NAME

segmap – map device memory into user space

SYNOPSIS

#include <sys/types.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/vm.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>

int *prefix***segmap(dev_t** *dev,* **off_t** *off,* **struct as** **asp,* **caddr_t** **addrp,* **off_t** *len,*
unsigned int *prot,* **unsigned int** *maxprot,* **unsigned int** *flags,* **cred_t** **cred_p***);**

INTERFACE LEVEL

Architecture independent level 2 (DKI only).

ARGUMENTS

dev Device whose memory is to be mapped.

off Offset within device memory at which mapping begins.

asp Pointer to the address space into which the device memory should be mapped.

addrp Pointer to the address in the address space to which the device memory should be mapped.

len Length (in bytes) of the memory to be mapped.

prot A bit field that specifies the protections. Possible settings are:

- **PROT_READ** Read access is desired.
- **PROT_WRITE** Write access is desired.
- **PROT_EXEC** Execute access is desired.
- **PROT_USER** User-level access is desired (the mapping is being done as a result of a **mmap**(2) system call).
- **PROT_ALL** All access is desired.

maxprot Maximum protection flag possible for attempted mapping (the **PROT_WRITE** bit may be masked out if the user opened the special file read-only). If **(max-prot & prot) != prot** then there is an access violation.

flags Flags indicating type of mapping. Possible values are (other bits may be set):

- **MAP_SHARED** Changes should be shared.
- **MAP_PRIVATE** Changes are private.

cred_p Pointer to the user credentials structure.

DESCRIPTION

The **segmap( )** entry point is an optional routine for character drivers that support memory mapping. The **mmap**(2) system call, when applied to a character special file, allows device memory to be mapped into user space for direct access by the user application.

Typically, a character driver that needs to support the **mmap**(2) system call supplies either an **mmap**(9E) entry point, or both an **mmap** (9E) and a **segmap( )** entry point routine (see the **mmap**(9E) reference page). If no **segmap( )** entry point is provided for the driver, **ddi_segmap**(9F) is used as a default.

A driver for a memory-mapped device would provide a **segmap( )** entry point if it:

- needs to maintain a separate context for each user mapping. See **ddi_mapdev**(9F) for details.
- needs to assign device access attributes to the user mapping. See **ddi_segmap_setup**(9F) and **ddi_mapdev_set_device_acc_attr**(9F) for details.

The responsibilities of a **segmap( )** entry point are:

- Verify that the range, defined by *offset* and *len*, to be mapped is valid for the device. Typically, this task is performed by calling the **mmap**(9E) entry point. Note that if you are using **ddi_segmap**( ) to set up the mapping, it will call your **mmap**(9E) entry point for you to validate the range to be mapped.
- Assign device access attributes to the mapping. See **ddi_mapdev_set_device_acc_attr**(9F), **ddi_segmap_setup**(9F), and **ddi_device_acc_attr**(9S) for details.
- Set up device contexts for the user mapping if your device requires context switching. See **ddi_mapdev**(9F) for details.
- Perform the mapping with **ddi_segmap**( ), **ddi_segmap_setup**( ), or **ddi_mapdev**( ) and return the status if it fails.

RETURN VALUES

The **segmap( )** routine should return **0** if the driver is successful in performing the memory map of its device address space into the specified address space.

The **segmap**( ) must return an error number on failure. For example, valid error numbers would be **ENXIO** if the offset/length pair specified exceeds the limits of the device memory, or **EINVAL** if the driver detects an invalid type of mapping attempted.

If one of the mapping routines **ddi_segmap**( ), **ddi_segmap_setup**( ), or **ddi_mapdev**( ) fails, you must return the error number returned by the respective routine.

SEE ALSO

mmap(2), **mmap**(9E), **ddi_mapdev**(9F), **ddi_mapdev_set_device_acc_attr**(9F), **ddi_segmap**(9F), **ddi_segmap_setup**(9F), **ddi_device_acc_attr**(9S)

Writing Device Drivers