Gotchas when using linker sections to arrange data, part 2

devblogs.microsoft.com/oldnewthing/20181109-00

November 9, 2018

Raymond Chen

We saw last time that you need to accommodate potential padding between fragments within a section when walking through an array of pointers. Fortunately, it's a simple matter of skipping over null pointer entries.

Dealing with padding between fragments when you have a sequence of structures is more complicated, because the amount of padding may not be an exact multiple of the structure size.

```
struct THING
{
    const char* name;
    int value;
};

#pragma section("mydata$a", read)
__declspec(allocate("mydata$a")) \
    const THING firstThing{};

#pragma section("mydata$m", read)
#define ADD_THING(x, y) \
__declspec(allocate("mydata$m")) \
    const THING thing##x{#x, y}

#pragma section("mydata$z", read)
__declspec(allocate("mydata$z")) \
    const THING lastThing{};

// file1.cpp
ADD_THING(Red, 3);

// file2.cpp
ADD_THING(Blue, 4);

// file3.cpp
ADD_THING(Green, 0);
```

We would be tempted to write

```
// Code in italics is wrong.
void LessNaiveRegisterAllTheThings()
{
    auto begin = (uintptr_t)&firstThing + sizeof(firstThing);
    auto end = (uintptr_t)&lastThing;
    for (auto current = begin; current < end;
         current += sizeof(THING)) {
      auto thing = (const THING*)current;
      if (thing->name) RegisterThing(thing->name, thing->value);
    }
}
```

However this will run into trouble if padding is inserted that is not a multiple of `sizeof(THING)`. In that case, advancing by `sizeof(THING)` would eventually cause us to step over some padding bytes as well as the initial bytes of a valid `THING`.

We will have to walk the pointer past any null bytes until we find the start of a "good" structure.

This also means that zero cannot be a legitimate value for the first member of a "good" structure, because we wouldn't be able to figure out whether a zero value is the start of a "good" structure, or whether it's just padding.

In the above example, we know that the name is never null, so we can use that as our signal as to whether we have the start of a valid `THING`. If not, then we advance by the alignment of a `THING` and try again.

```
void RegisterAllTheThings()
{
    auto begin = (uintptr_t)&firstThing + sizeof(THING);
    auto end = (uintptr_t)&lastThing;
    auto current = begin;
    while (current < end) {
        auto thing = (const THING*)current;
        if (thing->name) {
            RegisterThing(thing->name, thing->value);
            current += sizeof(THING);
        } else {
            current += alignof(THING);
        }
    }
}
```

A less complicated alternative is to avoid generating structures into ordered segments and just use pointers exclusively.

```
#pragma section("mydata$a", read)
__declspec(allocate("mydata$a")) \
    const THING* const firstThing = nullptr;

#pragma section("mydata$m", read)
#define ADD_THING(x, y, s) \
    const THING thing##x{#x, y}; \
__declspec(allocate("mydata$m")) \
    const THING* const thing##x##ptr = &thing##x;

#pragma section("mydata$z", read)
__declspec(allocate("mydata$z")) \
    const THING* const lastThing = nullptr;

// file1.cpp
ADD_THING(Red, 3);

// file2.cpp
ADD_THING(Blue, 4);

// file3.cpp
ADD_THING(Green, 0);
```

At this point, we can use the “pointers” pattern.

```
void RegisterAllTheThings()
{
    auto begin = (uintptr_t)&firstThing
                 + sizeof(firstThing);
    auto end = (uintptr_t)&lastThing;
    for (auto current = begin; current < end;
         current += sizeof(const THING*)) {
      auto thing = *(const THING* const*)current;
      if (thing) RegisterThing(thing->name, thing->value);
    }
}
```

For extra style points, you could move the `firstThing` to `mydata$b` and generate the `THING` objects into `mydata$a`. This keeps all the `THING` objects contiguous in memory, which is more cache-friendly. It also keeps them close to the pointer table, which means that they will all page in/out together. Since this data is probably going to be used only at process startup, putting them all together lets them page out once and stay out.

Raymond Chen

Follow