suspension egg

Serialized partial continuations
Extension for Chicken Scheme
Version 0.1

felix

Table of Contents

1 About this egg

1.1 Version history

`0.1` Initial release

1.2 Requirements

This egg requires the following extensions:

`s11n`, `CHICKEN must be configure with support for procedure tables`

1.3 Usage

Load this egg like so:

```
(require-extension suspension)
```

2 Documentation

A *suspension* is a serialized partial continuation. This extension provides basic tools for working with suspensions and may be useful for implementing continuation-based *modal* behaviour in applications like web-servers.

Note that suspensions can only be reliably created for compiled code, as continuations created by the interpreter will contain more references to live data as needed - in particular the environment representation of the interpreter is simpler and retains more garbage. Continuations created by compiled code only reference the minimal set of live data required to continue execution and thus are much smaller.

Not all data is serializable, ports and foreign pointers are not, for example. To be able to create a suspension from a pending continuation, no references to such data may exist in the continuation (say, in local bindings which are accessed by code executed when the continuation is resumed).

The implementation of delimited execution contexts creates a separate thread so `(current-thread)` will not be `eq?` to the thread that created the delimited execution context. Parameters modified in the new execution context are propagated back to the parent thread, though. Also, any exceptions raised in the new context will be delivered to the parent thread.

The default ports (`current-[input,output,error]-port`) are bound to the default STDIO-ports during execution of a limited execution context.

`with-limited-continuation` [procedure]

```
(with-limited-continuation THUNK)
```

Creates a limited execution context by spawning a new thread and suspending the current one. THUNK should be a procedure of no arguments and will be executed in th new context. Any results returned by THUNK will be returned by the `with-limited-continuation` form.

`continuation-suspend` [procedure]

```
(continuation-suspend STORE)
```

Captures the current continuation up to the point of the innermost enclosing `with-limited-continuation` form and invokes the 1-argument procedure STORE with the serialized representation (a string) of the continuation. The execution context will be exited and the enclosing `with-limited-continuation` form will return any results returned by STORE.

The effect of invoking `continuation-suspend` outside of the dynamic context of a `with-limited-continuation` is undefined.

`continuation-resume` [procedure]

```
(continuation-resume SK RESULTS ...)
```

Deserializes the continuation in the string SK and invokes it, continuing the execution suspended and stored in the serialized continuation. The given results will be returned from the `continuation-suspend` form that suspended SK.

The effect of invoking `continuation-resume` outside of the dynamic context of a `with-limited-continuation` is undefined.

`continuation-drop` [procedure]

```
(continuation-drop RESULTS ...)
```

Exits the execution context created by the innermost enclosing `with-limited-continuation` form and returns RESULTS ...

The effect of invoking `continuation-drop` outside of the dynamic context of a `with-limited-continuation` is undefined.

3 Examples

This example demonstrates how to use suspensions for continuation-based programming in the spiffy web-server:

```
;;;; webtest.scm
;
; Compile and start and point your browser to "http://localhost:8080/count".
; Click "Next" to invoke the continuation and try cloning the window.
; (Note how several windows share the global "j" but have private "i"
; bindings).


(use spiffy suspension posix utils)


(unless (file-exists? "continuations")
  (create-directory "continuations") )


(define-http-resource (k id)
  (with-limited-continuation
   (lambda ()
     (if id
         (let ((k (with-input-from-file (make-pathname "continuations" id) read-all)))
           (spiffy-debug "read: ~a" id)
           (continuation-resume k #f) )
         (error "missing `id' argument") ) ) ) )

(define (send/suspend proc)
  (continuation-suspend
   (lambda (k)
     (let ((id (conc (current-seconds) (random 1000000))))
       (with-output-to-file (make-pathname "continuations" id)
         (cut display k) )
       (spiffy-debug "written continuation ~a ..." id)
       (proc (conc "k?id=" id)) ) ) ) )

(define j 1)

(define-http-resource (count)
  (with-limited-continuation
   (lambda ()
     (let loop ((i 1))
       (send/suspend
        (lambda (kurl)
          (conc "<h1>Count: " i " / " j "</h1><a href='" kurl "'>next</a>") ) )
       (set! j (+ j 1))
```

```
       (loop (+ i 1)) ) ) ) )

(start-server debug: #t root: "." port: 8080)
```

4 License

Index

C

W