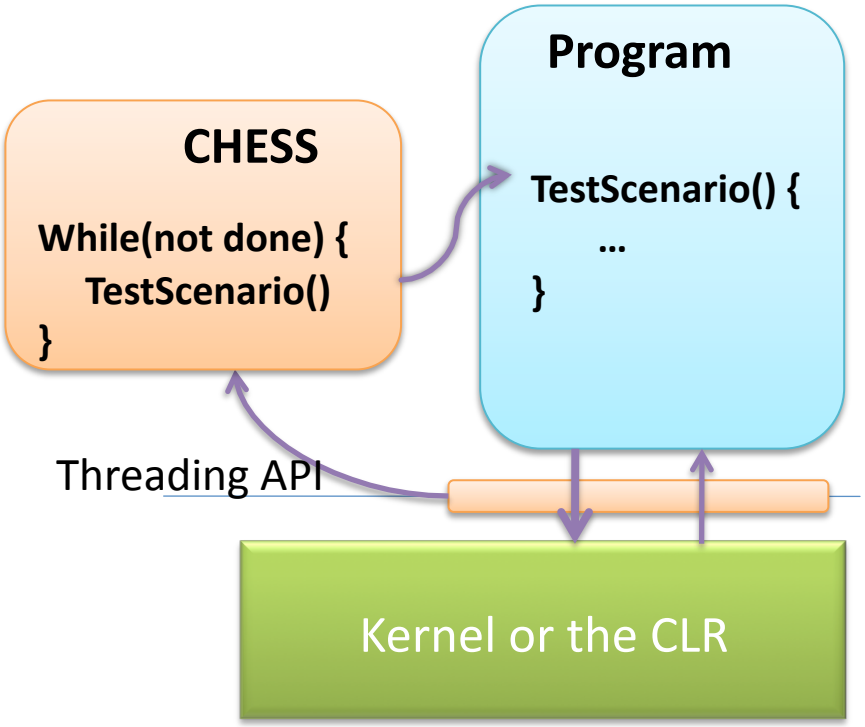
SCHEDULE EXPLORATION ALGORITHMS

A Brief Introduction to CHESS



Tester Provides a Test Scenario

CHESS runs the scenario in a loop

- Every run takes a different interleaving
- Every run is repeatable

CHESS 'hijacks' the scheduler

- Detour Win32 API calls
- To control and introduce nondeterminism

Enumerating thread interleavings

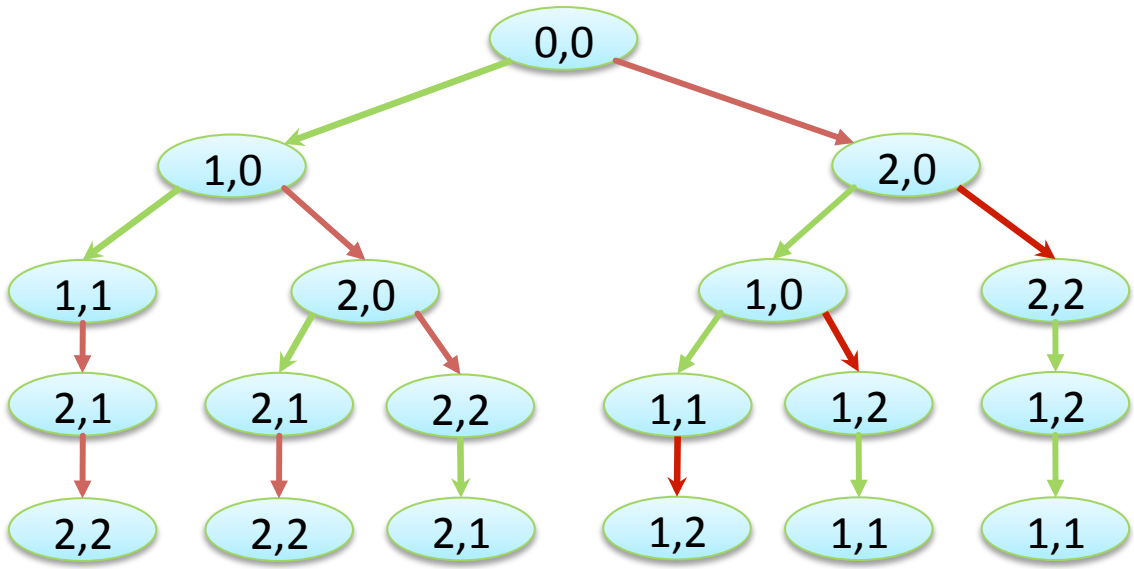
Thread 1

```
x = 1;
y = 1;
```

Thread 2

```
x = 2;
y = 2;
```

```
x = 1;
y = 1;
x = 2;
y = 2;
```



Running CHESS

- mchess <test.exe> /tc:<testfn> /mp:n

Visual Studio Command Prompt (2010)

```
\Debug>mchess /tc:ThreadSafetyChecker.Test1 /mp:7 ThreadSafetyChecker.exe
WARNING: Race Detection Disabled. Races May Hide Bugs.
Tests: 1 Threads: 3 ExecSteps: 35 Time: 0.156
Tests: 2 Threads: 3 ExecSteps: 35 Time: 0.156
Tests: 3 Threads: 3 ExecSteps: 35 Time: 0.172
Tests: 4 Threads: 3 ExecSteps: 35 Time: 0.172
Tests: 5 Threads: 3 ExecSteps: 35 Time: 0.188
Tests: 6 Threads: 3 ExecSteps: 35 Time: 0.188
Tests: 7 Threads: 3 ExecSteps: 35 Time: 0.203
Tests: 8 Threads: 3 ExecSteps: 35 Time: 0.203
Tests: 9 Threads: 3 ExecSteps: 35 Time: 0.203
Tests: 10 Threads: 3 ExecSteps: 35 Time: 0.219
Tests: 20 Threads: 3 ExecSteps: 35 Time: 0.250
Tests: 30 Threads: 3 ExecSteps: 35 Time: 0.297
Tests: 40 Threads: 3 ExecSteps: 35 Time: 0.344
Tests: 50 Threads: 3 ExecSteps: 35 Time: 0.390
Tests: 60 Threads: 3 ExecSteps: 35 Time: 0.422
Tests: 70 Threads: 3 ExecSteps: 35 Time: 0.468
Tests: 80 Threads: 3 ExecSteps: 35 Time: 0.515
Tests: 90 Threads: 3 ExecSteps: 35 Time: 0.562
Tests: 100 Threads: 3 ExecSteps: 35 Time: 0.593
Tests: 200 Threads: 3 ExecSteps: 35 Time: 1.46
^C
C:\Users\madanm\Desktop\CSEP506\assignments\Assignment 3\ThreadSafetyChecl
\Debug>
```

Repro using CHESS

- Mchess /repro …
  - Runs the last schedule explored

```
Tests: 100 Threads: 3 ExecSteps: 35 Time: 0.593
Tests: 200 Threads: 3 ExecSteps: 35 Time: 1.46
^C
C:\Users\madanm\Desktop\CSEP506\assignments\Assignment 3\ThreadSafetyChecker\bi
\Debug>mchess /repro /tc:ThreadSafetyChecker.Test1 /mp:7 ThreadSafetyChecker.ex
 > out
Tests: 1 Threads: 3 ExecSteps: 35 Time: 0.203

C:\Users\madanm\Desktop\CSEP506\assignments\Assignment 3\ThreadSafetyChecker\bi
\Debug>
```

Use Concurrency Explorer

- Mchess /repro /trace
- ConcurrencyExplorer trace

Concurrency Explorer II

File Font Help

tid	op	obj
1	TASK_FORK	2
1	TASK_RESUME	2
1	TASK_FORK	3
1	TASK_RESUME	3
1	Thread.Join(-1) (BLO...	3
2	TASK_BEGIN	2
2	vol.read	512
2	vol.read	512
2	Interlocked.Compare...	512
2	vol.read	512
Pre	emp	tion
3	TASK_BEGIN	3
3	vol.read	512
3	vol.read	512
3	Interlocked.Compare...	512
3	vol.read	512
3	vol.read	513
3	vol.write	513
3	vol.read	512
3	vol.read	512
3	vol.read	513
Pre	emp	tion
2	vol.read	513
2	vol.write	513
Pre	emp	tion

ChessTask1: Program.cs, line 171

```
            Observations.ThreadName.Value = "1";

            sw.Set();
            int wh;
            sw.Wait(out wh);
        }));

        Thread t2 = new Thread(new ThreadStart(() =>
        {
            Observations.ThreadName.Value = "2";

            sw.Set();
            int wh;
            sw.Wait(out wh);
        }));

        t1.Start();
        t2.Start();
        t2.Join();
        t1.Join();

        return Helper.CheckThreadSafety(Observations.End(
    }


    public static void Main(string[] args)
    {
        Test1.Startup(args);
        Test1.Run();
        Test1.Shutdown();
    }
}
```

ChessTask2: Program.cs, line 57

```
    public static void StartObservations()
    {
        m_memstream.SetLength(0);
    }

    public static void ObserveCall(string method, params string[] vals
    {
        if (m_serialMode) Monitor.Enter(m_serialModeLock);
        ObserveEvent(ThreadName.Value, method, "call", vals);
    }

    public static void ObserveReturn(string method, params string[]
    {
        ObserveEvent(ThreadName.Value, method, "ret", vals);
        if (m_serialMode)
            Monitor.Exit(m_serialModeLock);
        else
            m_numReturns++;
    }

    public static string EndObservations()
    {
        m_memstream.Seek(0, SeekOrigin.Begin);
        StreamReader st = new StreamReader(m_memstream);
        return st.ReadToEnd();
    }
}

public class MyManualResetEventWrapper
{
    MyManualResetEvent m_mre = null;
```

What thread schedules are explored

- /mp:n => all explorations with n preemptions
- Schedule space is exponential in n

Randomized Delay Insertion

- Insert delays to increase interleaving coverage
- Randomized algorithm for inserting delays
  - Probabilistic guarantees for finding race conditions
  - Scalable and effective

Delay Insertion

Parent

```
void* p = 0;
CreateThd(child);
p = malloc(…);
```

Child

```
Init();
DoMoreWork();
p->f ++;
```

Delay Insertion

Parent

```
void* p = 0;
RandDelay();
CreateThd(child);
RandDelay();
p = malloc(…);
```

Child

```
Init();
RandDelay();
DoMoreWork();
RandDelay();
p->f ++;
```

1. Instrument random delays

Delay Insertion

Parent

```
void* p = 0;
RandDelay();
CreateThd(child);
```

Child

```
Init();
RandDelay();
DoMoreWork();
RandDelay();
p->f ++;
```

```
RandDelay();
p = malloc(…);
```

1. Instrument random delays
2. Lucky => bug

Delay Insertion

Parent

```
void* p = 0;
RandDelay();
CreateThd(child);
```

Child

```
Init();
RandDelay();
DoMoreWork();
```

1. Instrument random delays
2. Lucky => bug
3. Unlucky => no bug

```
RandDelay();
p = malloc(…);
```

```
RandDelay();
p->f ++;
```

Delay Insertion Algorithm

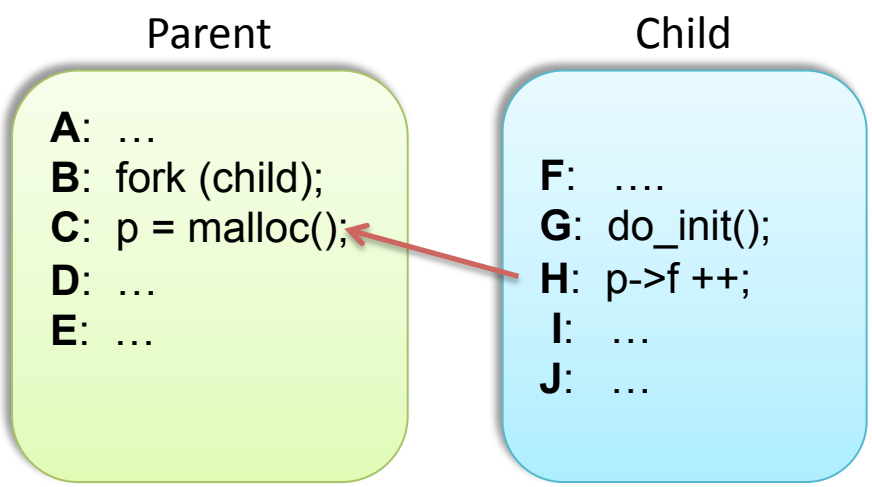
- Maintain a priority per thread
  - Different from OS priority
- Periodically assign random priorities
- Consistently delay the lowest priority thread
- Sometimes change the priority of the lowest priority thread

Bug Depth Definition

- Bug Depth = number of ordering constraints sufficient to find the bug
- Best explained through examples

A Bug of Depth 1

- Bug Depth = no. of ordering constraints sufficient to find the bug



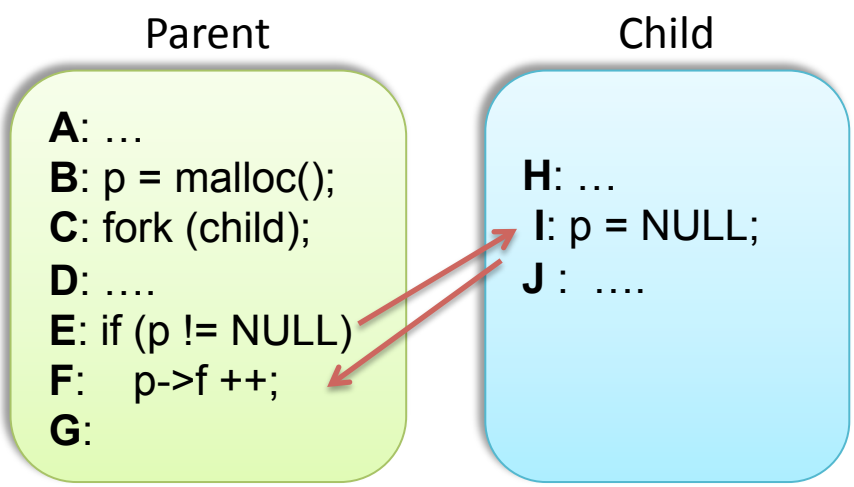
Possible schedules

A B C D E F G H I J ✓
A B F G H C D E I J ✗
A B F G C D E H I J ✓
A B F G C H D E I J ✓
A B F G H I J C D E ✗
…

- Probability of bug >= 1/n
  - n: no. of threads (~ tens)

A Bug of Depth 2

- Bug Depth = no. of ordering constraints sufficient to find the bug



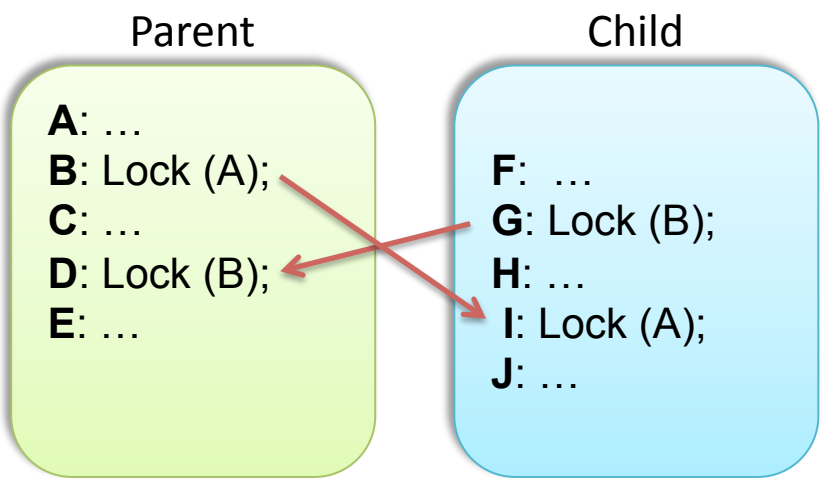
Possible schedules

A B C D E F G H I J ✓
A B C D E H I J F G ✗
A B C H I D E G J ✓
A B C D H E F I J G ✓
A B C H D E I J F G ✗
…

- Probability of bug >= 1/nk
  - n: no. of threads (~ tens)
  - k: no. of instructions (~ millions)

Another Bug of Depth 2

- Bug Depth = no. of ordering constraints sufficient to find the bug



- Probability of bug >= 1/nk
  - n: no. of threads (~ tens)
  - k: no. of instructions (~ millions)