The Region Trap Library: Handling Traps on Application Defined Regions of Memory

Tim Brecht
Harjinder Sandhu

www.cs.uwaterloo.ca/~brecht
Problem

• Mismatch in granularity
  – Page size fixed (OS / Architecture)
  – Data Structure sizes vary (Application)

• Manage data at granularity of application

• Region = contiguous addresses of memory
Application Domains

- Distributed Shared Memory (DSM)
- Persistent Objects / Storage
- Garbage Collection
- Checkpointing
Example Domain: DSM

- Separate memory / host
- Logically shared
- Cache for performance
Data Larger than a Page

char array[40000];

• One page fault per page
• Action taken for each page
Data Smaller than a Page

char array[40000]; int x; int y;

- Can’t control faults when objs on same page
- “FALSE FAULTS” - determine if wanted fault
Region Trap: Goals

- User-level traps on regions not pages
- Similar to VM functions (mprotect)
- Wide Use:
  - C library (no lang dependent tricks)
  - Standard OS (no kernel mods)
  - Arch independent mechanisms
    (Linux/Pentium, AIX/PowerPC, Solaris/Sparc)
Requirements

• Control & detect access to regions
• Distinguish read and write traps
• Call handler on access (per region)
• Continue execution

• Can’t use mprotect (page based)
• Trap and call handler ? Use signals
Generating a Trap

• How to generate SIGSEGV?

```c
char *A = malloc(N);
A[0] = 'a';
A[1] = 'b';
```
Trapping: Pointer Swizzling

Kernel Space

User Space

region_ptr

0x1000 3000

Read / Write

USENIX, 1999
Trapping: Pointer Swizzling

Kernel Space

User Space

region_ptr

0x9000 3000

0x0000 0000

0x8000 0000

0x7FFFF FFFF

0x9000 3000

0x7FFFF FFFF

0x0000 0000

Read / Write

Invalid
Trapping: Pointer Swizzling

Kernel Space

0x4000 3000
region_ptr

User Space

0x0000 0000
0x7FFF FFFF
0x8000 0000
0xFFFF FFFF

Invalid
Read Only
Read / Write

USENIX, 1999
Region Trap Library

1 char A[N], *B, *C;
2 B = reg_define(A, N, &B);
3 C = reg_alloc(N, &C);
4 reg_handler(B, B_handler);
5 reg_handler(C, C_handler);
6 reg_protect(B, PROT_NONE);
7 reg_protect(C, PROT_READ);
8 B[i] = C[i];  /* B traps */
9 C[i] = i;    /* C traps */
10 A[i] = i;    /* does not trap */
Determining the Region

- AVL tree of regions (start and end addr)
- If faulting addr lies inside a region
  Find region in AVL tree
- Otherwise real SIGSEGV
  Stop catching SIGSEGV and resume
- User can’t use sigaction for SIGSEGV
  reg_handler(NO_REGION, handler);
Continue Execution

• Change protection level
  Fix (swizzle) region pointer and aliases
  \[\text{ptr} = \text{reg\textunderscore alloc}(N, &\text{ptr});\]
  \[\text{reg\textunderscore ptr}(\&\text{newptr});\]
  Fix (swizzle) register
  \[\text{ld } r2, (r3) \quad \text{st } r2, (r3)\]

• Call installed handler with useful params

• Leave sig handler (re-execute instruction)
Related Work

- Explicit checking [Hoskings & Moss, 93]
  [Zekauskas et al., 94] [Scales et al., 96]
- Program Annotation
  [Sandhu et al., 93] [Bershad et al., 93]
- Unaligned traps [Thekkath & Levy, 94]
- MultiView [Itzkovitz & Schuster, 99]
Example with Annotation

```c
for (i=0; i<N; i++)
ReadAccess(B[i]);

for (i=start; i<end; i++) {
  ReadAccess(A[i]);
  WriteAccess(C[i]);
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      C[i][j] += A[i][k] * B[k][j];
}
```
Example without Annotation

for (i = 0; i < N; i++)
    ReadAccess(B[i]);

for (i = start; i < end; i++)
    ReadAccess(A[i]);
    WriteAccess(C[i]);

for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
        C[i][j] += A[i][k] * B[k][j];
Application Performance

VM
RT
AN

1.0
0.8
0.6
0.4
0.2
0.0

IS
TSP
LU
SOR
MM
Application Performance

<table>
<thead>
<tr>
<th></th>
<th>MM</th>
<th>SOR</th>
<th>LU</th>
<th>FLOYD</th>
<th>TSP</th>
<th>IS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>RT</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

VM
RT
Conclusions

- Control / detect access to independent regions
- Machine speed access to unprotected regions
- RTL works on many Architectures and OSes
- Prototype implementation quite efficient
- Significant performance gains for some apps

http://www.cs.uwaterloo.ca/~brecht