NAME

ddi_mapdev – create driver-controlled mapping of device

SYNOPSIS

#include <sys/ddi.h>
#include <sys/sunddi.h>

int ddi_mapdev(dev_t *dev*, **off_t** *offset*, **struct as** **asp*, **caddr_t** **addrp*, **off_t** *len*, **u_int** *prot*, **u_int** *maxprot*, **u_int** *flags*, **cred_t** **cred*, **struct ddi_mapdev_ctl** **ctl*, **ddi_mapdev_handle_t** **handlep*, **void** **devprivate*);

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

dev	The device whose memory is to be mapped.
offset	The offset within device memory at which the mapping begins.
as	An opaque pointer to the user address space into which the device memory should be mapped.
addrp	Pointer to the starting address within the user 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.
maxprot	Maximum protection flag possible for attempted mapping.
flags	Flags indicating type of mapping.
cred	Pointer to the user credentials structure.
ctl	A pointer to a **ddi_mapdev_ctl**(9S) structure. The structure contains pointers to device driver-supplied functions that manage events on the device mapping.
handlep	An opaque pointer to a device mapping handle. A handle to the new device mapping is generated and placed into the location pointed to by **handlep*. If the call fails, the value of **handlep* is undefined.
devprivate	Driver private mapping data. This value is passed into each mapping call back routine.

DESCRIPTION

ddi_mapdev( ) sets up user mappings to device space. The driver is notified of user events on the mappings via the entry points defined by *ctl*.

The user events that the driver is notified of are:

access	User has accessed an address in the mapping that has no translations.
duplication	User has duplicated the mapping. Mappings are duplicated when the process calls **fork**(2).
unmapping	User has called **munmap**(2) on the mapping or is exiting.

See **mapdev_access**(9E), **mapdev_dup**(9E), and **mapdev_free**(9E) for details on these entry points.

The range to be mapped, defined by *offset* and *len* must be valid.

The arguments *dev*, *asp*, *addrp*, *len*, *prot*, *maxprot*, *flags*, and *cred* are provided by the **segmap**(9E) entry point and should not be modified. See **segmap**(9E) for a description of these arguments. Unlike **ddi_segmap**(9F), the drivers **mmap**(9E) entry point is not called to verify the range to be mapped.

With the handle, device drivers can use **ddi_mapdev_intercept**(9F) and **ddi_mapdev_nointercept**(9F) to inform the system of whether or not they are interested in being notified when the user process accesses the mapping. By default, user accesses to newly created mappings will generate a call to the **mapdev_access( )** entry point. The driver is always notified of duplications and unmaps.

The device may also use the handle to assign certain characteristics to the mapping. See **ddi_mapdev_set_device_acc_attr**(9F) for details.

The device driver can use these interfaces to implement a device context and control user accesses to the device space. **ddi_mapdev**( ) is typically called from the **segmap**(9E) entry point.

RETURN VALUES

ddi_mapdev( ) returns zero on success and non-zero on failure. The return value from **ddi_mapdev( )** should be used as the return value for the drivers **segmap( )** entry point.

CONTEXT

This routine can be called from user or kernel context only.

SEE ALSO

fork(2), **mmap**(2), **munmap**(2), **mapdev_access**(9E), **mapdev_dup**(9E), **mapdev_free**(9E), **mmap**(9E), **segmap**(9E), **ddi_mapdev_intercept**(9F), **ddi_mapdev_nointercept**(9F), **ddi_mapdev_set_device_acc_attr**(9F), **ddi_segmap**(9F), **ddi_mapdev_ctl**(9S)

Writing Device Drivers

NOTES

Only mappings of type **MAP_PRIVATE** should be used with **ddi_mapdev( )**.