Processes without Partitions

Matthew Flatt
University of Utah

Adam Wick
University of Utah

Robert Bruce Findler
University of Chicago
This talk has been modified from its original version. It has been edited for content and formatted to fit your screen.
Machines and Processes
Machines and Processes
Machines and Processes
Machines and Processes

\[ \lambda \]

\[ \lambda \]

\[ 101010 \\
010101 \]
Machines and Processes

λ

λ

OS

101010
010101

101010
010101
Machines and Processes
Process Examples
Process Examples
Process Examples
Process Examples

gregs@ai.mit.edu (Gregory T. Sullivan)
To: LL1 Discussion <ll1-discuss@ai.mit.edu>
Subject: LL2, Sat. Nov. 9, 2002 - webcast info.
Date: 06 Nov 2002 11:51:57 -0500

Thanks to the generous sponsorship of Microsoft and Dr. Dobbs Journal, we will be able to webcast the LL2 proceedings this coming Saturday, Nov. 9. The morning session (10am to 12:35pm EST) can be viewed at:

http://web.mit.edu/webcast/ailab/mit-ll2-s1-05nov02-80k.ram

and the afternoon, from 1:45pm to 6pm will be at:

http://web.mit.edu/webcast/ailab/mit-ll2-s2-05nov02-80k.ram
Process Examples

DrScheme

user's program

Run DrScheme
Languages with Termination

Pilot [Redell80]
JKernel [Hawblitzel98]
KaffeOS [Back00]
SPIN [Bershad95]
Alta [Tullman99]
JSR-121 [Soper03]

.NET application domains

...
Languages with Termination

PLT Scheme
Motivation and Approach

Processes in PLT Scheme

- Threads
- Parameters
- Eventspaces
- Custodians

Memory Accounting
Threads

Concurrent execution

```
(require "spin-display.scm")  eval

(define (spin)
  (rotate-a-little)
  (sleep 0.1)
  (spin))

(define spinner (thread spin))  eval

(kill-thread spinner)  eval
```
Parameters (a.k.a. Fluid Variables)

Thread-local state

(printf "Hello\n")
_printf (current-output-port) "Hola\n")
(fprintf (current-error-port) "Goodbye\n")
(error "Ciao")

(eval
(parameterize ((current-error-port (current-output-port)))
  (error "Au Revoir")))

(eval
(parameterize ((current-error-port (current-output-port)))
  (thread
    (lambda ()
      (error "Zai Jian"))))

(eval
(thread (lambda () (message-box "One" "Hi")))
(thread (lambda () (message-box "Two" "Bye")))  eval

(thread (lambda () (message-box "One" "Hi")))
(parameterize ((current-eventspace (make-eventspace)))
  (thread (lambda () (message-box "Two" "Bye")))  eval)
Custodians

Termination and clean-up

(define c (make-custodian))
(parameterize ((current-custodian c))...

(custodian-shutdown-all c) eval
Etc.

• Security Guards
  
  \textit{Resource access control}

• Namespaces

  \textit{Global bindings}

• Will Executors

  \textit{Timing of finalizations}

• Inspectors

  \textit{Debugging access}
Building a Programming Environment

SchemeEsq, a mini DrScheme [ICFP 99]
GUI - Frame

(define frame
  (new frame%
      [label "SchemeEsq"]
      [width 400] [height 175]))

(send frame show #t)

eval
GUI - Reset Button

(new button%
  [label "Reset"]
  [parent frame]
  [callback (lambda (b e) (reset-program))])

(eval)
GUI - Interaction Area

(define repl-display-canvas
  (new editor-canvas%
      [parent frame]))

(eval)
(define esq-text%)
  (class text% ... (evaluate str) ...))

(define repl-editor (new esq-text%))
(send repl-display-canvas set-editor repl-editor)
Evaluator

(define (evaluate expr-str)
  (thread
    (lambda ()
      (print (eval (read (open-input-string expr-str)))))
    (newline)
    (send repl-editor new-prompt))))
(define user-output-port
  (make-output-port ... repl-editor ...))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port))
    (thread
      (lambda ()
        (lambda ()
          ...))))))
Evaluating GUIs

(define user-eventspace (make-eventspace))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port)
                 (current-eventspace user-eventspace))
    (thread
      (lambda ()
        ...
      ))))

eval
Custodian for Evaluation

(define user-custodian (make-custodian))

(define user-eventspace
  (parameterize ((current-custodian user-custodian))
    (make-eventspace)))

(define (evaluate expr-str)
  (parameterize ((current-output-port user-output-port)
      (current-eventspace user-eventspace)
      (current-custodian user-custodian))
    (thread
      (lambda ()
        ...))))

(eval)
(define (reset-program)
  (custodian-shutdown-all user-custodian)

  (set! user-custodian (make-custodian))
  (parameterize ((current-custodian user-custodian))
    (set! user-eventspace (make-eventspace)))
  (send repl-editor reset))
Motivation and Approach

Processes in PLT Scheme

Memory Accounting

• Without partitions [ISMM 04]
Resource Consumption

DrScheme

user's program
Resource Consumption

DrScheme

user's program
Resource Consumption

DrScheme

use your program
Resource Consumption

DrScheme

λ

us

λ

am
Resource Consumption

DrScheme
Resource Accounting

- Conventional OS: process memory use = size of partition

- Accounting is easy
- Trading data is difficult
Resource Accounting

- **Language as OS**: process memory use = size of owned data

- Trading data is easy

- Accounting *appears* difficult: sharing, real-time tracking
Our strategy: compute accounting charges during GC

See also [Price03]
Basic Accounting

\[ \lambda = \text{custodian A} \]
\[ \lambda = \text{custodian B} \]
Basic Accounting

\[ \lambda = \text{custodian A} \]

\[ \lambda = \text{custodian B} \]

thread 1

\[ x \rightarrow q \]

\[ A \rightarrow A \]

thread 2

\[ y \rightarrow r \]

\[ A \rightarrow A \]

thread 3

\[ z \rightarrow s \]

\[ B \rightarrow B \]
Sharing

\[ \lambda = \text{custodian A} \quad \text{thread 1} \quad x \quad z \]

\[ \lambda = \text{custodian B} \quad \text{thread 2} \quad y \]

\[ 46 \]
Sharing

\[ \lambda = \text{custodian A} \]

\[ \lambda = \text{custodian B} \]

thread 1

thread 2

A

B

A or B
Sharing: Charge the Child
Threads, Custodians, and Weak References

$\lambda = \text{custodian A} \rightarrow \text{thread 1}$

$\lambda = \text{custodian B} \rightarrow \text{thread 2}$
Threads, Custodians, and Weak References

\[ \lambda = \text{custodian A} \rightarrow \text{thread 1} \]

\[ \lambda = \text{custodian B} \rightarrow \text{thread 2} \]
Threads, Custodians, and Weak References

![Diagram showing relationships between threads, custodians, and weak references.]
Threads, Custodians, and Weak References

\[ \lambda = \text{custodian A} \rightarrow \text{thread 1} \]

\[ \lambda = \text{custodian B} \rightarrow \text{thread 2} \]
Why Charge the Child?

- Parent is responsible for children
- Parent may allocate for children
  - GUI objects
  - File descriptors
  - ...

\[
\lambda = \text{custodian A} \xrightarrow{\text{thread 1}} x \xrightarrow{A} z \xrightarrow{B, A} \\
\lambda = \text{custodian B} \xrightarrow{\text{thread 2}} y \xrightarrow{B} \]
Initial Experience: DrScheme

```
(define (deep n)
  (+ (deep n)));
> (deep 0)
```

Bad Loop

```
(define (loop n)
  (loop (+ n 1)));
> (loop 0)
```

Normal

```
(define (loop n)
  (loop (+ n 1)));
> (loop 0)
```

Normal
Initial Experience: DrScheme

Bad Loop

Normal

Shut Down

(define (deep n) (+ (deep n)))
> (deep 0)

(define (loop n) (loop (+ n 1)))
> (loop 0)

(define (loop n) (loop (+ n 1)))
> (loop 0)
DrScheme Bug

\[ \lambda = \text{DrScheme} \rightarrow \text{thread 0} \]

\[ \lambda = \text{User1} \rightarrow \text{thread 1} \]

\[ \lambda = \text{User2} \rightarrow \text{thread 2} \]
DrScheme Repair

\[ \lambda = \text{DrScheme} \rightarrow \text{thread 0} \]

\[ \lambda = \text{User1} \rightarrow \text{thread 1} \]

\[ \lambda = \text{User2} \rightarrow \text{thread 2} \]
DrScheme Repair

\[ \lambda = \text{DrScheme} \quad \text{thread 0} \]
\[ \lambda = \text{User1} \quad \text{thread 1} \]
\[ \lambda = \text{User2} \quad \text{thread 2} \]

Changed 5 references:
- Weakened 2
- Removed 2
- Moved 1 into child
Current Experience: DrScheme

Bad Loop

Normal

Normal
Current Experience: DrScheme

```
(define (deep n)
  (+ (deep n))))

> (deep 0)
```

Shut Down

```
(define (loop n)
  (loop (+ n 1)))

> (loop 0)
```

Normal

```
(define (loop n)
  (loop (+ n 1)))

> (loop 0)
```

Normal
Accounting without Partitions

Useful accounting

• Doesn't need partitions
• Does need hierarchy
Conclusion

• Programmers need OS-like constructs in languages
  ○ concurrency
  ○ adjust run-time environment
  ○ easy termination

• Multiple language constructs for “process”
  ○ programmer can mix and match to balance isolation and cooperation