Red Hat Enterprise Linux Release 9.2 Manual Pages on 'vdso.7' command

$ man vdso.7

VDSO(7)   Linux Programmer's Manual   VDSO(7)

NAME

    vdso - overview of the virtual ELF dynamic shared object

SYNOPSIS

    #include <sys/auxv.h>

    void *vdso = (uintptr_t) getauxval(AT_SYSINFO_EHDR);

DESCRIPTION

    The "vDSO" (virtual dynamic shared object) is a small shared library

    that the kernel automatically maps into the address space of all user-

    space applications. Applications usually do not need to concern them?

    selves with these details as the vDSO is most commonly called by the C

    library. This way you can code in the normal way using standard func?

    tions and the C library will take care of using any functionality that

    is available via the vDSO.

    Why does the vDSO exist at all? There are some system calls the kernel

    provides that user-space code ends up using frequently, to the point

    that such calls can dominate overall performance. This is due both to

    the frequency of the call as well as the context-switch overhead that

    results from exiting user space and entering the kernel.

    The rest of this documentation is geared toward the curious and/or C

    library writers rather than general developers. If you're trying to

    call the vDSO in your own application rather than using the C library,

    you're most likely doing it wrong.

Example background

Making system calls can be slow. In x86 32-bit systems, you can trig‐
ger a software interrupt (int $0x80) to tell the kernel you wish to
make a system call. However, this instruction is expensive: it goes
through the full interrupt-handling paths in the processor's microcode
as well as in the kernel. Newer processors have faster (but backward
incompatible) instructions to initiate system calls. Rather than re‐
quire the C library to figure out if this functionality is available at
run time, the C library can use functions provided by the kernel in the
vDSO.

Note that the terminology can be confusing. On x86 systems, the vDSO
function used to determine the preferred method of making a system call
is named "__kernel_vsyscall", but on x86-64, the term "vsyscall" also
refers to an obsolete way to ask the kernel what time it is or what CPU
the caller is on.

One frequently used system call is gettimeofday(2). This system call
is called both directly by user-space applications as well as indi‐
rectly by the C library. Think timestamps or timing loops or polling‐
all of these frequently need to know what time it is right now. This
information is also not secret‐any application in any privilege mode
(root or any unprivileged user) will get the same answer. Thus the
kernel arranges for the information required to answer this question to
be placed in memory the process can access. Now a call to gettimeof‐
day(2) changes from a system call to a normal function call and a few
memory accesses.

Finding the vDSO

The base address of the vDSO (if one exists) is passed by the kernel to
each program in the initial auxiliary vector (see getauxval(3)), via
the AT_SYSINFO_EHDR tag.

You must not assume the vDSO is mapped at any particular location in
the user's memory map. The base address will usually be randomized at
run time every time a new process image is created (at execve(2) time).
This is done for security reasons, to prevent "return-to-libc" attacks.

For some architectures, there is also an AT_SYSINFO tag. This is used only for locating the vsyscall entry point and is frequently omitted or set to 0 (meaning it's not available). This tag is a throwback to the initial vDSO work (see History below) and its use should be avoided.

File format

Since the vDSO is a fully formed ELF image, you can do symbol lookups on it. This allows new symbols to be added with newer kernel releases, and allows the C library to detect available functionality at run time when running under different kernel versions. Oftentimes the C library will do detection with the first call and then cache the result for subsequent calls.

All symbols are also versioned (using the GNU version format).   This allows the kernel to update the function signature without breaking backward compatibility. This means changing the arguments that the function accepts as well as the return value. Thus, when looking up a symbol in the vDSO, you must always include the version to match the ABI you expect.

Typically the vDSO follows the naming convention of prefixing all sym? bols with "__vdso_" or "__kernel_" so as to distinguish them from other standard symbols.   For example, the "gettimeofday" function is named "__vdso_gettimeofday".

You use the standard C calling conventions when calling any of these functions. No need to worry about weird register or stack behavior.

NOTES

Source

When you compile the kernel, it will automatically compile and link the vDSO code for you. You will frequently find it under the architecture-specific directory:

    find arch/$ARCH/ -name '*vdso*.so*' -o -name '*gate*.so*'

vDSO names

The name of the vDSO varies across architectures. It will often show up in things like glibc's ldd(1) output. The exact name should not matter to any code, so do not hardcode it.

```
       user ABI   vDSO name
       ???????????????????????????????
       aarch64    linux-vdso.so.1
       arm        linux-vdso.so.1
       ia64       linux-gate.so.1
       mips       linux-vdso.so.1
       ppc/32     linux-vdso32.so.1
       ppc/64     linux-vdso64.so.1
       riscv      linux-vdso.so.1
       s390       linux-vdso32.so.1
       s390x      linux-vdso64.so.1
       sh         linux-gate.so.1
       i386       linux-gate.so.1
       x86-64     linux-vdso.so.1
       x86/x32    linux-vdso.so.1
```

strace(1), seccomp(2), and the vDSO

When tracing systems calls with strace(1), symbols (system calls) that
are exported by the vDSO will not appear in the trace output.   Those
system calls will likewise not be visible to seccomp(2) filters.

ARCHITECTURE-SPECIFIC NOTES

The subsections below provide architecture-specific notes on the vDSO.
Note that the vDSO that is used is based on the ABI of your user-space
code and not the ABI of the kernel. Thus, for example, when you run an
i386 32-bit ELF binary, you'll get the same vDSO regardless of whether
you run it under an i386 32-bit kernel or under an x86-64 64-bit ker‐
nel.   Therefore, the name of the user-space ABI should be used to de‐
termine which of the sections below is relevant.

ARM functions

The table below lists the symbols exported by the vDSO.

```
symbol                version
?????????????????????????????????????????????????????????????????????
__vdso_gettimeofday    LINUX_2.6 (exported since Linux 4.1)
__vdso_clock_gettime   LINUX_2.6 (exported since Linux 4.1)
```

Additionally, the ARM port has a code page full of utility functions.
Since it's just a raw page of code, there is no ELF information for do?
ing symbol lookups or versioning. It does provide support for differ?
ent versions though.

For information on this code page, it's best to refer to the kernel
documentation as it's extremely detailed and covers everything you need
to know: Documentation/arm/kernel_user_helpers.txt.

aarch64 functions

The table below lists the symbols exported by the vDSO.

symbol   version

???????????????????????????????????????

__kernel_rt_sigreturn   LINUX_2.6.39

__kernel_gettimeofday   LINUX_2.6.39

__kernel_clock_gettime   LINUX_2.6.39

__kernel_clock_getres   LINUX_2.6.39

bfin (Blackfin) functions (port removed in Linux 4.17)

As this CPU lacks a memory management unit (MMU), it doesn't set up a
vDSO in the normal sense. Instead, it maps at boot time a few raw
functions into a fixed location in memory. User-space applications
then call directly into that region. There is no provision for back?
ward compatibility beyond sniffing raw opcodes, but as this is an em?
bedded CPU, it can get away with things?some of the object formats it
runs aren't even ELF based (they're bFLT/FLAT).

For information on this code page, it's best to refer to the public
documentation:

http://docs.blackfin.uclinux.org/doku.php?id=linux-kernel:fixed-code

mips functions

The table below lists the symbols exported by the vDSO.

symbol   version

??

__kernel_gettimeofday   LINUX_2.6 (exported since Linux 4.4)

__kernel_clock_gettime   LINUX_2.6 (exported since Linux 4.4)

ia64 (Itanium) functions

The table below lists the symbols exported by the vDSO.

```
symbol                      version
?????????????????????????????????????????
__kernel_sigtramp           LINUX_2.5
__kernel_syscall_via_break  LINUX_2.5
__kernel_syscall_via_epc    LINUX_2.5
```

The Itanium port is somewhat tricky. In addition to the vDSO above, it also has "light-weight system calls" (also known as "fast syscalls" or "fsys"). You can invoke these via the __kernel_syscall_via_epc vDSO helper. The system calls listed here have the same semantics as if you called them directly via syscall(2), so refer to the relevant documen? tation for each.   The table below lists the functions available via this mechanism.

```
function
????????????????????
clock_gettime
getcpu
getpid
getppid
gettimeofday
set_tid_address
```

parisc (hppa) functions

The parisc port has a code page with utility functions called a gateway page.   Rather than use the normal ELF auxiliary vector approach, it passes the address of the page to the process via the SR2 register. The permissions on the page are such that merely executing those ad? dresses automatically executes with kernel privileges and not in user space. This is done to match the way HP-UX works.

Since it's just a raw page of code, there is no ELF information for do? ing symbol lookups or versioning. Simply call into the appropriate offset via the branch instruction, for example:

```
    ble <offset>(%sr2, %r0)
```

offset   function

```
     ????????????????????????????????????????????
     00b0    lws_entry (CAS operations)
     00e0    set_thread_pointer (used by glibc)
     0100    linux_gateway_entry (syscall)
```

ppc/32 functions

The table below lists the symbols exported by the vDSO. The functions

marked with a * are available only when the kernel is a PowerPC64

(64-bit) kernel.

```
symbol                  version

?????????????????????????????????????????
     __kernel_clock_getres     LINUX_2.6.15

     __kernel_clock_gettime    LINUX_2.6.15

     __kernel_datapage_offset  LINUX_2.6.15

     __kernel_get_syscall_map  LINUX_2.6.15

     __kernel_get_tbfreq       LINUX_2.6.15

     __kernel_getcpu *         LINUX_2.6.15

     __kernel_gettimeofday     LINUX_2.6.15

     __kernel_sigtramp_rt32    LINUX_2.6.15

     __kernel_sigtramp32       LINUX_2.6.15

     __kernel_sync_dicache     LINUX_2.6.15

     __kernel_sync_dicache_p5  LINUX_2.6.15
```

The CLOCK_REALTIME_COARSE and CLOCK_MONOTONIC_COARSE clocks are not

supported by the __kernel_clock_getres and __kernel_clock_gettime in?

terfaces; the kernel falls back to the real system call.

ppc/64 functions

The table below lists the symbols exported by the vDSO.

```
symbol                  version

??????????????????????????????????????????
     __kernel_clock_getres     LINUX_2.6.15

     __kernel_clock_gettime    LINUX_2.6.15

     __kernel_datapage_offset  LINUX_2.6.15

     __kernel_get_syscall_map  LINUX_2.6.15

     __kernel_get_tbfreq       LINUX_2.6.15
```

```
__kernel_getcpu          LINUX_2.6.15

__kernel_gettimeofday    LINUX_2.6.15

__kernel_sigtramp_rt64   LINUX_2.6.15

__kernel_sync_dicache    LINUX_2.6.15

__kernel_sync_dicache_p5 LINUX_2.6.15
```

The CLOCK_REALTIME_COARSE and CLOCK_MONOTONIC_COARSE clocks are not

supported by the __kernel_clock_getres and __kernel_clock_gettime in?

terfaces; the kernel falls back to the real system call.

riscv functions

The table below lists the symbols exported by the vDSO.

```
symbol              version

?????????????????????????????????????

__kernel_rt_sigreturn   LINUX_4.15

__kernel_gettimeofday   LINUX_4.15

__kernel_clock_gettime  LINUX_4.15

__kernel_clock_getres   LINUX_4.15

__kernel_getcpu         LINUX_4.15

__kernel_flush_icache   LINUX_4.15
```

s390 functions

The table below lists the symbols exported by the vDSO.

```
symbol              version

?????????????????????????????????????????

__kernel_clock_getres   LINUX_2.6.29

__kernel_clock_gettime  LINUX_2.6.29

__kernel_gettimeofday   LINUX_2.6.29
```

s390x functions

The table below lists the symbols exported by the vDSO.

```
symbol              version

??????????????????????????????????????????

__kernel_clock_getres   LINUX_2.6.29

__kernel_clock_gettime  LINUX_2.6.29

__kernel_gettimeofday   LINUX_2.6.29
```

sh (SuperH) functions

The table below lists the symbols exported by the vDSO.

```
symbol              version
????????????????????????????????????
__kernel_rt_sigreturn   LINUX_2.6
__kernel_sigreturn      LINUX_2.6
__kernel_vsyscall       LINUX_2.6
```

i386 functions

The table below lists the symbols exported by the vDSO.

```
symbol              version
??????????????????????????????????????????????????????????????????
__kernel_sigreturn      LINUX_2.5
__kernel_rt_sigreturn   LINUX_2.5
__kernel_vsyscall       LINUX_2.5
__vdso_clock_gettime    LINUX_2.6 (exported since Linux 3.15)
__vdso_gettimeofday     LINUX_2.6 (exported since Linux 3.15)
__vdso_time             LINUX_2.6 (exported since Linux 3.15)
```

x86-64 functions

The table below lists the symbols exported by the vDSO. All of these symbols are also available without the "__vdso_" prefix, but you should ignore those and stick to the names below.

```
symbol              version
????????????????????????????????????
__vdso_clock_gettime    LINUX_2.6
__vdso_getcpu           LINUX_2.6
__vdso_gettimeofday     LINUX_2.6
__vdso_time             LINUX_2.6
```

x86/x32 functions

The table below lists the symbols exported by the vDSO.

```
symbol              version
??????????????????????????????????????
__vdso_clock_gettime    LINUX_2.6
__vdso_getcpu           LINUX_2.6
__vdso_gettimeofday     LINUX_2.6
```

__vdso_time   LINUX_2.6

   History

      The vDSO was originally just a single function?the vsyscall. In older

      kernels, you might see that name in a process's memory map rather than

      "vdso". Over time, people realized that this mechanism was a great way

      to pass more functionality to user space, so it was reconceived as a

      vDSO in the current format.

SEE ALSO

      syscalls(2), getauxval(3), proc(5)

      The documents, examples, and source code in the Linux source code tree:

         Documentation/ABI/stable/vdso

         Documentation/ia64/fsys.txt

         Documentation/vDSO/* (includes examples of using the vDSO)

         find arch/ -iname '*vdso*' -o -iname '*gate*'

COLOPHON

      This page is part of release 5.10 of the Linux man-pages project. A

      description of the project, information about reporting bugs, and the

      latest   version   of   this   page,   can   be   found   at

      https://www.kernel.org/doc/man-pages/.

Linux   2019-08-02   VDSO(7)