What is a static chain pointer in the context of calling convention ABI?

devblogs.microsoft.com/oldnewthing/20231204-00

December 4, 2023

Raymond Chen

Buried deep in the System V Application Binary Interface document for the AMD64 Architecture, there is a footnote on page 24 that says, “`%r10` is used for passing a function’s static chain pointer.” What is a static chain pointer?

Some languages, such as Pascal, supported nested functions such that the nested function is permitted to access variables from its parent.

```
function Outer(n: integer) : integer;
    var i: integer;

    procedure Inner(m: integer);
    begin
        i := i + m
    end;

(* Outer body begins here *)
begin
    i := 0;
    Inner(n);
    Outer := i
end;
```

The `Outer` function doesn’t do anything useful, but it does it in an interesting way.

It begins with a local variable declaration for `i`, and then defines a local procedure `Inner` which adds its parameter `m` to the `Outer` variable `i`. The `Outer` function then initializes `i` to zero, calls `Inner(n)` (which adds `n` to `i`), and then returns the modified value of `i`.

This is just the identity function, but it calculates the result with the help of an inner function.

The way this works is that the `Inner` function receives a hidden parameter that tells it where the `Outer` procedure’s local variables are.

In practice, what is passed is a pointer to the `Outer` procedure’s stack frame.

Now, if the containing function or precedure happens itself to be nested, then you can use the parent’s frame to access the local variables of the grandparent.

```
function Outer(n: integer) : integer;
    var i: integer;

    procedure Inner(m: integer);

        procedure MoreInner
        begin
            i := i + m
        end
    begin
        MoreInner
    end;

(* Outer body begins here *)
begin
    i := 0;
    Inner(n);
    Outer := i
end;
```

In this case, the `MoreInner` receives a hidden pointer to `Inner`‘s stack frame, which lets it access the `m` parameter from `Inner`. But `Inner` is itself a nested procedure and therefore received a pointer to `Outer`‘s stack frame. Therefore, `MoreInner` can use that pointer to access `Outer`‘s local variable `i`.

Here’s what it looks like in a diagram:

MoreInner	Inner	Outer
chain →	chain →	`n`
	`m`	`i`

This is called a *static* chain because the structure of the chain is based on lexical scoping, not dynamic scoping. You can see the difference in this example:

```
function Outer(n: integer) : integer;
    var i: integer;

    procedure Update(j: integer);
    begin
        i := i + j
    end;

    procedure Inner(m: integer);

        procedure MoreInner;
        begin
            Update(m)
        end;

    (* Inner body begins here *)
    begin
        MoreInner
    end;

(* Outer body begins here *)
begin
    i := 0;
    Inner(n);
    Outer := i
end;
```

This version is even more useless than the previous one: `MoreInner` doesn't update `i` directly, but instead asks `Update` (an uncle procedure) to do it.

At the point that `MoreInner` calls `Update`, it does not pass its own stack frame as the static chain pointer. Instead, it passes `Outer`'s stack frame, because `Update`'s parent is `Outer`.

The static chain does not match the dynamic call stack: The call stack says that `Update`'s caller is `MoreInner` but the static chain says that `Update`'s parent is `Outer`.

Update		MoreInner		Inner		Outer
frame pointer	→	frame pointer	→	frame pointer	→	frame pointer
chain						`n`
		chain	→	chain		`i`
				`m`		

The authors of the Application Binary Interface document assume you are familiar with how nested functions are implemented and are just noting that the calling convention for nested functions is to pass the static chain in the `%r10` register.