Lecture 12 10/15/2010 – Baggy Bounds Checking

Baggy Bounds Checking – A referent object approach

This system is similar to the Jones & Kelly system with an alternative data structure. The system invariant is the fact that the pointer is always in bounds therefore you can find the bounds by looking up the enclosing interval.

Structure

- Allocated regions of memory are aligned to the next power of 2.

\[2^{\lfloor \log_2 l \rfloor}\]

- The advantage of this restriction (which has the consequence of potentially wasting nearly 50% on an allocation) is that the bounds are stored using the log of the allocation size.
- If aligned to \(2^l\) bits then the \(l\) least significant bits are 0

Assume a pointer \(p\) is a in a region \(l\) and the region then the bounds are determined by-

\[
\begin{align*}
L_0 &= p \& \neg(2^l - 1) \\
H_0 &= L_0 + 2^l
\end{align*}
\]

Storage

- We need to map \(p \rightarrow l\)
- This is implemented as a special purpose hash table as shown below

```cpp
byte L[2 ^ slot_size] // Byte array
l = L(p / slot_size) // Provides the lookup for l
```

- The slot size is set to 16.
Updates

\[
p = \text{malloc}(16);
L[p/slot\_size] = 4;
\]

- \(L\) is the log of the allocated size.

- Multiple 16 byte slots present a problem, all information must be updated.

Ex:

\[
p = \text{malloc}(32);
\text{for} \ (i = 0; i < 32/slot\_size \ ; ++i)\{
L[p/slot\_size++] = 5
\}
\]

Performance

- \(\text{lookup} = \mathcal{O}(1)\)
- \(\text{update} = \mathcal{O}(n)\)
- Lookups occur more often than updates so the slower time of update is amortized over the course of execution.
- Allocations are usually small so multi-slot updates occur less frequently.

Bounds checking

- Arithmetic check

\[
q = p + i;
\text{assert}( (p \ ^ \ q) \gg L[p/slot\_size]) == 0)
\]

- All pointers in the region have the same \(32 - \log_2\) most significant bits.

\[
\text{wyzw1111}(end) \quad \text{wyzw0000}(start)
\]

\(\text{wyzw0000} - \text{wyzw1111} = \text{The largest value is } 2^\log_2 - 1\)
Deference check

```c
assert (q ^ q + sizeof(*q - 1) >> L[q/size_size] ) == 0)
*q = 0;
```

Out of bounds pointers

System memory layout in a standard modern 32-bit operating system

```
The upper 2GB of memory are inaccessible (1xxx...x) bits
```

When a pointer goes out of bounds it the most significant bit can be set to 1 with
```
q = q | 0x80000000
```
This allows us to mark the pointer without causing major disruption (deference would
cause a segmentation fault or access violation depending on the operating system.

- If the pointer is out of bounds you can recover it by resetting the most significant bit to 0
and running a bounds check on the resulting value. This method allows the pointer to go out
of bounds during operations.

```c
if (q & 0x80000000) {
    q` = q & 0x7fffffff;
    if q` % 16 < 8;
        l = L[q/slot_size - 1];
    else
        l = L[q/slot_size++];
```