



Multithreaded programs

The Good, the Bad and the Buggy



Mamy Ratsimbazafy
mamy@numforge.co

Weave
<https://github.com/mratsim/weave>



Hello!

I am Mamy Ratsimbazafy

During the day Blockchain/Ethereum 2 developer (in Nim)

During the night, multithreading, deep learning,
cryptography (in Nim) and data scientist (in Python)

You can contact me at mamy@numforge.co

Github: mratsim

Twitter: m_ratsim





Why and why not multithreading?



- Why: Time is precious (~3 hours to generate on 18 cores)
 - $10800\text{s} * 18 = 194400\text{s}$
- But: Use logarithmic data structures: $\log_2(194400) = 17.56\text{s}$



My goal today

Help your inner cat deal with threads





Motivating examples (1/3)

A working channel output:

What if I told you you can't reliably debug echo?

<https://github.com/nim-lang/Nim/issues/13936>

Araq commented 7 hours ago

Member

Works for me on Windows, here is the output:

```
Sender sent: 42
          Receiver got: 42
          Receiver got: 53
          Receiver got: 64

Sender sent: 53
          Receiver got: 75
          Receiver got: 86
          Receiver got: 97

Sender sent: 64
          Receiver got: 75
          Receiver got: 86
          Receiver got: 97

Sender sent: 75
          Receiver got: 75
          Receiver got: 86
          Receiver got: 97

Sender sent: 86
          Receiver got: 75
          Receiver got: 86
          Receiver got: 97

Sender sent: 97
          Receiver got: 108
          Receiver got: 119
          Receiver got: 130

Sender sent: 108
          Receiver got: 108
          Receiver got: 119
          Receiver got: 130

Sender sent: 119
          Receiver got: 108
          Receiver got: 119
          Receiver got: 130

Sender sent: 130
          Receiver got: 108
          Receiver got: 119
          Receiver got: 130

Sender sent: 141
          Receiver got: 141
```

Time.travel() ?

1



Motivating examples (2/3)

Unintuitive low-level behavior

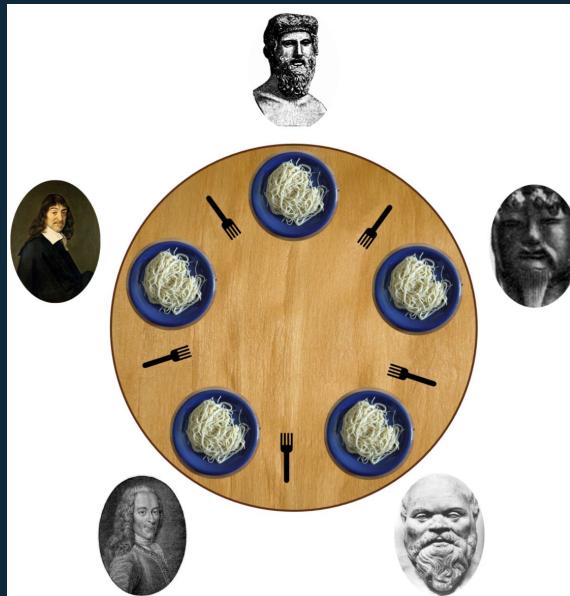
```
1  var A, B = 0
2
3  proc thread1() =
4      A = 1
5      echo B
6
7  proc thread2() =
8      B = 1
9      echo A
10 |
```

This can print 11, 10, 01 but unintuitively it can also display 00

Motivating examples (3/3)

Dining Philosophers

Deadlock and Starvation





Agenda



- ◊ Target audience and example threaded programs
- ◊ Debugging multithreaded programs
- ◊ Designing to reduce the potential multithreaded bug surface
- ◊ Correct-by-constructions programs



1

Target audience

And example threaded programs



Concurrent Data Structures

- ◊ A (hash)-table that allow access from multiple threads.
- ◊ An event bus (i.e. multithreaded publish/subscribe)
- ◊ An object pool for game objects, for example particles.
- ◊ A tree-like data structure, on a GPU, to accelerate raytracing (and make ray collisions logarithmic).





Concurrent systems

- ◊ Transactional code: “commit-rollback”
 - ◊ Collecting metrics and/or logs from multiple threads
 - ◊ Building a service architecture
 - User Interface
 - Sound
 - Graphics
 - Networking, File, IO
 - ... (physics for a game, JIT for a web browser, logs)
- 



2

Debugging

Multithreaded programs



Tests are not enough!

- ◇ This can be tested for all inputs on a single-core

```
1 proc foo(a: int16) =  
2 | discard  
3
```

- ◇ But on 4-core, it's $(2^{16})^4 = 2^{64}$ states

```
: 264 = 18 446 744 073 709 551 616
```

- ◇ Way to many states to check exhaustively

- ◇ Also threads are not deterministic so bugs are hard to reproduce



Introducing AOP

Assertion-oriented programming

```
# Worker - Tasks handling
# ----

proc restartWork*() =
    preCondition: myThefts().outstanding == WV_MaxConcurrentStealPerWorker
    preCondition: myTodoBoxes().len == WV_MaxConcurrentStealPerWorker

    # Adjust value of outstanding by MaxSteal-1, the number of steal
    # requests that have been dropped:
    # outstanding = outstanding - (MaxSteal-1) =
    #           = MaxSteal - MaxSteal + 1 = 1

myThefts().outstanding = 1 # The current steal request is not fully fulfilled yet
myWorker().isWaiting = false
myThefts().dropped = 0
```

```
16
17 # Signals
18 #
19
20 proc detectTermination*() {.inline.} =
21     preCondition: myID() == RootID
22     preCondition: myWorker().leftIsWaiting and myWorker().rightIsWaiting
23     preCondition: not workerContext.runtimeIsQuiescent
24
25 debugTermination:
26     log("">>>> Worker %2d detects termination <<<\n", myID())
27
28 workerContext.runtimeIsQuiescent = true
29
30 proc asyncSignal(fn: proc (_: pointer) {.nimcall, gcsafe.}, chan: var ChannelSpscSinglePtr[T]
31     ## Send an asynchronous signal 'fn' to channel 'chan'
32
```



Assertion-Oriented Programming

- ◊ Gate them behind “defined” or {.strdefine.} and display as much context as possible.
- ◊ Remember, log order between 2 threads is random
- ◊ but logs produced by 1 thread is (sequentially consistent)

Rabbit hole: Serializability - <https://en.wikipedia.org/wiki/Serializability>

```
83  result = quote do:
84    .noSideEffect.:
85    when compileOption("assertions"):
86      assert' predicate', 'debug' & $`values` & " [Worker " & `myID` & "]\\n"
87    elif defined(WLAsserts):
88      if unlikely(not(`predicate`)):
89        raise newException(AssertionError, 'debug' & $`values` & '\\n')
90
91  # A way way to get the caller function would be nice.
92
93 template precondition*(require: untyped) =
94  ## Optional runtime check before returning from a function
95  assertContract("pre-condition", require)
96
97 template postCondition*(ensure: untyped) =
98  ## Optional runtime check at the start of a function
99  assertContract("post-condition", ensure)
100
101 template ascertain*(check: untyped) =
102  ## Optional runtime check in the middle of processing
103  assertContract("transient condition", check)
```



Introducing LOP

Log-oriented programming

```

E memory_pools.nim x
weave > memory > E memory_pools.nim > ...
299 func collect(arena: var Arena, force: bool) =
300   ## Collect garbage from the arena
301   ## We only move localFree to free when it's 0(1)
302   ## except on thread teardown
303
304 let beforeCollect(_used:) = arena.meta.used
305
306 if not arena.meta.localFree.isNil:
307   if likely(arena.meta.free.isNil):
308     # Fast path
309     arena.meta.free = arena.meta.localFree
310     arena.meta.localFree = nil
311   elif force:
312     # Very slow path: only on thread teardown
313     for memBlock in arena.meta.localFree:
314       guardedAccess(memBlock):
315         arena.meta.free.prepend(memBlock)
316         arena.meta.localFree = nil
317       debugMem:
318         log("Arena 0x%.08x - TID %d - collecting localFree, slow path (reclaimed %d)\n",
319             arena.addr, arena.meta.threadID, arena.blocks.len - arena.meta.used)
320
321 debugMem:
322 log("Arena 0x%.08x - TID %d - collect, remoteFree.peek() %d blocks (%d used or pending)\n",
323     arena.addr, arena.meta.threadID.load(moRelaxed), arena.meta.remoteFree.peek(), arena.meta.used)
324
325 # TODO: while in the MPSC queue, the memory is not poisoned
326 var first, last: ptr MemBlock
327 let count = arena.meta.remoteFree.tryRecvBatch(first, last)
328
329 debugMem:
330 log("Arena 0x%.08x - TID %d - collect, batched remoteFree %d blocks (%d blocks used)\n",
331     arena.addr, arena.meta.threadID.load(moRelaxed), count, arena.meta.used - count)
332
333 if count > 0:
334   if arena.meta.free.isNil:
335     arena.meta.free = first
336     last.next.store(nil, moRelaxed)
337   else:
338     guardedAccess(arena.meta.free):
339       arena.meta.free.prepend(last)
340     arena.meta.used -= count
341 when WV_SanitizeAddr:
342   var cur = first
343   while true:
344     if cur == last:
345       poisonMemRegion(cur, WV_MemBlockSize)
346       break
347     let next = cur.next.load(moRelaxed)
348     poisonMemRegion(cur, WV_MemBlockSize)
349     cur = cast[ptr MemBlock](next)
350
351 debugMem:
352 log("Arena 0x%.08x - TID %d - collected garbage, reclaimed %d blocks (%d used)\n",
353     arena.addr, arena.meta.threadID.load(moRelaxed), beforeCollect - arena.meta.used, arena.meta.used)

```



```

E victims.nim x
weave > E victims.nim > ...
escalcTask: task_stop > task_end
profilerSendRecvTask:
  if upperSplit.hasFuture:
    # This task has a future so it depends on both splitted tasks.
    let fvNode = newFlowvarNode(upperSplit.futureSize)
    fvNode.next = cast[FlowvarNode](task.futures)
    task.futures = cast[pointer](fvNode)
    # Redirect the result channel of the upperSplit
    LazyFV:
      cast[ptr ptr LazyFlowVar](upperSplit.data.addr)[] = fvNode.lfv
    EagerFV:
      cast[ptr ptr ChannelLSPCSingle](upperSplit.data.addr)[] = fvNode.chan
    fvNode.next = cast[FlowvarNode](task.futures)
    task.futures = cast[pointer](fvNode)
    # Don't share the required futures with the child
    upperSplit.futures = nil
  debugSplit:
    let steps = (upperStop.stop-upperSplit.start + upperSplit.stride-1) div upperSplit.stride
    log("Worker %d: Sending [%ld, %ld] to worker %d (%d steps) (hasFuture: %d, dependsOnFutures: 0x%.08x)\n", i,
        req.send(upperSplit)
  incCounter(loopSplit)
  debug:
    let steps = (task.stop-task.cur + task.stride-1) div task.stride
    log("Worker %d: Continuing with [%ld, %ld] (%d steps) (hasFuture: %d, dependsOnFutures: 0x%.08x), myID(), %d\n", i,
        myID(), steps)
  parallelForIn:
weave > parallel_forIn > ...
loadBalance(Weave)
debugLog: log("Worker %d: Finished loop task 0x%.08x (iterations [%ld, %ld]) (futures: 0x%.08x)\n", myID(), i,
block: # Wait for the child loop iterations
while not this.futures.isNil:
  let fvNode = cast[FlowvarNode](this.futures)
  this.futures = cast[pointer](fvNode.next)
LazyFV:
  let dummyFV = cast[Flowvar[bool]](fvNode.lfv)
EagerFV:
  let dummyFV = cast[Flowvar[bool]](fvNode.chan)
debugSplit: log("Worker %d: loop task 0x%.08x (iterations [%ld, %ld]) waiting for the remainder\n", myID())
let isLastIter = sync(dummyFV)
assert: not isLastIter
debugSplit: log("Worker %d: loop task 0x%.08x (iterations [%ld, %ld]) complete\n", myID(), this.fn, this.str,
# The "sync" in the merge statement should have recycled the flowvar channel already
# For LazyFlowvar, the LazyFlowvar itself was allocated on the heap, so we need to recycle it as well
# 2 dealloc for eager FV and 3 for Lazy FV
recycleFVN(fvNode)
proc parallelForSplitted(index, start, stop, stride, captured, capturedTy, dependsOnEvent, body: NimNode): NimNode
  ## In case a parallelFor depends on an iteration event indexed by the loop variable
  ## we can't use regular parallel loop with lazy splitting

```

Log-oriented programming

Semantic logs

- debug, debugSplit, debugMem



```
memory_pool.nim
wave> memory > memory_pool.nim ...
299 func collect(arena: var Arena, force: bool) =
300     # Collect garbage memory in the arena
301     # We only move localFree to free when it's 0
302     # except on thread teardown
303
304 let beforeCollect {used} = arena.meta.used
305
306 if not arena.meta.localFree.isNil:
307     if likely(arena.meta.free.isNil):
308         # Fast path
309         arena.meta.free = arena.meta.localFree
310         arena.meta.localFree = nil
311     elif force:
312         # Very slow path: only on thread teardown
313         for memBlock in arena.meta.localFree:
314             guardedAccess(memBlock):
315                 arena.meta.free.prepend(memBlock)
316                 arena.meta.localFree = nil
317             debugMem:
318                 log("Arena: 0x%08x - TID %d - collecting localFree, slow path (reclaimed %d)\n",
319                     arena.addr, arena.meta.threadID.load(moRelaxed), arena.meta.remoteFree.peek(), arena.meta.used)
320
321 debugMem:
322     log("Arena: 0x%08x - TID %d - collect, remoteFree.peek() %d blocks (%d used or pending)\n",
323         arena.addr, arena.meta.threadID.load(moRelaxed), arena.meta.remoteFree.peek(), arena.meta.used)
324
325 # TODO: while in the MPSQ queue, the memory is not poisoned
326 var first, last: ptr MemBlock
327 let count = arena.meta.remoteFree.tryRecvBatch(first, last)
328
329 debugMem:
330     log("Arena: 0x%08x - TID %d - collect, batched remoteFree %d blocks (%d blocks used)\n",
331         arena.addr, arena.meta.threadID.load(moRelaxed), count, arena.meta.used - count)
332
333 if count > 0:
334     if arena.meta.free.isNil:
335         arena.meta.free = first
336         last.nextStore(nil, moRelaxed)
337     else:
338         guardedAccess(arena.meta.free):
339             arena.meta.free.prepend(last)
340         arena.meta.used -= count
341         when MW_SanitizeAddress:
342             var cur = first
343             while cur != last:
344                 if cur == last:
345                     poisonMemRegion(cur, MW_MemBlockSize)
346                     break
347                 let next = cur.next.load(moRelaxed)
348                 poisonMemRegion(cur, MW_MemBlockSize)
349                 cur = cast[ptr MemBlock](next)
350
351 debugMem:
352     log("Arena: 0x%08x - TID %d - collected garbage, reclaimed %d blocks (%d used)\n",
353         arena.addr, arena.meta.threadID.load(moRelaxed), beforeCollect - arena.meta.used, arena.meta.us ...

wave> memory > memory_pool.nim ...
271
272     profileSendRecvTask()
273     if upperSplit.hasFuture:
274         # The task has a Future so it depends on both splitted tasks.
275         let fvNode = newFlowvarNode(upperSplit.futureSize)
276         # Redirect the result channel of the upperSplit
277         LazyFV:
278             cast[ptr LazyFlowVar] (upperSplit.data.addr)[] = fvNode.lfv
279             EagerFV:
280                 cast[ptr ChannelSPSCSingle] (upperSplit.data.addr)[] = fvNode.chan
281             fvNode.next = cast[FlowvarNode](task.futures)
282             task.futures = cast[pointer](fvNode)
283             # Don't share the required futures with the child
284             upperSplit.futures = nil
285
286         debugSplit:
287             let steps = (upperSplit.stop-upperSplit.start + upperSplit.stride-1) div upperSplit.stride
288             log("Worker %d: Sending [%d, %d] to worker %d (%d steps) (hasFuture: %d, dependsOnFutures: 0x%08x)\n",
289                 myID(), steps, myID(), steps, hasFuture, dependsOnFutures)
290
291         req.send(upperSplit)
292
293         incCounter(loopSplit)
294         debug:
295             let steps = (task.stop-task.cur + task.stride-1) div task.stride
296             log("Worker %d: Continuing with [%d, %d] (%d steps) (hasFuture: %d, dependsOnFutures: 0x%08x)\n",
297                 myID(), steps, myID(), steps, hasFuture, dependsOnFutures)
298
299         EagerFV:
300             parallelForRange X:
301                 loadBalance(weave)
302
303             debugSplit:
304                 log("Worker %d: Finished loop task 0x%08x (iterations [%d, %d]) (futures: 0x%08x)\n",
305                     myID(), iterations, myID(), iterations, futures)
306             block: # Wait for the child loop iterations
307                 while not this.futures.isNil:
308                     let fvNode = cast[FlowvarNode](this.futures)
309                     this.futures = cast[pointer](fvNode.next)
310
311                     LazyFV:
312                         let dummyFV = cast[FlowvarBool](fvNode.lfv)
313                         EagerFV:
314                             let dummyFV = cast[FlowvarBool](fvNode.chan)
315
316                     debugSplit:
317                         log("Worker %d: loop task 0x%08x (iterations [%d, %d]) waiting for the remainder\n",
318                             myID(), iterations, myID(), iterations)
319                         let isLastIter = sync(dummyFV)
320                         ascertain: not isLastIter
321                         debugSplit:
322                             log("Worker %d: loop task 0x%08x (iterations [%d, %d]) complete\n",
323                                 myID(), iterations, myID(), iterations)
324
325             # The "sync" in the merge statement should have recycled the flowvar channel already
326             # For LazyFlowVar, the Lazyflowvar itself was allocated on the heap, so we need to recycle it as well
327             # 2 deletes: for eager FV and 3 for Lazy FV
328             recycleFV((fvNode))
329
330         proc parallelForParallel(index, start, stop, stride, captured, capturedly, dependsOnEvent, body: NimNode):
331             # In case of parallelFor depends on iteration event indexed by the loop variable
332             # we can't use regular parallel loop with lazy splitting
333             ...

wave> memory > memory_pool.nim ...
333
```

Log-oriented programming

Analyzing philosophers on parallel matrix transpose

```
$ nim c -r -d:release -d:WV_debugSplit --outdir:build benchmarks/matrix_transposition/weave_transposes.nim
```

```
93
94 proc weave2DTiledNestedTranspose(M, N: int, bufIn, bufOut: ptr UncheckedArray[float32]) =
95     ## Transpose with 2D tiling and nested
96
97 const blck = 64 # const do not need to be captured
98
99 parallelForStrided j in 0 ..< N, stride = blck:
100     captures: {M, N, bufIn, bufOut}
101     parallelForStrided i in 0 ..< M, stride = blck:
102         captures: {j, M, N, bufIn, bufOut}
103         for jj in j ..< min(j+blck, N):
104             for ii in i ..< min(i+blck, M):
105                 bufOut[jj*M+ii] = bufIn[ii*N+jj]
```

```
Worker 25: has 0 steal requests queued
Worker 25: 23 steps left (start: 0, current: 2560, stop: 4000, stride: 64, 3 thieves)
Worker 25: Sending [3648, 4000) to worker 2 (6 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 8: Sending [2496, 4000) to worker 31 (24 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 8: has 5 steal requests queued
Worker 8: 22 steps left (start: 0, current: 1088, stop: 2496, stride: 64, 6 thieves)
Worker 8: Sending [2304, 2496) to worker 20 (3 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 8: has 4 steal requests queued
Worker 1: has 0 steal requests queued
Worker 1: 22 steps left (start: 2048, current: 2624, stop: 4000, stride: 64, 1 thieves)
Worker 5: has 0 steal requests queued
Worker 5: 52 steps left (start: 0, current: 704, stop: 4000, stride: 64, 1 thieves)
Worker 5: Sending [2304, 4000) to worker 2 (27 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 8: 19 steps left (start: 0, current: 1088, stop: 2304, stride: 64, 5 thieves)
Worker 8: Sending [2112, 2304) to worker 34 (3 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 8: has 3 steal requests queued
Worker 8: 16 steps left (start: 0, current: 1088, stop: 2112, stride: 64, 4 thieves)
Worker 8: Sending [1920, 2112) to worker 22 (3 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 30: has 0 steal requests queued
Worker 7: has 0 steal requests queued
Worker 7: 13 steps left (start: 0, current: 3200, stop: 4000, stride: 64, 1 thieves)
Worker 7: Sending [3584, 4000) to worker 10 (7 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 7: has 2 steal requests queued
Worker 7: 6 steps left (start: 0, current: 3200, stop: 3584, stride: 64, 3 thieves)
Worker 7: Sending [3520, 3584) to worker 18 (1 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
Worker 7: has 1 steal requests queued
Worker 7: 5 steps left (start: 0, current: 3200, stop: 3520, stride: 64, 2 thieves)
Worker 7: Sending [3456, 3520) to worker 26 (1 steps) (hasFuture: 0, dependsOnFutures: 0x00000000)
```

Log-oriented programming

Buffer logs and send at once via flushFile(stdout)

```
1 import ../../weave
2
3 init(Weave)
4
5 parallelFor i in 0 ..< 100:
6     echo "Enjoy (", i, ")"
7     echo "event (", i, ")"
8     echo "ordering (", i, ")"
9
10 exit(Weave)
11
```

weave > instrumentation > loggers.nim
You, 6 months ago | 1 author (You)

```
1 # Weave          Mamy André-Ratsimbazafy [7 months ago]
2 # Copyright (c) 2019 Mamy André-Ratsimbazafy
3 # Licensed and distributed under either of
4 #   * MIT license (license terms in the root directory)
5 #   * Apache v2 license (license terms in the root directory)
6 # at your option. This file may not be copied, modified or
7 # distributed without the express permission of the copyright owner.
8 import system/ansi_c
9
10 {.used.}
11
12 template log*(args: varargs[untyped]): untyped =
13     c_printf(args)
14     flushFile(stdout)
15
```

```
Enjoy (0)
event (0)
ordering (0)
Enjoy (82)
Enjoy (52)
event (52)
Enjoy (16)
event (16)
ordering (16)
event (82)
Enjoy (17)
Enjoy (76)
event (76)
ordering (76)
Enjoy (77)
event (77)
Enjoy (64)
event (64)
ordering (64)
Enjoy (88)
event (88)
Enjoy (40)
event (40)
ordering (40)
Enjoy (94)
event (94)
ordering (94)
ordering (82)
Enjoy (83)
```

Sanitizers – Thread Sanitizer

```
channels_spcc_single_ptr.nim  buggy_channel.nim x
build: # buggy_channel.nim ...
1 import std/atomic
2
3 const MemBlockSize = 256
4 type
5     ChannelSPCCSingle* = object
6         full_align: 128.; Atomic[bool]
7         itemSize*: uint8
8         buffer{.align: 8.}: UncheckedArray[byte]
9
10 proc `=(chan: var ChannelSPCCSingle,
11         dest: var ChannelSPCCSingle,
12         source: ChannelSPCCSingle)
13 ) {.error: "A channel cannot be copied".}
14
15 proc initialize(chan: var ChannelSPCCSingle, itemsize: SomeInteger) {.inl.}
16 assert itemsize.int < 8 || int high(uint8)
17 assert itemSize + sizeof(chan.itemsize) +
18     sizeof(chan.full) < MemBlockSize
19
20 chan.itemSize = uint8 itemsize
21 chan.full.store(false, moRelaxed)
22
23 func acqPryc(chan: var ChannelSPCCSingle): bool {.inline.} =
24     not chan.full.load(moRelaxed) # <----- This should be moAcquire
25
26 func tryRecv*[T](chan: var ChannelSPCCSingle, dst: var T): bool {.inline.} =
27     assert(sizeof(T) == chan.itemsize.int) or
28         # Support dummy object
29         (sizeof(T) == 0 and chan.itemsize == 0)
30
31     let full = chan.full.load(moRelaxed) # <----- This should be moAcquire
32     if not full:
33         return false
34     dst = cast[ptr T](chan.buffer.addr[])
35     chan.full.store(false, moRelaxed) # <----- This should be moRelease
36     return true
37
38 func trySend*[T](chan: var ChannelSPCCSingle, src: sink T): bool {.inline.}:
39     assert(sizeof(T) == chan.itemsize.int) or
40         # Support dummy object
41         (sizeof(T) == 0 and chan.itemsize == 0)
42
43     let full = chan.full.load(moRelaxed) # <----- This should be moAcquire
44     if full:
45         return false
46     cast[ptr T](chan.buffer.addr[]) = src
47     chan.full.store(true, moRelaxed) # <----- This should be moRelease
48     return true
49
50 # Sanity checks
51 #
52 when isMainModule:
53
54     when not compileOption("threads"):
55         {.error: "This requires --threads:on compilation flag".}
56
57     template sendLoop*[T](chan: var ChannelSPCCSingle,
58                           data: sink T,
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Testing if 2 threads can data

Sender sent: 42

WARNING: ThreadSanitizer: data race (pid=262799)
Read of size 8 at 0x7f2b30780058 by thread T1:
#0 tryRecv..._C19ae11e29jx312UGebuggy_channel(tyObject_ChannelSPCCSingle_Np479cQ{q0wIToLJ1KvR4g6, longG} /home/beta/Programming/nim/weave/build/buggy_channel.nim:6
#1 thread_func..._669E61nyj9j9c50uR2z1AQRobj_Thread.Args._r4JP9pCjRFDjaPUidFdY0g) /home/beta/Programming/nim/weave/build/buggy_channel.nim:6
#2 buggy_channel+0xe7793

#3 threadProcWrap..._6c05jNUFpsX9IoMPMuUA(tyObject_Thread..._KyvESSExzj9ahVayuVg5g*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:146:3 (buggy_channel+0xe8588)

#4 threadProcWrap..._6c05jNUFpsX9IoMPMuUA(tyObject_Thread..._KyvESSExzj9ahVayuVg5g*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:162:2 (buggy_channel+0xe8585)

#4 threadProcWrapper..._KwUyWV00QDGGR2cngjGA(void*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:171:2 (buggy_channel+0xe855)

63)

Previous write of size 8 at 0x7f2b30780058 by thread T2:
#0 trySend..._C19ae11e29jx312UGebuggy_channel(tyObject_ChannelSPCCSingle_Np479cQ{q0wIToLJ1KvR4g6, longG} /home/beta/Programming/nim/weave/build/buggy_channel.nim:6
#1 buggy_channel+0xe7792

#2 threadProcWrap..._6c05jNUFpsX9IoMPMuUA(tyObject_Thread..._r4JP9pCjRFDjaPUidFdY0g) /home/beta/Programming/nim/weave/build/buggy_channel.nim:6
#3 threadProcWrap..._6c05jNUFpsX9IoMPMuUA(tyObject_Thread..._KyvESSExzj9ahVayuVg5g*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:146:3 (buggy_channel+0xe8588)

#3 threadProcWrap..._6c05jNUFpsX9IoMPMuUA(tyObject_Thread..._KyvESSExzj9ahVayuVg5g*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:162:2 (buggy_channel+0xe8585)

#4 threadProcWrapper..._KwUyWV00QDGGR2cngjGA(void*) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:171:2 (buggy_channel+0xe855)

63)

Thread T1 (tid=262801, running) created by main thread at:
#0 pthread_create_could... (buggy_channel+0xd0b7):
ct.ThreadArgs._r4JP9pCjRFDjaPUidFdY0g) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:32:14 (buggy_channel+0xe8588)

#1 main..._JP9pCjRFDjaPUidFdY0g) /home/beta/Programming/nim/weave/build/buggy_channel.nim:122:2 (buggy_channel+0xe808d)

#2 _NimMainModule() /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:133:2 (buggy_channel+0xe8ccf)

#3 _NimMainInner() /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:213:2 (buggy_channel+0xe8ccf)

#4 _NimMain() /home/beta/choosenim/toolchains/nim+devel/lib/system.nim:2146:2 (buggy_channel+0xe8e30)

#5 main /home/beta/choosenim/toolchains/nim+devel/lib/system.nim:2153:2 (buggy_channel+0xe8e30)

Thread T2 (tid=262802, running) created by main thread at:
#0 pthread_create_snail... (buggy_channel+0xd0b7):
ct.ThreadArgs._r4JP9pCjRFDjaPUidFdY0g) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:32:14 (buggy_channel+0xe8588)

#1 main..._JP9pCjRFDjaPUidFdY0g) /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:133:2 (buggy_channel+0xe8ccf)

#2 _NimMainModule() /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:133:2 (buggy_channel+0xe8ccf)

#3 _NimMainInner() /home/beta/choosenim/toolchains/nim+devel/lib/system/threads.nim:213:2 (buggy_channel+0xe8ccf)

#4 _NimMain() /home/beta/choosenim/toolchains/nim+devel/lib/system.nim:2146:2 (buggy_channel+0xe8e30)

#5 main /home/beta/choosenim/toolchains/nim+devel/lib/system.nim:2153:2 (buggy_channel+0xe8e30)

SUMMARY: ThreadSanitizer: data race: /home/beta/Programming/nim/weave/build/buggy_channel.nim:35:9 in tryRecv..._C19ae11e29jx312UGebuggy_channel(tyObject_ChannelSPCCSingle_Np479cQ{q0wIToLJ1KvR4g6, longG})

Sender sent: 53 Receiver got: 42

Sender sent: 64 Receiver got: 53

Sender sent: 75 Receiver got: 64

Sender sent: 86 Receiver got: 75

Sender sent: 97 Receiver got: 86

Sender sent: 108 Receiver got: 97

Sender sent: 119 Receiver got: 108

Sender sent: 130 Receiver got: 119

Sender sent: 141 Receiver got: 130

Receiver got: 101

Success

ThreadSanitizer: reported 1 warnings

Error: execution terminated: program failed: '/home/beta/Programming/nim/weave/build/buggy_channel' /beta ~/Programming/nim/weave [master] \$ beta ~/Programming/nim/weave [master] \$

We are luckily correct

Sanitizers – Thread Sanitizer



A screenshot of a terminal window showing the execution of a Nim program. The terminal output indicates a successful build and execution of the `buggy_channel.nim` file.

```
beta ~/Programming/Nim/weave [master] $ nim cpp --cc:clang -r -d:release --debugger:native --passC:"-fsanitize=thread" --passL:"-fsanitize=thread" -o:outdir:build/buggy_channel.nim
Hint: used config file '/home/beta/.choosenim/toolchains/nim-#devel/config/nim.cfg' [Conf]
Hint: used config file '/home/beta/.choosenim/toolchains/nim-#devel/config/config.nims' [Conf]
Hint: used config file '/home/beta/Programming/Nim/weave/nim.cfg' [Conf]
....CC: buggy_channel.nim
Hint: [Link]
Hint: 41857 lines; 0.504s; 58.996MiB peakmem; Release build; proj: /home/beta/Programming/Nim/weave/build/buggy_channel.nim; out: /home/beta/Programming/Nim/weave/build/buggy_channel [Success]
Hint: /home/beta/Programming/Nim/weave/build/buggy_channel [Exec]
Testing if 2 threads can send data
Receiver got: 42
Sender sent: 42
Receiver got: 53
Sender sent: 53
Sender sent: 64
Sender sent: 75
Receiver got: 64
Receiver got: 75
Sender sent: 86
Receiver got: 86
Sender sent: 97
Receiver got: 97
Sender sent: 108
Receiver got: 108
Sender sent: 119
Receiver got: 119
Sender sent: 130
Receiver got: 130
Sender sent: 141
Receiver got: 141
-----
Success
beta ~/Programming/Nim/weave [master] $
```

The terminal also displays the code for the `buggy_channel.nim` file, which contains various channel operations and assertions related to memory safety.

```
channels_spse_single_ptr.nim  buggy_channel.nim
```

```
1 import std/atomics
2 const MemBlockSize = 256
3 type
4     ChannelSPSCSingle* = object
5         full*: align(128): Atomic[bool]
6         itemSize*: uint8
7         buffer*: align(8): UncheckedArray[byte]
8
9 proc `=(dest: var ChannelSPSCSingle,
10      source: ChannelSPSCSingle)
11 {error: "A channel cannot be copied".}
12
13 proc initialize(chan: var ChannelSPSCSingle, itemsize: SomeInteger) {.inline.}
14 assert itemsize.int in 0 .. int high(uint)
15 assert itemsize.int +
16     sizeof(chan.itemsize) +
17     sizeof(chan.full) < MemBlockSize
18
19 chan.itemSize = uint8 itemsize
20 chan.full.store(false, moRelaxed)
21
22 func isEmpty(chan: var ChannelSPSCSingle): bool {.inline.}
23 not chan.full.load(moAcquire) # <----- This should be moAcquire
24
25 func tryRecv*[T](chan: var ChannelSPSCSingle, dst: var T): bool {.inline.}
26 assert sizeof(T) == chan.itemsize.int or
27     # Support dummy object
28     (sizeof(T) == 0 and chan.itemsize == 1)
29
30 let full = chan.full.load(moAcquire) # <----- This should be moAcquire
31 if not full:
32     return false
33 dst = cast[ptr T](chan.buffer.addr[])
34 chan.full.store(false, moRelease) # <----- This should be moRelease
35 return true
36
37 func trySend*[T](chan: var ChannelSPSCSingle, src: sink T): bool {.inline.}
38 assert sizeof(T) == chan.itemsize.int or
39     # Support dummy object
40     (sizeof(T) == 0 and chan.itemsize == 1)
41
42 let full = chan.full.load(moAcquire) # <----- This should be moAcquire
43 if full:
44     return false
45 cast[ptr T](chan.buffer.addr[])[] = src
46 chan.full.store(true, moRelease) # <----- This should be moRelease
47 return true
48
```



Sanitizers – Thread Sanitizer

- ◊ Always use thread sanitizer it only requires

```
nim cpp --cc:clang -r -d:release --debugger:native  
--passC:"-fsanitize=thread" --passL:"-fsanitize=thread"  
--outdir:build build/buggy_channel.nim
```

- ◊ Use the C++ target if you use atomics
- ◊ Easy way to detect data races for concurrent data structure, for example a concurrent hash table





Sanitizers – Address Sanitizer

- ◇ Detects incorrect memory addresses (uninitialized memory, buffer overflows, use after free, leaks).
- ◇ Can be made aware of custom memory management scheme (memory pools, ...)



Sanitizers - Address Sanitizer

```
weave > instrumentation > sanitizers.nim > ...
Many Ratsimbazafy, 6 months ago | author [Many Ratsimbazafy]
1 # Weave - Many Ratsimbazafy [6 months ago] • Add AddressSanitizer support for the memory pool (#78)
2 # Copyright (c) 2019 Many André-Ratsimbazafy
3 # Licensed and distributed under either of
4 #   * MIT license (license terms in the root directory or at http://opensource.org/licenses/MIT).
5 #   * Apache v2 license (license terms in the root directory or at http://www.apache.org/licenses/LICENSE-2.0).
6 #   at your option. This file may not be copied, modified, or distributed except according to those terms.
7
8 # Address sanitizer
9 #
10
11 proc asan_poison_memory_Region(region: pointer, size: int){.nodecl, imports:"__asan_poison_memory_region"}
12 proc asan_unpoison_memory_Region(region: pointer, size: int){.nodecl, imports:"__asan_unpoison_memory_region"}
13 const WV_SanitizeAddr*:booldefine.= false
14
15 when WV_SanitizeAddr:
16     when not defined(gcc) or defined(clang) or defined(llvm_gcc):
17         {.error: "Address Sanitizer requires GCC or Clang."}
18     else:
19         {.passc:"-fsanitize=address"}
20         {.passl:"-fsanitize=address"}
21         when defined(clang) and defined(amd64):
22             {.passl:"-fclang_rt.asan_dynamic-x86_64"}
23         else:
24             {.error: "Compiler + CPU arch configuration missing."}
25
26 template poisonMemRegion*(region: pointer, size: int) =
27     when WV_SanitizeAddr:
28         asan_poison_memory_region(region, size)
29     else:
30         discard
31
32 template unpoisonMemRegion*(region: pointer, size: int) =
33     when WV_SanitizeAddr:
34         asan_unpoison_memory_region(region, size)
35     else:
36         discard
37
weave > memory > memory_pool.nim > ...
Many Ratsimbazafy, 6 months ago | author [Many Ratsimbazafy]
158 # MostUsedArenaRatio = 8
159     ## Beyond 1/8 of its capacity an arena is considered mostly used.
160     MaxSlowReqs = 8 !8
161     ## In the slow path, up to 8 arenas can be considered for release at
162     ## once.
163
164 # Sanitizer
165 template guardedAccess(memBlock: MemBlock, body: untyped): untyped =
166     unpoisonMemRegion(memBlock.addr, WV_MemBlockSize)
167     body
168     poisonMemRegion(memBlock.addr, WV_MemBlockSize)
169
170 template guardedAccess(memBlock: ptr MemBlock, body: untyped): untyped =
171     unpoisonMemRegion(memBlock, WV_MemBlockSize)
172     body
173     poisonMemRegion(memBlock, WV_MemBlockSize)
174
175 # Data structures
176 #
177
178 iterator backward(tail: ptr Arena): ptr Arena =
179     ## Doubly-linked list backward iterator
180     ## Note: we assume that the list is not circular
181     ## and terminates via a nil pointer
182
183 # sanitizers.nim > memory_pool.nim > ...
184
185 when WV_SanitizeAddr:
186     if arena.meta.free.isNil:
187         arena.meta.free = first
188         last.next.store(nil, moRelaxed)
189     else:
190         guardedAccess(arena.meta.free):
191             arena.meta.free.prepend(last)
192     arena.meta.used -= count
193
194 when WV_SanitizeAddr:
195     var cur = first
196     while true:
197         if cur = last:
198             poisonMemRegion(cur, WV_MemBlockSize)
199             break
200         let next = cur.next.load(moRelaxed)
201         poisonMemRegion(cur, WV_MemBlockSize)
202         cur = cast[ptr MemBlock](next)
203
204 debugMem:
205     log("Arena 0x%08x - TID %d - collected garbage, reclaimed %d block"
206         arena.addr, arena.meta.threadID.load(moRelaxed), beforeCollect - ar-
207         ena.meta.used)
208
209 func allocBlock(arena: var Arena): ptr MemBlock {.inline.} =
210     ## Allocate from an arena
211     preCondition: not arena.meta.free.isNil
212     preCondition: arena.meta.used < arena.blocks.len
213
214     arena.meta.used += 1
215     result = arena.meta.free
216     unpoisonMemRegion(result, WV_MemBlockSize)
217     ## The following acts as prefetching for the block that we are returning
218     arena.meta.free = cast[ptr MemBlock](result.next.load(moRelaxed))
219
220 postCondition: arena.meta.used in 0 .. arena.blocks.len
```



Sanitizers

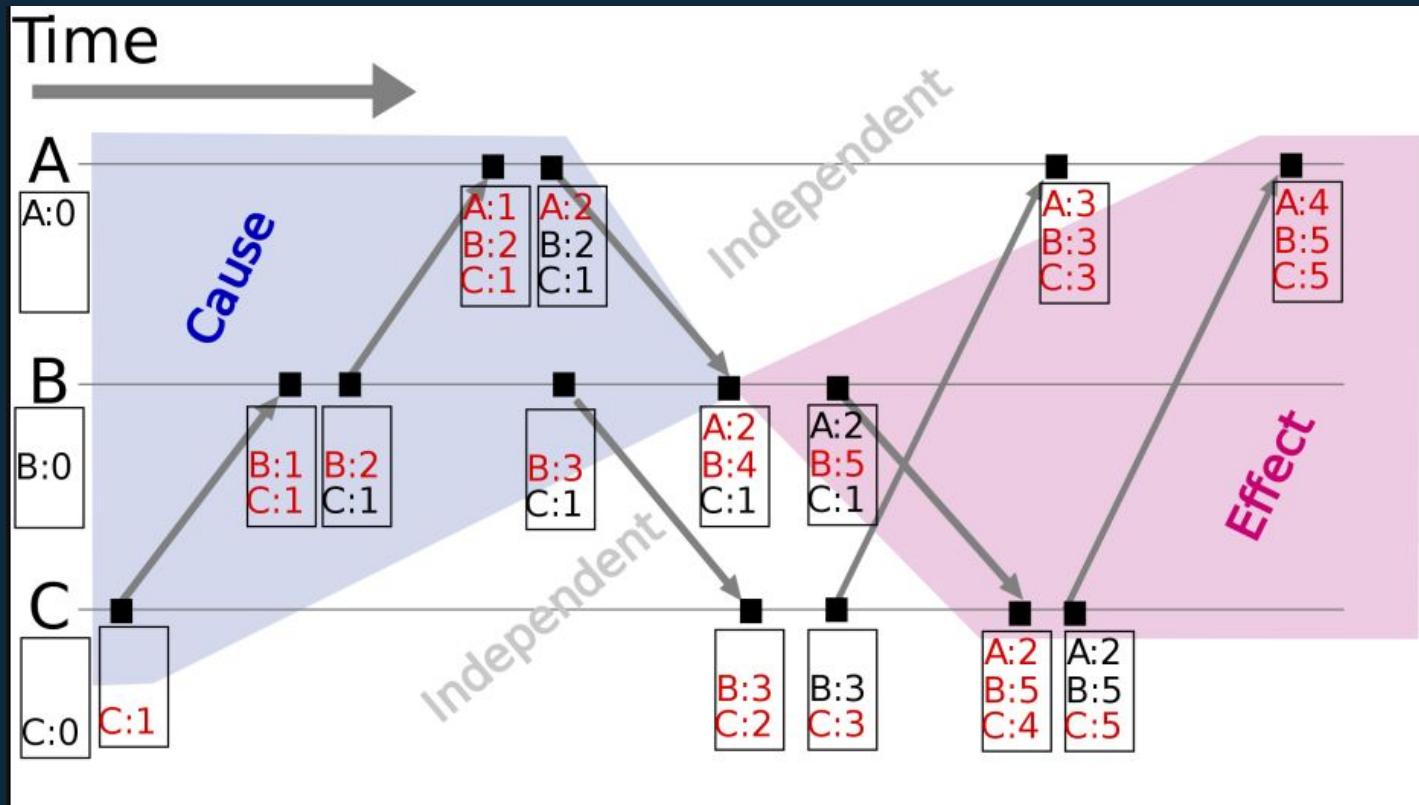
- ◇ `nim c --cc:clang -r -d:release --debugger:native
--passC:"-fsanitize=address"
--passL:"-fsanitize=address" --outdir:build
weave/memory/memory_pools.nim`
 - ◇ Also `valgrind --tool=helgrind build/mybinary` for
deadlocks (no need for recompilation but slow)
 - ◇ And `valgrind --tool=drd build/mybinary` for data races
- 



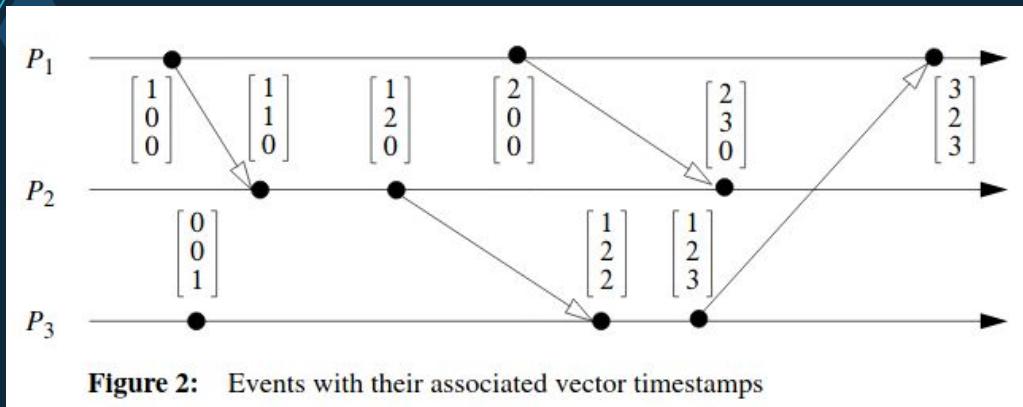
The limits

- ◊ At the very least we miss a notion of relative time between events that lead to the bad situation
 - ◊ Ideally we have a trace or replay of the problematic sequence of events
 - ◊ This does not prove the absence of synchronization bugs, what if the race is triggered once 1/1000000
 - ◊ We debugged the low-level but what if we're having the wrong approach at a high-level?
- 

Relative time via Vector Clock



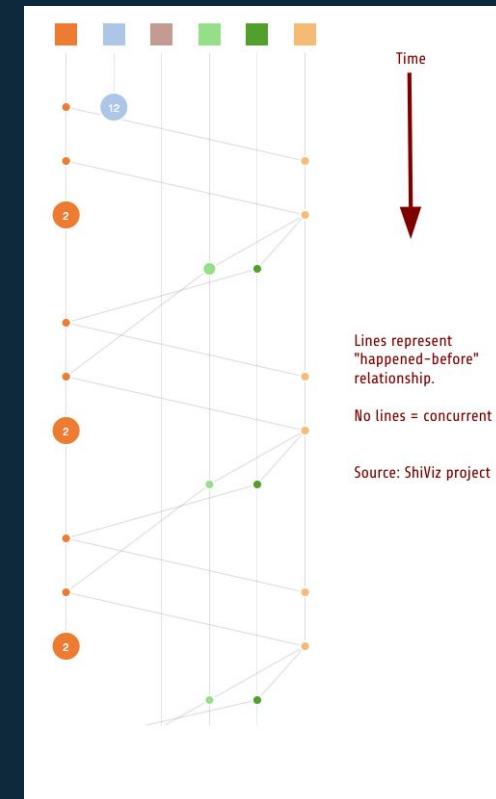
Tracking causality of distributed events



- Detecting causal relationships in distributed computations: In Search of the Holy Grail
Schwarz, Mattern, 1994

<http://www.vs.inf.ethz.ch/publ/papers/holygrail.pdf>

- <https://bestchai.bitbucket.io/shiviz/>



Implementing Vector Clock in ~70 lines

```
72 type
73   Timestamp = distinct int64
74
75   CausalOrderings = enum
76     Identical
77     HappenedBefore
78     HappenedAfter
79     Concurrent
80
81   VectorClock = object
82     ## A vector clock tracks the partial ordering of event in a distributed system.
83     ## 2 clocks can be compared to check if
84     ## - 1 event must have happened before the other
85     ## They could happen concurrently
86     ## - They violate the laws of causality
87     ## [a, b] <= [c, d] iff a[i] <= c[i] and vector clock for causal ordering of events
88     ## The length is the number of processes/threads tracked
89     ## The array of known local timestamps is indexed by the threadID (< len)
90     localTS: array[MaxThreads, Timestamp]
91     len: int8
92
93   proc inc(time: var Timestamp{borrow})
94     proc s {time: Timestamp} string{borrow}
95     proc <=(a, b: Timestamp) bool{borrow}
96     proc ==(a, b: Timestamp) bool{borrow}
97
98   proc init*clock: var VectorClock {.inline.} =
99     ## Initialize a vector clock
100    zeroMem(clock.addr, sizeof(clock))
101
102  func tick*clock: var VectorClock, tid: ThreadId {.inline.=}
103    ## Local tick in thread tid
104    if tid > clock.len then
105      clock.len = intOf(tid) + 1
106      clock.locals[tid.int].inc()
107
108  func synchronize(a: var VectorClock, b: VectorClock) {.inline.=}
109    ## Synchronize clock 'a' with clock 'b'
110    if b.len > a.len:
111      a.len = b.len
112      for i in 0 ..< a.len:
113        if a.localTS[i] < b.localTS[i]:
114          a.localTS[i] = b.localTS[i]
115
116  func causality*a, b: VectorClock: CausalOrdering =
117    ## Returns the causal order between 2 vector clocks
118
119    result = Identical
120    for i in 0 ..< MaxThreads:
121      let tsA = a.localTS[i]
122      let tsB = b.localTS[i]
123
124      if tsA == tsB:
125        continue
126      elif tsA < tsB:
127        if result == HappenedAfter:
128          return Concurrent
129        result = HappenedBefore
130      else:
131        if result == HappenedBefore:
132          return Concurrent
133        result = HappenedAfter
134
135  proc # -----
136    # Sanity checks
137    when isMainModule:
138      import random
139
140    proc clock[N: static int]{a: array[N, int]}: VectorClock =
141      static: doassert N ≤ MaxThreads
142      result: = N#0
143      for i in 0 ..< N:
144        result.localTS[i] = TimeStamp(a[i])
145
146    proc randomClock(rng: var Rand, N: static int): VectorClock =
147      static: doassert N ≤ MaxThreads
148      result: = N#0
149      for i in 0 ..< N:
150        result.localTS[i] = TimeStamp(rng.rand(high(iint)))
151
152    proc causality(a: static int[], b: array[N, int]): CausalOrdering =
153      clock(a).causality(clock(b))
154
155    proc test_causality() =
156      # Free: https://en.wikipedia.org/wiki/Vector\_clock#/media/File:Vector\_Clock.svg
157
158      doassert causality([0, 0, 0], [0, 0, 0]) == HappenedBefore
159      doassert causality([0, 0, 0], [0, 1, 0]) == HappenedBefore
160      doassert causality([0, 1, 0], [0, 2, 0]) == HappenedBefore
161      doassert causality([0, 2, 0], [1, 0, 0]) == HappenedBefore
162      doassert causality([1, 0, 0], [1, 2, 0]) == HappenedBefore
163
164      doassert causality([0, 2, 1], [0, 3, 0]) == HappenedBefore
165      doassert causality([0, 3, 0], [0, 3, 1]) == HappenedBefore
166      doassert causality([0, 2, 1], [0, 3, 1]) == HappenedBefore
167      doassert causality([0, 1, 1], [0, 2, 1]) == HappenedBefore
168      doassert causality([0, 2, 1], [1, 0, 1]) == HappenedBefore
169      doassert causality([1, 0, 1], [1, 2, 1]) == HappenedBefore
170
171      doassert causality([0, 2, 1], [0, 3, 1]) == HappenedBefore
172      doassert causality([0, 3, 1], [0, 3, 2]) == HappenedBefore
173      doassert causality([0, 2, 1], [0, 3, 2]) == HappenedBefore
174      doassert causality([0, 3, 1], [1, 0, 1]) == HappenedBefore
175
176      doassert causality([0, 3, 2], [0, 3, 3]) == HappenedBefore
177      doassert causality([0, 4, 1], [0, 3, 3]) == HappenedBefore
178      doassert causality([0, 3, 3], [0, 3, 3]) == HappenedBefore
179      doassert causality([0, 5, 1], [0, 3, 3]) == HappenedBefore
180      doassert causality([0, 3, 3], [0, 5, 1]) == HappenedBefore
181
182      doassert causality([0, 5, 0], [0, 5, 1]) == HappenedBefore
183      doassert causality([0, 3, 3], [0, 5, 1]) == HappenedBefore
184      doassert causality([0, 5, 0], [0, 5, 1]) == HappenedBefore
185
186      echo "Causality tests - Happened before - SUCCESS"
187
188  # -----
189
190  doassert causality([0, 0, 1], [1, 2, 0]) == Concurrent
191  doassert causality([0, 0, 1], [1, 2, 1]) == Concurrent
192  doassert causality([1, 2, 1], [0, 3, 0]) == Concurrent
193  doassert causality([0, 0, 1], [0, 3, 0]) == Concurrent
194
195  doassert causality([0, 0, 1], [0, 3, 0]) == Concurrent
196  doassert causality([0, 2, 0], [0, 3, 0]) == Concurrent
197  doassert causality([0, 2, 0], [0, 3, 0]) == Concurrent
198  doassert causality([0, 2, 0], [0, 3, 0]) == Concurrent
199
200  doassert causality([0, 5, 0], [0, 5, 1]) == Concurrent
201  doassert causality([0, 3, 0], [0, 5, 1]) == Concurrent
202
203  doassert causality([0, 3, 0], [1, 2, 0]) == Concurrent
204  doassert causality([0, 3, 0], [1, 2, 1]) == Concurrent
205
206  doassert causality([0, 3, 0], [1, 2, 1]) == Concurrent
207  doassert causality([0, 3, 0], [1, 2, 1]) == Concurrent
208  doassert causality([0, 2, 1], [0, 3, 0]) == Concurrent
209
210  echo "Causality tests - Concurrency - SUCCESS"
```



Vector Clock – The gist

- ◊ In multithreaded programs or distributed systems or micro-services, classic timestamps are unreliable to know the order of events.
 - ◊ Vector clocks give you partial ordering (Happened Before, Happened After, Concurrent)
 - ◊ If you have 4 services, you only need to add
 - “clock(0, 0, 0, 0)” at time 0
 - “clock(0, 2, 1, 0)” when the logging thread knows that thread 1 advanced twice and thread 3 advanced once.
- 



3

Designing

Multithreaded programs



Engineering

- ◆ Civil engineering
- ◆ Mechanical engineering
- ◆ Material engineering
- ◆ Industrial engineering
- ◆ Electrical engineering
- ◆ Nuclear engineering
- ◆ Chemical engineering
- ◆ Aerospace engineering
- ◆ Marine engineering

Years of study, design, checks, and audits go before building any of the other “engineering” projects.

The higher the risks and the stakes, the longer the design phase.





Software engineering

- ◊ Multithreaded programs must be designed
 - I.e. We want a **blueprint** that we can check
- ◊ Many of their properties are emergent at a high-level

Example behaviour:

- “Always open the door and let the other pass before you”

What if 2 programs exhibits this behavior?

- Contention to open the door (“Always open”).
- Deadlock as no one wants to go through.
- Deadlock if the door is already opened.





Blueprints

- ◇ Code changes fast:
 - Comments, documentation go stale
 - ◇ In many cases the devil is in the implementation details (try reproducing an academic paper, for example on garbage collection)
 - ◇ A blueprint of what we have implemented may be more useful than a planned design, in particular to find corner cases
- 



State Machines

- ◊ The hardest part of multithreading is handling state
 - ◊ Visualizing the possible states and events your components react to is invaluable to detect “design bugs”
 - ◊ State machines (but also Petri Nets) lend themselves to formal verification
- 



Hardware engineering

- ◇ Designing a chip, router, memory model, cache coherence protocol, embedded device requires modelling.
- ◇ They use state machines to validate the design before thousands are poured into a doomed to fail prototypes

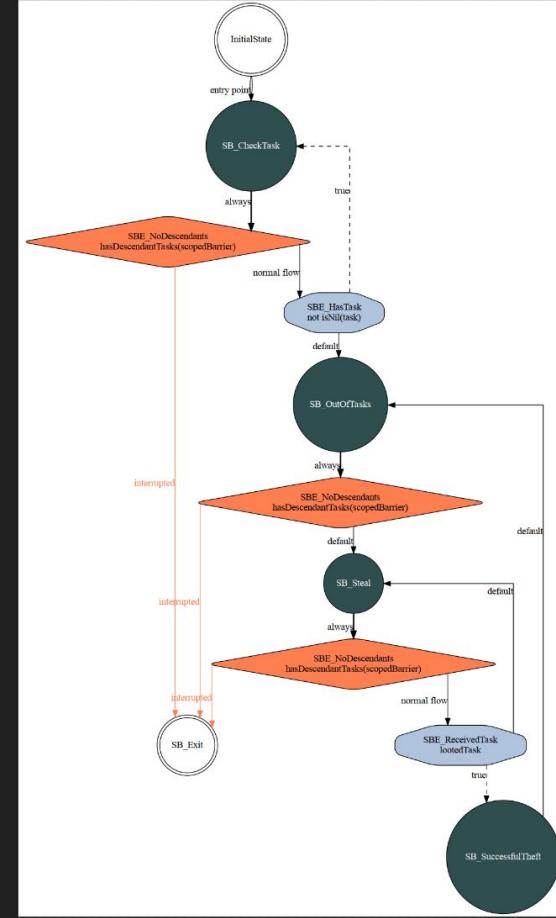


Synthesis - A compile-time state machine generator

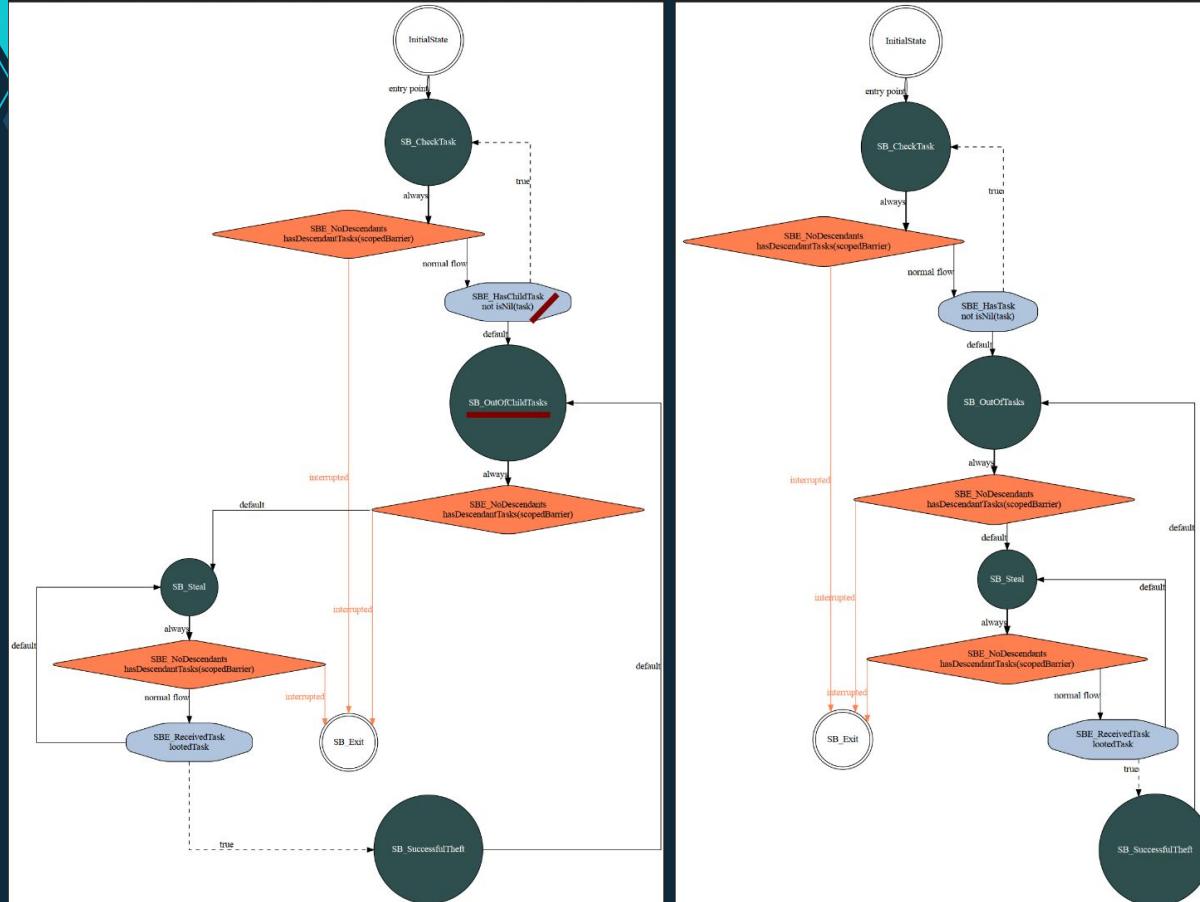
```

@ sync_scope.nim
wave> state_machines > sync_scope.nim
  debug: logf worker %d: syncScope 2 - DECOMMISSION & CHILDREN, myID()
191 fin: SB_OutOfTasks
192
193 # These states are interrupted when the scope has no more descendant
194
195 behavior(syncScopeFSA):
196   ini: SB_OutOfTasks
197   transition:
198     # Steal and hope to advance towards the child tasks in other workers' queues.
199     trySteal(isOutOfTask = false) # Don't sleep here or we might stall the runtime
200     profile_start(idle)
201   fin: SB_Steal
202
203 onEntry(syncScopeFSA, SB_Steal):
204   let lootedTask = recvLootSteal(task, isOutOfTasks = false) # Don't sleep here or we might stall the runtime
205
206 implEvent(syncScopeFSA, SBE_ReceivedTask):
207   lootedTask
208
209 behavior(syncScopeFSA):
210   steady: SB_Steal
211   event: SBE_ReceivedTask
212   transition: profile_stop(idle)
213   dispatchOnChildrenAndThieves()
214   profile_start(idle)
215   fin: SB_SuccessfulTheft
216
217 behavior(syncScopeFSA):
218   steady: SB_Steal
219   transition:
220     profile_stop(idle)
221     dispatchOnChildrenAndThieves()
222     profile_start(idle)
223
224 # -----
225 # This is not interrupted when scoped barrier has no descendant tasks
226
227 behavior(syncScopeFSA):
228   ini: SB_SuccessfulTheft
229   transition:
230     ascertain: not task fn.isNil
231     debug: logf[Worker %d: syncScope 3 - stole tasks\n, myID()]
232     TargetLastVictim:
233       if task.victim != NOT_A_WORKER:
234         myThefts().lastVictim = task.victim
235       ascertain: myThefts().lastVictim != myID()
236
237     if not task.next.isNil:
238       profile(enq_dog_task):
239         # Add everything
240         myWorker().deque.addListFirst(task)
241         # And then only use the last
242         task = myWorker().deque.popFirst()
243
244     StealAdaptive:
245       myThefts().recentThefts += 1
246
247     # Share loot with children workers
248     debug: logf[Worker %d: syncScope 4 - sharing work\n, myID()]
249     shareWork()
250
251     # Run the rest
252     profile(run_task):
253       execute(task)
254       profile(enq_dog_task):
255         # The memory is re-used but not zero-ed
256         localCtx.taskCache.add(task)
257   fin: SB_OutOfTasks
258
259 # -----
260
261 synthesize(syncScopeFSA):
262   proc wait(scopedBarrier: var ScopedBarrier)
263

```



Visual bug hunting with state machines



This is a barrier where a worker helps other until all descendants tasks spawned are done.

On the left, the old implementation, the worker was sometimes trying to “help” while all the other workers actually finished, i.e. deadlock.

The design bug was that the worker only drained direct child tasks and may have grandchildren in its queue





Share nothing or be SLOW

Starvation

Latencies

Overheads

Waiting for contention

(Credit: The HPX multithreading runtime)



Functions & messages over mutable objects

- ◊ Prefer side-effect free functions
- ◊ Avoid mutable objects or at least don't share mutable fields across threads (one branch of a tree data structure per-thread for example)
- ◊ Avoid OOP unless you are ready to split your “Person” and “Dog” instances across threads and deal with the contention.
Prefer plain objects like object variants.
Copy is cheaper than contention and synchronization

https://github.com/mratsim/trace-of-radiance/blob/99f7d85d/trace_of_radiance/support/emulate_classes_with_ADTs.nim#L251-L268





Functions & messages over mutable objects

**Isolate synchronization to few data structures
that will be under higher scrutiny, like channels.**



Functions & messages over mutable objects



MULTITHREADING

THREADS ARE NOT GOING TO SYNCHRONIZE THEMSELVES



Functions & messages over mutable objects

- ◊ Function and messages allows you to do black-box testing and ease fuzzing.
 - ◊ For example you can write a runner that has different scenarios, provided in a convenient format (JSON, YAML) and produces certain sequences of messages and verifying the output messages.
 - ◊ Randomize the message order and verify that output is still consistent.
- 



3

Correct-by-construction

Multithreaded programs



Correct-by-construction multithreading

- ◊ By design, your software is free of multithreading bugs:
 - Deadlocks
 - Livelocks
 - Data races
 - Object lifetime bugs (use before init, use-after-free)
 - ◊ More importantly it does what you specified
- 



Once you eliminate the impossible,
whatever remains, no matter how
improbable, must be the truth.

- ◊ The story of how I found a deadlock bug in glibc and musl.
- ◊ I was implementing a “2-phase commit” protocol to put a thread to sleep if there is no work.
 - The events:
 - I want to sleep
 - Sleeping
 - Wakeup!
- ◊ A race between sleep and wakeup will deadlock your thread



80 lines of code, what could go wrong?

```
67 type
68     # Multi Producers, Single Consumer event notification
69     # This can be seen as a wait-free condition variable for producers
70     # that avoids them spending time in expensive kernel land due to mutexes.
71     #
72     #
73     # This data structure should be associated with a MPSC channel
74     # to notify that an "event" happened in the channel.
75     # It avoid spurious polling of empty channels,
76     # and allow parking of threads to save on CPU power.
77     #
78     # See also: binary semaphores, eventcounts
79     # On Windows: ManualResetEvent and AutoResetEvent
80 when supportsFutex:
81     # ----- Consumer specific -----
82     ticket{.align: WV_CacheLinePadding.}: uint8 # A ticket for the consumer to sleep in a phase
83     # ----- Contention ----- no real need for padding as cache line should be reloaded in case of contention anyway
84     futex: Futex # A Futex (atomic int32 that can put thread to sleep)
85     phase: Atomic<uint> # A binary timestamp, toggles between 0 and 1 (but there is no atomic "not")
86     signaled: Atomic<bool> # Signaling condition
87 else:
88     # ----- Consumer specific -----
89     lock{.align: WV_CacheLinePadding.}: Lock # The lock is never used, it's just there for the condition variable
90     ticket: uint8 # A ticket for the consumer to sleep in a phase
91     # ----- Contention ----- no real need for padding as cache line should be reloaded in case of contention anyway
92     cond: Cond # Allow parking of threads
93     phase: Atomic<uint> # A binary timestamp, toggles between 0 and 1 (but there is no atomic "not")
94     signaled: Atomic<bool> # Signaling condition
95
96 func initialize*(en: var EventNotifier) {.inline.} =
97 when supportsFutex:
98     en.futex.initialize()
99 else:
100     en.cond.initCond()
101     en.lock.initLock()
102     en.ticket = 0
103     en.phase.store(0, moRelaxed)
104     en.signaled.store(false, moRelaxed)
105
106 func '=destroy'*(en: var EventNotifier) {.inline.} =
107 when not supportsFutex:
108     en.cond.deinitCond()
109     en.lock.deinitLock()
110
111 func '='*(dst: var EventNotifier, src: EventNotifier) {.error: "An event notifier cannot be copied".}
112 func '=sink'*(dst: var EventNotifier, src: EventNotifier) {.error: "An event notifier cannot be moved".}
```

```
114 func prepareToPark*(en: var EventNotifier) {.inline.} =
115     ## The consumer intends to sleep soon.
116     ## This must be called before the formal notification
117     ## via a channel.
118     if not en.signaled.load(moRelaxed):
119         en.ticket = en.phase.load(moRelaxed)
120
121 proc park*(en: var EventNotifier) {.inline.} =
122     ## Wait until we are signaled of an event
123     ## Thread is parked and does not consume CPU resources
124     ## This may wakeup spuriously.
125     if not en.signaled.load(moRelaxed):
126         if en.ticket == en.phase.load(moRelaxed):
127             when supportsFutex:
128                 en.futex.wait(0)
129             else:
130                 en.cond.wait(en.lock) # Spurious wakeup are not a problem
131             en.signaled.store(false, moRelaxed)
132     when supportsFutex:
133         en.futex.initialize()
134     # We still hold the lock but it's not used anyway.
135
136 proc notify*(en: var EventNotifier) {.inline.} =
137     ## Signal a thread that it can be unparked
138
139     if en.signaled.load(moRelaxed):
140         # Another producer is signaling
141         return
142     en.signaled.store(true, moRelease)
143     discard en.phase.fetchXor(1, moRelaxed)
144     when supportsFutex:
145         en.futex.store(1, moRelease)
146         en.futex.wake()
147     else:
148         en.cond.signal()
```

The version with lock + condition variable deadlocks on Linux but not Mac or Windows.

Futex is the raw kernel primitive that glibc and Windows synchronization use to implement locks and condition variables.

Futex don't deadlock.



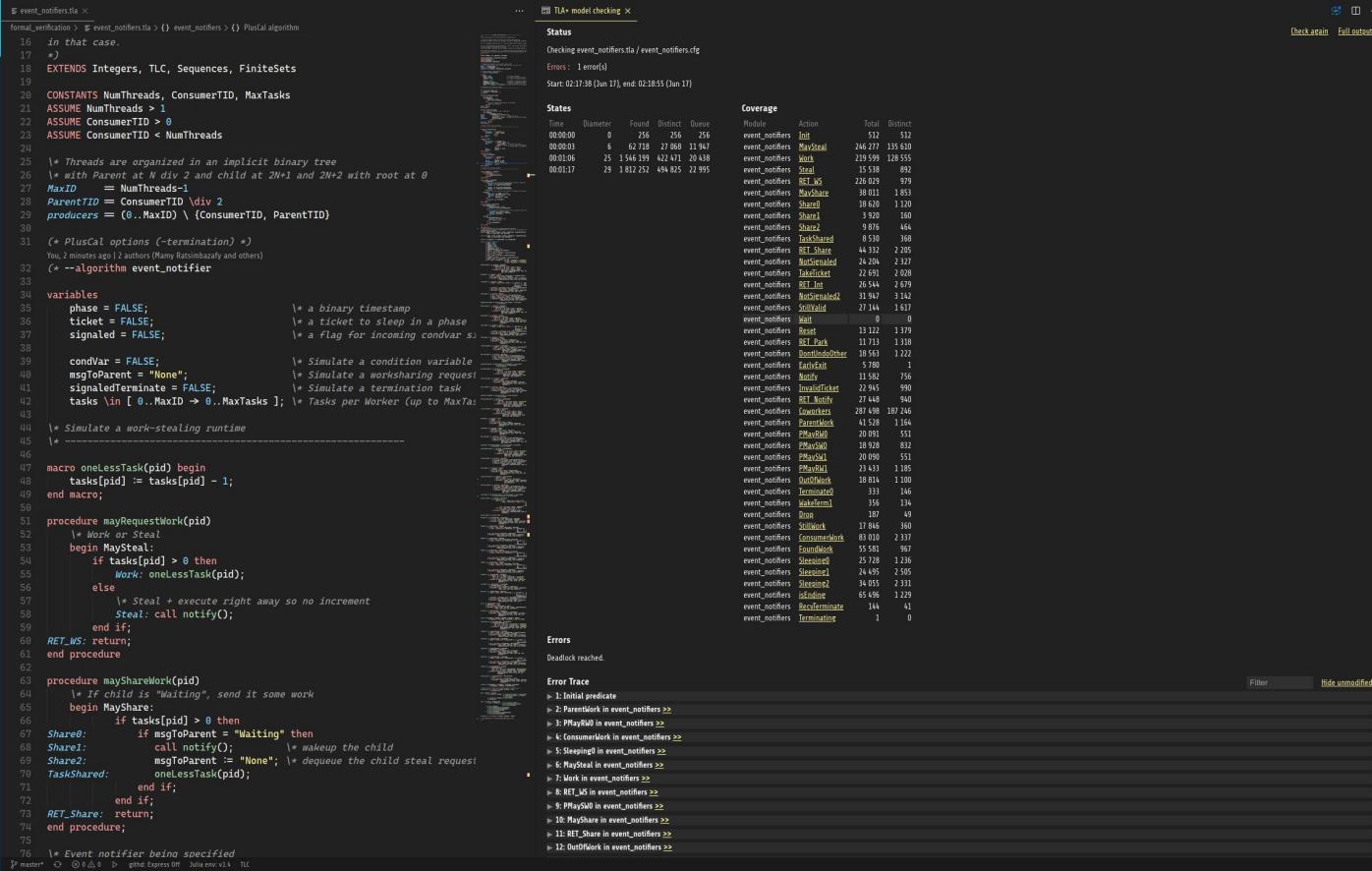


Proving my code bug-free

- ◊ Formal verification via model checking: verifying all possible state my threads can be (i.e. preparing to sleep, sleeping or waking another thread) and trying all possible threads interleaving
- ◊ Using TLA+



Proving code bug-free



```

event_notifiers.tla
formal_verification > event_notifiers.tla > {} event_notifiers > {} PlusCal algorithm
15  in that case.
16  */
17  */
18  EXTENDS Integers, TLC, Sequences, FiniteSets
19
20  CONSTANTS NumThreads, ConsumerTID, MaxTasks
21  ASSUME NumThreads > 1
22  ASSUME ConsumerTID > 0
23  ASSUME ConsumerTID < NumThreads
24
25  /* Threads are organized in an implicit binary tree
26  */ with Parent at N div 2 and child at 2N+1 and 2N+2 with root at 0
27  MaxID      = NumThreads-1
28  ParentTID  = ConsumerTID \div 2
29  producers  = {..,MaxID} \ {ConsumerTID, ParentTID}
30
31 (* PlusCal options {-termination *} *)
32 You 2 minutes ago | 2 authors (Many Ratsimbazafy and others)
33 (* --algorithm event_notifier
34
35 variables
36   phase = FALSE;           /* a binary timestamp
37   ticket = FALSE;          /* a ticket to sleep in a phase
38   signaled = FALSE;         /* a flag for incoming condvar signals
39
40   condvar = FALSE;          /* Simulate a condition variable
41   msgToParent = "None";     /* Simulate a worksharing request
42   signaledTerminate = FALSE; /* Simulate a termination task
43   tasks \in [0..MaxID \rightarrow 0..MaxTasks]; /* Tasks per Worker (up to MaxTasks)
44
45 /* Simulate a work-stealing runtime
46
47 macro oneLessTask(pid) begin
48   tasks[pid] := tasks[pid] - 1;
49 end macro;
50
51 procedure mayRequestWork(pid)
52   /* Work or Steal
53   begin MaySteal:
54   if tasks[pid] > 0 then
55     Work: oneLessTask(pid);
56   else
57     /* Steal + execute right away so no increment
58     Steal: call notify();
59   end if;
60   RET_WS: return;
61 end procedure;
62
63 procedure mayShareWork(pid)
64   /* If child is "Waiting", send it some work
65   begin MayShare:
66   if tasks[pid] > 0 then
67     Share0:  if msgToParent = "Waiting" then
68       call notify();           /* wakeup the child
69     Share1:  msgToParent := "None"; /* dequeue the child steal request
70     TaskShared: oneLessTask(pid);
71   end if;
72   end if;
73   RET_Share: return;
74 end procedure;
75
76 /* Event notifier being specified

```

Notice how this simple data structure has millions of states

Proving code bug-free

event_notifiers.tla

```

formal_verification > event_notifiers.tla > {} event_notifiers > {} PlusCal algorithm
75
76  /* Event notifier being specified
77 */
78
79 procedure prepareParking()
80 begin
81     NotSignaled: if ~signaled then
82         TakeTicket: ticket := phase;
83         end if;
84     RET_Int: return;
85 end procedure
86
87 procedure park()
88 begin
89     NotSignaled2: if ~signaled then
90         StillValid: if ticket = phase then
91             Wait: await condVar; /* next line is atomic
92             condVar := FALSE; /* we don't model the spurious wake
93             end if;
94         end if;
95     Reset: signaled := FALSE;
96     RET_Park: return;
97 end procedure;
98
99 procedure notify()
100 variables prevState
101 begin
102     DontUndoOther: if signaled then
103         EarlyExit: return;
104     end if;
105     Notify: signaled := TRUE;
106     InvalidTicket: phase := ~phase;
107     /* Awaken: condVar := TRUE;
108     RET_Notify: return;
109 end procedure;
110
111 /* Simulate the runtime lifetime
112 */
113
114 process producer |in producers
115 begin Coworkers:
116     while tasks[Self] > 0 do
117         call mayRequestWork(Self);
118     end while;
119 end process;
120
121 process parent = ParentTID
122 begin ParentWork:
123     either | The order of work sharing and work stealing is arbitrary
124     PMayW0: call mayRequestWork(ParentTID);
125     PMaySW0: call mayShareWork(ParentTID);
126     or
127     PMayW1: call mayShareWork(ParentTID);
128     PMayW1: call mayRequestWork(ParentTID);
129     end either;
130     /* But it will for sure tell the consumer to terminate at one point
131     OutOfWork:
132     if tasks = [ x \in 0..MaxID -> 0 ] then
133         Terminated0: signaledTerminate := TRUE;

```

TLA+ model checking

- 18: RET_Int in event_notifiers >
- 19: Sleeping1 in event_notifiers >
- condVar
msgToParent M
 pc (i) M
 phase
 pid (i)
 pid_ (i)
 prevState (i)
 signaled
 signaledTerminate
 stack (i)
 tasks (i)
 ticket
- FALSE
 "Waiting"
 (0 > "Coworkers" @ 0 1 > "Notify" @ 0 2 > "Coworkers" @ 0 3 > "Sleeping2")
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
- 20: Sleeping2 in event_notifiers >
- condVar
msgToParent
 pc (i) M
 phase
 pid (i)
 pid_ (i)
 prevState (i)
 signaled
 signaledTerminate
 stack (i) M
 tasks (i)
 ticket
- FALSE
 "Waiting"
 (0 > "Coworkers" @ 0 1 > "Notify" @ 0 2 > "Coworkers" @ 0 3 > "NotSignaled2")
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
- 21: NotSignaled2 in event_notifiers >
- condVar
msgToParent
 pc (i) M
 phase
 pid (i)
 pid_ (i)
 prevState (i)
 signaled
 signaledTerminate
 stack (i)
 tasks (i)
 ticket
- FALSE
 "Waiting"
 (0 > "Coworkers" @ 0 1 > "Notify" @ 0 2 > "Coworkers" @ 0 3 > "StillValid")
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
- 22: Notify in event_notifiers >
- condVar
msgToParent
 pc (i) M
 phase
 pid (i)
 pid_ (i)
 prevState (i)
 signaled
 signaledTerminate
 stack (i)
 tasks (i)
 ticket
- FALSE
 "Waiting"
 (0 > "Coworkers" @ 0 1 > "InvalidTicket" @ 0 2 > "Coworkers" @ 0 3 > "StillValid")
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
 (0 > defaultInitValue @ 0 1 > defaultInitValue @ 0 2 > defaultInitValue @ 0 3 > defaultInitValue)
- 23: StillValid in event_notifiers >
- condVar
msgToParent
- FALSE
 "Waiting"



Proving my code bug-free

- ◊ This helped me debug 2 things I probably wouldn't be able to debug otherwise:
 - Livelock sequences of 20 to 40 steps that repeated themselves when I tried to properly order my wakeups.
 - Proving that my code was bug-free and so that even if it seemed improbable, it was glibc that was buggy.





Future goal

- ◊ Bring model checking to Nim
 - ThreadCollider: Formal Verification of Nim concurrent programs
 - Test a program correctness by testing all possible thread interleaving without having to rewrite it in another language.
 - <https://github.com/nim-lang/RFCs/issues/222>
 - <https://github.com/mratsim/weave/pull/127>



Multithreaded programs

The Good, the Bad and the Buggy



Mamy Ratsimbazafy
mamy@numforge.co

Weave
<https://github.com/mratsim/weave>