Implementace cyklu

```
;; implementace cyklu
(define-macro my-do
  (lambda (binding condition . body)
    `(letrec ((my-loop (lambda ,(map car binding)
                         (if ,(car condition)
                             (begin ,@(cdr condition))
                             (begin ,@body
                                    (my-loop ,@(map caddr binding)))))))
       (my-loop ,@(map cadr binding)))))

;; implementace cyklu s okamžitým opuštěním
(define-macro my-do
  (lambda (binding condition . body)
    `(call/cc
      (lambda (break)
        (letrec ((my-loop (lambda ,(map car binding)
                            (if ,(car condition)
                                (begin ,@(cdr condition))
                                (begin ,@body
                                       (my-loop ,@(map caddr binding)))))))
          (my-loop ,@(map cadr binding)))))))

;; implementace continue a break
(define-macro my-do
  (lambda (binding condition . body)
    `(call/cc
      (lambda (break)
        (letrec ((my-loop (lambda ,(map car binding)
                            (call/cc (lambda (continue)
                                       (if ,(car condition)
                                           (break ,@(cdr condition))
                                           (begin ,@body))))
                            (my-loop ,@(map caddr binding)))))
          (my-loop ,@(map cadr binding)))))))
```

Ukázka použití

```
;; příklad použití
(my-do ((x '(1 3 5 7 9) (cdr x))
        (sum 0 (+ sum (car x))))
       ((null? x) sum)
       (if (= (car x) 3)
           (continue))
       (if (> sum 10)
           (break sum))
       (display sum)
       (newline))
```

Jednoduchý paralelní systém

```
;; definice symbolů pro funkce
(define round-robin #f)
(define stop #f)
(define add-task #f)

;; paralelní motor
(let ((queue ())     ; fronta procesů
      (restart #f)) ; restart cyklické obsluhy procesů

  ;; přidej úlohu do fronty
  (set! add-task
        (lambda (task)
          (set! queue (append queue (list task)))))

  ;; zastav vykonávání aktuálního procesu, zařaď proces na konec fronty
  ;; a vyvolej pokračování dalšího cyklu
  (set! stop
        (lambda ()
          (call/cc (lambda (stop-point)
                     (add-task stop-point)
                     (restart)))))

  ;; cyklická obsluha procesů: vyber z fronty první proces,
  ;; vyvolej jej v aktuálním pokračování s navázaným restartem
  (set! round-robin
        (lambda ()
          (if (not (null? queue))             ; dokud není fronta prázdná
              (let ((first-task (car queue)) ; první proces
                    (the-rest (cdr queue)))  ; zbytek fronty

                (set! queue the-rest)         ; zkrať frontu

                (call/cc
                 (lambda (next-iter)
                   (set! restart next-iter)  ; nastav restart
                   (first-task)))            ; zavolej první proces ve frontě

                (round-robin))))))           ; cyklická obsluha procesů
```

Ukázka použití

```
;; definuj funkci pro výpočet faktoriálu
(define (f x n)
  (begin
    (display x)
    (newline)
    (display n)
    (newline)
    (newline)
    (stop)
    (if (= n 0)
        1
        (* n (f x (- n 1))))))

;; přidej dvě úlohy
(add-task (lambda () (f "PRVNI" 4)))
(add-task (lambda () (f "DRUHY" 5)))

;; spusť paralelní systém
(round-robin)
```