Using FS and GS segments in user space applications

The x86 architecture supports segmentation. Instructions which access memory can use segment register based addressing mode. The following notation is used to address a byte within a segment:

Segment-register:Byte-address

The segment base address is added to the Byte-address to compute the resulting virtual address which is accessed. This allows to access multiple instances of data with the identical Byte-address, i.e. the same code. The selection of a particular instance is purely based on the base-address in the segment register.

In 32-bit mode the CPU provides 6 segments, which also support segment limits. The limits can be used to enforce address space protections.

In 64-bit mode the CS/SS/DS/ES segments are ignored and the base address is always 0 to provide a full 64bit address space. The FS and GS segments are still functional in 64-bit mode.

Common FS and GS usage

The FS segment is commonly used to address Thread Local Storage (TLS). FS is usually managed by runtime code or a threading library. Variables declared with the '__thread' storage class specifier are instantiated per thread and the compiler emits the FS: address prefix for accesses to these variables. Each thread has its own FS base address so common code can be used without complex address offset calculations to access the per thread instances. Applications should not use FS for other purposes when they use runtimes or threading libraries which manage the per thread FS.

The GS segment has no common use and can be used freely by applications. GCC and Clang support GS based addressing via address space identifiers.

Reading and writing the FS/GS base address

There exist two mechanisms to read and write the FS/GS base address:

- the arch_prctl() system call
- the FSGSBASE instruction family

Accessing FS/GS base with arch_prctl()

The arch_prctl(2) based mechanism is available on all 64-bit CPUs and all kernel versions.

Reading the base:

arch_prctl(ARCH_GET_FS, &fsbase); arch_prctl(ARCH_GET_GS, &gsbase);

Writing the base:

arch_prctl(ARCH_SET_FS, fsbase); arch_prctl(ARCH_SET_GS, gsbase);

The ARCH_SET_GS prctl may be disabled depending on kernel configuration and security settings.

Accessing FS/GS base with the FSGSBASE instructions

With the Ivy Bridge CPU generation Intel introduced a new set of instructions to access the FS and GS base registers directly from user space. These instructions are also supported on AMD Family 17H CPUs. The following instructions are available:

RDFSBASE %reg	Read the FS base register
RDGSBASE %reg	Read the GS base register
WRFSBASE %reg	Write the FS base register
WRGSBASE %reg	Write the GS base register

The instructions avoid the overhead of the arch_prctl() syscall and allow more flexible usage of the FS/GS addressing modes in user space applications. This does not prevent conflicts between threading libraries and runtimes which utilize FS and applications which want to use it for their own purpose.

FSGSBASE instructions enablement

The instructions are enumerated in CPUID leaf 7, bit 0 of EBX. If available /proc/cpuinfo shows 'fsgsbase' in the flag entry of the CPUs.

The availability of the instructions does not enable them automatically. The kernel has to enable them explicitly in CR4. The reason for this is that older kernels make assumptions about the values in the GS register and enforce them when GS base is set via arch_prctl(). Allowing user space to write arbitrary values to GS base would violate these assumptions and cause malfunction.

On kernels which do not enable FSGSBASE the execution of the FSGSBASE instructions will fault with a #UD exception.

The kernel provides reliable information about the enabled state in the ELF AUX vector. If the HWCAP2_FSGSBASE bit is set in the AUX vector, the kernel has FSGSBASE instructions enabled and applications can use them. The following code example shows how this detection works:

```
#include <sys/auxv.h>
#include <elf.h>

/* Will be eventually in asm/hwcap.h */
#ifndef HWCAP2_FSGSBASE
#define HWCAP2_FSGSBASE        (1 << 1)
#endif

....

unsigned val = getauxval(AT_HWCAP2);

if (val & HWCAP2_FSGSBASE)
     printf("FSGSBASE enabled\n");
```

FSGSBASE instructions compiler support

GCC version 4.6.4 and newer provide instrinsics for the FSGSBASE instructions. Clang 5 supports them as well.

_readfsbase_u64()	Read the FS base register
_readfsbase_u64()	Read the GS base register
_writefsbase_u64()	Write the FS base register
_writegsbase_u64()	Write the GS base register

To utilize these instrinsics <immintrin.h> must be included in the source code and the compiler option -mfsgsbase has to be added.

Compiler support for FS/GS based addressing

GCC version 6 and newer provide support for FS/GS based addressing via Named Address Spaces. GCC implements the following address space identifiers for x86:

__seg_fs	Variable is addressed relative to FS
__seg_gs	Variable is addressed relative to GS

The preprocessor symbols __SEG_FS and __SEG_GS are defined when these address spaces are supported. Code which implements fallback modes should check whether these symbols are defined. Usage example:

```
#ifdef __SEG_GS

long data0 = 0;
long data1 = 1;

long __seg_gs *ptr;

/* Check whether FSGSBASE is enabled by the kernel (HWCAP2_FSGSBASE) */
....

/* Set GS base to point to data0 */
_writegsbase_u64(&data0);

/* Access offset 0 of GS */
ptr = 0;
printf("data0 = %ld\n", *ptr);

/* Set GS base to point to data1 */
_writegsbase_u64(&data1);
/* ptr still addresses offset 0! */
printf("data1 = %ld\n", *ptr);
```

Clang does not provide the GCC address space identifiers, but it provides address spaces via an attribute based mechanism in Clang 2.6 and newer versions:

__attribute__((address_space(256))	Variable is addressed relative to GS
__attribute__((address_space(257))	Variable is addressed relative to FS

FS/GS based addressing with inline assembly

In case the compiler does not support address spaces, inline assembly can be used for FS/GS based addressing mode:

```
mov %fs:offset, %reg
mov %gs:offset, %reg

mov %reg, %fs:offset
mov %reg, %gs:offset
```