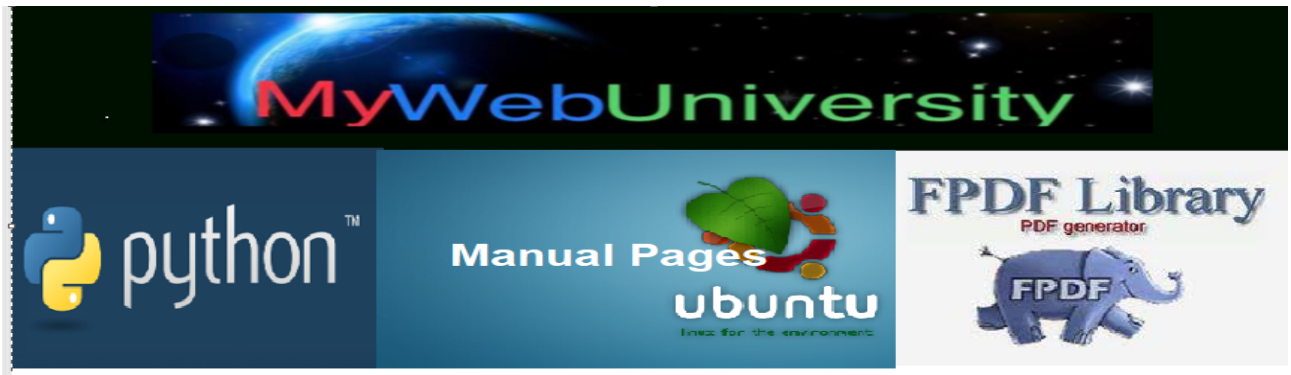

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'mbind.2'

$ man mbind.2

mbind(2) System Calls Manual mbind(2)

NAME

mbind - set memory policy for a memory range

LIBRARY

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

SYNOPSIS

#include <numaif.h>

long mbind(unsigned long size, unsigned long maxnode;

void addr[size], unsigned long size, int mode,

const unsigned long nodemask[(maxnode + ULONG_WIDTH - 1)

/ ULONG_WIDTH],

unsigned long maxnode, unsigned int flags);

DESCRIPTION

mbind() sets the NUMA memory policy, which consists of a policy mode and zero or more nodes, for the memory range starting with addr and continuing for size bytes. The memory policy defines from which node memory is allocated.

If the memory range specified by the addr and size arguments includes an "anonymous" region of memory?that is a region of memory created using the mmap(2) system call with the MAP_ANONYMOUS?or a memory-mapped file, mapped using the mmap(2) system call with the MAP_PRIVATE flag, pages will be allocated only according to the specified policy when the application writes (stores) to the page. For anonymous regions, an initial read access will use a shared page in the kernel containing all zeros. For a file mapped with MAP_PRIVATE, an initial read access will allocate pages according to the memory policy of the thread that

causes the page to be allocated. This may not be the thread that called mbind().

The specified policy will be ignored for any MAP_SHARED mappings in the specified memory range. Rather the pages will be allocated according to the memory policy of the thread that caused the page to be allocated. Again, this may not be the thread that called mbind().

If the specified memory range includes a shared memory region created using the shmget(2) system call and attached using the shmat(2) system call, pages allocated for the anonymous or shared memory region will be allocated according to the policy specified, regardless of which process attached to the shared memory segment causes the allocation. If, however, the shared memory region was created with the SHM_HUGETLB flag, the huge pages will be allocated according to the policy specified only if the page allocation is caused by the process that calls mbind() for that region.

By default, mbind() has an effect only for new allocations; if the pages inside the range have been already touched before setting the policy, then the policy has no effect. This default behavior may be overridden by the MPOL_MF_MOVE and MPOL_MF_MOVE_ALL flags described below.

The mode argument must specify one of MPOL_DEFAULT, MPOL_BIND, MPOL_INTERLEAVE, MPOL_WEIGHTED_INTERLEAVE, MPOL_PREFERRED, MPOL_PREFERRED_MANY, or MPOL_LOCAL (which are de?

scribed in detail below). All policy modes except MPOL_DEFAULT require the caller to spec? ify 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 flags are:

MPOL_F_NUMA_BALANCING (since Linux 5.15)

When mode is MPOL_BIND, enable the kernel NUMA balancing for the task if it is sup? ported 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_STATIC_NODES (since Linux-2.6.26)

A nonempty nodemask specifies physical node IDs. Linux does not remap the nodemask when the thread moves to a different cpuset context, nor when the set of nodes al? lowed by the thread's current cpuset context changes.

MPOL_F_RELATIVE_NODES (since Linux-2.6.26)

A nonempty nodemask specifies node IDs that are relative to the set of node IDs al? lowed by the thread's current cpuset.

nodemask points to a bit mask of nodes containing 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 thread's cur? rent cpuset context (unless the MPOL_F_STATIC_NODES mode flag is specified), and contains memory.

The mode argument must include one of the following values:

MPOL_DEFAULT

This mode requests that any nondefault policy be removed, restoring default behavior. When applied to a range of memory via mbind(), this means to use the thread memory policy, which may have been set with set_mempolicy(2). If the mode of the thread memory policy is also MPOL_DEFAULT, the system-wide default policy will be used. The system-wide default policy allocates pages on the node of the CPU that triggers the allocation. For MPOL_DEFAULT, the nodemask and maxnode arguments must be specify the empty set of nodes.

MPOL_BIND

This mode specifies 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 sufficient free memory that is closest to the node where the allocation takes place. Pages will not be allocated from any node not specified in the IR nodemask . (Before Linux 2.6.26, page allocations came from the node with the lowest numeric node ID first, until that node contained no free memory. Alloca? tions then came from the node with the next highest node ID specified in nodemask and so forth, until none of the specified nodes contained free memory.)

MPOL_INTERLEAVE

This mode specifies that page allocations be interleaved across the set of nodes specified in nodemask. This optimizes for bandwidth instead of latency by spreading out pages and memory accesses to those pages across multiple nodes. To be effective the memory area should be fairly large, at least 1 MB or bigger with a fairly uniform access pattern. Accesses to a single page of the area will still be limited to the memory bandwidth of a single node.

MPOL_WEIGHTED_INTERLEAVE (since Linux 6.9)

This mode interleaves page allocations across the nodes specified in nodemask accord?

ing to the weights in /sys/kernel/mm/mempolicy/weighted_interleave. For example, if bits 0, 2, and 5 are set in nodemask, and the contents of /sys/kernel/mm/mempol? icy/weighted_interleave/node0, /sys/.../node2, and /sys/.../node5 are 4, 7, and 9, respectively, then pages in this region will be allocated on nodes 0, 2, and 5 in a 4:7:9 ratio.

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 other nodes if the preferred nodes 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 arguments specify the empty set, then the memory is allocated on the node of the CPU that trig? gered the allocation.

MPOL_PREFERRED_MANY (since Linux 5.15)

Specifies a set of nodes for allocation; see set_mempolicy(2).

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 argu? ments 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 thread'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 thread's current cpuset context. By con? trast, MPOL_DEFAULT reverts to the memory policy of the thread (which may be set via set_mempolicy(2)); that policy may be something other than "local allocation".

If MPOL_MF_STRICT is passed in flags and mode is not MPOL_DEFAULT, then the call fails with the error EIO if the existing pages in the memory range don't follow the policy.

If MPOL_MF_MOVE is specified in flags, then the kernel will attempt to move all the existing pages in the memory range so that they follow the policy. Pages that are shared with other processes will not be moved. If MPOL_MF_STRICT is also specified, then the call fails with the error EIO if some pages could not be moved. If the MPOL_INTERLEAVE policy was speci? fied, pages already residing on the specified nodes will not be moved such that they are in? terleaved.

If MPOL_MF_MOVE_ALL is passed in flags, then the kernel will attempt to move all existing pages in the memory range regardless of whether other processes use the pages. The calling thread must be privileged (CAP_SYS_NICE) to use this flag. If MPOL_MF_STRICT is also speci? fied, then the call fails with the error EIO if some pages could not be moved. If the MPOL_INTERLEAVE policy was specified, pages already residing on the specified nodes will not be moved such that they are interleaved.

RETURN VALUE

On success, mbind() returns 0; on error, -1 is returned and errno is set to indicate the er? ror.

ERRORS

EFAULT Part or all of the memory range specified by nodemask and maxnode points outside your accessible address space. Or, there was an unmapped hole in the specified memory range specified by addr and size.

EINVAL An invalid value was specified for flags or mode; or addr + size was less than addr; or addr is not a multiple of the system page size. Or, mode is MPOL_DEFAULT and nodemask specified a nonempty set; or mode is MPOL_BIND or MPOL_INTERLEAVE and node? mask is empty. Or, maxnode exceeds a kernel-imposed limit. 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 thread'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.

EIO MPOL_MF_STRICT was specified and an existing page was already on a node that does not follow the policy; or MPOL_MF_MOVE or MPOL_MF_MOVE_ALL was specified and the kernel was unable to move all existing pages in the range.

ENOMEM Insufficient kernel memory was available.

EPERM The flags argument included the MPOL_MF_MOVE_ALL flag and the caller does not have the CAP_SYS_NICE privilege.

STANDARDS

Linux.

HISTORY

Linux 2.6.7.

Support for huge page policy was added with Linux 2.6.16. For interleave policy to be ef? fective on huge page mappings the policied memory needs to be tens of megabytes or larger.

Before Linux 5.7. MPOL_MF_STRICT was ignored on huge page mappings.

MPOL_MF_MOVE and MPOL_MF_MOVE_ALL are available only on Linux 2.6.16 and later.

NOTES

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

NUMA policy is not supported on a memory-mapped file range that was mapped with the MAP_SHARED flag.

The MPOL_DEFAULT mode can have different effects for mbind() and set_mempolicy(2). When MPOL_DEFAULT is specified for set_mempolicy(2), the thread's memory policy reverts to the system default policy or local allocation. When MPOL_DEFAULT is specified for a range of memory using mbind(), any pages subsequently allocated for that range will use the thread's memory policy, as set by set_mempolicy(2). This effectively removes the explicit policy from the specified range, "falling back" to a possibly nondefault policy. To select ex? plicit "local allocation" for a memory range, specify a mode of MPOL_LOCAL or MPOL_PREFERRED with an empty set of nodes. This method will work for set_mempolicy(2), as well.

SEE ALSO

get_mempolicy(2), getcpu(2), mmap(2), set_mempolicy(2), shmat(2), shmget(2), numa(3), cpuset(7), numa(7), numactl(8)