NAME

maus, getmaus, freemaus, enabmaus, dismaus, switmaus − multiple access user space operations

SYNOPSIS

getmaus (name, mode)
char *name;

freemaus (mausdes)

char *enabmaus (mausdes)

dismaus (vaddr)
char *vaddr;

char *switmaus (mausdes, vaddr)
char *vaddr;

DESCRIPTION

MAUS is a dedicated portion of core memory, which may be subdivided in logical subsections. See *maus*(4) for a discussion of MAUS layout. These subsections are referenced via entries in the UNIX file system which may be used as arguments to the UNIX *open* system call or the *getmaus* system call. Opening such a special file results in a file descriptor being returned which may be subsequently used with other file system calls like *read*, *write*, *seek* and *close* in the standard manner.

Performing the *getmaus* primitive on a maus special file returns a maus descriptor which is analogous to a file descriptor in many ways. This maus descriptor may be subsequently used with the *enabmaus* primitive to attach the described maus subsection to the user's address space. If the *enabmaus* primitive is used repetitively on the same maus descriptor different virtual addresses will be returned on each call until all memory mapping registers have been used; at which time an error is returned. Note that every active instance of maus requires the allocation of a separate memory mapping register since no register may point to more than one maus segment at a time.

Once *enabmaus* has been used the *dismaus* primitive may be utilized to remove active instances of maus from the user's address space. (In reality, *enabmaus* and *dismaus* are special cases of the *switmaus* primitive described below.)

Finally, *freemaus*, deallocates a maus descriptor so that it may be reassigned by *getmaus*. Note that if a maus descriptor has been enabled it may still be freed: the virtual address returned by *enabmaus* remains in the user's address space until a *dismaus* primitive is utilized on the virtual address in question.

The maus primitives are defined as follows:

if *function* is a **0**, **1**, or **2** (*getmaus*(*name*,*mode*) from C), the maus file described by *argy* (*name* from C) is accessed to determine if the read, write, or read/write permission as specified by *function* (*mode* from C) should be granted to the specified user. This permission check is in accordance with the standard UNIX file protection. The file specified must be a special maus file. This primitive returns a maus descriptor which must be saved for future use with *freemaus* and *enabmaus*. This primitive is similar to the open system call in many respects.

if *function* is **3** (*freemaus*(*mausdes*) from C), the maus descriptor described by *argy* (*mausfes* from C) is deallocated from the process. Any further attempts to use the value as a maus descriptor will result in an error being returned.

if *function* is **4** (*switmaus*(*mausdes*,*vaddr*) from C), the system will select the user data memory mapping register specified by *argy* (*vaddr* from C), and load it so that the maus segment specified by *argx* (*mausdes* from C), becomes part of the user's virtual address

space. When using the C interface, the value returned by *switmaus* is the old maus descriptor associated with *vaddr*; if *vaddr* had not been associated with a maus descriptor, −2 is returned. For the assembly interface, the value returned is a pointer to the start of this maus area which may be used like any assembly pointer, but should be preserved for future maus system calls. If *argx* is a −1, (*dismaus*(*vaddr*) from C), the specified virtual address is removed from the user's address space. The C interface returns the maus descriptor which had been associated with *vaddr*; if *vaddr* had not been enabled then -2 is returned. The assembly interface returns some value not equal to −1 (unless there has been an error). If *argy* is −1 (*enabmaus*(*mdes*) from C), the first available memory mapping register is allocated and used. If both arguments are −1, an error will be returned only if there are no unused user memory mapping registers. An error indication is always returned if no memory mapping registers are available or if an address is specified which is in use for program text, data or stack. When expecting a maus descriptor to be returned, for example after a *dismaus (vaddr)*, a -2 return means that no maus descriptor had been enabled with the virtual address given. In all cases, a −1 return means error.

FILES

/dev/maus/*

RULES OF THE ROAD

1) Maus descriptors are inherited across forks and executes. Note that if the new process executed has text or data which wants to occupy the memory currently open to maus, the execute will fail.

2) Maus virtual addresses are inherited across forks.

3) If the *break* system call is used to increase the user's size to the point where an additional memory mapping register is needed and maus is utilizing the next contiguous memory mapping register, the *break* will fail. The user may then utilize *enabmaus* and *dismaus* to reassign the maus virtual address(es). This can be done by doing successive *enabmaus* system calls until the desired virtual address is reached and then disabling the unneeded addresses before using the *break* system call. Alternatively, the user could disable all the active maus segments, use the *break* system call, and then reenable the maus segments.

4) Since the memory mapping hardware does not allow a write-only segment, when the user requests write-only maus via the *getmaus* primitive he is actually granted read-write permission assuming the file system protection tests pass. Only write permission of the maus special file is tested in this case.

SEE ALSO

break(2), open(2), maus(4)

DIAGNOSTICS

From assembler the error bit is set for any error. From C, a −1 return indicates an error.

ASSEMBLER

(maus = 58.; not in assembler)
(function in R0)
(argx in R1)
sys maus; argy