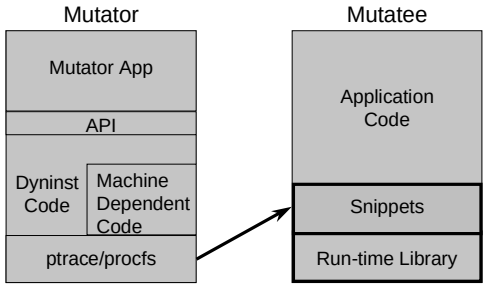
dyninstAPI: Features and Future

Jeffrey K. Hollingsworth

© Copyright 1998, Jeffrey K. Hollingsworth, All Rights Reserved.



## The Dyninst Library






## dyninstAPI

- Provides:
  - Functions for control of mutatee
  - Runtime code generation
  - Information about mutatee
  - requires mutator/mutatee to be on the same machine
- Leverages code from Paradyn
- A set of C++ classes
  - BPatch_thread - process/thread control
  - BPatch_image - program code abstractions
  - BPatch_snippet - code placed into a program
  - BPatch_*Expr - bits of code be placed into a program



## BPatch_thread Class

- Represents a thread of execution
  - Unix process for now, support for threads soon
- Starts/stops thread
  - stopExecution(), continueExecution()
  - terminateExectution()
- Inserts code snippets into threads
  - insertSnippet(snippet, point, pre or post, snippet order)
- Allocates memory in thread's address space
  - malloc(number of bytes)
  - malloc(type)



## BPatch_image Class

- Represents an executable program
  - Symbol table and line numbers info
- Locate functions
  - findFunction(name)
  - Returns a BPatch_function
- Locate points for snippets
  - findProcedurePoint(procedure name, location)
  - Returns a list of BPatch_point
  - Location can be one of:
    BPatch_entry, BPatch_exit, BPatch_subroutine,
    BPatch_longJump, BPatch_allLocations



## BPatch_image Class Continued

- Locate variables
  - findVariable(scope, name)
  - If scope is omitted, searches only global scope
  - Returns a BPatch_variableExpr
- Locate application defined types
  - findType(name)
  - Returns a BPatch_type

## BPatch_snippet

- Code to be inserted into program
- Base class representing statements
  - BPatch_funcCallExpr: a function call
  - BPatch_ifExpr: a conditional expression
  - BPatch_arithExpr: an arithmetic expression
  - BPatch_constExpr: a numerical or string constant
- Example:

```
aVar = appImage->findVariable("fooCount");
BPatch_arithExpr expr(BPatch_plus,
                      *aVar, BPatch_constExpr(1));
```



## BPatch_variableExpr Class

- Variables in the thread's address space
- Mutator allocated dynamic memory
  - Returned by BPatch_thread
- Type system
  - Base types: int, float, constant string
  - Types defined in instrumented program



## API Example

```
// Attach to the program
BPatch_thread *appThread = new BPatch_thread("foo", pid);

// Read the program's image
BPatch_image *appImage = appThread->getImage();

// Create a snippet to increment a counter
…

// Find the entry point to "bar"
BPatch_Vector<BPatch_point *> *points =
   appImage->findProcedurePoint("bar", BPatch_entry);

// Insert the snippet
appThread->insertSnippet(incrCount, *points);
```



## Status

- Released version:
  - Platforms: Sparc(Solaris), x86(NT,Solaris), Power(AIX)
  - Features:
    - process control (including attaching to running code)
    - code generation: integer expressions, function calls
    - variable manipulation: read/write scalars and blocks
- Current Issues:
  - missing features
    - one-shot (aka inferrior RPC), goto
    - recent features:
      - uniqueString, mangledName, baseAddr
    - Boolean and floating point types
    - Block and loop instrumentation
  - Ports: Linux, Alpha, MIPS



## API is at a cross road

- Has performance centric features
- New features needed for
  - correctness debuggers
  - performance steering
- Current effort is by a few people
  - how should evolution be driven?
- API vs. Implementation
  - is the value in the interface or its implementation?



## Command Interpreter

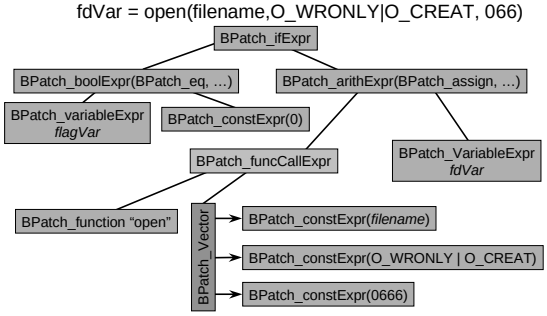
- API is good for building tools
  - create tools to construct snippets
- Expressions are hard to build directly
  - prototype TCL interpreter exists
    - useful for "small" tools
    - easier than writing expression by hand
- Would statement parser help the API?
  - This would extend the scope of the API
  - What would the interface look like? Strings?



© 1998 hollings@cs.umd.edu

## A Case for Improved Statements

```
if (flagVar == 0)
        fdVar = open(filename,O_WRONLY|O_CREAT, 066)
```




University of Maryland 13

## Type System

- Current API has primitive type checking:
  - integers: 32 bit signed
  - strings: variable length, null terminated (ala C)
  - hooks for complex types in API, but not implemented
- Also includes mechanism to bypass types
  - everything is a word
  - support for structs/classes via:
    - base pointer to complex types
    - arithmetic expressions for pointer arithmatic
- Question: Should there be a full type system?

University of Maryland 14

## What should snippets be for?

- Just enough to call functions
  - glue code for parameter passing
- Functions plus "simple code"
  - *current target of API*
  - includes simple integer expressions and variables
  - branch (goto) designed for looping
- Full statements and expressions
  - floating point expressions
  - runtime access to complex types
  - strong type system

University of Maryland 15

## Data collection

- Current API
  - use ptrace/procfs to implement reads
  - lacks synchronization with application
  - can be expensive
- Shared Memory
  - memory region shared by mutator/mutatee
  - still lacks synchronization
  - easier for mutator to create updates seen by mutatee
- Read/Write
  - application would need to produce data
  - no synchronization required

University of Maryland 16

## Remote Interface

- For distributed memory machines
  - would be helpful to use API remotely
  - permits "front-end" node to instrument all nodes
- Issues
  - RPC
    - what package to use?
      - Paradyn RPC, Java RMI, Corba, DCE, …
      - interaction with parallel operating environment
  - heterogeneous systems
    - basic snippet mechanism is machine independent
    - but, word sizes and other features vary.

University of Maryland 17