Dynamic space relocation

David Lichteblau

December 15, 2009
SBCL memory layout

+------------------------------------+
| | read-only | static | dynamic | sbcl.core |
+------------------------------------+ (not to scale...)

mmap each of them at startup to (Linux/x86):

- A few bytes of read-only space at #x01000000
- A few bytes of static space at #x01100000
- 512 MB of dynamic space #x09000000 to #x29000000
Issues # 1

- Why maintain this table manually in the first place?
Issues # 2

- What if Linux/nonmainstream hardware now has the stack at #x10000000?

- Would like sbcl.so, runnable in an existing process. What if the host process has something mmap()ed at #x25000000?

- Microsoft Windows
**Solution**

**Relocate spaces at startup to a suitable locations**
(Dynamic space only at this point.)
The relocation patch in a nutshell

```c
os_vm_address_t
-os_validate(os_vm_address_t addr, os_vm_size_t len)
+os_validate(os_vm_address_t addr, os_vm_size_t len, int fixedp)
{
    int flags = MAP_PRIVATE | MAP_ANON;

    - if (addr)
    + if (addr && fixedp)
        flags |= MAP_FIXED;

    addr = mmap(addr, len, OS_VM_PROT_ALL, flags, -1, 0);

    if (addr == MAP_FAILED) {
        perror("mmap");
        return NULL;
    }

    return addr;
}

(*BSD version)
Describe which part of memory moves where (can relocate multiple such segments simultaneously).

```c
struct relocation_segment {
    long *old_start;
    long *old_end;
    long displacement;
};
```

fixme: currently assumes sizeof(long) = sizeof(void*), will break on 64bit Windows
Dynamic space relocation: Segment at $ptr$, relocate from old position $old\_start$ to new position $old\_start + displacement$:

```c
void relocate_single(
    long *ptr, long nwords, long *old_start, long displacement);
```

Static space fixup: Segment unchanged, put points to something that has moved:

```c
void relocation_fixup(long *fixup_ptr,
    long n_fixup_words,
    int nsegments,
    struct relocation_segment *segments);
```

See other talk for use case: Segments still at old position, each relocated in place for a future position:

```c
void relocate_all(int nsegments, struct relocation_segment *segments);
```
looks mostly just like scav_* or ptrans_*

```c
static void
sub_relocate(long *ptr, long nwords, struct relocator *ctx)
{
    int nsegments = ctx->nsegments;
    struct relocation_segment *segments = ctx->segments;

    long *p;
    long *q = ptr + nwords;
    long nrelocated;
    int i;

    for (p = ptr; p < q; p += nrelocated) {
        long word = *p;
        if (is_lisp_pointer(word)) {
            long *address = (long *) native_pointer(word);
            for (i = 0; i < nsegments; i++)
                if (segments[i].old_start <= address
                    && address < segments[i].old_end)
                {
                    *p += ctx->segments[i].displacement;
                    break;
                }
            nrelocated = 1;
        } else {
            relocfn fn = reloctab[widetag_of(word)];
            if (fn)
                nrelocated = fn(p, ctx);
            else
                nrelocated = 1;
        }
    }
}
```
Demonstration