If *mm_ptcreate* successfully created the partition, the *ptid* and *bnum* are filled in and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Too many partitions.

NOTES

Not callable from ISR.

Will not cause a preempt.

3.9 Dual-ported Memory

Dual-ported memory is commonly found in multiprocessor systems. The executive provides a method for converting internal addresses to external, and external addresses to internal, to accommodate the use of dual-ported memory and allow tasks to exchange addresses between processors.

The *internal* address will be defined as the address of a memory resource, relative to the local node which needs to access the memory resource.

The *external* address will be defined as the address of a memory resource, relative to a remote node which needs to access the memory resource.

The directives provided for dual-ported memory are:

Directive	Function
m_ext2int	Convert external address
m_int2ext	Convert internal address

3.9.1 M_EXT2INT

NAME

m_ext2int -- "Convert external address to internal address"

SYNOPSIS

uint m_ext2int (external, &internal)

```
        char *external;     /* external address */
        char *internal;     /* internal address - returned by this call */
```

DESCRIPTION

The *m_ext2int* call is used to convert the physical address contained in *external* into an internal address, so it can be used by the local node. The internal address is returned to the caller in *internal.*

The external (VMEbus) address is normally an address received by the local node, and the requester may not know whether its internal (local) or not. If the address contained in *external* is internal, the returned address will be same as the address in *external.*

RETURN VALUE

The *m_ext2int* directive always succeeds, the internal address is returned in *internal,* and 0 is returned.

ERROR CONDITIONS

None.

NOTES

Can be called from within an ISR.

Will not cause a preempt.

In a MMU system, a task will need to execute *mm_p2l* following this call to obtain a logical internal address.

3.9.2 M_INT2EXT

NAME

m_int2ext -- "Convert internal address to external address"

SYNOPSIS

uint m_int2ext (internal, &external)

```
        char *internal;     /* internal address */
        char *external;     /* external address - returned by this call */
```

DESCRIPTION

The *m_int2ext* call is used to convert the physical address contained in *internal* into an external address, so it can pass the address to a remote node within the system. The external address is returned to the requester in *external*.

The internal address is a physical address accessible by the local node within its dual-ported memory, and the external (VMEbus) address will be different.

RETURN VALUE

The *m_int2ext* directive always succeeds, the external address is returned in *external,* and 0 is returned.

ERROR CONDITIONS

None.

NOTES

Can be called from within an ISR.

Will not cause a preempt.

In a MMU system, a task will need to execute *mm_l2p* preceding this call to obtain a physical address.

4. I/O INTERFACE

This section describes a set of I/O Interface services for the RTEID. These services provide a well defined mechanism for installing and calling device drivers. They provide a structured methodology for writing drivers which both simplifies and assists in the development of drivers and enhances their portability between RTEID based ~~system~~ systems. The RTEID does not make any assumptions about the construction or operation of a driver itself.

The directives provided by the I/O Interface are:

Directive	Description
de_init	Initialize a device driver
de_open	Open a device for I/O
de_close	Close a device
de_read	Read from a device
de_write	Write to a device
de_cntrl	Special device services

4.1 Driver Properties

Device drivers shall have the following properties:

1. A driver is always called by a task and is considered to run on behalf of the task which called it.
2. A driver can make any and all RTEID calls, including additional I/O calls. I/O calls may not be called from within the driver's ISR.
3. If the driver makes a blocking service call, (e.g. *q_receive*), the calling task blocks.
4. Drivers always execute in supervisor mode regardless of the mode of the caller. Designers should account for driver stack usage when determining supervisor stack sizes for new tasks.
5. A driver may temporarily enter user mode but must return to supervisor mode prior to exiting.
6. Other than item (4) above, drivers retain the *mode* of the calling task. Thus on entry they have the same interrupt mask level, preemption, asr and time-slicing status as the caller. The driver may change any or all of these but is responsible for restoring them prior to exiting.

4.2 Data Structures

The data structures used by drivers which are supported by the I/O Interface are:

- **Driver Address Table**
  - Used by the I/O Interface to locate the driver's INIT, OPEN, CLOSE, READ, WRITE, and CNTRL routines.