NAME

dl_iterate_phdr – walk through list of shared objects

SYNOPSIS

```
#define _GNU_SOURCE       /* See feature_test_macros(7) */
#include <link.h>

int dl_iterate_phdr(
      int (*callback) (struct dl_phdr_info *info,
                       size_t size, void *data),
      void *data);
```

DESCRIPTION

The **dl_iterate_phdr**() function allows an application to inquire at run time to find out which shared objects it has loaded, and the order in which they were loaded.

The **dl_iterate_phdr**() function walks through the list of an application's shared objects and calls the function *callback* once for each object, until either all shared objects have been processed or *callback* returns a nonzero value.

Each call to *callback* receives three arguments: *info*, which is a pointer to a structure containing information about the shared object; *size*, which is the size of the structure pointed to by *info*; and *data*, which is a copy of whatever value was passed by the calling program as the second argument (also named *data*) in the call to **dl_iterate_phdr**().

The *info* argument is a structure of the following type:

```
struct dl_phdr_info {
    ElfW(Addr)        dlpi_addr;  /* Base address of object */
    const char       *dlpi_name;  /* (Null-terminated) name of
                                     object */
    const ElfW(Phdr) *dlpi_phdr;  /* Pointer to array of
                                     ELF program headers
                                     for this object */
    ElfW(Half)        dlpi_phnum; /* # of items in dlpi_phdr */

    /* The following fields were added in glibc 2.4, after the first
       version of this structure was available.  Check the size
       argument passed to the dl_iterate_phdr callback to determine
       whether or not each later member is available.  */

    unsigned long long int dlpi_adds;
                    /* Incremented when a new object may
                       have been added */
    unsigned long long int dlpi_subs;
                    /* Incremented when an object may
                       have been removed */
    size_t dlpi_tls_modid;
                    /* If there is a PT_TLS segment, its module
                       ID as used in TLS relocations, else zero */
    void  *dlpi_tls_data;
                    /* The address of the calling thread's instance
                       of this module's PT_TLS segment, if it has
                       one and it has been allocated in the calling
                       thread, otherwise a null pointer */
};
```

(The *ElfW*() macro definition turns its argument into the name of an ELF data type suitable for the hardware architecture. For example, on a 32-bit platform, *ElfW(Addr)* yields the data type name *Elf32_Addr*.

Further information on these types can be found in the *<elf.h>* and *<link.h>* header files.)

The *dlpi_addr* field indicates the base address of the shared object (i.e., the difference between the virtual memory address of the shared object and the offset of that object in the file from which it was loaded). The *dlpi_name* field is a null-terminated string giving the pathname from which the shared object was loaded.

To understand the meaning of the *dlpi_phdr* and *dlpi_phnum* fields, we need to be aware that an ELF shared object consists of a number of segments, each of which has a corresponding program header describing the segment. The *dlpi_phdr* field is a pointer to an array of the program headers for this shared object. The *dlpi_phnum* field indicates the size of this array.

These program headers are structures of the following form:

```
typedef struct {
    Elf32_Word  p_type;    /* Segment type */
    Elf32_Off   p_offset;  /* Segment file offset */
    Elf32_Addr  p_vaddr;   /* Segment virtual address */
    Elf32_Addr  p_paddr;   /* Segment physical address */
    Elf32_Word  p_filesz;  /* Segment size in file */
    Elf32_Word  p_memsz;   /* Segment size in memory */
    Elf32_Word  p_flags;   /* Segment flags */
    Elf32_Word  p_align;   /* Segment alignment */
} Elf32_Phdr;
```

Note that we can calculate the location of a particular program header, *x*, in virtual memory using the formula:

```
addr == info->dlpi_addr + info->dlpi_phdr[x].p_vaddr;
```

Possible values for *p_type* include the following (see *<elf.h>* for further details):

```
#define PT_LOAD         1    /* Loadable program segment */
#define PT_DYNAMIC      2    /* Dynamic linking information */
#define PT_INTERP       3    /* Program interpreter */
#define PT_NOTE         4    /* Auxiliary information */
#define PT_SHLIB        5    /* Reserved */
#define PT_PHDR         6    /* Entry for header table itself */
#define PT_TLS          7    /* Thread-local storage segment */
#define PT_GNU_EH_FRAME 0x6474e550 /* GCC .eh_frame_hdr segment */
#define PT_GNU_STACK  0x6474e551 /* Indicates stack executability */
#define PT_GNU_RELRO  0x6474e552 /* Read-only after relocation */
```

RETURN VALUE

The **dl_iterate_phdr**() function returns whatever value was returned by the last call to *callback*.

VERSIONS

dl_iterate_phdr() has been supported in glibc since version 2.2.4.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	Attribute	Value
dl_iterate_phdr()	Thread safety	MT-Safe

CONFORMING TO

The **dl_iterate_phdr**() function is not specified in any standard. Various other systems provide a version of this function, although details of the returned *dl_phdr_info* structure differ. On the BSDs and Solaris, the structure includes the fields *dlpi_addr*, *dlpi_name*, *dlpi_phdr*, and *dlpi_phnum* in addition to other implementation-specific fields.

NOTES

Future versions of the C library may add further fields to the *dl_phdr_info* structure; in that event, the *size* argument provides a mechanism for the callback function to discover whether it is running on a system with added fields.

The first object visited by *callback* is the main program. For the main program, the *dlpi_name* field will be an empty string.

EXAMPLE

The following program displays a list of pathnames of the shared objects it has loaded. For each shared object, the program lists some information (virtual address, size, flags, and type) for each of the objects ELF segments.

The following shell session demonstrates the output produced by the program on an x86-64 system. The first shared object for which output is displayed (where the name is an empty string) is the main program.

```
$ ./a.out
Name: "" (9 segments)
     0: [      0x400040; memsz:    1f8] flags: 0x5; PT_PHDR
     1: [      0x400238; memsz:     1c] flags: 0x4; PT_INTERP
     2: [      0x400000; memsz:    ac4] flags: 0x5; PT_LOAD
     3: [      0x600e10; memsz:    240] flags: 0x6; PT_LOAD
     4: [      0x600e28; memsz:    1d0] flags: 0x6; PT_DYNAMIC
     5: [      0x400254; memsz:     44] flags: 0x4; PT_NOTE
     6: [      0x400970; memsz:     3c] flags: 0x4; PT_GNU_EH_FRAME
     7: [         (nil); memsz:      0] flags: 0x6; PT_GNU_STACK
     8: [      0x600e10; memsz:    1f0] flags: 0x4; PT_GNU_RELRO
Name: "linux-vdso.so.1" (4 segments)
     0: [0x7ffc6edd1000; memsz:    e89] flags: 0x5; PT_LOAD
     1: [0x7ffc6edd1360; memsz:    110] flags: 0x4; PT_DYNAMIC
     2: [0x7ffc6edd17b0; memsz:     3c] flags: 0x4; PT_NOTE
     3: [0x7ffc6edd17ec; memsz:     3c] flags: 0x4; PT_GNU_EH_FRAME
Name: "/lib64/libc.so.6" (10 segments)
     0: [0x7f55712ce040; memsz:    230] flags: 0x5; PT_PHDR
     1: [0x7f557145b980; memsz:     1c] flags: 0x4; PT_INTERP
     2: [0x7f55712ce000; memsz: 1b6a5c] flags: 0x5; PT_LOAD
     3: [0x7f55716857a0; memsz:   9240] flags: 0x6; PT_LOAD
     4: [0x7f5571688b80; memsz:    1f0] flags: 0x6; PT_DYNAMIC
     5: [0x7f55712ce270; memsz:     44] flags: 0x4; PT_NOTE
     6: [0x7f55716857a0; memsz:     78] flags: 0x4; PT_TLS
     7: [0x7f557145b99c; memsz:   544c] flags: 0x4; PT_GNU_EH_FRAME
     8: [0x7f55712ce000; memsz:      0] flags: 0x6; PT_GNU_STACK
     9: [0x7f55716857a0; memsz:   3860] flags: 0x4; PT_GNU_RELRO
Name: "/lib64/ld-linux-x86-64.so.2" (7 segments)
     0: [0x7f557168f000; memsz:  20828] flags: 0x5; PT_LOAD
     1: [0x7f55718afba0; memsz:   15a8] flags: 0x6; PT_LOAD
     2: [0x7f55718afe10; memsz:    190] flags: 0x6; PT_DYNAMIC
     3: [0x7f557168f1c8; memsz:     24] flags: 0x4; PT_NOTE
     4: [0x7f55716acec4; memsz:    604] flags: 0x4; PT_GNU_EH_FRAME
     5: [0x7f557168f000; memsz:      0] flags: 0x6; PT_GNU_STACK
     6: [0x7f55718afba0; memsz:    460] flags: 0x4; PT_GNU_RELRO
```

Program source

```
#define _GNU_SOURCE
#include <link.h>
#include <stdlib.h>
```

```
#include <stdio.h>

static int
callback(struct dl_phdr_info *info, size_t size, void *data)
{
    char *type;
    int p_type, j;

    printf("Name: \"%s\" (%d segments)\n", info->dlpi_name,
           info->dlpi_phnum);

    for (j = 0; j < info->dlpi_phnum; j++) {
        p_type = info->dlpi_phdr[j].p_type;
        type =  (p_type == PT_LOAD) ? "PT_LOAD" :
                (p_type == PT_DYNAMIC) ? "PT_DYNAMIC" :
                (p_type == PT_INTERP) ? "PT_INTERP" :
                (p_type == PT_NOTE) ? "PT_NOTE" :
                (p_type == PT_INTERP) ? "PT_INTERP" :
                (p_type == PT_PHDR) ? "PT_PHDR" :
                (p_type == PT_TLS) ? "PT_TLS" :
                (p_type == PT_GNU_EH_FRAME) ? "PT_GNU_EH_FRAME" :
                (p_type == PT_GNU_STACK) ? "PT_GNU_STACK" :
                (p_type == PT_GNU_RELRO) ? "PT_GNU_RELRO" : NULL;

        printf("    %2d: [%14p; memsz:%7lx] flags: 0x%x; ", j,
                (void *) (info->dlpi_addr + info->dlpi_phdr[j].p_vaddr),
                info->dlpi_phdr[j].p_memsz,
                info->dlpi_phdr[j].p_flags);
        if (type != NULL)
            printf("%s\n", type);
        else
            printf("[other (0x%x)]\n", p_type);
    }

    return 0;
}

int
main(int argc, char *argv[])
{
    dl_iterate_phdr(callback, NULL);

    exit(EXIT_SUCCESS);
}
```

SEE ALSO

ldd(1), **objdump**(1), **readelf**(1), **dladdr**(3), **dlopen**(3), **elf**(5), **ld.so**(8)

Executable and Linking Format Specification, available at various locations online.

COLOPHON

This page is part of release 5.05 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/.