Calling Handlers by Name in WebAssembly

August 25, 2025

Ishan Mishra, The University of Edinburgh

Resume & Suspend

Unnamed Handlers	Named Handlers
`suspend $e :` `  [t1*] -> [t2*]`	
`resume $ct (on $e $l)* :` `  [t1* (ref $ct)] -> [t2*]`	

```
where:
  $e is a control tag
  $ct is some continuation
  $l is some label
```

Resume & Suspend (ii)

Unnamed Handlers

```
suspend $e :
  [t1*] -> [t2*]
```

```
resume $ct (on $e $l)* :
  [t1* (ref $ct)] -> [t2*]
```

Named Handlers

```
suspend_to $hname $e :
        [t1* (ref $ht)] -> [t2*]
```

```
resume_with $ct (on $e $l)* :
        [t1* (ref $ct)] -> [t2*]
```

```
where:
  $e is a control tag
  $ct is some continuation
  $l is some label
  $hname is the handler name (new)
  $ht is the handler name ty (new)
    - $ht : (handler (result t*))
    - t* are the types left on the stack post suspend
```

`suspend_to` enforces specifying a handler by name

`resume_with` generates a handler name to pass down

Nested Unnamed Handlers, Same Control Tag

Example Structure[1] :

```
outer_handler[read() -> 10]:
  inner_handler[read() -> 42]:
    print(read() + read())
```

Client Code

```
(func $client
  (local $a i32)
  (suspend $read)
  (local.set $a)
  (suspend $read)
  (i32.add (local.get $a))
  (call $print-i32)
)
```

[1] *Adapted from Biernacki D, Piróg M, Polesiuk P, Sieczkowski F. (1)*

Nested Unnamed Handlers, Same Control Tag (ii)

Outer handling function:

```
(func $environ-outer (param $k (ref $ct_inner))
 (param $k_client (ref $ct_client))
  ;; call inner handler function
  ...
  (local.set $k_read) ;; on a suspend $read
  ;; outer handler returns 10
  (loop $loop
    (block $on_read (result (ref $ct_client_read))
      (resume $ct_client_read (on $read $on_read) (i32.const 10) (local.get
$k_read))
      (return)
    ) ;; on_read
    (local.set $k_read)
    (br $loop)
))
```

- Inner handler function is identical, except it returns 42 on a read()
- Client will always print 84, outer handler is completely ignored

Nested Named Handlers, Same Control Tag

Modified client code with named handlers

```
(func $client (param $hname_inner (ref $ht)) (param $hname_outer (ref $ht))
  (local $a i32)

  ;; specifying inner handler and receiving an updated name for it
  (suspend_to $ht $read (local.get $hname_inner))
  (local.set $hname_inner)
  (local.set $a)

  ;; specifying outer handler and receiving an updated name for it
  (suspend_to $ht $read (local.get $hname_outer))
  (local.set $hname_outer)
  (i32.add (local.get $a))

  (call $print-i32) ;; now prints 52!
)
```

For `$hname_inner` or `$hname_outer` to get passed to client, handling code can replace `resume` for `resume_with`

Accidental Handling

Possible to erroneously utilise the “wrong” handler. Consider the following higher-order tracing function.

```
(type $ft_unit (func))
(type $ct_unit (cont $ft_unit))
(type $ft_trace (func (param (ref $ct_unit))))
(type $ct_trace (cont $ft_trace))
(tag $log)

(func $tracer (type $ft_trace) (param $f (ref $ct_unit))
  (local $k (ref $ct_unit))
  ;; sets up its own handler for $log, clashing with any outer handler for $log
  (block $on_log (result (ref $ct_unit))
    (resume $ct_unit (on $log $on_log) (local.get $f))
    (return)
  )
  (local.set $k)
  (call $print-i32 (i32.const 1))
)
```

Accidental Handling (ii)

Client Code:

```
(func $foo (type $ft_unit)
  (suspend $log)  ;; expects to log 0
)
```

Application Code:

```
(func (export "application")
  (local $k (ref $ct_unit))
  (block $on_log (result (ref $ct_unit))
    (resume $ct_trace (on $log $on_log) (cont.new $ct_unit (ref.func $foo))
(cont.new $ct_trace (ref.func $tracer)) )
    (return)
  ) ;; below is shadowed by tracer's own handler
  (local.set $k)
  (call $print-i32 (i32.const 0))
)
```

Explicit Handling via Named Handlers

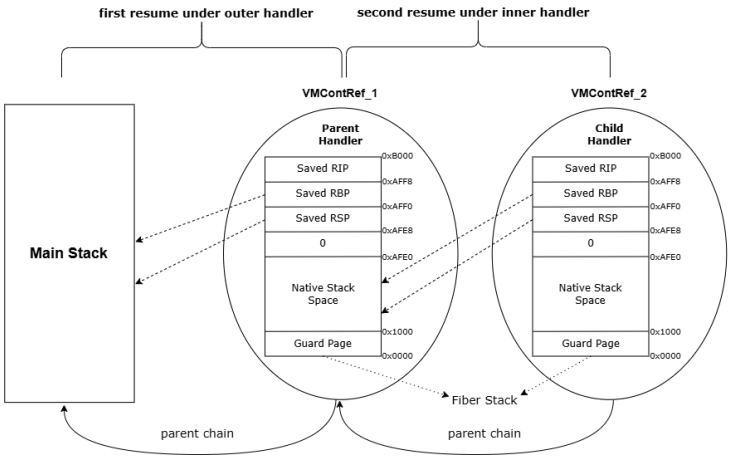
```
(type $ht (handler)) ;; handler name type

(func $foo (type $ft_unit)
  (param $hname_app (ref $ht))
  (param $hname_tracer (ref $ht))
    ;; explicitly using handler defined by the application
    (suspend_to $ht $log (local.get $hname_app))
    (local.set $hname_app)
)
```

As the names `hname_app` and `hname_tracer` get passed down on each `resume_with`, no ambiguity when suspending.

Named Handler Wasmtime Implementation

Piggybacks on the current segmented stacks model for representing continuations. Handler names are the address in memory of the continuation's `VMContRef`.



Benchmarks

Backend	Itersum	Treesum	Sieve	Sieve (SP)
Unnamed WasmFX	1.02x	0.33x	5.67x	6.33x
Named WasmFX	1.04x	0.36x	7.62x	8.65x
Unnamed Asyncify	1.04x	0.92x	0.86x	0.97x
Named Asyncify	**Baseline**			

Table 5.1: Runtime Increase Comparisons (Lower is Better)

Backend	Itersum	Treesum	Sieve	Sieve (SP)
Unnamed WasmFX	1.04x	1.34x	4.45x	5.16x
Named WasmFX	1.06x	1.51x	5.29x	6.00x
Unnamed Asyncify	1.00x	0.91x	1.00x	1.00x
Named Asyncify	**Baseline**			

Table 5.2: Heap Allocation Increase Comparisons (Lower is Better)

Backend	Itersum	Treesum	Sieve
Unnamed WasmFX	0.57x	0.59x	0.60x
Named WasmFX	0.59x	0.59x	0.60x
Unnamed Asyncify	0.98x	1.00x	1.00x
Named Asyncify	**Baseline**		

Table 5.3: Binary Size Increase Comparisons (Lower is Better)

Extension: Static Handler Shapes

Searching for a named handler requires matching both the name and tag

Some improvement could be made if the handler name type also encoded tags that the associated handler must define logic for

```
(type $handler_name (handler (tags $read $write) ts*)
```

When searching for a relevant named handler, only the name would then be required

Extension: Switch and Resume Throw

Yet to figure out proper semantics for named variants: `switch_with` or `resume_throw_with`.

This project restricted its scope to just `resume` and `suspend` for a minimal but working extension.

Alternative: First-Class Tags

Idea: Allow creation of tags[2]

```
(type $t (tag (result i32)))
(type $ft (func (param (tagref $t))))
(type $ct (cont $ft))

(func $f (param $k (ref $ct))
   (tag $read (result i32)) ;; locally-generated tag
   ...
    (resume $ct (on $read $l) (local.get $read) (local.get $k))
   ...
)

(func $g (param $read (tagref $t))
   (suspend $t (local.get $read))
)
```

[2]Inspired from Vilhena PE de, Pottier F. (2)

Alternative: First-Class Tags (ii)

Named Handlers can be encoded as:

```
resume_with $ct (on $e $l) = (tag $t' (type-of $t'))
                             (resume $ct' (on $t' …) …)

suspend_to $hname $tag = suspend $tag …
```

TODO: Need to investigate further if this level of expressivity is necessary

Resources

- Reference Interpreter PR
- Binaryen w/ Handler Names PR
- Wasmfxtime w/ Handler Names Fork
- Wasmfx-tools w/ Handler Names Fork
- My UG4 Thesis
- Fiber-C PR

Code from Examples (links to Github Gists):

- Nested Unnamed Handlers
- Nested Named Handlers
- Accidental Handling
- Explicit Handling via Named Handlers

References

1. Biernacki D, Piróg M, Polesiuk P, Sieczkowski F. Binders by day, labels by night: effect instances via lexically scoped handlers. Proc ACM Program Lang [Internet]. 2020;4(POPL):1–29. Available from: https://doi.org/10.1145/3371116

2. Vilhena PE de, Pottier F. A Type System for Effect Handlers and Dynamic Labels. In: Wies T, editor. Springer; 2023. pp. 225–52. (Lecture Notes in Computer Science; vol. 13990). Available from: https://doi.org/10.1007/978-3-031-30044-8/_9

3. Xie N, Cong Y, Ikemori K, Leijen D. First-class names for effect handlers. Proc ACM Program Lang [Internet]. 2022;6(OOPSLA2):30–59. Available from: https://doi.org/10.1145/3563289