NAME

set_mempolicy − set default NUMA memory policy for a thread and its children

LIBRARY

NUMA (Non-Uniform Memory Access) policy library (*libnuma*, *−lnuma*)

SYNOPSIS

#include <numaif.h>

long set_mempolicy(int *mode***, const unsigned long ***nodemask***,**
unsigned long *maxnode***);**

DESCRIPTION

set_mempolicy() sets the NUMA memory policy of the calling thread, which consists of a policy mode and zero or more nodes, to the values specified by the *mode*, *nodemask*, and *maxnode* arguments.

A NUMA machine has different memory controllers with different distances to specific CPUs. The memory policy defines from which node memory is allocated for the thread.

This system call defines the default policy for the thread. The thread policy governs allocation of pages in the process's address space outside of memory ranges controlled by a more specific policy set by **mbind**(2). The thread default policy also controls allocation of any pages for memory-mapped files mapped using the **mmap**(2) call with the **MAP_PRIVATE** flag and that are only read (loaded) from by the thread and of memory-mapped files mapped using the **mmap**(2) call with the **MAP_SHARED** flag, regardless of the access type. The policy is applied only when a new page is allocated for the thread. For anonymous memory this is when the page is first touched by the thread.

The *mode* argument must specify one of **MPOL_DEFAULT**, **MPOL_BIND**, **MPOL_INTERLEAVE**, **MPOL_PREFERRED**, or **MPOL_LOCAL** (which are described in detail below). All modes except **MPOL_DEFAULT** require the caller to specify the node or nodes to which the mode applies, via the *nodemask* argument.

The *mode* argument may also include an optional *mode flag*. The supported*mode fla gs* are:

MPOL_F_NUMA_BALANCING (since Linux 5.12)
: When *mode* is **MPOL_BIND**, enable the kernel NUMA balancing for the task if it is supported by the kernel. If the flag isn't supported by the kernel, or is used with *mode* other than **MPOL_BIND**, −1 is returned and *errno* is set to **EINVAL**.

MPOL_F_RELATIVE_NODES (since Linux 2.6.26)
: A nonempty *nodemask* specifies node IDs that are relative to the set of node IDs allowed by the process's current cpuset.

MPOL_F_STATIC_NODES (since Linux 2.6.26)
: A nonempty *nodemask* specifies physical node IDs. Linux will not remap the *nodemask* when the process moves to a different cpuset context, nor when the set of nodes allowed by the process's current cpuset context changes.

nodemask points to a bit mask of node IDs that contains up to *maxnode* bits. The bit mask size is rounded to the next multiple of *sizeof(unsigned long)*, but the kernel will use bits only up to *maxnode*. A NULL value of *nodemask* or a *maxnode* value of zero specifies the empty set of nodes. If the value of *maxnode* is zero, the *nodemask* argument is ignored.

Where a *nodemask* is required, it must contain at least one node that is on-line, allowed by the process's current cpuset context, (unless the **MPOL_F_STATIC_NODES** mode flag is specified), and contains memory. If the**MPOL_F_ST ATIC_NODES** is set in *mode* and a required *nodemask* contains no nodes that are allowed by the process's current cpuset context, the memory policy reverts to *local allocation*. This effectively overrides the specified policy until the process's cpuset context includes one or more of the nodes specified by *nodemask*.

The *mode* argument must include one of the following values:

MPOL_DEFAULT

This mode specifies that any nondefault thread memory policy be removed, so that the memory policy "falls back" to the system default policy. The system default policy is "local allocation"—that is, allocate memory on the node of the CPU that triggered the allocation. *nodemask* must be specified as NULL. If the "local node" contains no free memory, the system will attempt to allocate memory from a "near by" node.

MPOL_BIND

This mode defines a strict policy that restricts memory allocation to the nodes specified in *nodemask*. If*nodemask* specifies more than one node, page allocations will come from the node with the lowest numeric node ID first, until that node contains no free memory. Allocations will then come from the node with the next highest node ID specified in *nodemask* and so forth, until none of the specified nodes contain free memory. Pages will not be allocated from any node not specified in the *nodemask*.

MPOL_INTERLEAVE

This mode interleaves page allocations across the nodes specified in *nodemask* in numeric node ID order. This optimizes for bandwidth instead of latency by spreading out pages and memory accesses to those pages across multiple nodes. However, accesses to a single page will still be limited to the memory bandwidth of a single node.

MPOL_PREFERRED

This mode sets the preferred node for allocation. The kernel will try to allocate pages from this node first and fall back to "near by" nodes if the preferred node is low on free memory. If*nodemask* specifies more than one node ID, the first node in the mask will be selected as the preferred node. If the*nodemask* and *maxnode* ar guments specify the empty set, then the policy specifies "local allocation" (like the system default policy discussed above).

MPOL_LOCAL (since Linux 3.8)

This mode specifies "local allocation"; the memory is allocated on the node of the CPU that triggered the allocation (the "local node"). The *nodemask* and *maxnode* arguments must specify the empty set. If the "local node" is low on free memory, the kernel will try to allocate memory from other nodes. The kernel will allocate memory from the "local node" whenever memory for this node is available. If the "local node" is not allowed by the process's current cpuset context, the kernel will try to allocate memory from other nodes. The kernel will allocate memory from the "local node" whenever it becomes allowed by the process's current cpuset context.

The thread memory policy is preserved across an **execve**(2), and is inherited by child threads created using **fork**(2) or **clone**(2).

RETURN VALUE

On success, **set_mempolicy**() returns 0; on error, −1 is returned and *errno* is set to indicate the error.

ERRORS

EFAULT

Part of all of the memory range specified by *nodemask* and *maxnode* points outside your accessible address space.

EINVAL

mode is invalid. Or, *mode* is **MPOL_DEFAULT** and *nodemask* is nonempty, or *mode* is **MPOL_BIND** or **MPOL_INTERLEAVE** and *nodemask* is empty. Or, *maxnode* specifies more than a page worth of bits. Or, *nodemask* specifies one or more node IDs that are greater than the maximum supported node ID. Or, none of the node IDs specified by *nodemask* are on-line and allowed by the process's current cpuset context, or none of the specified nodes contain memory. Or, the *mode* argument specified both **MPOL_F_STATIC_NODES** and **MPOL_F_RELATIVE_NODES**. Or, the **MPOL_F_NUMA_BALANCING** isn't supported by the kernel, or is used with *mode* other than **MPOL_BIND**.

ENOMEM
Insufficient kernel memory was available.

VERSIONS

The **set_mempolicy**() system call was added in Linux 2.6.7.

STANDARDS

This system call is Linux-specific.

NOTES

Memory policy is not remembered if the page is swapped out. When such a page is paged back in, it will use the policy of the thread or memory range that is in effect at the time the page is allocated.

For information on library support, see **numa**(7).

SEE ALSO

get_mempolicy(2), **getcpu**(2), **mbind**(2), **mmap**(2), **numa**(3), **cpuset**(7), **numa**(7), **numactl**(8)