Implementing Thread Local Storage for HP PA-RISC Linux

Randolph Chung, Carlos O'Donell, John David Anglin

April 14, 2006

1 Introduction

This document describes the specific thread local storage (TLS) implementations for HP PA-RISC Linux. It is meant to be read after reading:

http://people.redhat.com/drepper/tls.pdf

Hereafter the HP PA-RISC architecture will be referred to as *hppa*.

2 hppa Specific

The thread-local storage structure follows variant I. The size of the TCB is 8 bytes on hppa32 (16 bytes in hppa64). The first 4 (8) bytes contain the pointer to the dynamic thread vector, the remianing 4 (8) bytes are reserved for the implementation.

The TLS blocks for all modules present at startup time are created consecutively following the TCB. The $tlsoffset\_x$ values are computed as follows:

$$\begin{aligned} tlsoffset_1 &= \texttt{round}(tlssize_1, align_1) \\ tlsoffset_{m+1} &= \texttt{round}(tlsoffset_m + tlssize_{m+1}, align_{m+1}) \end{aligned}$$

for all m in $1 \leq m \leq M$, where M is the total number of modules.

The `__tls_get_addr` function is defined as:

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

`tls_index` is an internal data structure with the following definition:

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

The thread pointer is held in a control register (`cr27`). The control register must be transfered to a general register before it can be used. The thread pointer cannot be written to in user mode; the kernel provides an interface to set the thread pointer (`set_thread_pointer`).

Note: the Linux TLS ABI differs from the HP-UX TLS ABI. In HP-UX, a single TLS access model is defined, which is similar to the "Initial Exec TLS model" described below.

3 hppa General Dynamic TLS model

The hppa general dynamic access model is similar to SPARC. 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
`addil`	`LT'x-$tls_gdidx$, gp`	`R_PARISC_TLS_GD21L`	x
`ldo`	`RT'x-$tls_gdidx$(%r1), %arg0`	`R_PARISC_TLS_GD14R`	x
`b`	`__tls_get_addr`	`R_PARISC_TLS_GDCALL`	
`nop`			
		Outstanding Relocations	
`GOT[n]`		`R_PARISC_TLS_DTPMOD32`	x
`GOT[n+1]`		`R_PARISC_TLS_DTPOFF32`	x

The expressions `LT'x-$tls_gdidx$` and `RT'x-$tls_gdidx$` causes the linker to create a `tls_index` object in the GOT. The GOT offset of the first entry is loaded into `%r26` using the `addil/ldo` instructions, and passed to `__tls_get_addr`. The `tls_index` object occupies two entries in the GOT, the first entry (marked with `R_PARISC_TLS_DTPMOD32`) will be filled in at runtime with the module id; the second entry (`R_PARISC_TLS_DTPOFF32`) is filled in with the offset. On hppa64, the relocations become `R_PARISC_TLS_DTPMOD64` and `R_PARISC_TLS_DTPOFF64`.

The nop in the delay slot of the branch to `__tls_get_addr` is needed to reserve space in case an optimization needs to convert the above sequence into one of the other sequences.

4 hppa Local Dynamic TLS Model

The local dynamic TLS model is useful in case multiple variables are used. The code sequence is similar to that for the general dynamic model:

Local Dynamic Model Code Sequence		**Initial Relocation**	**Symbol**
`addil`	`LT'x1-$tls_ldidx$, gp`	`R_PARISC_TLS_LDM21L`	`x1`
`b`	`__tls_get_addr`	`R_PARISC_TLS_LDMCALL`	
`ldo`	`RT'x1-$tls_ldidx$(%r1), %arg0`	`R_PARISC_TLS_LDM14R`	`x1`
`addil`	`LR'x1-$tls_dtpoff$, %ret0`	`R_PARISC_TLS_LDO21L`	`x1`
`ldo`	`RR'x1-$tls_dtpoff$(%r1), %tmp1`	`R_PARISC_TLS_LDO14R`	`x1`
`addil`	`LR'x2-$tls_dtpoff$, %ret0`	`R_PARISC_TLS_LDO21L`	`x2`
`ldo`	`RR'x2-$tls_dtpoff$(%r1), %tmp2`	`R_PARISC_TLS_LDO14R`	`x2`
		Outstadning Relocations	
`GOT[n]`		`R_PARISC_TLS_DTPMOD32`	`x1`
`GOT[n+1]`			

The first three instructions are similar to the general dynamic model. A single entry in the GOT is created to hold the module id (filled in at runtime by the dynamic linker). The offset passed in the initial call to `__tls_get_addr` will be 0.

The remaining instructions load the offset of the variables being accessed and add them to the address returned by `__tls_get_addr`. `x-$tls_dtpoff$` is replaced by the offset to the symbol x; and the addil/ldo sequence adds the offset to the result of `__tls_get_addr`. The sequences are marked with the relocations `R_PARISC_TLS_LDO21L` and `R_PARISC_TLS_LDO14R` so that the linker can recognize it.

Compared to the general dynamic model, the local dynamic model saves one GOT entry, two instructions and one function call for each additional variable referenced. Avoiding the branch penality may make this optimization worthwhile if multiple TLS variables are referenced.

5 hppa Initial Exec TLS Model

The sequence to support the initial exec model on hppa is fairly straightforward:

Initial Exec Model Code Sequence	Initial Relocation	Symbol
`mfctl cr27, %t1`		
`addil LT'x-$tls_ieoff$, %gp`	R_PARISC_TLS_IE21L	x
`ldw RT'x-$tls_ieoff$(%r1), %t2`	R_PARISC_TLS_IE14R	x
`add %t1, %t2, %t3`		
	Outstanding Relocations	
GOT[n]	R_PARISC_TLS_TPREL32	x

Note: The `R_PARISC_TLS_IE21L` should be the same as `R_PARISC_LTOFF_TP21L`, `R_PARISC_TLS_IE14R` should be the same as `R_PARISC_LTOFF_TP14{D}R` and `R_PARISC_TLS_TPREL32` should be the same as `R_PARISC_TPREL32`.

The thread pointer needs to be loaded into a general register before it can be used for address manipulations. mfctl has a significant latency, so the compiler should optimize calls to load the thread pointer.

The `LT'x-$tls_ieoff$` expression causes the creation of a GOT entry marked with a `R_PARISC_TLS_TPREL32` relocation (`R_PARISC_TLS_TPREL64` on hppa64). At runtime, the entry is filled in with the offset of the TLS variable relative to its TCB block. The addil/ldw sequence loads this value and adds it to the thread pointer to produce the desired address.

Note that this is the sole TLS access method defined by the PA-RISC 64-bit runtime document. The relocations used here are renamed from the ones specified in that document to be consistent with glibc TLS implementations on other architectures.

6 hppa Local Exec TLS Model

The simplest case is the local exec model. The code sequence is as follows:

Local Exec Model Code Sequence		Initial Relocation	Symbol
`mfctl`	`cr27, %t1`		
`addil`	`LR'x-$tls_leoff$, %t1`	`R_PARISC_TLS_LE21L`	`x`
`ldo`	`RR'x-$tls_leoff$(%r1), %t2`	`R_PARISC_TLS_LE14R`	`x`
		Outstanding Relocations	
		None	

Note: The relocation `R_PARISC_TLS_LE21L` should be the same as `R_PARISC_TPREL21L` and `R_PARISC_TLS_LE14R` should be the same as `R_PARISC_TPREL14R`.

The `x-$tls_leoff$` expression is translated into immediate values by the linker which represent the offset of the variable x from the thread pointer. The addil/ldo instructions adds the resulting value to the thread pointer to get the effective address of the TLS variable. As in the case for the initial exec TLS model, loading the thread pointer from cr27 should be optimized for accesses to multiple variables in the same function.

`R_PARISC_TPREL21L`/`R_PARISC_TPREL14R` are the relocation types defined in the PA-RISC ELF supplement that describes the manipulations required here, but the names `R_PARISC_TLS_LE21L`/`R_PARISC_TLS_LE14R` were chosen to be consistent with the other glibc implementations.

7 hppa Linker Optimizations

General Dynamic to Initial exec:

GD → IE Code Transition		**Initial Relocation**	**Symbol**
addil	LT'x-tls_gdidx, gp	R_PARISC_TLS_GD21L	x
ldo	RT'x-tls_gdidx(%r1), %arg0	R_PARISC_TLS_GD14R	x
b	__tls_get_addr	R_PARISC_TLS_GDCALL	
nop			
	⇓	⇓	⇓
mfctl	cr27, %t1		
addil	LT'x-tls_ieoff, %gp	R_PARISC_TLS_IE21L	x
ldw	RT'x-tls_ieoff(%r1), %t2	R_PARISC_TLS_IE14R	x
add	%t1, %t2, %t3		
		Outstanding Relocations	
GOT[n]		R_PARISC_TLS_TPREL32	x

Note: The relocation R_PARISC_TLS_IE21L should be the same as R_PARISC_LTOFF_TP21L, R_PARISC_TLS_IE14R should be the same as R_PARISC_LTOFF_TP14{D}R and R_PARISC_TLS_TPREL32 should be the same as R_PARISC_TPREL32.

General Dyanmic to Local Exec:

GD → LE Code Transition		**Initial Relocation**	**Symbol**
addil	LT'x-tls_gdidx, gp	R_PARISC_TLS_GD21L	x
ldo	RT'x-tls_gdidx(%r1), %arg0	R_PARISC_TLS_GD14R	x
b	__tls_get_addr	R_PARISC_TLS_GDCALL	
nop			
	⇓	⇓	⇓
mfctl	cr27, %t1		
addil	LR'x-tls_leoff, %t1	R_PARISC_TLS_LE21L	x
ldo	RR'x-tls_leoff(%r1), %t2	R_PARISC_TLS_LE14R	x
nop			
		Outstanding Relocations	
		None	

Note: The relocation R_PARISC_TLS_LE21L should be the same as R_PARISC_TPREL21L, and R_PARISC_TLS_LE14R should be the same as R_PARISC_TPREL14R.

Local dynamic to local exec:

LD → LE Code Transition		Initial Relocation	Symbol
`addil`	`LT'x1-$tls_ldidx$, gp`	R_PARISC_TLS_LDM21L	x1
`b`	`__tls_get_addr`	R_PARISC_TLS_LDMCALL	
`ldo`	`RT'x1-$tls_ldidx$(%r1), %arg0`	R_PARISC_TLS_LDM14R	x1
`addil`	`LR'x1-$tls_dtpoff$, %ret0`	R_PARISC_TLS_LDO21L	x1
`ldo`	`RR'x1-$tls_dtpoff$(%r1), %tmp1`	R_PARISC_TLS_LDO14R	x1
	⇓	⇓	⇓
`nop`			
`nop`			
`mfctl`	`cr27, %t1`		
`addil`	`LR'x-$tls_leoff$, %t1`	R_PARISC_TLS_LE21L	x
`ldo`	`RR'x-$tls_leoff$(%r1), %t2`	R_PARISC_TLS_LE14R	x
		Outstanding Relocations	
		None	

Note: The relocation R_PARISC_TLS_LE21L should be the same as R_PARISC_TPREL21L, and R_PARISC_TLS_LE14R should be the same as R_PARISC_TPREL14R.

Initial exec to local exec:

IE → LE Code Transition		Initial Relocation	Symbol
`mfctl`	`cr27, %t1`		
`addil`	`LT'x-$tls_ieoff$, %gp`	R_PARISC_TLS_IE21L	x
`ldw`	`RT'x-$tls_ieoff$(%r1), %t2`	R_PARISC_TLS_IE14R	x
`add`	`%t1, %t2, %t3`		
	⇓	⇓	⇓
`mfctl`	`cr27, %t1`		
`addil`	`LR'x-$tls_leoff$, %t1`	R_PARISC_TLS_LE21L	x
`ldo`	`RR'x-$tls_leoff$(%r1), %t2`	R_PARISC_TLS_LE14R	x
`nop`			
		Outstanding Relocations	
		None	

Note: The relocation R_PARISC_TLS_IE21L should be same as R_PARISC_LTOFF_TP21L, R_PARISC_TLS_IE14R should be the same as R_PARISC_LTOFF_TP14{D}R, R_PARISC_TLS_LE21L should be the same as R_PARISC_TPREL21 and R_PARISC_TLS_LE14R should be the same as R_PARISC_TPREL14R.

8 New hppa ELF definitions

The following are the required additional ELF definitions to implement TLS on hppa.

```
#define R_PARISC_TLS_GD21L
#define R_PARISC_TLS_GD14R
#define R_PARISC_TLS_GDCALL
#define R_PARISC_TLS_LDM21L
#define R_PARISC_TLS_LDM14R
#define R_PARISC_TLS_LDMCALL
#define R_PARISC_TLS_LDO21L
#define R_PARISC_TLS_LDO14R
#define R_PARISC_TLS_IE21L
#define R_PARISC_TLS_IE14R
#define R_PARISC_TLS_LE21L
#define R_PARISC_TLS_LE14R
#define R_PARISC_TLS_DTPMOD32
#define R_PARISC_TLS_DTPMOD64
#define R_PARISC_TLS_DTPOFF32
#define R_PARISC_TLS_DTPOFF64
#define R_PARISC_TLS_TPREL32
#define R_PARISC_TLS_TPREL64
```

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

tls_gdidx Allocate two contiguous entries in the GOT to hold a TLS index structure for passing to __tls_get_addr. At runtime, the ti_module field (R_PARISC_TLS_DTPMOD32) and ti_offset (R_PARISC_TLS_DTPOFF32 fields are filled in to point to the correct module/offset.

tls_ldidx Allocate two contiguous entries in the GOT to hold a tls index structure for passing to __tls_get_addr. The ti_offset field is set to 0. The ti_module field is filled in at runtime. The call to __tls_get_addr will return the starting offset of the dynamic TLS block.

tls_dtpoff Calculate the offset of the variable relative to the TLS block it is contained in.

tls_ieoff Calculate the offset of the variable relative to the TLS block

tls_leoff Calculate the offset of the variable relative to the static TLS block