for (int i=thr_lower; i<thr_upper; i++) {
        if (is_prime(i)) {
            total = total + 1;    // st1
            set[total] = i;      // st2
        }
    }
}
```

# Example: data race 1

Assume total is 4, then thread 1 executes total++ and thread 2 executes total++ concurrently. After the execution, the value of total may be either 5 or 6. If total is 5, then we loose one count.

```
int total = 0;
int set[N];
void prime_work(int thr_lower, int thr_upper)
{
    for (int i=thr_lower; i<thr_upper; i++) {
        if (is_prime(i)) {
            total = total + 1;    // st1
            set[total] = i;      // st2
        }
    }
}
```

## Example: data race 2

A thread may not see, at st2, the value it updates at st1. It may get the value another thread updates.

```
int total = 0;
int set[N];
void prime_work(int thr_lower, int thr_upper)
{
    for (int i=thr_lower; i<thr_upper; i++) {
        if (is_prime(i)) {
            total = total + 1;    // st1
            set[total] = i;      // st2
        }
    }
}
```

## Example: data race 3

If two threads see the same value of `total`, then they will overwrite the same element of array `set[]`! Some elements of array `set[]` may be defined multiple times, while some may not even be defined!

```
int total = 0;
int set[N];
void prime_work(int thr_lower, int thr_upper)
{
    for (int i=thr_lower; i<thr_upper; i++) {
        if (is_prime(i)) {
            total = total + 1;    // st1
            set[total] = i;      // st2
        }
    }
}
```

# Sun Data Race Detection Tool (DRDT)

- Automatically detects data races in a multi-threaded program.
- Uses compile-time instrumentation.
- Detection is done at run-time.
- Supported on Sparc/Solaris, x86/x64/Solaris/Linux.
- A component of Sun Studio Express.
- Free download available at

<http://developers.sun.com/prodtech/cc/downloads/express.jsp>

# Sun Data Race Detection Tool (DRDT)

- Programming schemes supported
  - > OpenMP
  - > POSIX Threads
  - > Solaris Threads
  - > ... and any mixture of the above ...
- Programming languages supported
  - > F95
  - > C
  - > C++
- Additional APIs provided to allow programmers to inform DRDT of user-defined synchronizations.

# Using DRDT – Step 1

- Compile your program for instrumentation.
  - > Add “-xinstrument=datarace” to your compiler/linker options.

```
% cc -xinstrument=datarace -mt a1.c a2.c a3.c
```

```
% cc -xinstrument=datarace -xopenmp omp1.c omp2.c
```

## Using DRDT – Step 2

- Run the application under “collect” with option “-r on”.
  - > (Similar to what you do using “collect” for performance analysis.)

```
% collect -r on a.out arg1 arg2
```

- > “collect” will create an experiment data file that stores the data race detection results.

# Using DRDT – Step 3

- Check the experimental result
  - > Use “er\_print” for command line interface.
    - > For a summary report,

```
% er_print -races test.1.er
```
    - > For a detailed report on one particular data race detected,

```
% er_print -rdetail 3 test.1.er
```
  - > Use “rdt” for graphic user interface.

```
% rdt test.1.er
```

# “er\_print” output

```
% er_print -races test.1.er
```

```
Total Races: 2 Experiment: test.1.er
```

```
Race #1, Vaddr: 0x212c0
```

```
Access 1: Read, work + 0x000000A0,  
line 42 in "pthr_prime.c"
```

```
Access 2: Write, work + 0x000000DC,  
line 44 in "pthr_prime.c"
```

```
Total Traces: 3
```

```
Race #2, Vaddr: 0x212c0
```

```
Access 1: Write, work + 0x000000DC,  
line 44 in "pthr_prime.c"
```

```
Access 2: Write, work + 0x000000DC,  
line 44 in "pthr_prime.c"
```

```
Total Traces: 2
```

```
% er_print -rdetail 2 test.1.er
```

```
Race #2, Vaddr: 0x212c0
```

```
Access 1: Write, work + 0x000000D4,  
line 44 in "pthr_prime.c"
```

```
Access 2: Write, work + 0x000000D4,  
line 44 in "pthr_prime.c"
```

```
Total Traces: 2
```

```
Trace 1
```

```
Access 1: Write  
work + 0x000000C4, line 44 in "pthr_prime.c"  
_lwp_start + 0x00000000
```

```
Access 2: Write  
work + 0x000000C4, line 44 in "pthr_prime.c"  
main + 0x000000C8, line 27 in "pthr_prime.c"  
_start + 0x00000108
```

```
Trace 2
```

```
Access 1: Write  
work + 0x000000D4, line 44 in "pthr_prime.c"  
_lwp_start + 0x00000000
```

```
Access 2: Write  
work + 0x000000D4, line 44 in "pthr_prime.c"  
main + 0x000000D8, line 32 in "pthr_prime.c"  
_start + 0x00000108
```

```
Detailed Race Events Info: Experiment: test.13.er Id: 2
```

Sun Studio Analyzer [test.1.er]

File View Timeline Help

Races Race Source Experiments

Summary Race Details

Data for Selected Race

Id: Trace #1

Vaddr: 0x21d00

Access 1

Type: Write

work + 0x00000208, line 51 in "pthr\_prime.c"

\_lwp\_start + 0x00000000

Access 2

Type: Write

work + 0x00000208, line 51 in "pthr\_prime.c"

main + 0x00000428, line 71 in "pthr\_prime.c"

\_start + 0x00000108

Race Accesses	Source File: /import/iropt2/y1140942/rdt/d
0	47. end = start + N/THREADS;
0	48. for (i = start; i < end; i++) {
0	49. if ( is_prime(i) ) {
8	50. primes[total] = i;
15	51. total++;
	52. }
	53. }
0	54. return NULL;
	55. }

Race Accesses	Source File: /import/iropt2/y1140942/rdt/d
0	47. end = start + N/THREADS;
0	48. for (i = start; i < end; i++) {
0	49. if ( is_prime(i) ) {
8	50. primes[total] = i;
15	51. total++;
	52. }
	53. }
0	54. return NULL;
	55. }

# Tips of Using DRDT

- Expect significant (> 50x) slowdown when running your application under DRDT.
  - > Use a small input data set.
- Check DRDT webpage for updates.
  - > [http://developers.sun.com/prodtech/cc/downloads/drdt/drdt\\_index.html](http://developers.sun.com/prodtech/cc/downloads/drdt/drdt_index.html)
- Read DRDT tutorial.
  - > <http://developers.sun.com/prodtech/cc/downloads/drdt/using.html>
- Check DRDT FAQ.
  - > <http://developers.sun.com/prodtech/cc/downloads/drdt/faq.html>
- Provide feedback or ask questions on the Sun Studio Tools forum.
  - > <http://forum.sun.com/jive/forum.jspa?forumID=309>

# What to Do After a Data-race is Found?

- 1) Check whether it is a false positive.
  - > A false positive is a reported data race that actually does not exist in the program.

# What to Do After a Data-race is Found?

- 1) Check whether it is a false positive.
  - > A false positive is a reported data race that actually does not exist in the program.
  - > No tool is perfect.
  - > For example, the tool may not understand the synchronizations in your program.

thread 1

```
my_lock();  
acct1 += x;  
my_unlock();
```

thread 2

```
my_lock();  
acct1 -= y;  
my_unlock();
```

\* A user implements his/her own lock and unlock routines.  
\* DRDT may not be able to recognize my\_lock() and my\_unlock() are lock routines, and reports data race between accesses on 'acct1'.

# What to Do After a Data-race is Found?

- 2) Check whether it is a benign race.
  - > The programmer may put data race there in order to get better performance, e.g. using a lock free algorithm.
    - > Yes? Are you sure? Check again!
  - > Programs with intentional data races are very difficult to get right.
    - > Are you relying on sequential consistency?

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,
                  const pthread_attr_t *attr,
                  void *(*start_routine)(void*),
                  void *arg);
```

```
void *work(void *arg)
{
    unsigned long myid = *(unsigned long *)arg;
    data[myid] = update(data[myid]);
    return NULL;
}
for (unsigned long i=0; i<THREADS; i++)
    pthread_create(tids[i], NULL, work, &i);
```

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine)(void*),  
                  void *arg);
```

```
void *work(void *arg)  
{  
    unsigned long myid = *(unsigned long *)arg;  
    data[myid] = update(data[myid]);  
    return NULL;  
}  
for (unsigned long i=0; i<THREADS; i++)  
    pthread_create(tids[i], NULL, work, &i);
```

A data race will be reported

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine)(void*),  
                  void *arg);
```

```
void *work(void *arg)  
{  
    unsigned long myid = *(unsigned long *)arg;  
    data[myid] = update(data[myid]);  
    return NULL;  
}  
for (unsigned long i=0; i<THREADS; i++)  
    pthread_create(tids[i], NULL, work, &i);
```

A data race will be reported

Bug!

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine)(void*),  
                  void *arg);
```

One possible fix

```
void *work(void *arg)  
{  
    (unsigned long)arg;  
    unsigned long myid = *(unsigned long *)arg;  
    data[myid] = update(data[myid]);  
    return NULL;  
}  
for (unsigned long i=0; i<THREADS; i++)  
    pthread_create(tids[i], NULL, work, &i;);
```

(void \*)i

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.
  - > A tool can detect only certain kinds of data races.

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.
  - > A tool can detect only certain kinds of data races.

thread 1

```
acct1 += x;  
  
acct2 -= x;
```

thread 2

```
acct1 += y;  
  
acct2 -= y;
```

thread 3

```
print acct1, acct2
```

Data Race! Accounts are not balanced!  
“account1 + account2” is not constant.

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.
  - > A tool can detect only certain kinds of data races.

thread 1

```
lock();  
acct1 += x;  
unlock();  
lock();  
acct2 -= x;  
unlock();
```

thread 2

```
lock();  
acct1 += y;  
unlock();  
lock();  
acct2 -= y;  
unlock();
```

thread 3

```
lock();  
print acct1, acct2  
unlock();
```

A wrong fix. No data race is reported.

But accounts are not balanced at certain stages!

“account1 + account2” is not constant.

# What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race condition!
  - > A data race could be a bug or caused by a bug.
  - > A tool can detect only certain kinds of data races.

thread 1

```
lock();  
acnt1 += x;  
  
acnt2 -= x;  
unlock();
```

thread 2

```
lock();  
acnt1 += y;  
  
acnt2 -= y;  
unlock();
```

thread 3

```
lock();  
print acnt1, acnt2  
unlock();
```

A correct fix.

# What to Do After a Data-race is Found?

- 4) Run the detection tool again after code change!
  - > A tool may not be able to detect all data races in one run.
  - > The code change may introduce or reveal another data race.

# Summary

- Bugs caused by race conditions are very difficult to detect by hand.
- Use Sun Studio DRDT to detect data race conditions in your multi-threaded programs.
- Fix the bug, not the race condition.

**Thank you!**

**Yuan Lin**

yuan.lin@sun.com