myloop2.ss 2020-01-02 16:10:21

```
(shadowing-intern "bye" "sys")
(define *r-e-p-start* #f)
(set! bye (lambda()(if *r-e-p-start* (catch (save-
       his)))(newline)(sys:bye)))
(define quit sys:bye)
;;;(in-package "sys")
(define sys:undef (if #f #f))
(define (get-his-file)
  (let* ((home (tk-string *HOME* "/.ss2/"))(x #f))
    ;;;(format #t "Debug home ~s~%" home)
    (if (not (file-exists? home))
       (set! x (catch (mkdir home)))
    )
   (if x  ;; error
       #f
      (file-exists? (tk-string home 'line-history)
      )
   )
))

(define (save-his)
  (let* ((file (tk-string *HOME* "/.ss2/line-histo
  ry"))(x #f)
        (out 0) (v *line-history*))
      ;;;(format #t "Debug -save-his  ~s~%" file)
    (set! out (open-output-file file))
    ;;;  i=0   is (bye)   omitted!
  (when out
    (do ((i 1 (+ i 1)) (n (vector-size v)))
        ((>= i n))
        (set! x  (vector-ref v i))
        (if (eq? x sys:undef)
            (set! i n)
            (format out "~s~%" x)
        )
    )
  (close-port out)
 )
))

(define (get-my-his)
  (let* ((file (get-his-file))(in 0)(x #f)(n 0)(m
  0)
           (his ()) (v *line-history*))
    (when file
      ;;;(format #t "Debug his  ~s~%" file)
      (set! in (open-input-file file))
      (if verbose-mode
         (format *stdmsg* "  history-file:~s~%" fi
         le )
      )
      (set! x (read in))
      ;;;(format #t "x:~s ~%" x)
      (while (not (eof-object? x))
         (if (not (member x his))
            (push x his)
         )
         (set! x (read in))
     )
     (set! n (length his))
     (set! m (vector-size *line-history*))
     (when (> n (- m 5))
         (if (< n 50) (set! n (+ n 10)))
         (set! v (make-vector n  sys:undef))
         (set! *line-history* v)
     )
     (set! his (reverse his))
     (set! n 0)
     (while (pair? his)
        (vector-set! v n (pop his))
        (inc n)
     )
    (close-port in)
   )
))



(define (eval-hook x)
     (catch-error-only (lambda()(values->list (eva
     l x)))
         error-out))

(define (error-out e obj)
        ;;; (print-error e obj)
     (if (eq? e E_hookreturn)
         (format #t "hook return myloop~%")
         (my-print-error e obj)
     )
  ;;;*error-return*
)
```

```
(define (my-print-error e obj)
   (let* ((h (call-history)) (i 0))
     (print-error e obj)
     (format #t "ENV:~s~%" *errorenv*)
     (set! h (cdr h))
     (while (pair? h)
        (format #t "~s:~s~%" i (pop h))
        (inc i)
     )
))

(define (myloop)
  (let* ((x 0)(v 0)(etop ())(c 0))
   ;;; (vector-set! *line-history* 0 "(fact 10)")
   ;;;
          ;;;; if you need, you can change *line-h
          istory*
          ;;;;  by any vector ,any size initialize
          d by undef!!
    (if (not  *r-e-p-start*)
        (begin
         (if *tcl-version* (load "error-gui"))
         (get-my-his)
        )
    )
    (set! *r-e-p-start* #t)
    (set!  x (call/cc (lambda(e) (set! etop e) eto
    p)))
    ;;;(format #t "error-top ~s~%" x)
    (if (eq?  x etop)
       (begin  ;;; first
          (set! *syserrorhook* (lambda(e obj) (err
          or-out e obj) (etop e obj)))
        )
    )
     (set! x *errorhook*)
     (if (not (procedure? x))
        (set! *errorhook* *syserrorhook*)
     )
    (checkstack) ;;; Not Use!
  (while (not (eq? v :topsystem))
    ;;; (format #t "*errorhook* ~s~%" *errorhook*)
    ;;;(checkstack) ;;; Not Use!
     ;(display (package-name sys:*package*))
     ;(display "> ")
     ;;(display "::")
     (display *prompt*)
     (flush)
    ;; (while (or (eof-object?  (begin (set! c (in
    key #f #f)) c))
    ;;                 (char-whitespace? c))
    (update 'ss)
    (while (not (char-ready?))
         (update 'ss)
         (after 100)
         ;;;(display "wait ")
         (flush)
        (if *tcl-version*
             (if (top-visible?) (update))
            ;;; (update)
        )
     )
    ;;;(format #t "~s ~s~%" c (char->integer c))
    ;;(unread-char c)
     (system-input #t)  ;;; for line edit
     (set! x (read))
     (line-history)
     (system-input #f)
     ;;(set! x (eval-hook x))
     ;;(set! x (values->list (eval x)))
       (reset-history)
      (set! x (syntax-expand x ()))
      (set! x (vm-compile (list 'begin x '(INLINE
      MULTI->SCM)) ()))
       (reset-history)
      ;;(set! x (values->list (CALL-VM x ())))
      (set! x (CALL-VM x ()))
     (cond
          ( (pair? x)
            (display "Value: ")
            (set! v (pop x))
            (write v)
            (while (pair? x)
            (newline)
            (write (pop x))
           )
          )
        (else
           (display "NoValue: ")
           (write sys:undef)
         )
     )
     (newline)
)
```

```
))

;;;(myloop)
#|
 (if (not  *r-e-p-start*)
        (begin
         ;;;(if *tcl-version* (load "error-gui"))
          (get-my-his)
        )
  )
|#
(get-my-his)
(set! *r-e-p-start* #t)
(set! sys:*r-e-p* myloop)
```