NEC

SX-Aurora TSUBASA

ELF Handling For Thread-Local Storage

VE Architecture Processor Supplement

Proprietary Notice

Remarks

- This document is based on ELF Handling For Thread-Local Storage, Version 0.21, Ulrich Drepper, August 22, 2013.
- All product, brand, or trade names in this publication are the trademarks or registered trademarks of their respective owners.

Contents

Chapter1 Motivation

VE should support the thread local storage for the keyword __thread of C and C++. This document is the supplement to “ELF handling For Thread-Local Storage” about VE specific definitions.

Chapter2　Run-Time Handling of TLS

VE uses “Variant I” for the thread local storage data structures described in “ELF Handling For Thread-Local Storage”.

2.1　Architecture Specific Definitions

2.1.1　VE specific

The %tp (%s14) is used as the thread register containing tpt. Accessing the dynamic thread vector with the thread register’s help is implementation defined. Each element of the dtvt is 8 bytes in size.

The __tls_get_addr function has the same interface as on x86_64.

The prototype is:

```
extern void *__tls_get_addr (tls_index *ti);
```

where the type tls_index is defined as:

```
tyepdef struct
  {
    unsigned long int ti_module;
    unsigned long int ti_offset;
  } tls_index;
```

Chapter3 TLS Access Models

3.1 General Dynamic TLS Model

In the following sections the code shown in determining a address of a thread-local variable x:

```
extern __thread int x;
&x;
```

3.1.1 VE General Dynamic TLS Model

The VE general dynamic access model is very similar to the IA-32 GNU variant. The __tls_get_addr function is called with one parameter which is a pointer to an object of type tls_index.

General Dynamic Model Code Sequence	*Initial Relocation*	*Symbol*
0x00 lea %s0,x@TLS_GD_LO(-24)	R_VE_TLS_GD_LO32	x
0x08 and %s0,%s0,(32)0		
0x10 sic %lr		
0x18 lea.sl %s0,x@TLS_GD_HI(%s0,%lr)	R_VE_TLS_GD_HI32	x
0x20 lea %s12,__tls_get_addr@PLT_LO(8)	R_VE_PLT_LO32	__tls_get_addr
0x28 and %s12,%s12,(32)0		
0x30 lea.sl %s12,__tls_get_addr@PLT_HI(%s12,%lr)	R_VE_PLT_HI32	__tls_get_addr
0x38 bsic %lr,(,%s12)		
	Outstanding Relocations	
GOT[n]	R_VE_DTPMOD64	x
GOT[n+1]	R_VE_DTPOFF64	x

Due to VE's architecture the offset has to be loaded in two steps in the register %s0.
The offset so loaded is that of the first of two consecutive words in the GOT which the linker will add when creating an executable or shared object and which get the relocations R_VE_DTPMOD64 and R_VE_DTPOFF64 assigned. These relocations will instruct the dynamic linker to look up the thread-local symbol x and store the module ID of the module it is found in into the first word and the offset in the TLS block into the second word.

3.2 Local Exec TLS Model

Optimizations for the local dynamic model, similar to those the local dynamic model adds to the generic dynamic model, lead to the local exec model. Its use is even more restricted than that of the local dynamic model. It can only be used for code in the executable itself and to access variables in the executable itself.

Restricting the use to the executable means that just as for the local exec model that the TLS block can be addressed relative to the thread pointer. Restricting the variables to only those defined in the executable means that always the first TLS block, the one for the executable, is used and therefore the size of all the other TLS blocks is irrelevant for the address computation. It also means that the linker knows when creating the final executable what the offset from the TCB is. The formula for the actual offset depends on the architecture but it consists of a sum or difference of the thread pointer, the offset of the first TLS block tlsoffset1 and the offset of the variable in in this TLS block offsetx. The result is known at link-time and is made available in the code as an immediate value.

The code in the architecture descriptions in the next sections implements something along the line of the following where the code must be in the executable itself:

```
static __thread int x;
&x;
```

3.2.1 VE Local Exec TLS Model

The VE code sequence is similar to x86_64 GNU. It is only an addition of the offset which is available as an immediate value to the thread pointer value.

Local Exec Model Code Sequence	*Initial Relocation*	*Symbol*
0x00 lea %s0,x@TPOFF_LO	R_VE_TPOFF_LO32	x
0x08 and %s0,%s0,(32)0		
0x10 lea.sl %s0,x@TPOFF_HI(%tp,%s0)	R_VE_TPOFF_HI32	x
	Outstanding Relocations	

Chapter4　Linker Optimizations

This chapter describes optimization, automatic vectorization, inlining and automatic parallelization which are useful in making user programs execute quickly.

4.1　VE Linker Optimizations

4.1.1　General Dynamic To Local Exec

This transition is similar to the General Dynamic to Initial Exec optimization, just the offset can be stored directly into the instruction and no runtime relocation is required.

GE -> LE Code Transition	*Initial Relocation*	*Symbol*
0x00 lea %s0,x@TLS_GD_LO(-24)	R_VE_TPOFF_LO32	x
0x08 and %s0,%s0,(32)0		
0x10 sic %lr		
0x18 lea.sl %s0,x@TLS_GD_HI(%s0,%lr)	R_VE_TLS_GD_HI32	x
0x20 lea %s12,__get_tls_addr@PLT_LO(8)	R_VE_PLT_LO32	__get_tls_addr
0x28 and %s12,%s12,(32)0		
0x30 lea.sl %s12,__tls_get_addr@PLT_HI(%s12,%lr)	R_VE_PLT_HI32	__get_tls_addr
0x38 bsic %lr,(,%s12)		
0x00 lea　%s0,x@TPOFF_LO	R_VE_TPOFF_LO	x
0x08 and　%s0,%s0,(32)0		
0x10 lea.sl %s0,x@TPOFF_HI(%tp,%s0)	R_VE_TPOFF_HI	x
0x18 nop		
0x20 nop		
0x28 nop		
0x30 nop		
0x38 nop		

The resulting sequence is identical to the code for the Local Exec model when the address is computed. Generating the code in the compiler is nevertheless more efficient if the addressed variable can be used directly instead of first computing the address.

Chapter5　New ELF Definitions

This section shows the actual definitions for the newly introduced constants necessary to describe the extended ELF format. The generic extensions are:

```
#define SHF_TLS   (1 << 10)
#define STT_TLS   6
#define PT_TLS    7
```

5.1　New VE ELF Definitions

```
#define R_VE_DTPMOD64    22
#define R_VE_DTPOFF64    23
/* #define R_VE_TPOFF64      24 */
#define R_VE_TLS_GD_HI32 25
#define R_VE_TLS_GD_LO32 26
/* #define R_VE_TLS_LD_HI32  27 */
/* #define R_VE_TLS_LD_LO32  28 */
/* #define R_VE_DTPOFF32     29 */
/* #define R_VE_TLS_IE_HI32  30 */
/* #define R_VE_TLS_IE_LO32  31 */
#define R_VE_TPOFF_HI32  32
#define R_VE_TPOFF_LO32  33
/* #define R_VE_TPOFF32      34 */
```

The operators used in the code sequences are defined as follows:

- @TLS_GD_HI/@TLS_GD_LO
  Allocate two contiguous entries in the GOT to hold a tls_index structure (for passing to _tls_get_addr). It may be used only in the exact VE general dynamic code sequence shown above.
- @TPOFF_HI32/@TPOFF_LO32
  Calculate the offset of the variable relative to TLS block end, %tp. No GOT entry is created.

Appendix A History

A.1 History table

Feb. 2018	Rev. 1	Create a new entry.
Mar. 2018	Rev. 1.1	Fix the style of document.
Dec. 2018	Rev. 1.2	The design of document is changed.

A.2 Change notes

The following changes are done in this edition.

- The design of document is changed.